Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES INFORMATIQUES

Conception assistée par ordinateur d'un nœud SOPHO-NET

Dazard, Olivier; Mahiat, Cécile

Award date:
1986

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 14. May. 2024

Facultés Universitaires N. D. de la Paix Namur

Institut d'Informatique

CONCEPTION ASSISTEE PAR ORDINATEUR D'UN NOEUD SOPHO-NET

Dazard Olivier
Mahiat Cécile

Promoteur : Philippe Van Bastelaer

Mémoire présenté
en vue de l'obtention
du titre de
Licencié et Maitre
en Informatique

Année Académique 1985 - 1986

PREFACE

Nous tenons tout d'abord à remercier monsieur Philippe Van Bastelaer, promoteur de ce mémoire, pour l'intérêt qu'il a porté à notre étude ainsi que pour les nombreux conseils qu'il nous a prodigués tout au long de la réalisation et de la rédaction du mémoire.

Nous remercions également messieurs Marc Dieudonné et Jules Georges de Philips Telesoft Internationnal pour l'accueil et l'aide qu'ils nous ont apportés tout au long des recherches nécessaires à la réalisation de ce mémoire.

Notre recherche bibliographique a été facilitée grâce à l'aimable collaboration du Laboratoire de Recherche Philips à Bruxelles et du professeur Ed. Coffman (Bell Labs).

Enfin, nous désirons exprimer toute notre gratitude envers les membres de l'institut d'informatique de Namur qui directement ou indirectement ont contribué à faire progresser notre travail. Nous pensons plus particulièrement à monsieur Jean-Pol Leclercq pour ses explications à nos problèmes mathématiques et à mademoiselle Béatrice Scoyer qui nous a fourni toute l'aide matérielle nécessaire à la rédaction de ce mémoire dans les meilleures conditions possibles.

INTRODUCTION GENERALE

L'objectif de ce mémoire est de présenter une méthode de conception assistée par ordinateur, (C.A.O.) pour une étape de la conception d'un noeud de réseau SOPHO-NET, développé par PHILIPS TELESOFT INTERNATIONAL. Un noeud SOPHO-NET est construit autour d'un ensemble de cartes possédant chacune une mémoire et un processeur.

Le projet de C.A.O. que nous présentons consiste à assigner à des cartes les programmes nécessaires à la mise en oeuvre des fonctions d'un noeud SOPHO-NET; cette assignation doit être telle que le nombre de cartes déterminé soit minimum et suffisant. La singularité de notre projet réside dans le fait que les quantités de mémoire et de CPU nécessaires pour réaliser chaque fonction d'un noeud ne sont pas constantes mais varient dans certaines conditions.

Dans le premier chapitre, nous décrivons les principaux concepts d'un réseau SOPHO-NET, tant sur le plan matériel que logiciel.

Dans le deuxième chapitre, nous analysons en profondeur les concepts logiciels de base entrant dans la conception d'un noeud SOPHO-NET.

Dans le troisième chapitre, nous analysons la démarche suivie actuellement pour concevoir un noeud de réseau SOPHO-NET. Mettant alors en évidence les lacunes d'une étape de cette démarche, nous montrons la pertinence du projet de C.A.O. que nous avons entrepris afin de modifier cette étape. Dans ce même chapitre, nous définissons le problème que nous avons à résoudre ainsi que ses données et résultats.

Nous sommes alors en mesure, au cours du quatrième chapitre, d'exposer la méthode retenue pour résoudre le problème défini; cette méthode est appelée la "méthode de fusion".

Dans le cinquième chapitre, nous décrivons les méthodes qui furent une progression vers la méthode de fusion, ainsi que les raisons pour lesquelles nous avons dû les abandonner.

Dans le sixième chapitre, nous comparons la méthode de fusion ou certaines de ses étapes à d'autres méthodes provenant de la littérature. Nous gardons une attitude critique et constructive en décrivant les limites de ces méthodes, mais également en dégageant des enseignements pour notre propre méthode.

Dans le septième et dernier chapitre, nous présentons le dossier de programmation relatif au programme en langage Pascal que nous avons développé pour mettre en oeuvre la méthode de fusion. Ce programme

porte le nom de SYANODE : System Aid for NOde DEsign.

En conclusion, nous présentons une synthèse de notre mémoire ainsi que ses limites et ses extensions possibles.

Nous signalons au lecteur qu'une liste des abréviations rencontrées au cours de ce mémoire se trouve en annexe I ainsi qu'une liste de définitions des concepts en annexe II. Remarquons que nous mettons entre crochets "[ ]" le numéro des références bibliographiques lorsqu'elles apparaissent dans le texte. Le code source de notre programme se trouve en annexe VIII et est disponible soit auprès de monsieur J. Georges de "Philips Telesoft Internationnal" à Bruxelles, soit auprès de monsieur Ph. Van Bastelaer, professeur aux Facultés Universitaires N.D. de la Paix à Namur.

INTRODUCTION AU RESEAU SOPHO-NET

CHAPITRE I

1 Introduction

Dans la première partie de ce chapitre, nous décrivons de manière générale le réseau SOPHO-NET. Nous partons de considérations générales sur les réseaux existants afin d'en mettre en évidence les lacunes. Nous énumérons ensuite les aspects originaux de SOPHO-NET qui se veut une solution souple et adaptée aux besoins d'un utilisateur de réseau ainsi qu'un moyen de combler les lacunes qui auront été énumérées.

La deuxième partie du chapitre est consacrée à la description matérielle du réseau. Nous y appliquons une analyse de type descendant. Nous commençons par une étude détaillée de la structure matérielle d'un noeud SOPHO-NET. Cela nous amène tout naturellement à décrire chaque composant d'un noeud et en particulier, son élément clé : la carte CP (Processing Card).

La troisième partie de ce chapitre nous permet d'analyser les notions logicielles propres à SOPHO-NET.

2 Description générale d'un réseau SOPHO-NET

2.1 Introduction générale aux réseaux

Un réseau SOPHO-NET, Synergetic Philips Open Network, est un réseau privé à commutation de paquets, de haute performance. Le mot privé doit ici être pris dans son sens général; en effet, ce réseau est constitué d'un ensemble de noeuds appartenant à une organisation, construits et installés par Philips, reliés par des moyens de transmission très diversifiés et en particulier publics. Dans cette optique, réseaux privé et public sont deux notions complémentaires plutôt que concurrentes. Les réseaux privés ont essayé de combler certaines lacunes des réseaux publics. Ces lacunes sont les suivantes :

- il y a une prolifération des technologies de communication offertes par les services publics. Ces technologies ont des performances fort différentes, elles sont aussi d'âges différents. Lorsque les utilisateurs de réseaux veulent faire coexister ces technologies, il est nécessaire de cacher à ces utilisateurs ainsi qu'à leurs applications, les particularités propres à ces technologies;

- le standard X25 est réalisé de manière différente selon les pays. Ainsi, certains réseaux utilisent le datagramme, d'autres le circuit virtuel, etc... . De plus, certaines facilités proposées par X25 ne sont pas mises en oeuvre partout. L'utilisation de ce protocole fait donc surgir des problèmes de compatibilité de service dans le cadre d'échanges internationnaux;

- les réseaux publics sont caractérisés par la richesse de leurs services, mais aussi par leur complexité à l'usage. De plus, l'utilisateur ayant déjà sa propre interface doit opérer une

conversion pour pouvoir utiliser le réseau public;

- l'usage des réseaux publics peut être coûteux. Cela incite les utilisateurs de ces réseaux à mettre en oeuvre des politiques assez lourdes pour en faire un usage plus économique. Ainsi, le multiplexage des données sur les circuits virtuels constitue une politique possible d'économie. Dans certaines conditions, le problème de coût est moins crucial : le coût d'usage d'un réseau public tel que "DCS" est avantageux lorsque le trafic y est faible;

- l'usage d'équipements de types et de natures variés engendre des difficultés d'adaptation de leurs protocoles lorsqu'ils utilisent le réseau public.

D'autre part, la naissance de besoins nouveaux a favorisé l'apparition de moyens plus performants. Ces nouveaux besoins sont notamment :

- l'usage de réseaux locaux qui utilisent des techniques tout à fait différentes de celles utilisées par les réseaux publics. Philips a développé un réseau local appelé "SOPHO-LAN";

- la nécessité d'échanger des messages de types de plus en plus variés. Nous pouvons songer à la transmission d'images, de facsimilés, etc... Or il semble que ces types de messages aient évolué plus rapidement que les moyens de les transmettre.

2.2 Aspects originaux de SOPHO-NET

Les concepteurs de SOPHO-NET affirment dans [12] et [15] que SOPHO-NET est un moyen approprié pour combler les lacunes énumérées dans le point précédent et pour répondre aux besoins nouveaux. Toutefois, ces affirmations ne peuvent être confirmées ou infirmées que sur base de l'expérience des utilisateurs d'un réseau SOPHO-NET. Nous ne disposons malheureusement pas de cette information. Nous présentons donc les caractéristiques de SOPHO-NET avec certaines réserves.

Un réseau SOPHO-NET, dont un exemple est présenté à la figure 1.1, est constitué d'un ensemble de noeuds (les Ni sur la figure 1.1) reliés par des moyens de transmission. La localisation géographique des noeuds est déterminée en fonction des besoins, du nombre et de la localisation des utilisateurs. Le concepteur du réseau tient compte également des performances et des coûts des moyens de transmission disponibles. Généralement, un réseau SOPHO-NET est maillé mais des structures plus simples sont évidemment possibles. Les utilisateurs du réseau, qui peuvent être des ordinateurs ou terminaux de n'importe quel type, sont connectés à certains noeuds; le noeud d'accès au réseau d'un utilisateur est appelé son noeud "parent". Les concepteurs de SOPHO-NET précisent que l'interface d'accès à ce réseau est conviviale et aisée. Sur la figure 1.1, le noeud N2 est le noeud parent de l'utilisateur #1.

Un réseau SOPHO-NET peut utiliser simultanément différents types de ressources de transmission telles qu'un réseau X25, un réseau téléphonique (RTCP), des lignes louées (ce moyen est le plus utilisé actuellement), un PABX, des satellites, etc...(voir figure 1.1). Le réseau masque complètement aux utilisateurs et aux applications les caractéristiques propres à chaque moyen de transmission utilisé.

Un réseau SOPHO-NET permet de communiquer des informations telles que des textes, des images etc... selon les potentialités des terminaux connectés et cela sur des distances quelconques.

Une caractéristique fondamentale d'un réseau SOPHO-NET est qu'il accepte tout protocole de départ de l'utilisateur. Le réseau réalise lui-même les conversions de protocole nécessaires pour permettre à des utilisateurs de types différents de communiquer; cette conversion est effectuée au niveau du noeud parent de l'utilisateur. Cela évite d'avoir à greffer au niveau de l'utilisateur des éléments matériels ou logiciels coûteux de conversion de protocole . C'est donc le réseau SOPHO-NET qui s'adapte à l'utilisateur et non l'inverse. Remarquons enfin que certains programmes de conversion de protocole n'ont cependant pas encore été développés.

Pour transmettre les données entre noeuds, le réseau travaille selon la méthode de datagramme. Le routage des données dans le réseau est adaptatif. Ce type de routage permet de choisir le chemin le plus rapide entre noeuds et d'éviter les voies de transmission congestionnées. Un noeud prend ses propres décisions de routage; il n'y a pas de contrôle centralisé.

Cette autonomie des noeuds permet des modifications rapides de la structure matérielle du réseau, notamment lors d'adjonctions et de suppressions de noeuds.

Toute transformation logicielle et tout contrôle administratif ou de comptabilité du réseau est assuré par un centre de télé-maintenance autonome appelé "Network Management Center" (NMC). Il existe toujours au moins un NMC par réseau (voir figure 1.1).

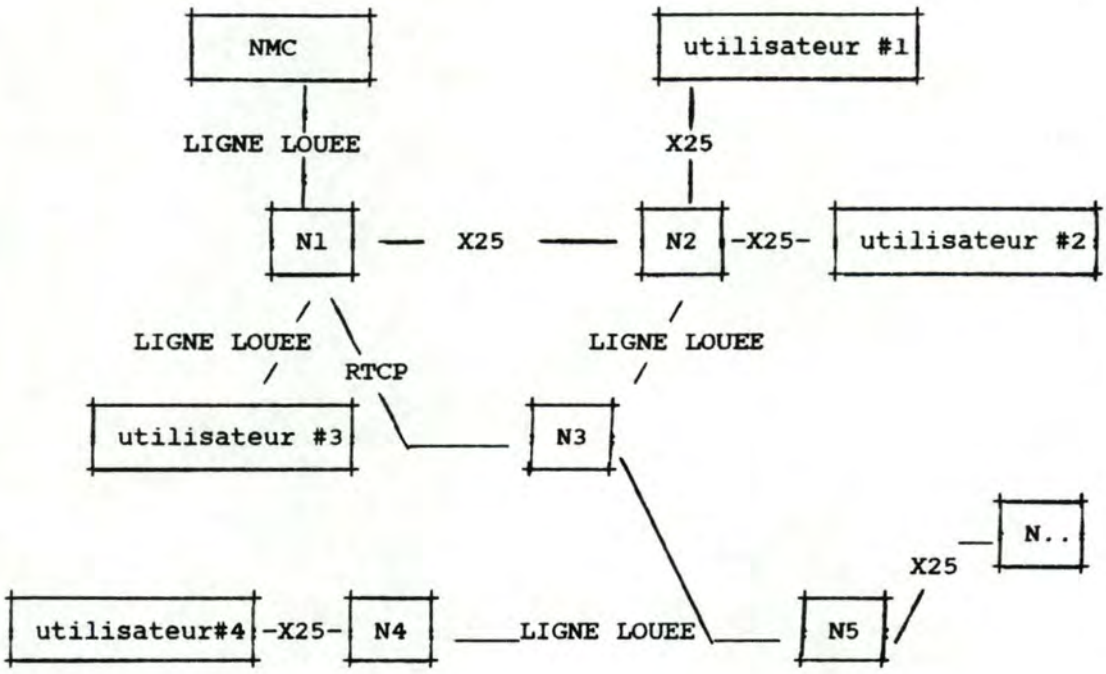


Figure 1.1 : exemple de structure d'un réseau SOPHO-NET

En résumé, les concepteurs du réseau SOPHO-NET affirment que celui-ci comble les lacunes citées dans le point 2.1 :

- en offrant une interface conviviale et aisée d'accès au réseau;
- en masquant entièrement aux utilisateurs et aux applications les moyens de transmission parfois très variés utilisés par le réseau;
- en permettant l'échange de n'importe quel type d'information;
- en permettant une combinaison très variée d'équipements au départ incompatibles, et en facilitant leur usage du réseau en respectant leur protocole;
- en offrant un coût d'usage compétitif par rapport aux réseaux concurrents;
- en offrant une technique puissante et décentralisée de routage adaptatif;
- en permettant des adjonctions et suppressions aisées de noeuds à un réseau;
- en offrant, via le "Network Management Center", des moyens autonomes de maintenance et d'extension du logiciel du réseau sur chaque noeud.

2.3 Cheminement des paquets dans un réseau SOPHO-NET

Les données fournies par un utilisateur sont décomposées en paquets qui peuvent suivre des chemins différents à l'intérieur du réseau jusqu'à leur utilisateur destinataire :

- ces paquets peuvent constituer du trafic local à destination d'un autre utilisateur connecté au même noeud parent;

- ces paquets peuvent être destinés à un utilisateur ayant un autre noeud parent, ils vont alors transiter de noeud en noeud en empruntant, autant que possible, le chemin le plus rapide. L'adresse du destinataire se trouvant dans chaque paquet est examinée par le noeud afin de déterminer si elle correspond à sa propre adresse. Dans l'affirmative, le paquet est transmis à l'utilisateur destinataire connecté à ce noeud. Dans le cas contraire, le noeud transmet le paquet à un de ses noeuds adjacents choisi sur base de sa table de routage;

- il existe également différents trafics de contrôle entre les noeuds afin de réaliser les fonctions de contrôles de flux, de récupérations d'erreurs, etc...

3 Environnement matériel de SOPHO-NET

3.1 Description d'un noeud SOPHO-NET

Chaque noeud SOPHO-NET est un réseau local de cartes de différents types reliées par des bus de natures variées (voir figure 1.2). Ce réseau est appelé "MARC 16 SYSTEM" (Multi ARrangement for Communication 16 bits System). L'élément moteur d'un noeud SOPHO-NET est la "carte CP" (Processing Card).

Les cartes CP contiennent les programmes qui réalisent les fonctions d'un noeud de réseau. Ces programmes peuvent s'échanger de l'information qu'ils soient sur la même carte CP ou non. Les cartes CP doivent donc pouvoir communiquer entre elles; un ou plusieurs bus appelés "bus CP" permettent la communication entre les cartes CP : il n'y a pas de mémoire commune. Sur la figure 1.2, les cartes CP #11 et #12 peuvent communiquer via le bus CP #1.

Un noeud SOPHO-NET comprend aussi des cartes appelées "cartes FTC" (Fast Transfer Card) qui permettent la communication entre les bus CP. Sur la figure 1.2, la communication entre les bus CP #1 et #2 n'est possible que via les cartes FTC #14 et #23.

La communication entre noeuds SOPHO-NET est possible grâce à des lignes externes connectées à des cartes appelées "cartes I/O". Les cartes I/O sont sous le contrôle des cartes CP. Sur la figure 1.2, les cartes I/O #1 et #2 sont sous le contrôle de la carte CP #13.

Les cartes I/O et CP sont reliées par un bus dit "bus I/O". Le bus I/O permet l'échange de caractères entre les cartes CP et I/O. Ces caractères proviennent des lignes externes connectées aux cartes I/O ou partent vers ces lignes externes. Ainsi, sur la figure 1.2, la carte CP #13 peut, via le bus I/O #1 et les cartes I/O #1 et #2, recevoir des caractères des lignes externes connectées à ces cartes I/O ou envoyer des caractères sur ces lignes externes.

Chaque bus CP est géré par une Bus Arbitration Unit (BAU). Une BAU est aussi une carte qui a pour fonction de gérer les conflits d'accès concurrents des cartes CP au bus CP. Lorsqu'une carte CP désire envoyer des informations à une autre carte CP, elle doit demander l'usage du bus CP à la BAU. Cette dernière avertit la carte CP lorsque le bus CP est libre. Ainsi, sur la figure 1.2, la carte CP #21 peut communiquer avec la carte CP #22 via le bus CP #2. Elle utilise ce bus CP après une demande d'usage du bus acceptée par la BAU #2. Il se peut que la carte CP de destination demandée ne se trouve pas sur le même bus CP que la carte CP d'origine de l'échange. Il est alors nécessaire de faire transiter les informations par les cartes FTC. Ainsi, sur la figure 1.2, si la carte CP #11 située sur le bus CP #1 désire envoyer de l'information à la carte CP #31 située sur le bus CP #3, elle doit faire transiter ses informations par les cartes FTC #15 et #34.

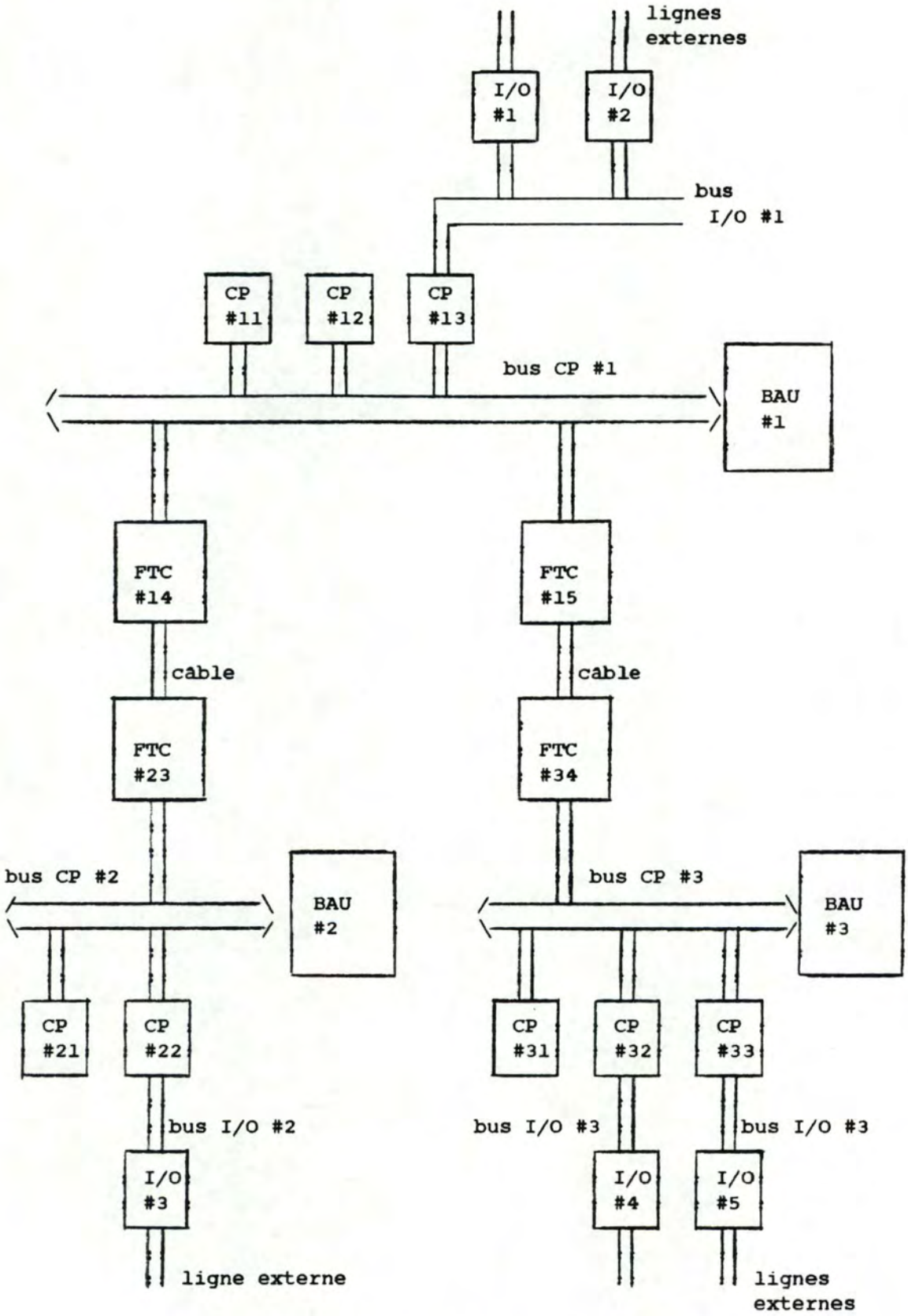


Figure 1.2 : schéma logique d'un noeud SOPHO-NET

Physiquement, un noeud SOPHO-NET est constitué d'une ou de plusieurs étagères, (voir figure 1.3). Sur chaque "panier" ou "shelf" de ces étagères, le constructeur place les cartes CP, I/O, FTC et BAU nécessaires dans des glissières prévues à cet effet. Les cartes placées dans les "paniers" sont ainsi connectées aux bus CP et I/O qui se trouvent au fond du "panier". Les câbles reliant les cartes FTC assurent les communications entre des cartes CP situées dans des "paniers" différents; chacun de ces "paniers" contient une carte BAU. Remarquons enfin que, étant donné que le nombre de glissières est limité, le nombre de cartes par "panier", par étagère et par noeud est limité.

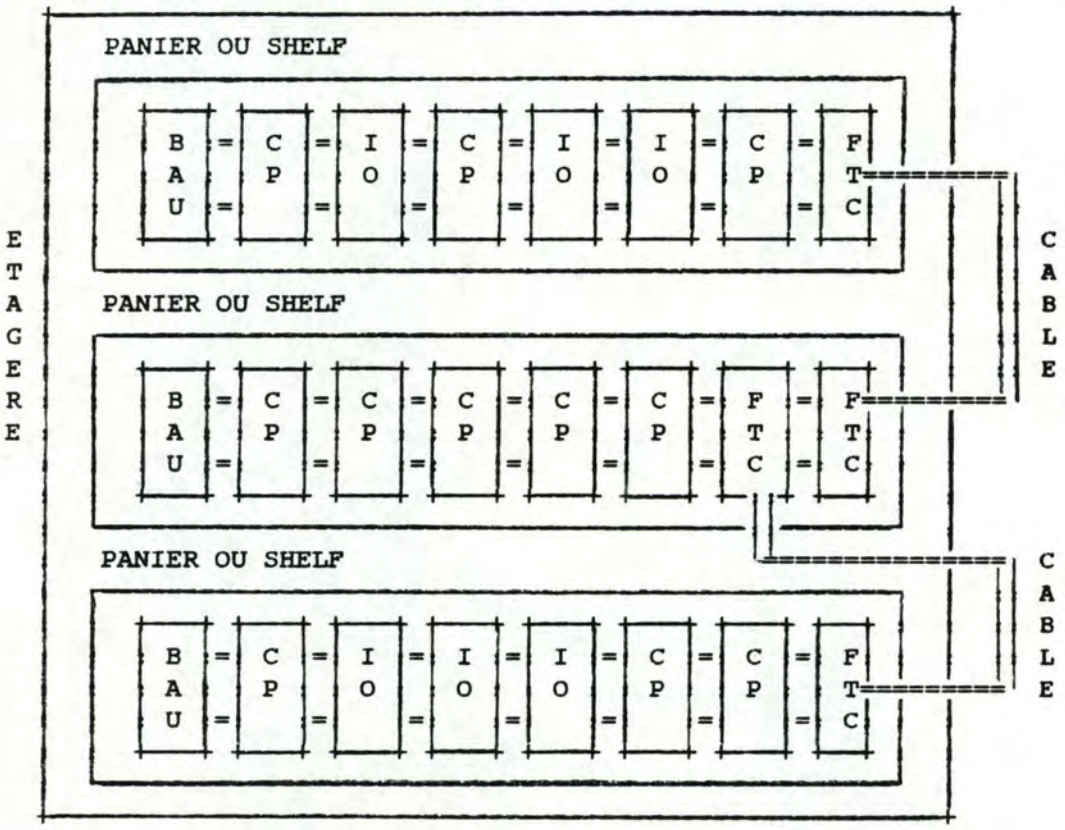


Figure 1.3 : structure physique simplifiée d'une étagère d'un noeud SOPHO-NET

3.2 Description de la carte CP

La carte CP que nous étudions ici est appelée "carte CP512", les cinq parties principales de cette carte sont :

- un microprocesseur;
- un système de mémoire;
- un système d'interruptions;
- un système d'interface entre la carte CP et le bus CP;
- un système d'interface entre la carte CP et le bus I/O.

Ces cinq parties sont reprises sur le schéma simplifié d'une carte CP de la figure 1.6. Chaque composant de la carte CP est accompagné du numéro du paragraphe où il est étudié.

3.2.1 Microprocesseur

Le microprocesseur d'une carte CP est de type ZILOG 8002. Il contient des registres 16 bits. Il ne permet pas la mémoire virtuelle. Le choix de ce type de microprocesseur pour la carte CP réside dans le fait qu'il était le seul disponible sur le marché au moment de la conception de SOPHO-NET et le seul pour lequel les concepteurs de SOPHO-NET pouvaient disposer, à ce moment, d'une bonne documentation.

3.2.2 Système de mémoire

Nous distinguons trois parties essentielles dans le système de mémoire d'une carte CP :

- l'EPROM, Erasable Programmable Read Only Memory, dans laquelle se trouvent les programmes résidants tels que le bootstrap, le programme de test, etc ... Sa taille est de 8 kB;

- la RAM, Random Acces Memory, est structurée en 512 pages de 1 kB (d'où le nom de CP512). Les pages des programmes sont disposées de façon quelconque dans la mémoire RAM de la carte CP. L'unité de base de la RAM est le byte (8 bits) auquel est toujours attaché un bit de parité. A chaque mot (deux bytes) est attaché un bit de coïncidence. Tout accès à un mot pour lequel ce bit est positionné provoque des sauts vers des routines d'interruption spécialisées. L'accès à la mémoire RAM s'effectue par l'intermédiaire d'une Memory Management Unit (MMU);

- la MMU, Memory Management Unit, est une unité particulière de gestion de la mémoire; cette unité permet la conversion d'adresse logique en adresse physique. La MMU comprend une table de 128 "MMU banks". Ces derniers sont eux-mêmes des tables décomposées chacune en 32 "locations" de 16 bits qui contiennent chacune un pointeur vers une page de la mémoire RAM. A chaque programme est associé un "Program number" (PNR) ainsi que 1, 2 ou 4 "MMU bank(s)" selon sa taille.

  L'accès à la mémoire RAM via la MMU s'effectue de la manière suivante : la sélection du ou des "MMU bank(s)" du

programme se fait sur base du PNR contenu dans un registre particulier.

D'autre part, le microprocesseur fournit une adresse logique pour accéder à la mémoire RAM. Cette adresse logique doit être traduite en adresse physique : à partir d'une partie de l'adresse logique, il est possible d'accéder au numéro de "location" dans le ou les "MMU bank(s)" sélectionné(s), et de là, aux références vers les pages du programme contenues dans ces "locations". L'adresse contenue dans une "location" (16 bits) ne permet toutefois pas d'accéder à toute une mémoire de 512 kB. Cette adresse doit donc être couplée à un déplacement dans la page. Ce déplacement est également trouvé dans l'adresse logique fournie par le microprocesseur. Le mécanisme de conversion d'adresse ainsi décrit est illustré sur la figure 1.4.

Remarquons qu'il est possible de trouver les mêmes adresses dans des "locations" différentes; cela signifie que certaines pages de programme peuvent être partagées.

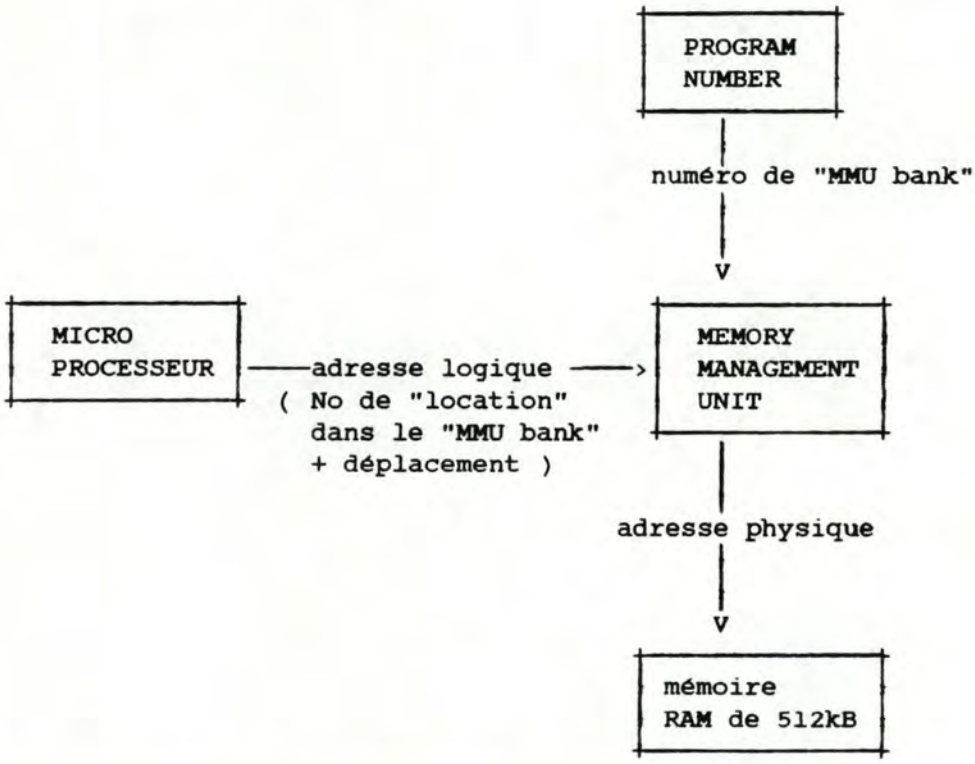


Figure 1.4 : mécanisme de conversion d'adresse logique en adresse physique

3.2.3 Interruptions

Nous distinguons quatre types d'interruption :

- l'interruption non masquable et non vectorisée;

- les interruptions vectorisées masquables;
- les interruptions non vectorisées masquables;
- les traps.

Ces interruptions se caractérisent par leur provenance, la façon dont les serveurs y répondent et leur degré de priorité.

L'unique interruption non masquable et non vectorisée provient de l'extérieur de la carte CP. Elle correspond à une alarme de chute de tension de l'alimentation électrique, elle est la plus prioritaire des interruptions et atteint le microprocesseur sans aucun intermédiaire. L'expression "non masquable" signifie que cette interruption ne peut jamais être ignorée par le microprocesseur même temporairement.

Les interruptions vectorisées sont masquables. Leur origine dans la carte CP est quelconque. En général, tous les événements normaux qui requièrent l'attention du microprocesseur sont connectés sur ce type d'interruption. Ces interruptions font réagir le système d'interruption qui fournit alors l'adresse d'une routine d'interruption (ou vecteur) au microprocesseur. Celui-ci peut alors activer la routine d'interruption correspondante.

Exemple : l'arrivée d'informations de l'extérieur de la carte CP.

Les interruptions non vectorisées masquables sont provoquées par des événements anormaux externes au microprocesseur. Ce type d'interruption est capté par le système d'interruption. Le système d'interruption informe le microprocesseur qu'un événement s'est produit. Le microprocesseur active alors une routine chargée d'examiner certains registres bien précis. Ces registres contiennent des informations importantes telles que la dernière instruction référencée, la raison de l'interruption, le dernier mot mémoire accédé, etc ... Le microprocesseur peut, sur base du contenu de ces registres, réaliser l'action appropriée en réponse à l'interruption.

Exemples : la violation d'espace mémoire, la violation de protection mémoire, les erreurs de parité, etc ...

Les "traps" sont les interruptions générées par l'exécution de certaines instructions.

Exemple : l'exécution d'une instruction dont le code opératoire n'est pas connu du microprocesseur ZILOG 8002.

3.2.4 Système d'interface de la carte CP avec le bus CP

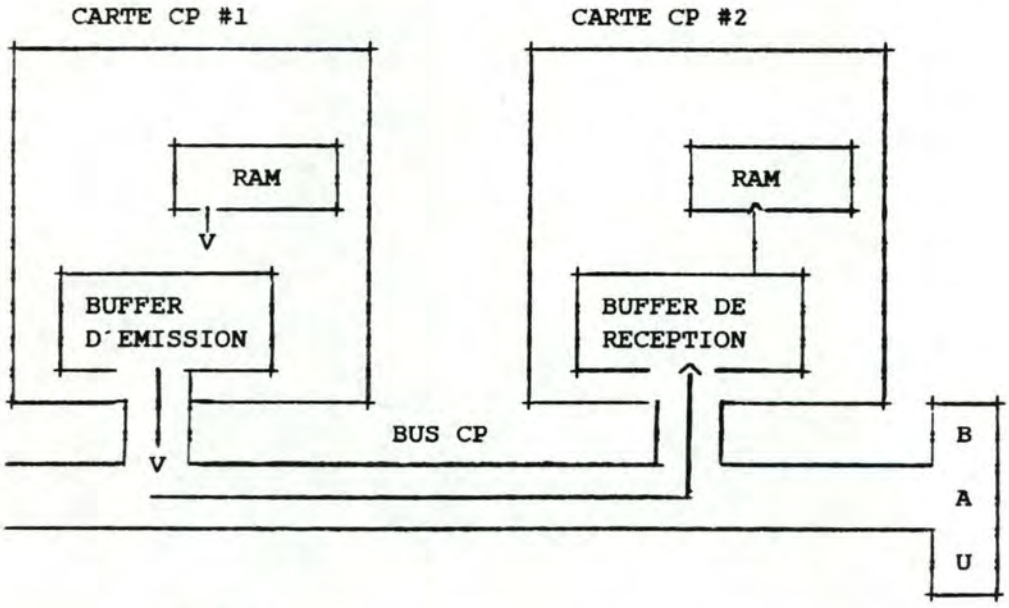
Le système d'interface de la carte CP avec le bus CP permet l'échange d'informations entre la carte CP et le bus CP. Un échange d'informations entre deux cartes CP ne peut se faire que via les bus CP du noeud. Chaque carte CP possède dans son système d'interface avec le bus CP deux buffers d'accès à ce bus : un buffer d'émission et un de réception. C'est via ces buffers que transitent les informations à émettre ou à recevoir.

Les données qu'une carte CP envoie à une autre carte CP sont transférées de sa mémoire RAM vers son buffer d'émission. L'accès à ce

dernier se fait soit par accès direct du CPU soit sous contrôle du "Direct Memory Access" (DMA). L'objectif de la DMA est de décharger le microprocesseur des opérations d'échanges de caractères coûteuses en temps CPU. L'arrivée de données dans le buffer d'émission provoque une demande d'utilisation du bus CP à la BAU. Plusieurs demandes d'usage du bus CP faites par d'autres cartes CP peuvent être en attente. La BAU avertit la carte CP quand elle peut disposer du bus CP. Moyennant accord de la carte CP destinataire, les données sont transférées via le bus CP vers le buffer de réception de cette carte CP. Le buffer de réception de la carte CP de destination de l'échange de données est vidé par la DMA dans sa mémoire RAM, les données sont ainsi arrivées à leur destination.

La figure 1.5 montre un exemple d'échange de données depuis la carte CP #1 vers la carte CP #2 après autorisation de la BAU pour l'usage du bus CP. Le flux de données part de la mémoire RAM de la carte CP #1 vers son buffer d'émission. Ce flux de données emprunte alors le bus CP pour arriver dans le buffer de réception de la carte CP #2 et de là est chargé dans la mémoire RAM de cette carte CP. La figure 1.5 ne reprend sur chaque carte CP que le buffer utile à l'échange de données.

Remarque : les interruptions non vectorisées masquables permettent la synchronisation des échanges de données.



où "-->" indique le flux de données et son sens.

Figure 1.5 : schéma d'un échange de données entre deux cartes CP

3.2.5 Système d'interface de la carte CP avec le bus I/O

Le système d'interface de la carte CP avec le bus I/O permet la communication entre la carte CP et les cartes I/O qu'elle gère. Les caractères reçus des lignes externes par une carte I/O sont temporairement stockés dans un buffer d'un caractère attaché à chaque ligne externe connectée à cette carte I/O. Leur arrivée sur la carte I/O produit une interruption vectorisée masquable sur la carte CP responsable de cette carte I/O. La carte CP se charge de faire venir les caractères de la carte I/O via le bus I/O.

Il existe sur la carte CP des programmes chargés de gérer les échanges avec les cartes I/O : les drivers; ceux-ci font l'objet d'une étude particulière au point 4 du chapitre II.

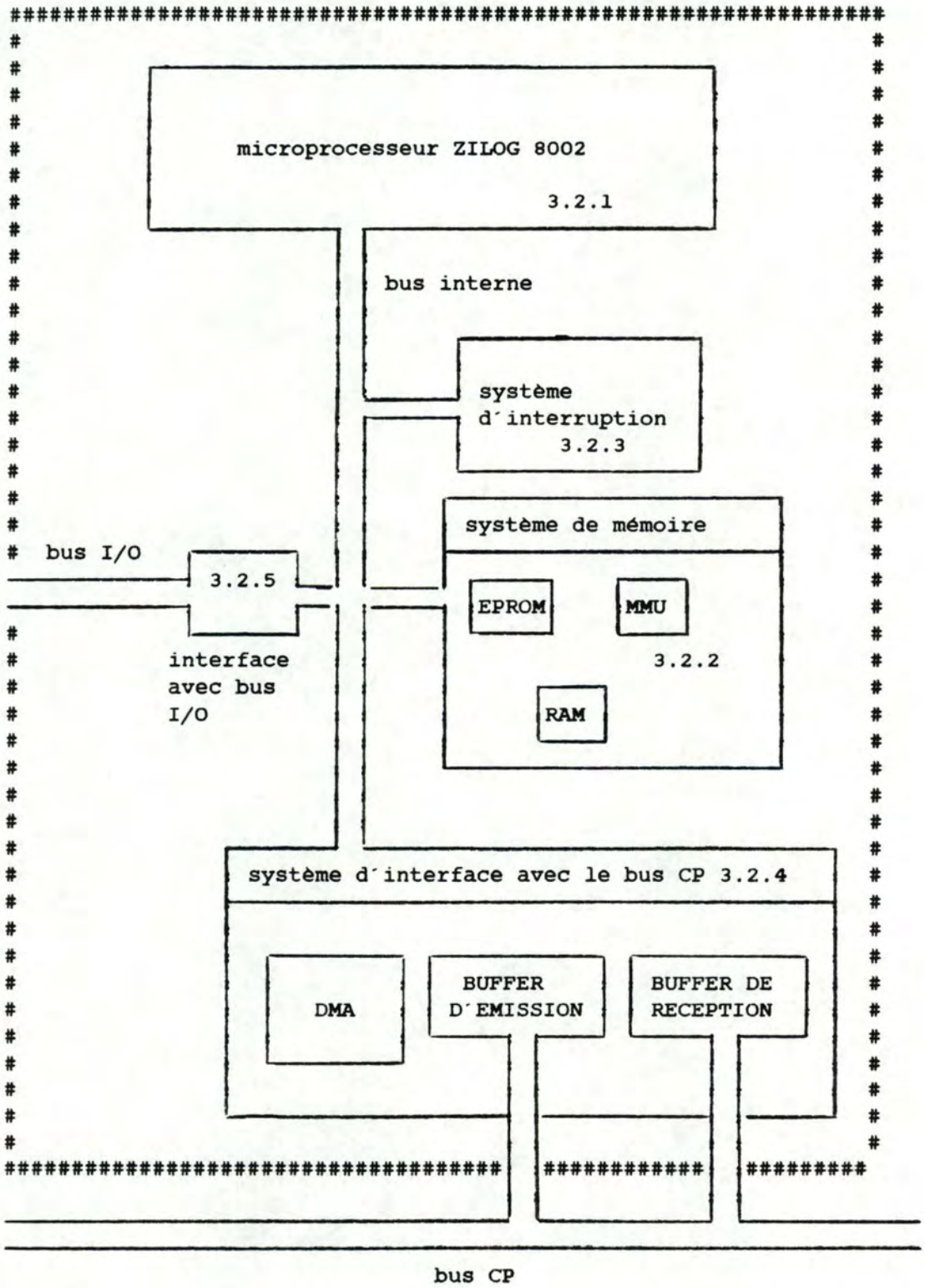


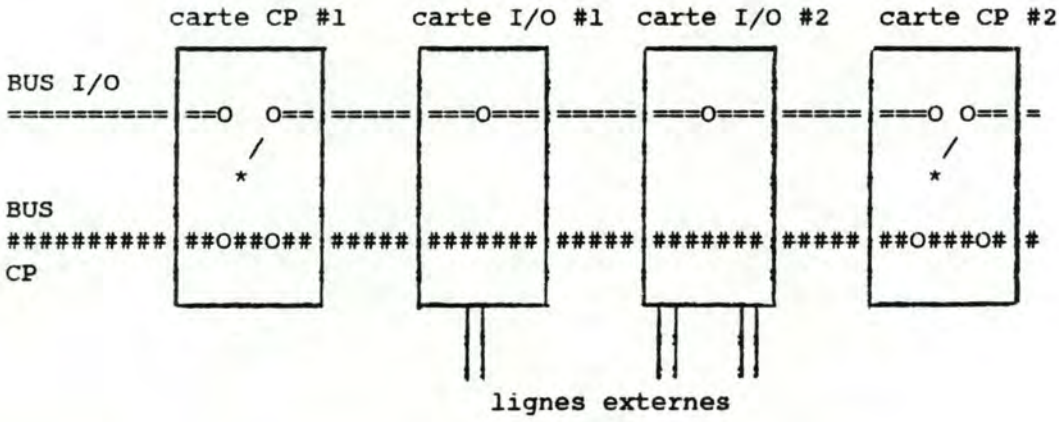
Figure 1.6 : schéma simplifié de la carte CP

3.3 Interfaces de connexion des cartes avec les bus

Chaque carte CP présente deux interfaces de connexion : la première permet à la carte CP de se connecter au bus CP et la seconde au bus I/O.

Chaque carte I/O ne dispose elle, que d'une seule interface de connexion : celle relative au bus I/O; elle n'est donc pas physiquement reliée au bus CP. En règle générale, la carte CP se trouve à gauche des cartes I/O avec lesquelles elle communique. Un multiplexeur placé sur chaque carte CP et commandé par logiciel permet de modifier ce sens en cas de défaillance de la carte CP. Les cartes I/O de la carte CP défaillante sont ainsi gérées par sa carte CP voisine de droite.

Ainsi, sur la figure 1.7, nous voyons que la carte CP #1 est en contact avec les cartes I/O #1 et #2 via le bus I/O car son multiplexeur est orienté à droite. Nous pouvons aussi observer que la carte CP #2, malgré qu'elle soit connectée au même bus I/O, ne peut pas communiquer avec ces mêmes cartes I/O à cause de l'orientation de son multiplexeur. La carte CP #1 est reliée à la carte CP #2 via le bus CP qui lui n'a pas de connexion avec les cartes I/O. En cas de défaillance de la carte CP #1, la carte CP #2 oriente son multiplexeur vers la gauche et gère ainsi les cartes I/O #1 et #2. Toutes les cartes CP éventuelles situées à droite de la carte CP #2 réorientent aussi vers la gauche leur multiplexeur.



où le symbole " / * " représente le multiplexeur orienté d'une carte CP et le symbole " O " une connexion d'une carte avec un bus

Figure 1.7 : structure des interfaces de connexion entre cartes et bus

4 Environnement logiciel de SOPHO-NET

4.1 Notion de domaine

A l'instar de la décomposition en couches logiques d'un noeud de réseau selon la norme des sept couches ISO, nous trouvons une décomposition logique en quatre niveaux appelés "DOMAINES" dans un noeud SOPHO-NET. Chacun de ces domaines est caractérisé par ses fonctions dans le noeud.

Les fonctions d'un domaine sont de deux types : les fonctions d'échange de données et celles de gestion du réseau. Nous faisons ici cette distinction pour chacun des domaines.

Nous trouvons dans un noeud SOPHO-NET la décomposition fonctionnelle suivante :

- le domaine d'information (ID) : sa fonction sur le plan de l'échange de données consiste à assurer une interface aisée et conviviale de communication avec les utilisateurs du réseau et à convertir éventuellement le protocole de ces utilisateurs. Ce domaine réalise un protocole appelé "protocole du domaine d'information" entre noeuds parents.

  Sur le plan de la gestion du réseau, la mission de ce domaine consiste à gérer les contrôles d'accès au réseau notamment au cours des phases de "LOG-ON" et "LOG-OFF";

- le domaine de communication (CD) : sur le plan de l'échange de données, ce domaine a pour fonction de gérer les "Virtual Circuit Paths" (VCP), circuits virtuels entre utilisateurs. Ce domaine réalise un protocole appelé "protocole du domaine de communication" entre noeuds parents. Ce protocole assure un échange de données sans erreur entre les noeuds parents des utilisateurs.

  Au niveau de la gestion du réseau, ce domaine s'occupe de la gestion du répertoire des utilisateurs et assure la gestion administrative du réseau; il s'agit essentiellement de la facturation de l'utilisation du réseau à ses utilisateurs;

- le domaine de transfert (TD) : sur le plan de l'échange de données, ce domaine s'occupe du routage des données et du contrôle de flux entre noeuds. Ce domaine applique un protocole appelé "protocole de transfert" de noeud à noeud. Cette couche utilise la ressource de transmission la plus appropriée pour un échange particulier. Le service fourni à ce niveau est un service de datagramme.

  La fonction de gestion du réseau de ce domaine consiste à gérer les tables de routage, les indicateurs de congestion et les ressources de transmission;

- le domaine de transmission (TRD) : il est déterminé par les moyens techniques utilisés pour la transmission. Ceux-ci peuvent être très variés; ainsi, nous rappelons qu'un réseau SOPHO-NET a la possibilité d'utiliser simultanément un réseau X25, un réseau téléphonique, ou des lignes louées sans aucune modification pour les domaines supérieurs. Cette polyvalence dans l'usage des ressources donne au réseau SOPHO-NET beaucoup de souplesse.

  A chaque type de ressource de transmission correspond un ou plusieurs sous-domaines. Classiquement le domaine de transmission est décomposé en trois sous-domaines : le "circuit domain", le "link domain" et facultativement le "packet domain":

  - le circuit domain : il reprend les éléments logiciels nécessaires à l'établissement des connexions physiques;

  - le link domain : il réalise les protocoles de liaison de données de type HDLC, BSC ou autres;

  - le packet domain : il est présent uniquement dans les cas où le réseau SOPHO-NET utilise un réseau de transmission de type X25. Ce domaine réalise les fonctions telles qu'elles ont été définies par la norme du niveau 3 de X25.

  Remarquons que le domaine de transmission peut être utilisé de deux manières :

- pour l'échange de données entre noeuds, le domaine de transmission se trouve alors au service du domaine de transfert;

- pour l'échange de données entre un utilisateur et son noeud parent, le domaine de transmission est alors au service du domaine d'information.

En guise d'exemple, la figure 1.8 illustre la situation où un terminal T1 est relié à son noeud parent, le noeud #1, par une ligne BSC. Ce noeud #1 est relié au noeud #2 par une ligne HDLC. Ce noeud #2 est le noeud parent d'un terminal T2. Ce terminal T2 et le noeud #2 sont reliés par un réseau X25.

Nous pouvons constater que le noeud #1 comporte deux domaines de transmission : un link domain BSC qui est au service du domaine d'information et qui lui permet de communiquer avec le terminal T1, et un link domain HDLC, qui est au service du domaine de transfert du noeud #1 et qui lui sert à dialoguer avec le noeud #2. Au niveau du noeud #2, nous retrouvons également deux domaines de transmission : un link domain HDLC, qui est le dual du link domain HDLC du noeud #1, et un packet domain X25 qui lui permet de dialoguer avec le terminal T2.

De manière générale, il suffit au noeud parent d'un utilisateur de comprendre un domaine de transmission de même type que la liaison le raccordant à cet utilisateur pour lui permettre d'accéder au réseau. De cette manière, SOPHO-NET n'impose aucune modification matérielle ou logicielle au terminal utilisateur et respecte son protocole. Nous voyons encore une fois que SOPHO-NET s'adapte à

l'utilisateur et non l'inverse.

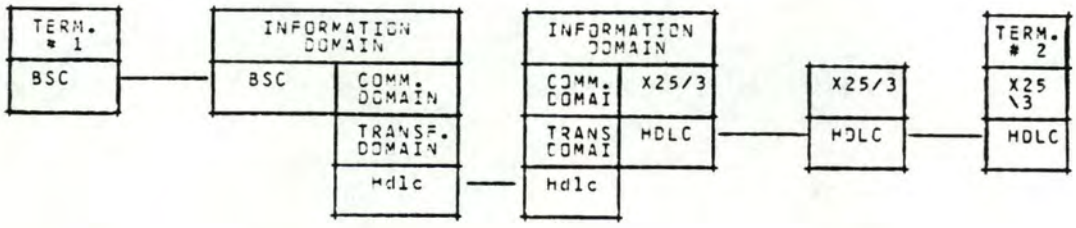


Figure 1.8 : exemple concret de réseau SOPHO-NET avec détail des domaines

Remarques :- à cette hiérarchie de domaines correspondent des formats d'information. Il s'agit de blocs d'information, de communication ou de transfert selon le domaine dans lequel ces blocs se trouvent. Lorsque les blocs descendent dans les domaines, des informations de contrôle leur sont ajoutées, ces informations de contrôle sont enlevées quand les blocs remontent vers le domaine d'information;

- la correspondance entre les quatre domaines décrits et la structure ISO est indiquée à la figure 1.9. Nous remarquons que cette correspondance n'est pas toujours biunivoque.

STRUCTURE ISO	FONCTIONS	STRUCTURE SOPHO-NET
COUCHE		DOMAINE
7 APPLICATION 6 PRESENTATION 5 SESSION	CONVERSION DU PROTOCOLE ET INTERFACE UTILISATEUR	INFORMATION
4 TRANSPORT	RECOUVREMENT D'ERREUR ET CONTROLE DE FLUX ENTRE NOEUDS PARENTS	COMMUNICATION
3 RESEAU	ADRESSAGE, ROUTAGE, CONTROLE DE CONGESTION	TRANSFERT
	ADAPTATION X25	PAQUET (X25)
2 LIAISON DE DONNEES	CONTROLE D'ERREUR ET DE FLUX NOEUD A NOEUD	LIAISON
1 PHYSIQUE	GESTION DES ELEMENTS PHYSIQUES	PHYSIQUE

Figure 1.9 : comparaison des couches ISO et SOPHO-NET

4.2 Notion d'élément

L'élément est une notion commerciale. Il constitue l'unité logicielle de vente. En plus de son nom, l'élément est caractérisé par sa fonction dans le noeud et par son prix. C'est la notion la plus proche de l'acquéreur d'un réseau SOPHO-NET; en effet, celui-ci ne connait le logiciel de ses noeuds que par le nom des éléments qui le constituent.

Exemple : l'élément BASOS qui réalise les fonctions de l'OS. Nous en retrouvons toujours au moins un par noeud.

CONCEPTION LOGICIELLE D'UN NOEUD SOPHO-NET

CHAPITRE II

1 Introduction

L'objectif de ce deuxième chapitre est de fournir au lecteur une analyse approfondie des notions logicielles de base qui entrent dans la conception d'un noeud SOPHO-NET.

Nous analysons successivement les notions de processus, de "FEP", et de driver. Ces notions sont constamment utilisées dans les chapitres suivants; c'est pourquoi nous insistons sur une bonne maitrise de celles-ci.

2 Notion de processus

2.1 Définition du processus

Les processus sont les composants logiciels chargés dans la mémoire RAM des cartes CP. A tout processus correspond une fonction précise (routage des paquets, gestion de protocole, etc...). Un processus est matérialisé par un code ou programme.

2.2 Consommations CPU d'un processus et intégration verticale

Un processus est caractérisé par un certain temps CPU qu'il requiert en moyenne chaque fois qu'il est activé. Ce temps est mesuré en millisecondes (ms). Il est possible de connaitre également le nombre de fois, en moyenne, que le processus est activé par seconde. Le produit du temps CPU requis par le processus, par le nombre moyen d'activations du processus par seconde nous donne le nombre de millisecondes par seconde (ms/s) en moyenne que le processus utilise le microprocesseur. Nous définissons ainsi un premier type de consommation CPU pour un processus :

- la consommation CPU intrinsèque (CPUI) d'un processus qui est le temps CPU moyen consommé par seconde par un processus. Ce temps est chiffré en milliseconde de temps CPU consommé par seconde de temps écoulé. Cette consommation peut aussi être mesurée en pourcentage par rapport à une puissance CPU totale disponible sur une carte CP. Cette consommation est fonction de la complexité des fonctions réalisées par le processus, des débits des données qu'il doit traiter et du type d'activation du processus (voir point 2.3.1).

De plus, la plupart des processus s'échangent de l'information sous forme de messages structurés appelés "blocs de communication". Les échanges de blocs de communication se déroulent sous le contrôle de l'OS. Les processus communicants sont dits "VOISINS". Pour réaliser ces échanges entre processus voisins, il est clair qu'une certaine puissance CPU doit être fournie par seconde par le microprocesseur. Cette puissance est calculée par le produit d'une consommation CPU moyenne du processus pour ses échanges par le nombre d'activations moyen de ce processus par seconde. Ceci nous amène ainsi à distinguer

un second type de consommation CPU du processus :

- la consommation CPU extrinsèque (CPUE) d'un processus qui est le temps CPU moyen nécessaire par seconde aux échanges de ce processus avec ses processus voisins. Ce temps correspond à la puissance CPU moyenne consommée par seconde par l'OS pour gérer ces échanges. La consommation CPU extrinsèque d'un processus est également mesurée en ms/s. Cette consommation est fonction du degré d'interaction entre les processus et des débits de données à traiter.

  De plus, cette consommation est étroitement liée au placement des processus : si un processus est placé sur la même carte CP que ses processus voisins, sa consommation extrinsèque est négligeable. Par contre, lorsqu'ils sont placés sur des cartes CP différentes, cette consommation n'est pas négligeable car il est alors nécessaire de consommer une certaine puissance CPU additionnelle pour permettre le passage des blocs de communication échangés sur le bus CP. Il est donc plus intéressant de placer des processus voisins sur une même carte CP : c'est la notion d'"INTEGRATION VERTICALE".

Ainsi, la formule de calcul de la consommation CPU totale d'un processus est :

$$CPUI + \sum_{j=1}^{N} CPUE(j)$$

où N est le nombre de processus voisins du processus considéré.

2.3 Consommation mémoire d'un processus et intégration horizontale

2.3.1 Composants de la consommation mémoire d'un processus

Chaque processus est caractérisé par une consommation de mémoire RAM chiffrable en kB. Lorsque des processus réalisent une même fonction mais travaillent sur des objets différents, ils sont identifiés par un même nom générique. Ces processus sont alors appelés processus "HOMONYMES". Lorsque ces processus homonymes sont placés sur une même carte CP, les pages de code de ces processus ne sont installées qu'une seule fois en mémoire RAM puisqu'elles leur sont identiques. Ces processus homonymes partagent aussi certaines pages de données communes. Cependant, chaque processus requiert des pages de données propres : celles-ci forment le "CONTEXTE" du processus.

Cette situation est illustrée sur la figure 2.1 où les processus P1, P2, P3 et P4 de fonction et nom P identiques, et indicés de 1 à 4 partagent les mêmes pages de code et de données communes. Toutefois, chacun de ces processus utilise son propre contexte.

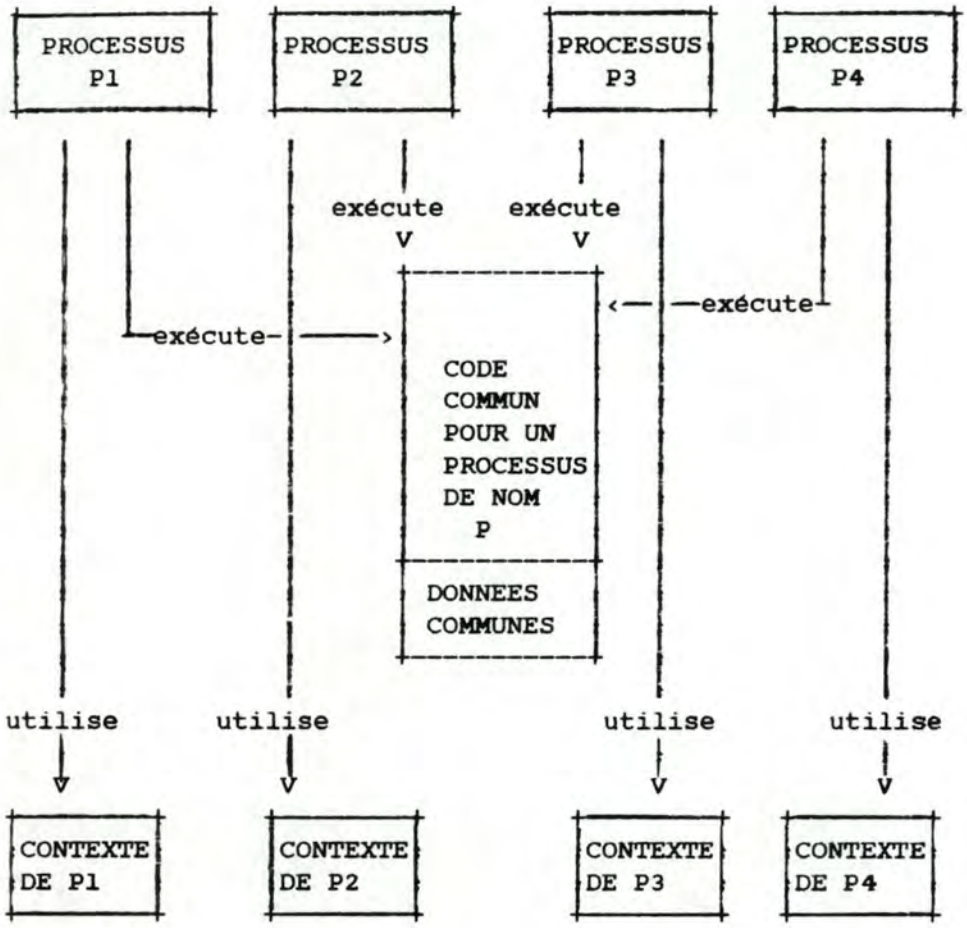


Figure 2.1 : schéma de consommation mémoire de processus homonymes placés sur une même carte CP

Le contexte d'un processus est constitué essentiellement :

- de données read-only;
- de données read-write;
- d'une zone d'adresses des processus voisins de ce processus.

Remarquons enfin qu'à chaque processus est associé une série de fichiers d'entrée; un fichier d'entrée est composé d'un ou plusieurs blocs de communication, il constitue en fait une notion logique permettant de définir un point d'entrée particulier dans un processus. Les fichiers d'entrée d'un processus sont définis en fonction de la nature des blocs de communication; en effet, les informations contenues dans les blocs de communication peuvent correspondre soit à des commandes de différents types (par exemple, une commande demandant l'envoi de données), soit à des données. Ces blocs de communication de natures différentes vont dans des fichiers d'entrée différents. Les types et durées d'activation d'un même processus varient en fonction de la nature du bloc de communication qu'il traite.

2.3.2 Notion d'intégration horizontale

Lorsque plusieurs processus homonymes se trouvent sur une même carte CP, leur consommation mémoire n'est pas la consommation mémoire d'un seul processus multipliée par le nombre de processus puisque ces processus partagent les mêmes pages de code et certaines pages de données. Nous avons seulement une quantité mémoire de contexte qui est ajoutée chaque fois qu'un processus supplémentaire homonyme est placé sur la carte CP. De plus, il est clair que ce gain de place mémoire par utilisation de certaines pages communes n'est maximal que si tous les processus homonymes sont placés sur une même carte CP; en effet, dès qu'un de ces processus est placé sur une autre carte CP, il faut recopier les pages de code du programme et de données communes sur celle-ci. D'où l'avantage à placer sur une même carte CP, dans la mesure du possible, tous les processus homonymes. C'est la notion d'"INTEGRATION HORIZONTALE".

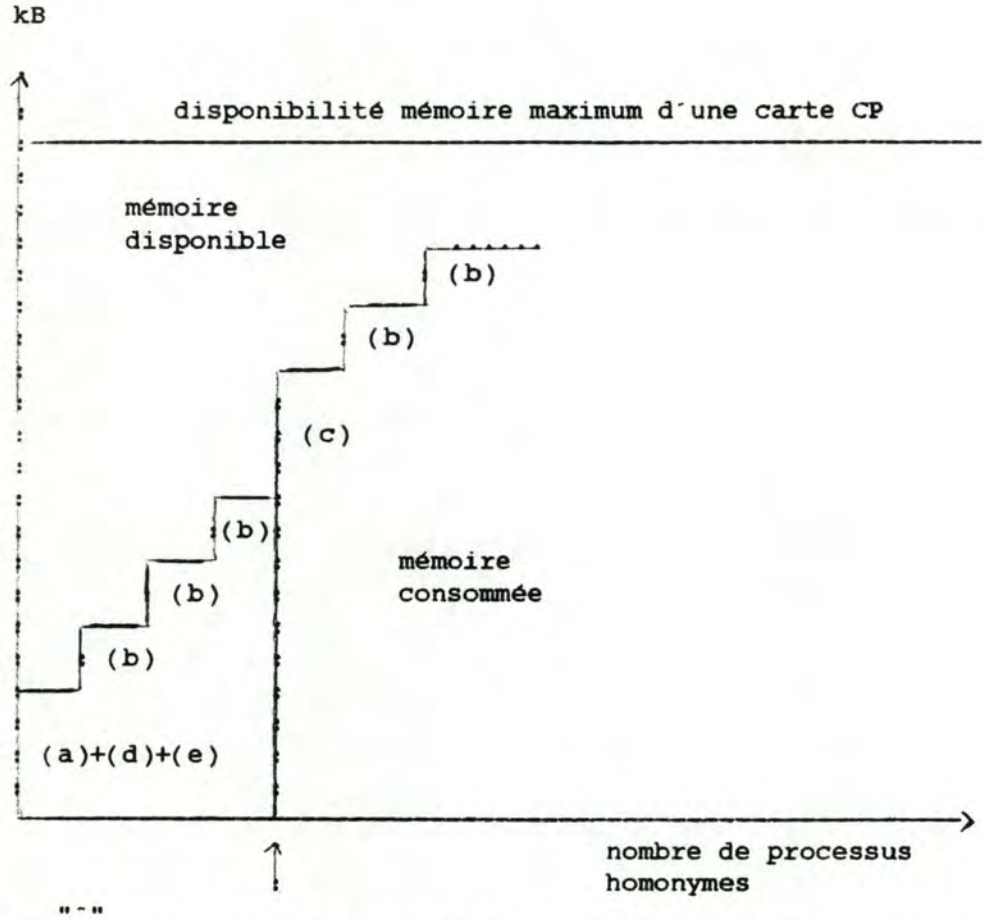
Remarquons que cette intégration horizontale permet également de réaliser un gain de "MMU banks". Ce gain est lié au fait que puisque les processus homonymes placés sur une même carte CP utilisent les mêmes pages de code et de données communes, ils utilisent implicitement les mêmes références à ces pages; il y a donc un partage de "MMU banks" par des processus homonymes placés sur une même carte CP.

2.3.3 Graphique de consommation mémoire de processus homonymes

Sur un graphique tel que celui présenté à la figure 2.2, la consommation mémoire d'un ensemble de processus homonymes placés sur une même carte CP en fonction de leur nombre évolue de façon linéaire par parties. Nous assistons à une réquisition mémoire importante pour le premier processus de cet ensemble : c'est la quantité mémoire nécessaire aux pages de code et de données communes partagées par tous les processus homonymes plus la taille du contexte du premier processus (a). Pour chaque processus subséquent, nous ajoutons une quantité mémoire relativement faible et constante : la taille du contexte de ces processus (b). Cette évolution est la même jusqu'à ce qu'on atteigne un point critique correspondant au nombre maximum de processus pour un même PNR. Un dépassement de ce maximum provoque l'octroi d'une pénalisation mémoire aux processus homonymes : ceux-ci reçoivent un PNR supplémentaire et un accroissement de place mémoire relativement peu important (c); la justification de cet espace supplémentaire est expliquée au point 2.3.5. Certains processus peuvent requérir des extensions de contexte, nous devons donc comptabiliser une consommation mémoire supplémentaire pour ces extensions; cette consommation est appelée "HRAM" (d). Les consommations décrites ici sont des consommations "statiques" dans le sens où le processus ne peut les étendre en cours d'activation.

Enfin, Il faut prévoir une consommation "dynamique" des processus (e). Cette consommation sert au stockage temporaire de blocs de communication partiellement traités par le processus. Cette consommation est dite "dynamique" car elle peut être étendue en cours d'activation du processus; elle ne peut donc être qu'estimée avant l'activation du processus.

Pour éviter de surcharger le graphique de la figure 2.2, nous avons rassemblé les consommations mémoire (d) et (e) avec la consommation mémoire initale (a).



où : indique le nombre maximum de processus acceptés sous un même PNR.

Figure 2.2 : évolution de la consommation mémoire d'un ensemble de processus homonymes placés sur une même carte CP en fonction de leur nombre

Dans le vocable SOPHO-NET, les processus homonymes sont appelés "THREADS" du programme. Le programme est alors dit "MULTI-THREAD". Pour connaitre la consommation mémoire d'un programme "multi-thread", nous devons connaitre le nombre de "threads" de ce programme, la quantité mémoire nécessaire au premier "thread" du programme, le nombre maximum de "threads" accepté sous un même PNR, l'incrément mémoire lorsque le nombre de "threads" dépasse ce maximum, les incréments mémoire correspondant aux "threads" additionnels (la taille du contexte), les ajouts mémoire correspondant à des extensions de contexte et enfin, l'ajout mémoire estimé pour la

consommation dynamique des "threads".

2.3.4 Calcul de la consommation mémoire de N processus homonymes

Le calcul de la consommation mémoire (en kB) de N processus homonymes ou "threads" d'un même programme s'effectue de la manière suivante :

soient :

- N : le nombre de "threads" du programme que nous voulons placer;
- MTNR : le nombre maximum de "threads" accepté sous un même PNR;
- QRAM : la quantité mémoire nécessaire pour le premier "thread" du programme. Il s'agit du code et des données communes plus le contexte du premier "thread". Cette quantité est mesurée en page de 1 kB;
- THRAM : la quantité mémoire nécessaire par "thread" supplémentaire. Il s'agit du contexte mesuré en page de 1 kB;
- PNRAM : la quantité mémoire nécessaire pour un PNR supplémentaire, cette quantité est mesurée en page de 1 kB;
- QDYN : la quantité mémoire dynamique estimée, mesurée en page de 1 kB;
- QHRAM : la quantité mémoire HRAM. Cette quantité est mesurée en kB.

La formule de calcul de la consommation de N "threads" placés sur une MEME carte CP est alors :

$$\left(QRAM + \left(N - 1 \right) THRAM \right)^{+} + \left(\left(\frac{N-1}{MTNR} \right)^{-} PNRAM \right) + QDYN + QHRAM$$

où $(\,)^{-}$ et $(\,)^{+}$ indiquent respectivement les parties entières inférieures ou supérieures

Remarquons que dans cette formule, il est nécessaire d'arrondir au kB supérieur car l'allocation mémoire se fait par page de 1 kB. Cette formule aboutit à une fonction de la forme de celle représentée à la figure 2.2.

Lorsque les N "threads" sont répartis sur C cartes CP, avec Nk "threads" sur la carte CP numéro k (où k= 1..C), nous appliquons la formule ci dessus pour chaque Nk et nous sommons les résultats obtenus pour avoir la consommation mémoire totale des N "threads".

2.3.5 Identification des consommations mémoire d'un processus

L'identification des consommations mémoire "statique" et "dynamique" d'un "thread" dans la mémoire RAM de la carte CP peut être visualisée à la figure 2.3. Nous y trouvons :

- une consommation de mémoire "statique" :

  - les pages de code du programme qui sont partageables par tous les "threads" d'un même programme se trouvant sur une même carte CP. Ces pages de code sont réparties dans la mémoire RAM de la carte CP et sont référencées par les adresses contenues dans les "locations" des "MMU banks" utilisés par ces "threads";

  - les pages de contexte des "threads", sont localisées aléatoirement dans la mémoire RAM;

  - les pages d'extension du contexte des "threads" sont également localisées aléatoirement dans la mémoire RAM;

  - les listes des références aux pages du contexte des "threads" forment les "Thread Control Table" (TCT). Ces TCT relatives à des "threads" de même PNR sont groupées dans une zone de mémoire de taille fixe : la " Program Control Table " (PCT), qui fait suite au code. La TCT du "thread" actif est chargée dans les "locations" de la MMU avant son activation. Lorsque la PCT est remplie de TCT, il est nécessaire d'attribuer un nouveau PNR pour les "threads" suivants, ceci entraine un coût mémoire supplémentaire à comptabiliser (la taille d'une PCT) et explique ainsi que le nombre de "threads" acceptés sous un même PNR est limité et que les "threads" reçoivent une pénalisation mémoire pour un dépassement de cette limite;

  - les références à la PCT des "threads" à même PNR se trouvent dans la MMU.

- Une consommation de mémoire "dynamique" :

  nous rappelons que certains "threads" peuvent demander dynamiquement des pages mémoire lors de leur activation.

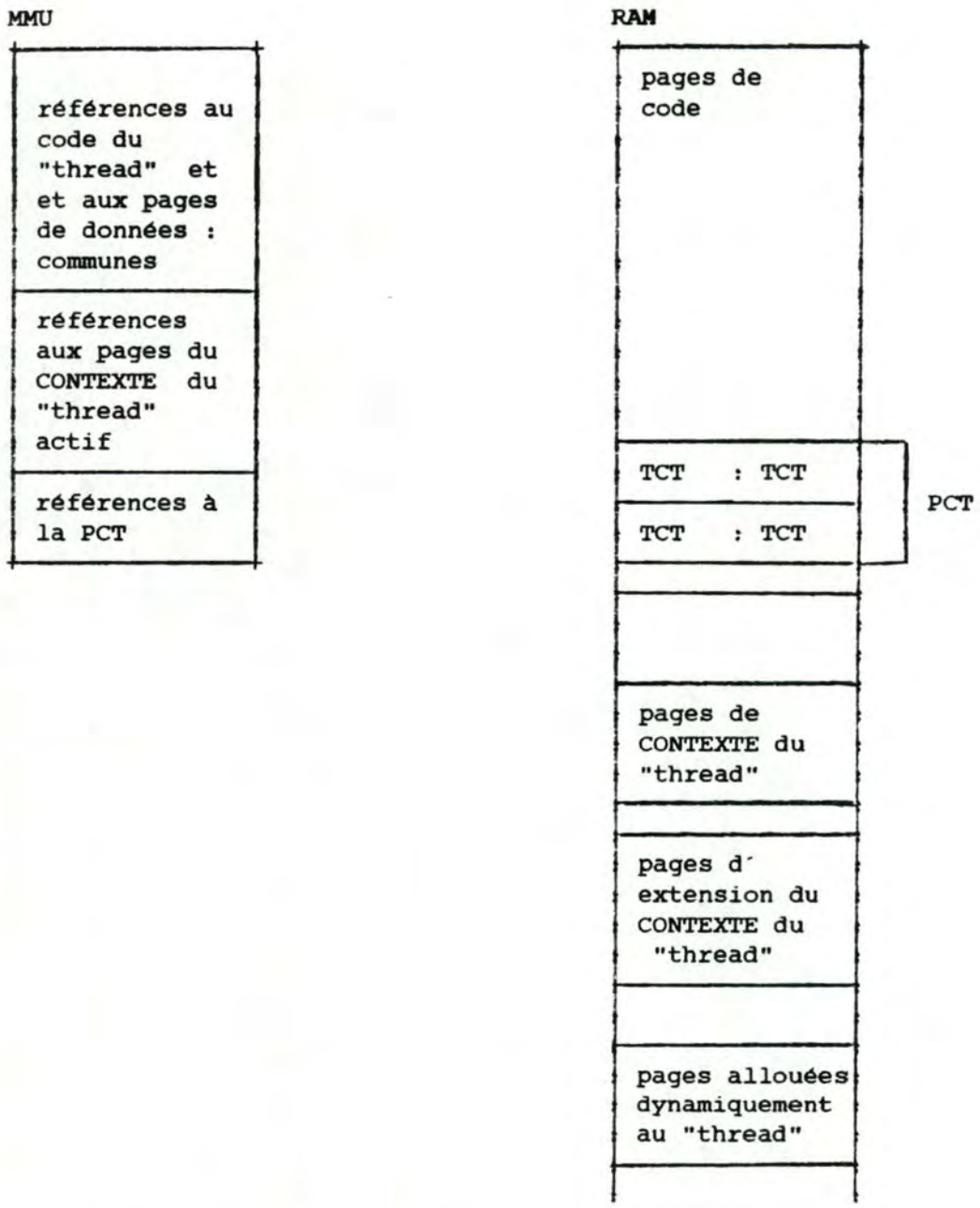


Figure 2.3 : schéma de la structure d'un "thread" en mémoire RAM et de ses références dans la MMU

2.4 Activation d'un processus

L'OS, via le "Scheduler", donne le contrôle à un processus en fonction de l'arrivée d'un événement. Les types d'événements possibles sont essentiellement la réception d'un bloc de communication et l'expiration d'un "timer". Nous nous intéressons ici plus particulièrement à l'activation du processus suite à l'arrivée d'un bloc de communication; dans ce cas, l'OS va placer dans un des fichiers d'entrée du processus le bloc de communication qui lui est

envoyé, il peut alors activer le processus. Dès que le processus est activé, les adresses de référence à son contexte sont chargées depuis la TCT correspondant à ce processus dans un de ses "MMU banks". Ce processus peut être interrompu en cours d'activation, suite aux différentes interruptions vues au point 3.2.3 du chapitre I. Cependant, dès que l'interruption a été traitée, le contrôle est rendu au processus interrompu. Il n'y a donc pas de réentrance; cela afin d'éviter de devoir mettre en oeuvre des mécanismes de choix de processus à activer après le traitement d'une interruption : de tels mécanismes risquent de dégrader les performances du système.

3 Notion de "FEP"

Un "FEP" (Functionnally Equivalent Processes) est une collection de processus qui ont même fonction et même nom. Les processus d'un "FEP" sont alloués dynamiquement par un gérant qui choisit arbitrairement un processus parmi les processus libres de cette collection lorsqu'un autre processus désire utiliser la fonction réalisée par le "FEP". Les "FEP" sont aussi appelés "processus dynamiques". Cette collection de processus peut être répartie sur plusieurs cartes CP; dans ce cas, la relation de voisinage entre cette collection et ses voisins n'est plus considérée : le gain de CPU extrinsèque est difficile à évaluer.

Exemple : les "Flow Manager" (FMGR) ont pour fonction de gérer les flux de données dans un circuit virtuel ouvert entre deux utilisateurs. Lorsqu'un processus désire passer un flux de données dans le circuit virtuel, un des FMGR libres lui est alloué par le gérant de ces FMGR. Il est possible qu'un autre FMGR lui soit alloué lors du passage d'un autre flux de données dans ce même circuit virtuel.

4 Notion de driver

4.1 Définition du driver

Parmi les processus, certains ont un aspect particulier : les drivers. Ceux-ci sont chargés des fonctions d'I/O avec les lignes externes du noeud. Les drivers s'occupent de la gestion des interruptions externes provoquées par l'envoi ou la réception de caractères à partir des lignes externes de transmission qu'ils gèrent.

4.2 Structure et fonctions du driver

Un driver est composé de deux parties : l'"Interruption Service Routine" (ISR) et le "background" (voir figure 2.4). Un buffer

ISR-background permet la communication entre ces deux parties.

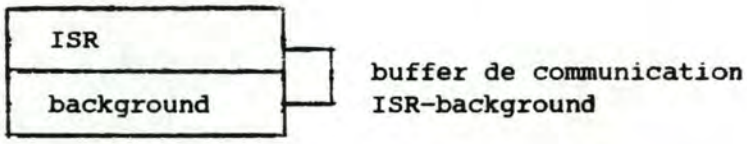


Figure 2.4 : structure d'un driver

La partie ISR est activée par une interruption externe tandis que la partie background est activée de la même façon qu'un processus normal (non driver), ou parce que l'ISR le demande.

Dans le sens de la réception de caractères à partir des lignes externes, l'ISR place les caractères reçus dans le buffer de communication ISR-background. Ce buffer est par la suite vidé par le background qui a pour mission de transmettre les caractères de ce buffer au processus de liaison de données HDLC, BSC ou autres. Le background accomplit cette tâche soit après réception d'un caractère particulier dans le cadre des protocoles HDLC, BSC ou autres (ex : réception d'un ETX), ou soit lorsque le buffer de communication ISR-background est plein.

Dans le sens de l'émission de caractères sur les lignes externes, le background va remplir le buffer de communication avec les caractères reçus des processus HDLC, BSC ou autres. l'ISR prend les caractères dans le buffer de communication ISR-background et les envoie sur la ligne externe via les cartes et le bus I/O.

4.3 Consommation mémoire d'un driver

Les éléments intervenant dans la consommation mémoire d'un driver sont exactement les mêmes que pour un processus non driver (cfr point 2.3.3). La formule de calcul de cette consommation reste la même également (cfr point 2.3.4). Par conséquent, l'intégration horizontale joue aussi dans leur cas. Remarquons enfin qu'étant donné la rapidité avec laquelle les interruptions externes doivent être traitées, chaque driver a son propre PNR. Ceci permet d'éviter des chargements des références à leur contexte dans les "MMU banks" coûteux en temps CPU. L'activation d'un driver nécessite donc uniquement le chargement de son PNR dans un registre. Toutefois, le code des drivers d'un même type placés sur une même carte CP reste commun.

4.4 Consommation CPU d'un driver

4.4.1 Charges extrinsèque et intrinsèque d'un driver

Au niveau du driver, nous retrouvons les mêmes types de consommation CPU intrinsèque et extrinsèque que pour un processus normal (cfr point 2.2). Un nouveau type de consommation CPU apparait cependant dans leur cas : les drivers sont activés suite à une interruption provoquée par l'arrivée, sur la carte I/O d'un caractère provenant des lignes externes connectées à cette carte I/O. La gestion de ce type d'interruption exige trois temps :

- un temps de sauvetage du contexte du processus interrompu : temps " Header ";

- un temps de service correspondant au traitement du caractère reçu, il s'agit essentiellement du placement de ce caractère dans le buffer de communication ISR-background. Ce temps est appelé temps "Service";

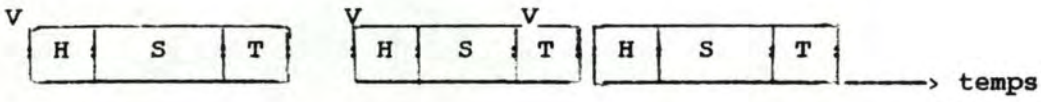
- un temps de restauration du contexte du processus interrompu : temps "Trailer".

Les temps Header et Trailer correspondent à une "charge extrinsèque" par rapport à la gestion des interruptions car ils n'y participent que de façon indirecte. Par opposition, le temps service correspond à une "charge intrinsèque". Nous appelons "consommation CPU extrinsèque d'un driver", la charge extrinsèque du driver.

4.4.2 Principe d'optimisation des charges d'un driver

Il est intéressant d'éviter que pour chaque interruption nous ayons les trois temps cités dans le point précédent (figure 2.5), d'où l'idée de regrouper les temps Service en une seule séquence (figure 2.6). Nous n'avons alors pour cette séquence qu'un seul temps Header et Trailer. Toutefois, pour pouvoir regrouper ces temps Service, il est nécessaire de fixer un certain délai avant le temps Header. Ce délai permet l'arrivée de caractères sur plusieurs lignes externes. Ce délai doit être calculé de manière telle que sur chaque ligne externe, le traitement d'un caractère soit réalisé avant la réception du suivant. Cette contrainte est liée au fait que les buffers de réception de caractères sur les cartes I/O ne peuvent accueillir qu'un seul caractère par ligne externe qui leur est connectée; une attente trop longue avant le traitement des caractères risque de provoquer un écrasement de ceux-ci. Le délai est assez complexe à calculer, il est borné supérieurement par 1/N seconde, où N est le débit en caractères par seconde sur la ligne externe la plus rapide.

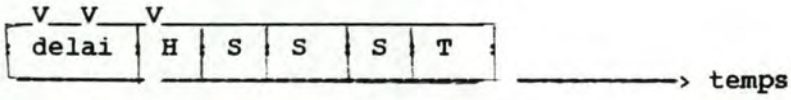
Ainsi, si la ligne externe la plus rapide a une vitesse de 9600 bps, alors N est égal à 1200 caractères par seconde et le délai maximum d'attente avant traitement des interruptions est égal à 1/1200 seconde.



où :

H indique le temps "Header";
S indique le temps "Service";
T indique le temps "Trailer";
V indique l'arrivée d'une interruption.

Figure 2.5 : schéma de gestion des interruptions SANS delai d'attente



où :

H indique le temps "Header";
S indique le temps "Service";
T indique le temps "Trailer";
V indique l'arrivée d'une interruption.

Figure 2.6 : schéma de gestion des interruptions AVEC délai d'attente et regroupement des gestions de ces interruptions

Un bon calcul du délai est fondamental pour optimiser l'usage du temps CPU consommé par les drivers : avec un délai bien calibré, il est possible de réduire la charge CPU extrinsèque des drivers utilisée lors de la gestion des interruptions et ainsi d'accroître la puissance CPU disponible pour les autres processus de la carte CP. Il existe encore pour l'instant beaucoup de problèmes pour déterminer la formule de calcul de ce délai. La formule présentée ci-dessus n'en est qu'une simplification extrême; elle ne définit qu'une borne supérieure de ce délai.

Cependant, à partir de statistiques, il est possible de construire un graphique de l'évolution de la charge CPU consommée pour la gestion des interruptions externes. Nous pouvons, sur base de ce graphique estimer la part des charges intrinsèque et extrinsèque pour un nombre donné d'interruptions. Un exemple d'un tel graphique est montré à la figure 2.7.

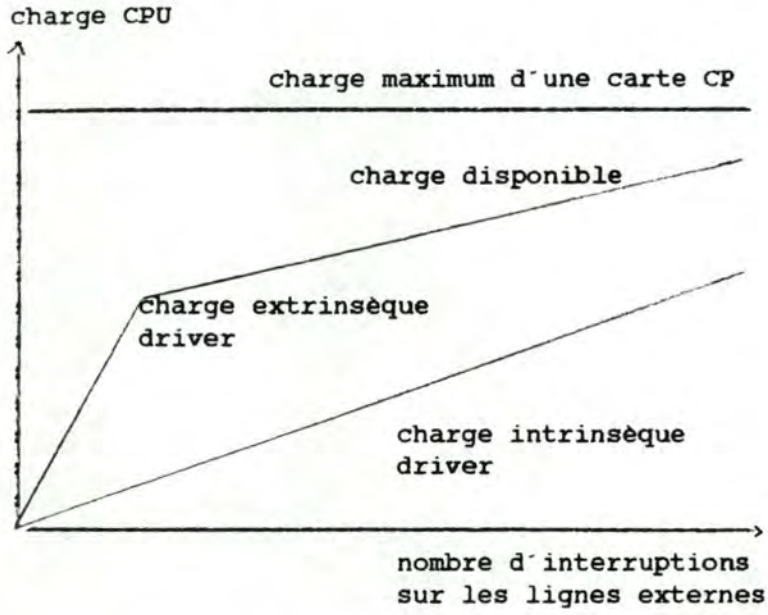


Figure 2.7 : exemple de graphique de l'évolution des charges extrinsèque et intrinsèque CPU du driver en fonction du nombre d'interruptions sur les lignes externes

Une remarque très importante à faire pour les drivers est donc que le regroupement des temps Service et donc la réduction de la charge extrinsèque est lié au placement de drivers sur la même carte CP.

4.4.3 Composants de la consommation CPU d'un driver

Nous pouvons distinguer quatre types de consommation CPU pour un driver :

- la consommation CPU intrinsèque de l'ISR (CPUI_ISR) qui est le temps CPU moyen nécessaire par seconde à l'exécution de l'ISR. Ce temps, correspondant au temps Service, dépend des débits de données à traiter et de la complexité des fonctions de l'ISR;

- la consommation CPU intrinsèque du background (CPUI_BCKG) qui est le temps CPU moyen nécessaire par seconde à l'exécution du background. Ce temps dépend des débits de données à traiter et de la complexité des fonctions que réalise le background;

- la consommation CPU extrinsèque du background (CPUE_BCKG) qui est le temps CPU moyen consommé par seconde par l'OS pour permettre les échanges du background avec ses processus voisins. Ce temps est fonction du degré d'interaction entre le background et ses processus voisins et des débits de données à traiter;

- la consommation CPU driver extrinsèque de l'ISR (DR_CPUE) qui est le temps CPU moyen nécessaire par seconde à l'initialisation

et à la terminaison du traitement des interruptions : charge extrinsèque ou temps Header et Trailer du processus interrompu par un processus driver. Ce temps est fonction de la vitesse de la ligne externe gérée par ce driver et du regroupement des drivers.

Enfin, la consommation totale d'un driver est calculée par la formule suivante :

$$CPUI\_ISR + CPUI\_BCKG + DR\_CPUE + \sum_{j=1}^{N} CPUE\_BCKG(j)$$

où N est le nombre de processus voisins du driver considéré

Note : Le placement des drivers sur les cartes CP est étroitement lié aux lignes externes de transmission connectées sur les cartes I/O. Ce placement fait intervenir des critères de vitesse de ligne externe de transmission, de limites de nombre de lignes externes de transmission par carte I/O, etc... Une fois que la carte CP d'accueil du driver est définie, elle ne peut plus être remise en question sous peine de devoir effectuer des modifications matérielles du noeud.

METHODE ACTUELLE DE CONFIGURATION D'UN NOEUD SOPHO-NET ET PROJET DE C.A.O.

CHAPITRE III

1 Introduction

L'objectif de ce troisième chapitre est d'introduire le lecteur à la méthode actuelle de configuration d'un noeud de réseau SOPHO-NET, aux problèmes liés à cette méthode et au projet de conception assistée par ordinateur (C.A.O.) d'un noeud de réseau SOPHO-NET.

Dans la première partie du chapitre, nous décrivons les étapes nécessaires à la conception d'un réseau SOPHO-NET depuis les spécifications du client jusqu'à la mise en oeuvre matérielle et logicielle finale du réseau.

Dans une deuxième partie, nous confrontons l'ancienne et la nouvelle philosophie suivies pour l'étape de la conception d'un noeud de réseau SOPHO-NET qui fait l'objet de notre travail. Cette confrontation nous permet de présenter le projet de conception assistée par ordinateur d'un noeud de réseau SOPHO-NET et ses objectifs.

Pour clôturer ce chapitre, nous donnons une définition du problème que nous devons résoudre, une description de ses données et des résultats attendus de la résolution de ce problème.

2 Etapes de la conception d'un noeud SOPHO-NET

2.1 Exigences du client

Tout client qui désire acheter un réseau SOPHO-NET remplit un questionnaire (voir annexe III) permettant de connaitre ses besoins exacts. Ceux-ci sont estimés au niveau :

- du nombre et de la localisation géographique des noeuds;
- pour chaque noeud :
  - des caractéristiques des lignes connectées (terminaux, liaisons inter-noeuds, liaison NMC);
  - du volume des échanges et de la répartition dans le temps de ceux-ci;
  - du nombre d'utilisateurs;
  - du nombre de circuits virtuels ou VCP;
  - etc ...

2.2 Des exigences du client à la configuration du réseau

Le questionnaire, fourni au constructeur, permet de définir une architecture de l'ensemble du réseau, une architecture pour chaque noeud et la liste des processus que le configurateur doit placer sur les cartes des différents noeuds en n'oubliant pas que certains processus se retrouvent sur chaque noeud (l'OS par exemple).

2.3 Création des fichiers de chargement

Une fois la liste des processus établie, se pose le problème de la localisation de ces processus à l'intérieur de chaque noeud. Il s'agit donc de déterminer pour chaque processus la carte CP sur laquelle il doit être placé. Ceci en sachant que ces processus consomment une certaine quantité mémoire, une certaine puissance calcul, étant donné que certains processus sont placés de façon conventionnelle ou obligatoire sur des cartes CP bien précises et en sachant que certains processus ne peuvent jamais être placés ensemble sur une même carte CP.

Pour chaque noeud du réseau est créé un fichier de chargement symbolique. Ce fichier indique pour chaque processus un certain nombre de paramètres et en particulier le numéro logique de la carte CP du noeud sur laquelle il doit être placé. La création de ce fichier de chargement est l'étape que nous avons modifiée. Le fichier de chargement symbolique est alors traduit en fichier de chargement effectif. Ce fichier de chargement effectif ainsi que les processus à placer sur le noeud sont ensuite chargés sur un disque dans ce noeud.

2.4 Localisation finale des processus

Sur base des informations contenues dans le fichier de chargement effectif, les processus peuvent être chargés depuis le disque sur les cartes CP du noeud.

3 Philosophies de création des fichiers de chargement

3.1 Ancienne philosophie

L'ancienne philosophie suivie pour la création du fichier de chargement consiste à placer une liste donnée de processus sur une architecture fixée au départ. En effet dans cette philosophie, le nombre de cartes disponibles pour le noeud à configurer est déterminé lors de la vente. Le contrat signé avec le client ne pouvant jamais être remis en question, les frais d'achat de toute carte supplémentaire

lors de la configuration d'un noeud doivent être pris en charge par Philips.

Le problème est alors d'utiliser au mieux les ressources de toutes les cartes CP en y plaçant un maximum de processus de la liste donnée. Lorsqu'il est impossible de placer tous les processus sur les cartes CP prévues, il est permis de se baser sur des probabilités d'utilisation simultanée des processus à fonction équivalente, c'est à dire les processus d'un même "FEP", et de n'en placer effectivement qu'un nombre moins important.

Par exemple, le nombre de FMGR, gérants des flux de données dans un circuit virtuel, peut être réduit si les "devices" qui produisent les flux de données ont peu de chance d'être tous utilisés en même temps.

Jusqu'à présent, une personne est chargée de déterminer manuellement et de façon assez intuitive la localisation des processus sur les cartes CP du noeud.

3.2 Nouvelle philosophie

Dans la nouvelle philosophie, nous disposons toujours d'une liste donnée de processus à placer mais plus d'une configuration imposée. L'étape de création du fichier de chargement doit déterminer, avant la signature du contrat de vente avec le client, le nombre minimum et suffisant de cartes CP nécessaires à la réalisation du noeud; ce nombre doit être minimum afin de proposer un contrat le plus avantageux possible pour le client, mais doit être suffisant pour éviter à Philips de devoir par après ajouter des cartes à ses propres frais.

Remarquons que pour éviter des risques d'engorgement ou de ralentissement du système, cette philosophie implique un rejet de tout calcul de probabilité d'utilisation simultanée des processus; ceci revient à dire que tout processus nécessaire pour satisfaire les besoins du client en toutes circonstances doit être placé dans le noeud.

Dans le cadre de cette nouvelle philosophie, le projet à réaliser doit permettre d'assister par ordinateur la création du fichier de chargement symbolique.

3.3 Objectifs du projet de C.A.O.

Par rapport à ce qui existe, les objectifs du projet de C.A.O. sont donc :

- d'éviter un travail manuel fastidieux et éventuellement erroné lors de la création du fichier de chargement symbolique;

- d'éviter des gaspillages en terme de place mémoire et de puissance CPU tant au niveau des processus eux-mêmes qu'au niveau des échanges entre ceux-ci. Pour cela nous savons déjà qu'il est intéressant de placer des processus voisins ou homonymes sur une même carte CP; c'est à dire qu'il faut profiter au mieux des intégrations verticale et horizontale. Le nombre de cartes CP doit donc être minimum;

- d'éviter des coûts supplémentaires qui seraient dus à des erreurs de prévision au niveau de la consommation CPU ou de la consommation mémoire des processus. Ce type d'erreur peut entraîner l'ajout d'une carte CP, d'un "panier" voire même d'une étagère. Le nombre de cartes CP doit donc être suffisant;

- d'équilibrer le noeud au niveau de la distribution de la charge CPU sur les cartes CP afin d'éviter la surcharge d'une carte CP et/ou l'inactivité d'une autre. Nous essayons également de conserver un équilibre au niveau de la distribution de l'occupation de la mémoire sur les cartes CP; il faut ici faire remarquer qu'il n'est pas certain que ce dernier équilibre soit intéressant dans la perspective d'une extension logicielle du noeud; en effet, vaut il mieux dans ce cas avoir des espaces libres de tailles sensiblement les mêmes sur les cartes CP ou non ?

Remarquons enfin que ce projet de C.A.O. s'inscrit dans le cadre d'un projet beaucoup plus large qui vise à automatiser toutes les étapes décrites dans le point 2. Ce projet porte le nom de "ANODE" (Automated NOde DEsign). Ceci implique qu'une méthode visant à réaliser les objectifs définis doit être caractérisée par sa rapidité.

4 Définition du problème

Notre problème est de placer, par une méthode rapide, une liste de processus à consommation mémoire et CPU connues sur un nombre minimum mais suffisant de cartes CP. Nous pouvons disposer d'une série de cartes CP sur lesquelles des processus ont déjà été placés et qu'il faut compléter, d'un nombre supposé illimité de cartes CP vierges disponibles. Nous connaissons également les avantages des intégrations verticale et horizontale. En bref, il s'agit donc de créer une partie du fichier de chargement symbolique.

Notons que le placement des processus de type "driver" n'entre pas dans le cadre de ce travail. Ces processus font intervenir des critères tels que la vitesse des lignes externes, le nombre maximum de lignes externes sur une carte I/O, etc. Tout processus de ce genre aura été localisé auparavant.

5 Données du problème

5.1 Données indépendantes de la configuration

5.1.1 Catalogue des processus

Le catalogue des processus reprend pour chaque processus son nom, le fait qu'il est de type " driver " ou non, le fait qu'il est de type "FEP" ou non et ses caractéristiques sur le plan de la mémoire. Il s'agit de la quantité mémoire nécessaire à l'installation du premier "thread" du programme, la quantité mémoire de contexte pour chaque "thread" supplémentaire, ce contexte est décomposé en contexte Read-only et Read-write, le nombre de "MMU banks" que le processus occupe, le nombre de "threads" qu'un même "PNR" peut accepter, ainsi que la quantité "HRAM" maximum qui peut lui être ajoutée. Ce catalogue est utilisé lors de toute conception d'une configuration.

Exemple : Le processus de nom "SR" n'est ni de type "driver" ni de type "FEP". Il occupe 24 kB pour le premier "thread", 1.1 kB de contexte Read-only et 3.43 kB de contexte Read-write par "thread" supplémentaire. Il consomme 4 "MMU banks", un PNR peut accepter 30 "threads" et la quantité HRAM maximum est de 0 kB.

Un exemple plus complet de catalogue des processus est présenté en annexe IV.

5.1.2 Catalogue des agrégats

Le catalogue des agrégats reprend une série de définitions d'agrégats. Un agrégat est défini par un ensemble de noms de processus généralement voisins de type "NON FEP".

Rassembler différents processus voisins dans un agrégat permet de favoriser l'intégration verticale des processus lors du placement de ceux-ci. Ceci donne également à l'utilisateur la possibilité d'influencer le placement des processus en intégrant dans le catalogue des agrégats des conventions qui lui sont propres. Il est aussi possible de former un agrégat avec plusieurs processus devant être placés sur une même carte.

Notons que, par convention, un nom d'agrégat commence toujours par le caractère " @ ".

Exemple : l'agrégat "@HDLCT" est composé des processus "DRHD", "HDLC" et "MDPER".

Un exemple plus complet de catalogue des agrégats est présenté en annexe IV.

5.1.3 Caractéristiques d'une carte CP

Les caractéristiques d'une carte CP sont une quantité mémoire, une quantité CPU ainsi qu'un nombre de "MMU banks" et un nombre de PNR. Il s'agit de quantités effectivement disponibles pour le placement des processus; les quantités nécessaires aux besoins de l'OS de la carte CP et aux marges de sécurité ayant été soustraites au préalable. Nous parlons aussi de quantités "conseillées". Nous disposons d'au plus 256 cartes CP; il s'agit du nombre maximum de cartes CP par noeud.

5.1.4 Bibliothèque des incompatibilités formelles

La bibliothèque des incompatibilités formelles est un ensemble d'interdictions de placement sur une même carte CP de processus donnés comme incompatibles au départ. Certains processus ne peuvent être placés sur une même carte CP parce que les temps d'exécution des uns peuvent entrainer des délais d'activation trop longs pour les autres. Ceci est le cas des processus de type "LNKA" et "EPST". Il est également interdit de placer deux processus "LUC" sur une même carte CP car leur contexte est inclus dans les pages de données communes.

Un exemple plus complet de bibliothèque des incompatibilités est présenté en annexe IV.

5.2 Données dépendantes de la configuration

5.2.1 Liste des groupes

La liste des groupes est en fait la liste comprenant l'ensemble des processus à placer sur les cartes CP d'un noeud.

Un groupe est composé soit d'un ou plusieurs processus homonymes, soit d'un ou plusieurs agrégats de même nom.

Définir des groupes permet au concepteur du noeud d'influencer le placement des processus vers une plus grande intégration horizontale. Il est clair que plus les tailles des groupes sont importantes et plus le nombre de groupes à placer est limité.

Tout commme les processus, les groupes ont une qualité de "FEP" ou non. Un groupe est de type "FEP" s'il est composé de processus de type "FEP". Il est alors dit "groupe dynamique" et peut être scindé. Par contre, un groupe de type "NON FEP", appelé "groupe statique", est indécomposable et ne peut jamais requérir plus de mémoire ou de CPU que les quantités conseillées sur une carte CP vierge. Une exception existe cependant : un groupe composé d'un seul processus peut consommer plus de CPU que la quantité conseillée afin

de ne pas surcharger une carte CP.

Deux groupes sont dits incompatibles dès qu'un processus composant le premier groupe est incompatible avec un processus du second.

Un groupe est voisin d'un autre dès qu'un des processus composant le premier groupe échange de l'information avec un processus du second.

De plus, un groupe est de type "driver" lorsqu'au moins un de ses processus est de type "driver".

La liste des groupes reprend pour chacun : son nom, les bornes inférieure et supérieure de ses "threads", sa consommation CPU intrinsèque, dans des cas exceptionnels le numéro de la carte CP où il doit être placé, la quantité "HRAM" qui lui correspond et, si le groupe est de type "driver", sa consommation CPU "driver" extrinsèque. La liste des groupes reprend également, pour chacun des groupes, le nombre de groupes voisins qu'il possède ainsi que l'identification de ses voisins et le CPU extrinsèque qui correspond aux échanges avec chacun de ses voisins.

Un groupe est identifié par le nom des processus ou agrégats qui le composent et par sa borne inférieure.

Rappelons que les processus et donc les groupes de type "driver" font l'objet d'un placement préalable. C'est ainsi que le numéro de carte CP de tout groupe de type "driver" est défini dans la liste des groupes.

La liste des groupes est propre à une configuration particulière puisque les relations entre groupes varient d'une configuration à l'autre.

Exemple : le groupe "SRM 1-4" dont la borne inférieure est 1 et la borne supérieure est 4, a une consommation CPU intrinsèque de 12 ms/s, son numéro de carte CP est indéterminé et sa quantité HRAM est égale à 0 kB. Ce groupe n'étant pas de type "driver" n'a pas de consommation CPU "driver" extrinsèque. Ses voisins sont les groupes "DS 1", "SR 1" et "SR 2" avec respectivement des consommations CPU extrinsèques de 15 ms/s, 14 ms/s et 15 ms/s.

Un exemple plus complet de liste des groupes est présenté en annexe IV.

5.2.2 Coefficient de rapport CPU-MEMOIRE

Le coefficient de rapport CPU-MEMOIRE calculé par le système ou fourni par l'utilisateur lorsqu'il le désire, constitue une information fondamentale; en effet, il permet d'orienter le placement des groupes dans le sens d'une plus grande intégration horizontale ou verticale. Sa formule de calcul ainsi que son effet seront développés au point 3.2.2

du chapitre IV. L'idée générale de ce coefficient est d'identifier la ressource mémoire ou CPU critique avant le placement des groupes, et d'en déduire un coefficient qui permet d'économiser cette ressource au moment du placement. Une ressource est critique lorsque prise indépendamment des autres, elle induit la consommation la plus importante d'un nombre de cartes CP.

6 Résultats demandés

6.1 Résultats de la configuration

Les résultats à obtenir pour la configuration sont :

a. le nombre total de cartes CP utilisées pour le placement réalisé;

b. la localisation des groupes donnés à l'entrée;

c. le nombre minimum de cartes CP nécessaires. Ce nombre est calculé en ne tenant pas compte du CPU nécessaire à la réalisation des échanges et en supposant que les processus homonymes sont placés sur une même carte CP;

d. le nombre maximum de cartes CP nécessaires. Ce nombre est calculé en supposant que tous les échanges nécessitent du CPU extrinsèque et que tous les processus homonymes sont placés sur des cartes CP différentes;

e. les moyennes des consommations mémoire et CPU des cartes CP;

f. les distributions des consommations mémoire et CPU des cartes CP;

g. les suppléments de consommation mémoire et CPU utilisés par rapport aux ressources minimales nécessaires pour le placement.

Un placement est d'autant plus performant que le nombre de cartes CP issu de ce placement est faible. A nombre de cartes CP égal, nous retenons le placement qui a les distributions mémoire et CPU les plus faibles.

6.2 Résultats par carte CP

Les résultats pour chaque carte CP sont :

a. l'occupation mémoire et CPU de la carte CP;

b. la quantité de CPU extrinsèque gagnée grâce au placement de groupes voisins sur cette carte CP;

c. un avertissement lorsque la quantité de CPU consommée dépasse la quantité conseillée pour une carte CP. Rappelons que dans ce cas, il ne peut y avoir qu'un seul groupe d'un seul processus sur cette carte CP;

d. la liste des groupes placés sur la carte CP.

6.3 Résultat par paire de cartes CP

Le résultat pour toute paire de cartes CP est :

le CPU extrinsèque nécessaire aux échanges au sein de la paire.

CREATION D'UN FICHIER DE CHARGEMENT

CHAPITRE IV

1 Introduction

Dans ce chapitre, nous présentons de manière générale la méthode que nous avons élaborée en vue d'apporter une solution au problème de la création d'un fichier de chargement, lors de la configuration d'un noeud. Nous appelons cette méthode : "la méthode de fusion". Ensuite, nous en précisons les différentes étapes. Nous montrons alors un organigramme qui est une visualisation synthétique de la méthode puis nous en calculons la complexité algorithmique. Pour terminer, nous critiquons cette méthode.

Remarquons dès à présent que la méthode que nous proposons est la prolongation d'une série d'autres méthodes qui ont dû être abandonnées pour des raisons d'inefficacité ou d'imperfection dans les résultats. Nous décrirons ces différentes méthodes et les raisons de leur abandon dans le chapitre V.

Nous signalons au lecteur qu'il lui est possible de consulter en annexe IV un catalogue des groupes et, en annexe V, les résultats obtenus, par la méthode de fusion, pour le placement de ces groupes.

2 Présentation générale

La méthode proposée est une heuristique qui nous semble réaliser une harmonie entre deux objectifs : concevoir un algorithme qui s'exécute rapidement tout en produisant des résultats suffisamment optimaux.

Nous conseillons au lecteur de se remémorer une série de définitions utiles pour une meilleure compréhension de la méthode. Ces définitions reprises en annexe II, sont celles de : "processus", "FEP", "driver", "carte CP", "groupe", "intégration horizontale" et "intégration verticale".

En un premier temps, nous nous assurons de la cohérence de nos sources d'informations. Ensuite nous construisons un graphe où les noeuds sont les groupes qui nous ont été donnés et où les arêtes représentent les gains mémoire et/ou CPU qui seront réalisés si les deux groupes qu'elles relient sont placés sur une même carte CP. Le poids des arêtes est alors influencé par le coefficient CPU-MEMOIRE dans le but d'économiser la ressource mémoire ou CPU la plus critique. Il est également utile à ce moment de rendre les arêtes mémoire comparables aux arêtes CPU et de réunir toutes les arêtes ayant deux

mêmes noeuds comme extrémités. Ceci permet d'avoir à notre disposition une mesure de gain absolue définissant l'intérêt à fusionner deux groupes.

Une fois ce graphe construit nous ne parlons plus de groupes mais bien de noyaux que nous allons fusionner de manière à réaliser le plus de gains mémoire et/ou CPU possibles. Nous définissons les concepts de noyau de type "FEP" et de noyau de type "NON FEP".

Dès que le procédé de fusion est terminé nous plaçons tous les noyaux de type "NON FEP" sur un nombre minimum de cartes CP. Ensuite nous plaçons les noyaux de type "FEP" en les découpant de sorte qu'ils puissent utiliser un maximum de ressources encore disponibles sur les cartes CP déjà entamées.

Signalons que nous sommes attentifs à empêcher la fusion de noyaux correspondants à des groupes incompatibles entre eux ou devant être placés sur des cartes CP dont les numéros sont différents.

Après ces étapes le placement proprement dit est réalisé, il nous faut alors l'évaluer et si cela nous semble utile, réaliser une nouvelle tentative de placement.

3 Différentes étapes de résolution

3.1 Analyse de cohérence

Avant tout traitement, il est nécessaire de vérifier si les informations disponibles sont cohérentes entre elles et respectent une syntaxe définie.

Les incohérences, contradictions entre nos sources d'informations, sont du type suivant :

- la bibliothèque des incompatibilités indique une incompatibilité entre des processus qui par ailleurs sont soit placés sur une même carte CP, soit se trouvent dans un même agrégat ou dans un même groupe de la liste de données;

- des groupes statiques excèdent la capacité d'une carte CP en ce qui concerne leur consommation de mémoire ou leur consommation de CPU intrinsèque. Rappelons que les groupes statiques composés d'un seul processus peuvent dépasser la limite CPU conseillée

d'une carte CP.

Il s'agit dans ces cas d'initialiser un dialogue avec l'utilisateur afin de retrouver cette cohérence.

3.2 Construction du graphe

3.2.1 Situation de départ

A partir des informations cohérentes dont nous disposons, il est possible de construire un graphe de représentation des groupes et des gains mémoire et CPU pouvant être réalisés lorsque deux groupes constitués de processus homonymes ou voisins, sont placés sur une même carte CP. Nous en présentons la définition complète suivie d'un exemple réel simple à la figure 4.3. Le graphe représentant les groupes est constitué de noeuds que nous appelons NOYAUX et d'arêtes de gain MEMOIRE ou CPU.

Un noyau, qui au départ correspond à un groupe, résulte de la fusion d'un ou plusieurs noyaux. Ses attributs sont la liste des groupes le composant, ses consommations mémoire et CPU intrinsèque et extrinsèque. Un noyau issu d'une fusion ne peut jamais requérir plus de mémoire ou de CPU que ce qui a été fixé comme limites mémoire et CPU d'un noyau. Un noyau est également caractérisé par une consommation de "MMU banks" et de "PNR". A chaque noyau est associé un numéro de carte CP qui est soit indéfini, soit égal au numéro de carte CP imposé par l'utilisateur pour le placement d'un ou plusieurs groupes, qui font partie de ce noyau.

De même que les groupes, tout noyau est de type "FEP" ou non. Au départ, le noyau est de même type que le groupe qu'il représente. Le noyau résultant de la fusion d'un noyau de type "FEP" et d'un noyau de type "NON FEP" est de type "NON FEP". Seuls les noyaux de type "FEP" sont par définition décomposables.

Remarquons que les tailles mémoire et CPU limites des noyaux peuvent varier lors de différents placements. Varier la taille des noyaux permet de varier les résultats obtenus. Suite à l'expérimentation de cette méthode, nous espérons pouvoir définir soit une norme idéale, soit un calcul précis de ces tailles limites des noyaux; ceci nous permettrait d'obtenir rapidement un résultat optimal.

Une arête du graphe représente :

- soit un gain de mémoire (en kB). Le poids indiqué sur l'arête représente un gain de place mémoire correspondant à un partage de code et de données communes, si les deux noyaux adjacents sont disposés sur une même carte CP. Remarquons que nous ne tenons pas compte des gains de "MMU banks" et de "PNR" possibles; en effet ces deux ressources sont toujours en quantités disponibles suffisantes sur les cartes CP.

  Exemple : placer les noyaux correspondants aux groupes "MDPER 1" et "MDPER 2" sur une même carte CP permet d'économiser 14 kB.

- soit un gain de CPU extrinsèque (en ms/s). Le poids indiqué sur cette arête est la puissance CPU extrinsèque relative aux échanges d'informations effectués entre les deux noyaux adjacents. Cette consommation CPU représente le gain de CPU extrinsèque réalisé lorsque les deux noyaux sont placés sur une même carte CP.

  Exemple : les noyaux correspondants aux groupes "DS 1" et "DSM 4" consomment 92 ms/s en échanges d'informations lorsqu'ils sont sur des cartes CP différentes.

Notons qu'il est possible de trouver à la fois une arête de gain MEMOIRE et une arête de gain CPU entre deux noyaux.

Voici quelques remarques au sujet de nos exemples :

les limites d'un noyau sont toujours caractérisées par une capacité mémoire de 512 kB et une capacité CPU de 1000 ms/s.

Nos représentations de noyaux et d'arêtes sont les suivantes :

un noyau est caractérisé par un nom, ses consommations de mémoire (MEM) et ses consommations de CPU intrinsèque (CPUI) et extrinsèque (CPUE). Nous ne représentons ni la consommation de "MMU banks" ni la qualité de "FEP"; ces caractéristiques étant sans importance au niveau des exemples choisis. Notons également que lorsqu'une quantité n'est pas intéressante pour l'exemple, nous la remplaçons par un point d'interrogation. Cette représentation est schématisée à la figure 4.1.

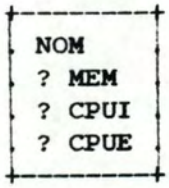


Figure 4.1 : représentation d'un noyau

Lorsque nous parlons d'"arête MEMOIRE" il s'agit d'une arête dont le poids représente un gain mémoire, tandis qu'"arête CPU" correspond à un gain CPU.

Une arête unissant deux noyaux est caractérisée par le gain mémoire et/ou CPU qu'elle représente. Ceci est montré à la figure 4.2.

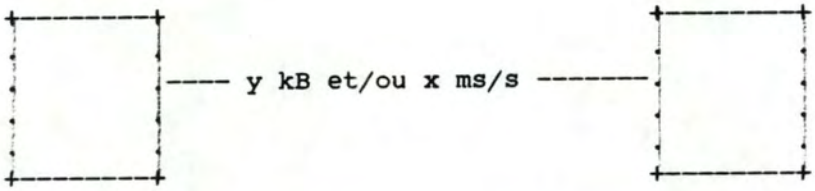


Figure 4.2 : représentation d'une arête.

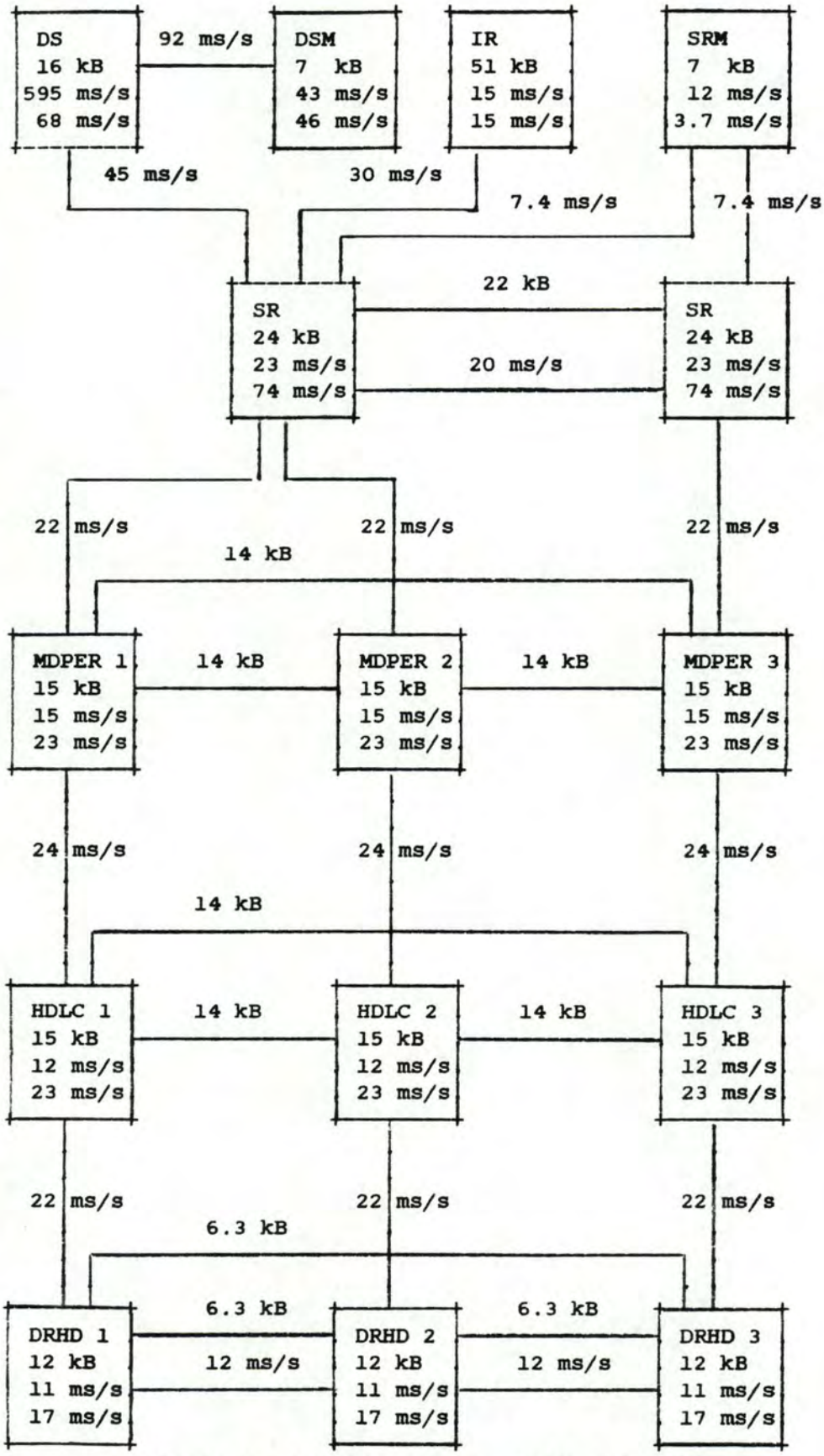


Figure 4.3 : exemple réel de graphe de noyaux

3.2.2 Calcul du rapport CPU-MEMOIRE

Pour nous permettre d'orienter le placement dans le sens d'une plus grande optimisation de la mémoire ou du CPU, nous nous servons du rapport CPU-MEMOIRE. Le rapport CPU-MEMOIRE indique quelle est la ressource CPU ou mémoire considérée comme étant la plus critique.

Lorsque le rapport CPU-MEMOIRE n'est pas fourni par l'utilisateur, le système le calcule lui-même. Dans ce cas, le rapport CPU-MEMOIRE est le rapport entre le nombre minimum de cartes CP nécessaires en ce qui concerne le CPU et le nombre minimum de cartes CP nécessaires en ce qui concerne la mémoire, pour placer les groupes. Le système calcule le nombre minimum de cartes CP nécessaires pour la mémoire : ce nombre est égal au quotient de la somme des besoins mémoire de chaque processus, en supposant que tous les "threads" d'un même processus sont sur une même carte, par la capacité mémoire d'une carte CP. Il calcule ensuite le nombre minimum de cartes CP nécessaires au point de vue du CPU : ce nombre est égal au quotient de la somme du CPU intrinsèque de chaque processus par la capacité CPU d'une carte CP. Notons que ne pas tenir compte des consommations de CPU extrinsèque suppose que tous les processus voisins entre eux sont sur une même carte CP. La ressource nécessitant le plus de cartes CP est celle à économiser dans le but de limiter le nombre de cartes CP à utiliser.

Nous formalisons le calcul du rapport CPU-MEMOIRE de la manière suivante :

- soit CPU-MIN = la somme du CPU intrinsèque de chaque processus, en négligeant les échanges d'informations entre processus;
- soit NBRE CPU-MIN = CPU-MIN divisé par la capacité CPU d'une carte CP, représente le nombre minimum de cartes CP nécessaires au point de vue du CPU pour pouvoir activer tous les processus;
- soit MEM-MIN = la somme des besoins mémoire de chaque processus, en supposant que tous les "threads" d'un même programme sont sur une même carte CP;
- soit NBRE MEM-MIN = MEM-MIN divisé par la capacité mémoire d'une carte CP, représente le nombre minimum de cartes CP nécessaires au point de vue de la mémoire pour pouvoir placer tous les processus.

Le rapport CPU-MEMOIRE est alors donné par la formule :

$$\frac{\text{NBRE CPU-MIN}}{\text{NBRE MEM-MIN}}$$

3.2.3 Comparabilité des arêtes

Pour pouvoir mesurer l'intérêt à fusionner deux noyaux, il est indispensable qu'il n'y ait qu'une arête entre eux et que toutes les arêtes soient comparables. A cette fin nous transformons le poids des arêtes, qui est un poids absolu, en une proportion d'utilisation des ressources d'une carte CP, le poids des arêtes est alors dit relatif. Ce poids relatif est ensuite transformé en un poids normalisé qui peut servir d'échelle de comparaison. Ceci se réalise de la manière suivante : le poids relatif des arêtes mémoire reste inchangé tandis que le poids relatif des arêtes CPU est multiplié par le rapport CPU-MEMOIRE. Normaliser le poids des arêtes nous permet de ne garder qu'une seule arête entre deux noyaux. Le poids de cette arête est égal à la somme des poids normalisés des arêtes qui les reliaient.

Nous pouvons remarquer que lorsque le rapport CPU-MEMOIRE est égal à 1, le poids de toute arête est inchangé et que nous n'influençons en rien le placement. Lorsque le rapport CPU-MEMOIRE est supérieur à 1, l'avantage est donné aux arêtes CPU; dans le cas contraire il est donné aux arêtes mémoire.

Nous illustrons cette transformation par l'exemple suivant :

Soient 2 noyaux A et B reliés par une arête MEMOIRE de poids absolu de 51 kB et par une arête CPU de poids absolu de 10 ms/s, ces données sont visualisées à la figure 4.4;

soit un rapport CPU-MEMOIRE égal à 1,5
soit (pour rappel) une capacité mémoire de 512 kB
soit (pour rappel) une capacité cpu de 1000 ms/s

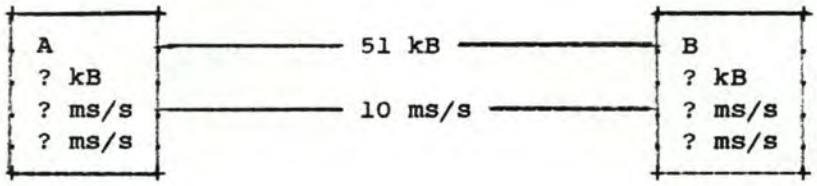


Figure 4.4 : graphe dont le poids des arêtes est absolu

Calculons les poids relatifs de ces arêtes

- 51 kB / 512 kB = 10 %

- 10 ms/s / 1000 ms/s = 1 %

Ces résultats sont illustrés à la figure 4.5

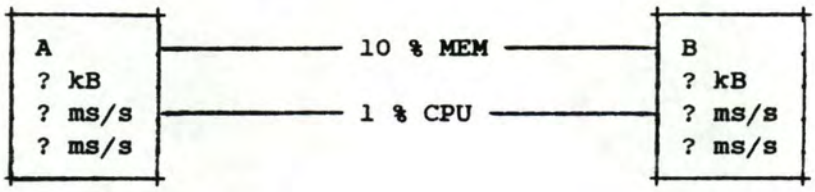


Figure 4.5 : graphe dont le poids des arêtes est relatif

Calculons ensuite les consommations normalisées par le rapport CPU-MEMOIRE :

- 10 * 1 = 10
- 1 * 1,5 = 1,5

Ces résultats sont illustrés à la figure 4.6.

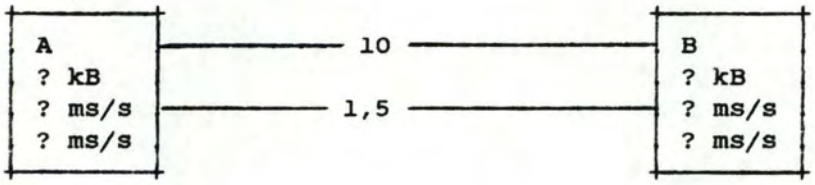


Figure 4.6 : graphe dont le poids des arêtes est normalisé

Regroupons alors les arêtes unissant 2 noyaux :

la somme des poids (A, B) = 10 + 1,5 = 11,5

Ce résultat est illustré à la figure 4.7.

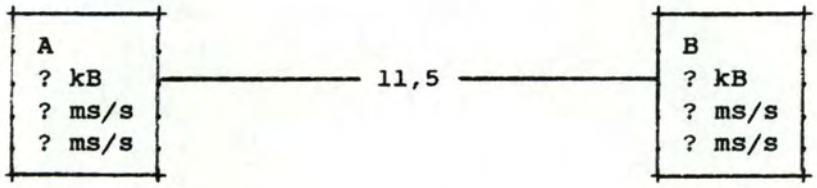


Figure 4.7 : graphe à arête unique entre 2 noyaux, arête dont le poids est normalisé

3.3 Fusion des noyaux du graphe

3.3.1 Stratégie générale

Notre but dans cette étape est de former à l'intérieur du graphe des noyaux de tailles mémoire et CPU limitées.

Le procédé utilisé est le suivant : nous recherchons dans le graphe l'arête de poids maximum; en effet cette arête représente le gain maximum qui peut être réalisé lors de la fusion de deux noyaux. Ensuite, nous déterminons si cette arête est "possible", "impossible absolue" ou "impossible temporaire" et réalisons le traitement adéquat. Si l'arête est "possible" nous fusionnons les noyaux qu'elle unit. Si elle est "impossible absolue", nous la détruisons : en effet, bien que l'arête les unissant soit l'arête maximum, il arrive que deux noyaux ne peuvent être fusionnés. Enfin, si l'arête est "impossible temporaire" nous ne faisons rien.

Remarquons que dans nos exemples nous avons choisi de représenter les arêtes en notant leur poids absolu mémoire ou CPU. Nous avons pris cette décision pour que le lecteur puisse facilement savoir ce que représente l'arête; dans la réalité nous ne traitons jamais que des poids normalisés.

3.3.2 Définition d'arête "POSSIBLE"

Une arête est dite "possible" si les quatre conditions suivantes sont vérifiées :

- il n'y a pas d'incompatibilité entre les groupes correspondants aux deux noyaux qui lui sont adjacents;

- lorsque les deux noyaux qui lui sont adjacents ont un numéro de carte CP défini, ils sont les mêmes;

- les besoins en mémoire des deux noyaux qui lui sont adjacents moins la quantité mémoire qu'elle représente ne dépassent pas la limite mémoire d'un noyau;

- la somme des besoins CPU intrinsèque et CPU extrinsèque des deux noyaux qui lui sont adjacents moins la quantité de CPU extrinsèque qu'elle représente ne dépasse pas la capacité CPU

limite d'un noyau.

Dans ce cas, il nous est possible de fusionner ces deux noyaux et de supprimer l'arête.

Les caractéristiques du nouveau noyau sont calculées de la manière suivante :

- la liste des groupes le composant est l'union des listes des groupes des deux noyaux fusionnés;
- les besoins en mémoire sont égaux à la somme des besoins en mémoire des deux noyaux fusionnés moins la quantité mémoire représentée par l'arête qui les unit;
- les besoins en puissance CPU intrinsèque sont la somme des besoins en puissance CPU intrinsèque des deux noyaux fusionnés;
- les besoins en puissance CPU extrinsèque sont la somme des besoins en puissance CPU extrinsèque des deux noyaux fusionnés moins la quantité de CPU extrinsèque représentée par l'arête qui les unit;
- le numéro de carte CP qui lui est associé est indéfini ou égal à celui associé à au moins un des deux noyaux fusionné;
- lorsque les deux noyaux sont de même type : "NON FEP" ou "FEP", le nouveau noyau est également de ce type. Le noyau issu d'une fusion d'un noyau de type "NON FEP" et d'un noyau de type "FEP" est de type "NON FEP".

Les nouvelles arêtes sont obtenues ainsi :

- les arêtes mémoire des noyaux fusionnés vers des noyaux tiers s'additionnent pour former une nouvelle arête vers ces mêmes noyaux, sauf lorsque deux arêtes représentent un gain mémoire correspondant à un même processus. Dans ce dernier cas, il ne faut alors comptabiliser ce gain qu'une seule fois : la nouvelle arête est égale à une des deux anciennes;
- les arêtes CPU des noyaux fusionnés vers des noyaux tiers s'additionnent pour former une nouvelle arête vers ces mêmes noyaux.

Nous proposons deux exemples concrets pour lesquels nous faisons l'hypothèse que les deux premières conditions de la définition d'"arête possible" sont vérifiées :

a. L'arête maximum est une arête CPU "possible"

Soient les noyaux :

- A nécessitant 100 kB mémoire, 500 ms/s de CPU intrinsèque et 125 ms/s de CPU extrinsèque;
- B nécessitant 100 kB mémoire, 200 ms/s de CPU intrinsèque et 125 ms/s de CPU extrinsèque;
- C et D dont les caractéristiques ne sont pas importantes.

Soient les arêtes :

- une arête (A, B) correspondant à 150 ms/s d'échange;
- une arête (A, C) correspondant à 90 ms/s d'échange;
- une arête (A, D) correspondant à 10 ms/s d'échange;
- une arête (B, D) correspondant à 100 ms/s d'échange.

Une visualisation de ces informations est présentée à la figure 4.8.

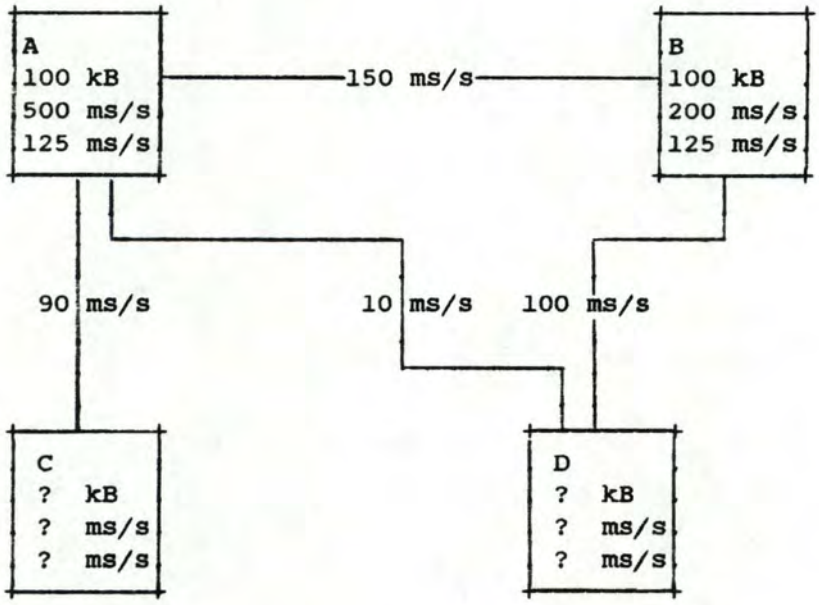


Figure 4.8 : graphe dont l'arête maximum est une CPU "possible"

L'arête maximum (A, B) est "possible", en effet :

- la somme des besoins mémoire de A et B est inférieure à 512 kB :
  100 kB + 100 kB - 0 kB = 200 kB < 512 kB

- la somme des besoins CPU de A et B est inférieure à 1000 ms/s :

  - le CPU intrinsèque nécessaire
    = 500 ms/s + 200 ms/s = 700 ms/s

  - le CPU extrinsèque nécessaire
    = 125 ms/s + 125 ms/s - 150 ms/s
    = 100 ms/s

  - le CPU total nécessaire est donc de 800 ms/s
    < 1000 ms/s

Nous pouvons fusionner ces deux noyaux et le résultat obtenu, visualisable à la figure 4.9, est le suivant :

Soient les noyaux :

- A+B nécessitant 200 kB mémoire, 700 ms/s de CPU intrinsèque et 100 ms/s de CPU extrinsèque;

- C et D dont les caractéristiques n'ont pas varié.

Soient les arêtes :

- une arête (A+B, C) correspondant à 90 ms/s d'échange;

- une arête (A+B, D) correspondant à 110 ms/s d'échange.

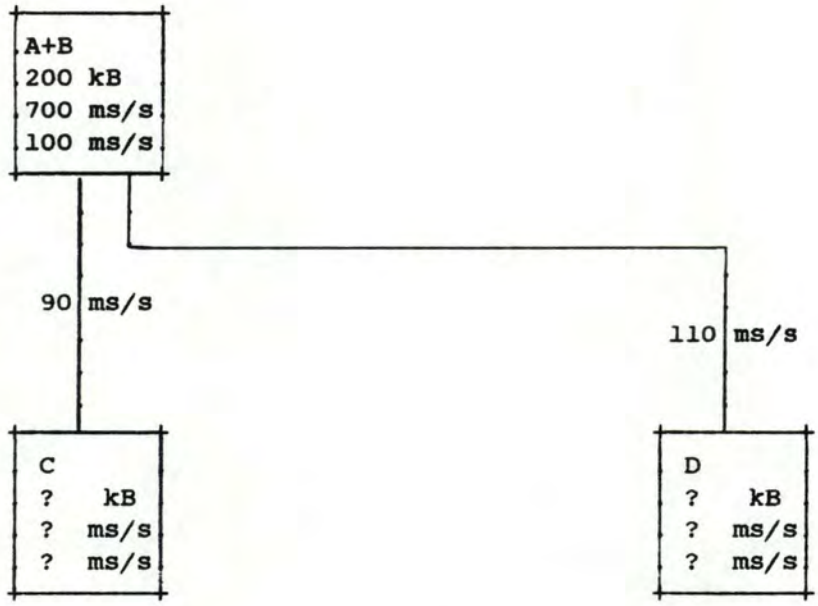


Figure 4.9 : résultat d'une fusion suivant une arête CPU maximum "possible"

b. L'arête maximum est une arête MEMOIRE "possible"

Soient les noyaux :

- A nécessitant 100 kB mémoire, 500 ms/s de CPU intrinsèque et 45 ms/s de CPU extrinsèque;
- B nécessitant 100 kB mémoire, 200 ms/s de CPU intrinsèque et 45 ms/s de CPU extrinsèque;
- C et D dont les caractéristiques ne sont pas importantes .

Soient les arêtes :

- une arête (A, B) correspondant à 80 kB;
- une arête (A, C) correspondant à 90 ms/s d'échange;
- une arête (A, D) correspondant à 50 kB;
- une arête (B, D) correspondant à 100 kB.

Une visualisation de ces informations est présentée à la figure 4.10.

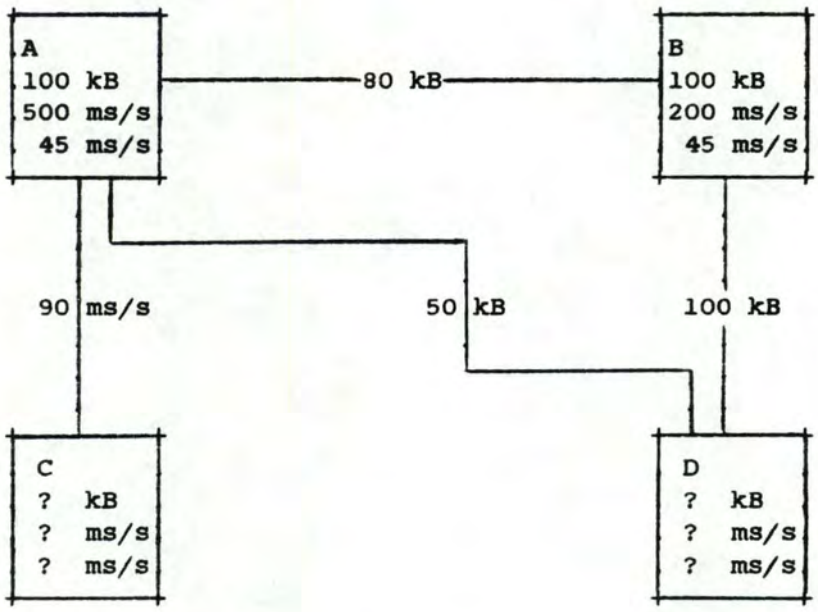


Figure 4.10 : graphe dont l'arête maximum est une MEMOIRE "possible"

L'arête maximum (A, B) est "possible", en effet :

- la somme des besoins mémoire de A et B est inférieure à 512 kB :
  100 kB + 100 kB - 80 kB = 120 kB < 512 kB

- la somme des besoins CPU de A et B est inférieure à 1000 ms/s :

  - le CPU intrinsèque nécessaire
    = 500 ms/s + 200 ms/s = 700 ms/s

  - le CPU extrinsèque nécessaire
    = 50 ms/s + 50 ms/s - 0 ms/s
    = 100 ms/s

  - le CPU total nécessaire est donc de
    800 ms/s < 1000 ms/s

Nous pouvons fusionner ces deux noyaux et le résultat obtenu, visualisable à la figure 4.11, est le suivant :

Soient les noyaux :

- A+B nécessitant 120 kB mémoire, 700 ms/s de CPU intrinsèque et 45 ms/s de CPU extrinsèque;

- C et D dont les caractéristiques n'ont pas varié.

Soient les arêtes :

- une arête (A+B, C) correspondant à 90 ms/s d'échange;
- une arête (A+B, D) correspondant à 100 kB si les 50 kB de l'arête (A, D) sont en fait un gain mémoire pour un processus compris dans la représentation des 100 kB de l'arête (B, D). En effet si par exemple, les noyaux A, B, D étaient tous constitués d'un même processus "x" apportant un gain de 50 kB lors d'une fusion, le fait de fusionner A et B implique que ce gain vis à vis de D ne doit être compté qu'une seule fois.

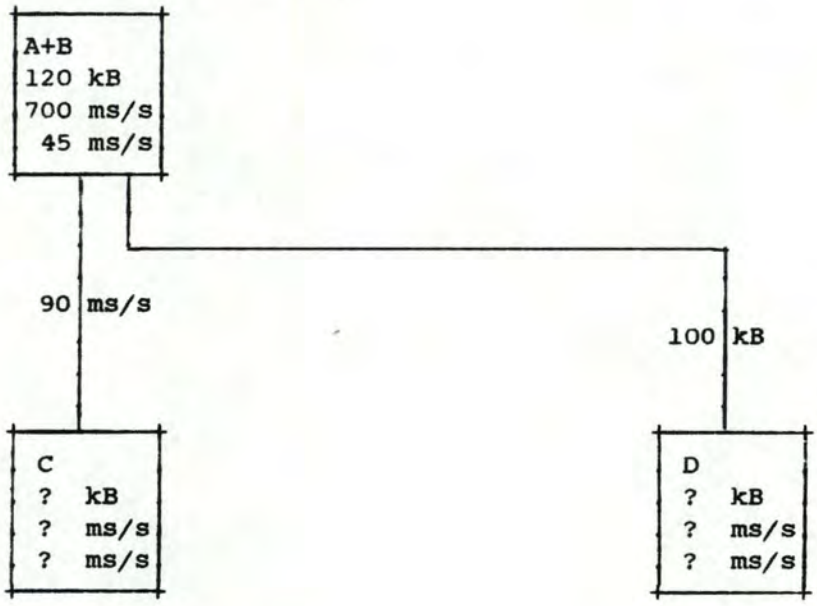


Figure 4.11 : résultat d'une fusion suivant une arête MEMOIRE maximum "possible"

3.3.3 Définition d'arête "IMPOSSIBLE ABSOLUE"

Une arête est dite "impossible absolue" dès qu'une des quatre propositions suivantes est vraie :

- il existe une incompatibilité entre des groupes correspondants aux noyaux qui lui sont adjacents;
- différents numéros de carte CP sont associés aux noyaux qui lui sont adjacents;

- l'addition des besoins CPU intrinsèque des deux noyaux qui lui sont adjacents dépasse la quantité CPU limite d'un noyau;

- la somme des besoins mémoire des deux noyaux qui lui sont adjacents, dont est soustraite la quantité représentée par l'arête, dépasse la capacité limite mémoire d'un noyau.

Sachant que ce gain est définitivement irréalisable, nous supprimons l'arête du graphe. Nous proposons deux exemples concrets illustrant les deux dernières propositions :

a. L'arête maximum est une arête CPU "impossible absolue"

Soient les noyaux :

- A nécessitant 100 kB mémoire, 550 ms/s de CPU intrinsèque et 100 ms/s de CPU extrinsèque;

- B nécessitant 100 kB mémoire, 600 ms/s de CPU intrinsèque et 100 ms/s de CPU extrinsèque.

Soit une arête :

- une arête (A, B) correspondant à 200 ms/s d'échange.

Une visualisation de ces informations est présentée à la figure 4.12.

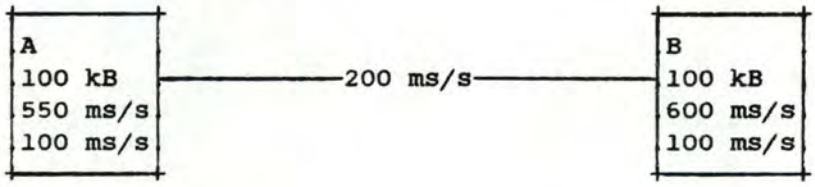


Figure 4.12 : graphe dont l'arête maximum est une CPU "impossible absolue"

L'arête maximum (A, B) est "impossible", en effet :

- la somme des besoins CPU intrinsèque de A et B est supérieure à 1000 ms/s :

  - le CPU intrinsèque nécessaire
    = 550 ms/s + 600 ms/s = 1150 ms/s

Comme il sera toujours impossible de fusionner ces deux noyaux, l'arête (A, B) peut être supprimée. Le résultat obtenu visualisable à la figure 4.13 est le suivant :

Soient les noyaux :

- A nécessitant 100 kB mémoire, 550 ms/s de CPU intrinsèque et 100 ms/s de CPU extrinsèque;
- B nécessitant 100 kB mémoire, 600 ms/s de CPU intrinsèque et 100 ms/s de CPU extrinsèque.

Figure 4.13 : résultat d'une rupture d'arête "impossible" CPU

b. L'arête maximum est une arête MEMOIRE "impossible absolue"

Soient les noyaux :

- A nécessitant 400 kB mémoire, 200 ms/s de CPU intrinsèque et 0 ms/s de CPU extrinsèque;
- B nécessitant 400 kB mémoire, 300 ms/s de CPU intrinsèque et 0 ms/s de CPU extrinsèque.

Soit une arête :

- une arête (A, B) correspondant à 100 kB.

Une visualisation de ces informations est présentée à la figure 4.14.

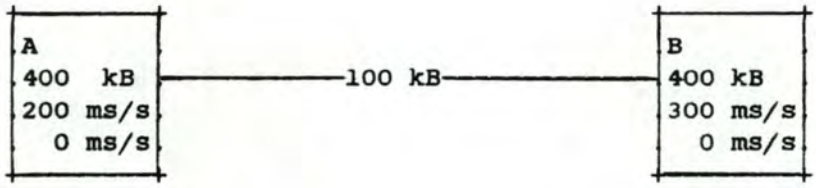


Figure 4.14 : graphe dont l'arête maximum est une arête MEMOIRE "impossible absolue"

L'arête maximum (A, B) est "impossible absolue", en effet :

- la somme des besoins mémoire de A et B est supérieure à 512 kB : 400 kB + 400 kB - 100 kB > 512 kB

Comme il sera toujours impossible de fusionner ces deux noyaux, l'arête (A, B) peut être supprimée. Le résultat obtenu, visualisable à la figure 4.15, est le suivant :

Soient les noyaux :

- A nécessitant 400 kB mémoire, 200 ms/s de CPU intrinsèque et 0 ms/s de CPU extrinsèque;
- B nécessitant 400 kB mémoire, 300 ms/s de CPU intrinsèque et 0 ms/s de CPU extrinsèque.

Figure 4.15 : résultat d'une rupture d'arête MEMOIRE "impossible absolue"

3.3.4 Définition d'arête "IMPOSSIBLE TEMPORAIRE"

Une arête est dite "impossible temporaire" si elle n'est ni "possible" ni "impossible absolue". En ce sens qu'il est à ce moment impossible de fusionner les noyaux qui lui sont adjacents mais que, suite à d'autres fusions, ils pourraient être réunis. Dans ce cas, aucun traitement n'est effectué et l'arête est temporairement ignorée.

Démontrons premièrement qu'il existe des arêtes CPU "impossibles temporaires", ensuite qu'il n'existe pas d'arête mémoire "impossible temporaire".

a. Existence d'arêtes CPU "impossibles temporaires"

Nous pouvons prouver l'existence d'arêtes CPU "impossibles temporaires" en montrant que la consommation CPU totale d'un noyau issu d'une fusion peut être inférieure à celle d'un des noyaux fusionnés.

En effet :

Soient 1 et 2, deux noyaux quelconques distincts.

CPU total (fusion 1-2) < CPU total (1) ?

ce qui équivaut à :

CPU Intr. (1) + CPU Intr. (2) + CPU Extr. (1)
+ CPU Extr. (2) - arête (1, 2)
<
CPU Intr. (1) + CPU Extr. (1) ?

ce qui équivaut à :

CPU Intr. (2) < arête (1, 2) - CPU Extr. (2) ?

Ceci est possible et signifie que la consommation de CPU intrinsèque du noyau (2) est inférieure à la consommation de CPU extrinsèque nécessaire aux échanges d'informations du noyau (1) vers le noyau (2).

b. Inexistence d'arêtes MEMOIRE "impossibles temporaires"

Nous pouvons prouver qu'il n'existe pas d'arêtes MEMOIRE "impossibles temporaires" en montrant que la consommation mémoire d'un noyau issu d'une fusion ne peut être inférieure à celle d'un des noyaux fusionnés.

En effet :

MEMOIRE (fusion 1-2) < MEMOIRE (1) ?

ce qui équivaut à :

MEMOIRE (1) + MEMOIRE (2) - arête (1, 2)
< MEMOIRE (1) ?

ce qui équivaut à :

MEMOIRE (2) - arête (1, 2) < 0 ?

ce qui équivaut à :

MEMOIRE (2) < arête (1, 2) ?

Ceci est impossible car la quantité "MEMOIRE (2)" qui représente le coût mémoire d'un premier "thread" d'un programme sur une carte CP est TOUJOURS plus grande que la quantité d'une

arête (1, 2) qui représente le gain de mémoire réalisé lors de l'installation d'un "thread" supplémentaire du même programme sur la même carte CP.

Nous proposons un exemple concret dans lequel l'arête maximum est une arête CPU "impossible temporaire".

Soient les noyaux :

- A nécessitant 100 kB mémoire, 700 ms/s de CPU intrinsèque et 275 ms/s de CPU extrinsèque;
- B nécessitant 100 kB mémoire, 200 ms/s de CPU intrinsèque et 200 ms/s de CPU extrinsèque;
- C nécessitant 100 kB mémoire, 50 ms/s de CPU intrinsèque et 175 ms/s de CPU extrinsèque.

Soient les arêtes :

- une arête (A, B) correspondant à 300 ms/s d'échange;
- une arête (A, C) correspondant à 250 ms/s d'échange;
- une arête (B, C) correspondant à 100 ms/s d'échange.

Une visualisation de ces informations est présentée à la figure 4.16.

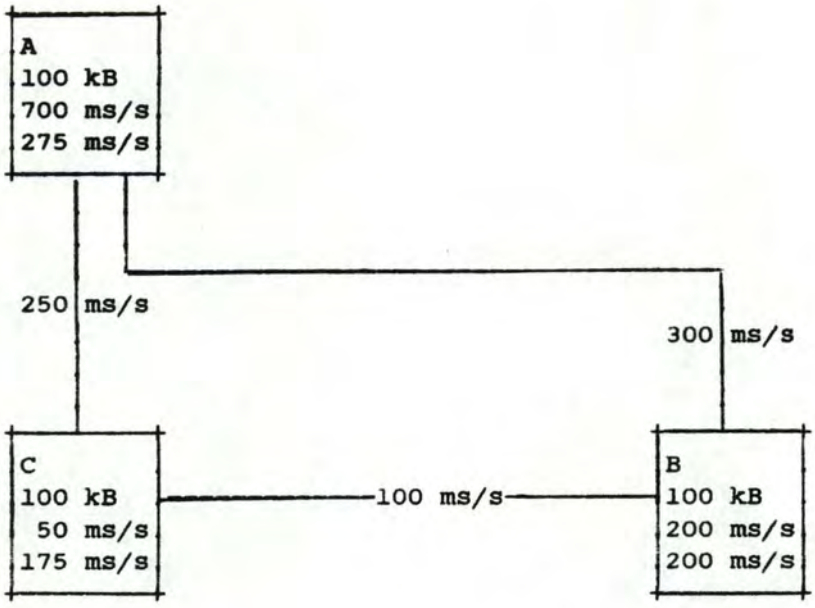


Figure 4.16 : graphe dont l'arête maximum est une arête CPU "impossible temporaire"

L'arête maximum (A, B) est "impossible temporaire", en effet :

- la somme des besoins mémoire de A et B est inférieure à 512 kB : 100 kB + 100 kB - 0 kB = 200 kB < 512 kB

- la somme des besoins CPU de A et B est supérieure à 1000 ms/s :

  - le CPU intrinsèque nécessaire = 700 ms/s + 200 ms/s = 900 ms/s

  - le CPU extrinsèque nécessaire = 275 ms/s + 200 ms/s - 300 ms/s = 175 ms/s

  - le CPU total nécessaire est donc de 1075 ms/s > 1000 ms/s

Suite à une fusion des noyaux A et C, les arêtes (A, B) et (C, B) vont disparaitre et une nouvelle arête (A+C, B) va être créée. Cette nouvelle arête pourrait être "possible".

En effet, l'arête (A, C) est une arête "possible" :

- la somme des besoins mémoire de A et C est inférieure à 512 kB : 100 kB + 100 kB - 0 kB = 200 kB < 512 kB

- la somme des besoins CPU de A et C est inférieure à 1000 ms/s :

  - le CPU intrinsèque nécessaire
    = 700 ms/s + 50 ms/s = 750 ms/s

  - le CPU extrinsèque nécessaire
    = 275 ms/s + 175 ms/s - 250 ms/s = 200 ms/s

  - le CPU total nécessaire est donc de 950 ms/s
    < 1000 ms/s

Les noyaux A et C peuvent être fusionnés et le résultat, visualisable à la figure 4.17, est le suivant :

Soient les noyaux :

- A+C nécessitant 200 kB mémoire, 750 ms/s de CPU intrinsèque et 200 ms/s de CPU extrinsèque;

- B nécessitant 100 kB mémoire, 200 ms/s de CPU intrinsèque et 200 ms/s de CPU extrinsèque.

Soit une arête :

- une arête (A+C, B) correspondant à 400 ms/s d'échange.

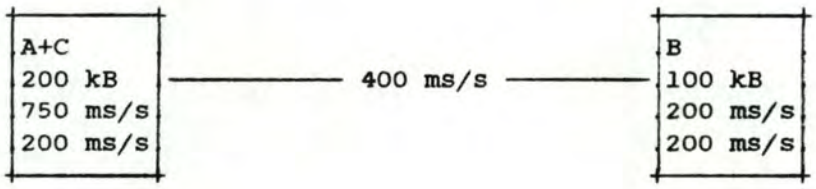


Figure 4.17 : résultat d'une fusion suivant une arête CPU "possible"

L'arête (A+C, B) est elle "possible", en effet :

- la somme des besoins mémoire de A+C et B est inférieure à 512 kB : 200 kB + 100 kB - 0 kB = 300 kB < 512 kB

- la somme des besoins CPU de A+C et B est inférieure à 1000 ms/s :

  - le CPU intrinsèque nécessaire
    = 750 ms/s + 200 ms/s = 950 ms/s

  - le CPU extrinsèque nécessaire
    = 200 ms/s + 200 ms/s - 400 ms/s = 0 ms/s

- le CPU total nécessaire est donc de 950 ms/s
  < 1000 ms/s

Les noyaux A+C et B peuvent être fusionnés et le résultat, visualisable à la figure 4.18, est le suivant :

Soit le noyau :

- A+C+B nécessitant 300 kB mémoire, 950 ms/s de CPU intrinsèque et 0 ms/s de CPU extrinsèque

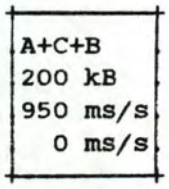


Figure 4.18 : résultat d'une fusion suivant une arête CPU "impossible temporaire" devenue "possible"

3.3.5 Transition de la qualité d'une arête

En résumé, nous présentons à la figure 4.19 un graphe de transition de la qualité d'une arête suite à une fusion.

Rappelons qu'une arête "impossible temporaire" ne devient elle-même jamais possible, mais que suite à une fusion, elle peut être remplacée par une arête "possible". C'est pourquoi à la figure 4.19, nous avons mis entre parenthèses ce type de transition.

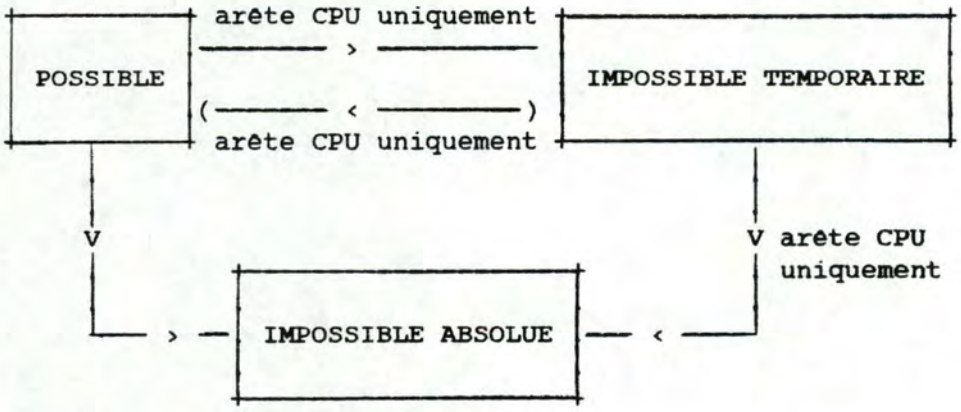


Figure 4.19 : graphe de transition de la qualité d'une arête

3.3.6 Condition d'arrêt du procédé de fusion

Le procédé de fusion est répété jusqu'à ce qu'il ne reste plus d'arêtes ou uniquement des arêtes "impossibles temporaires" entre les noyaux. Dans ce dernier cas, malgré la possibilité que celles-ci forment un cycle, nous considérons qu'elles n'ont plus aucune chance de devenir "possibles", vu que nous ne réalisons plus aucune fusion. Elles sont alors assimilées aux arêtes "impossibles absolues", c'est à dire supprimées. De cette sorte, il ne nous reste plus à la fin du procédé de fusion que des noyaux indépendants les uns des autres.

3.4 Placement des noyaux de type "NON FEP"

Sachant que les quantités de place mémoire et de puissance CPU sont limitées sur chaque carte CP et que les noyaux sont caractérisés par des besoins en mémoire et en CPU, notre problème, nous le rappelons, consiste à placer les noyaux sur un nombre minimum de cartes CP, tout en essayant d'obtenir des distributions mémoire et CPU optimales. Le résultat de cette étape est partiel puisque les noyaux de type "FEP", noyaux qui peuvent par définition être scindés, sont placés lors de l'étape suivante de manière à saturer les cartes CP.

Dans un premier temps nous plaçons les noyaux ayant dans leur composition un processus à placer sur une carte CP imposée par l'utilisateur. Ensuite, nous trions la liste des noyaux de type "NON FEP" restants selon les trois formules suivantes : l'ordre croissant selon la différence mémoire-CPU en valeur absolue, l'ordre décroissant selon la différence mémoire-CPU en valeur absolue et l'ordre alternatif favorisant la mémoire puis le CPU, calculé selon la différence mémoire-CPU en valeur absolue. Ces formules ont été choisies suite à un programme de simulation dont nous parlerons dans le chapitre de la justification de la méthode : le chapitre V. Pour terminer, suivant l'ordre de chaque liste triée, nous plaçons chaque noyau sur la première carte CP pouvant l'accueillir. S'il n'existe plus de carte CP disposant des ressources nécessaires, il est possible d'ajouter une carte CP vierge. Nous comparons alors les résultats obtenus selon les trois formules de tri et gardons uniquement le plus performant.

Notons que lors du placement d'un noyau sur une carte CP, il est possible de bénéficier d'un gain mémoire ou CPU. Ce gain est dû à la présence sur cette carte CP d'un noyau qui lui était adjacent dans le graphe des noyaux et lui aurait été fusionné si les limites des noyaux avaient été égales aux capacités disponibles d'une carte CP vierge. Pour que nos calculs soient corrects, nous tenons compte de ce gain. De plus, nous veillons à ce que le placement de différents noyaux sur

une même carte CP ne réunisse pas des groupes incompatibles.

3.5 Placement des noyaux de type "FEP"

Pour terminer, nous plaçons les noyaux de type "FEP", c'est à dire les noyaux pouvant ou devant être scindés. Il s'agit des noyaux correspondants à des groupes de type "FEP" n'ayant aucun voisin ou les ayant tous perdus suite au procédé de fusion. Placer les noyaux de type "FEP" en dernier lieu est capital pour saturer les cartes CP où des ressources sont encore disponibles.

Nous définissons le "coût de découpe" des "FEP". Ensuite nous expliquons le placement proprement dit des "FEP"; il se réalise en deux étapes : la première est une étape de tri qui permet d'ordonner les "FEP" principalement selon leur coût de découpe. La seconde est l'étape de placement de la liste triée des "FEP".

3.5.1 Coût de découpe des "FEP"

Chaque "FEP" est caractérisé par un coût de découpe. Ce coût est égal au supplément de mémoire nécessaire chaque fois qu'il faut installer un premier "thread" du "FEP" sur une nouvelle carte CP. Le CPU lui reste constant quelque soit le nombre de découpes du "FEP", il est réparti de manière proportionnelle. Ainsi la contrainte de placement des "FEP" se résume à minimiser le nombre de découpes de chaque "FEP".

3.5.2 Tri des "FEP"

Notre méthode consiste à classer les "FEP" selon trois critères puis à regrouper les "FEP" que nous définissons comme semblables.

Premièrement nous trions les "FEP" selon l'ordre décroissant du coût de découpe. Ensuite nous classons les "FEP" ayant un même coût de découpe selon l'ordre décroissant de la quantité de CPU qui leur est nécessaire par "thread". Enfin, lorsque différents groupes de "FEP" ont un même coût de découpe et ont besoin d'une même quantité de CPU par "thread", nous les trions selon l'ordre décroissant de la quantité "HRAM" nécessaire au "FEP".

Classer les "FEP" de cette manière nous permet de placer en premier lieu les "FEP" dont le coût de découpe est le plus important. Il est en effet intéressant de les placer lorsque certaines cartes CP

offrent encore assez de ressources pour pouvoir les accueillir dans leur totalité. Ensuite, nous favorisons le CPU avant de considérer un critère correspondant de nouveau à une consommation de mémoire.

Deux ou plusieurs "FEP" sont dits semblables, lorsqu'ils ont le même nom et donc un même coût de découpe, ainsi que la même quantité de CPU nécessaire par "thread" et une même quantité **HRAM**.

Nous regroupons des "FEP" semblables dans le but de les traiter en même temps. Ceci nous incite à choisir une carte CP pouvant accueillir ces "FEP". Les placer séparément nous aurait plus certainement mené à les placer sur des cartes CP différentes, nous forçant à installer plusieurs fois un premier "thread" de ce "FEP".

Nous présentons à la figure 4.20 un exemple de liste de "FEP" non triée puis à la figure 4.21 cette même liste classée.

NOM	NBRE DE "THREADS"	COUT DECOUPE	CPU / "THREADS"	HRAM
FMGR	5	42	20	0
FMGR	30	42	15	10
VMGR	10	52	50	0
FMGR	15	42	20	0
FMGR	30	42	15	0

Figure 4.20 : liste non triée de "FEP"

NOM	NBRE DE "THREADS"	COUT DECOUPE	CPU / "THREADS"	HRAM
VMGR	10	52	50	0
FMGR	20	42	20	0
FMGR	30	42	15	10
FMGR	30	42	15	0

Figure 4.21 : liste triée de "FEP"

3.5.3 Placement des "FEP"

Après avoir classé les "FEP" nous les traitons successivement selon la méthode exposée ci-après.

Nous essayons d'abord de placer un maximum de "threads" du "FEP" sur des cartes CP où nous avons déjà placé un "FEP" homonyme. De cette manière nous profitons de la première installation du "FEP" homonyme sur ces cartes CP.

Nous recherchons alors un minimum de cartes CP pouvant accueillir les "threads" restants. Nous présentons en un premier point le principe général de cette recherche. Nous la détaillons en un deuxième point et terminons en montrant quelques exemples très simples qui illustrent différentes étapes.

a. Principe général

Le principe général de la recherche d'un nombre minimum de cartes CP pouvant accueillir un nombre connu de "threads" d'un "FEP" déterminé, vise deux buts : minimiser le nombre de découpes de ce "FEP" et équilibrer dans la mesure du possible les distributions mémoire et CPU sur les cartes CP.

Pour minimiser le nombre de découpes du "FEP", nous calculons le nombre de "threads" du "FEP" que chaque carte CP peut accueillir et plaçons ce "FEP" sur un nombre minimum de cartes CP. Remarquons que nous ajoutons les cartes CP vierges indispensables pour pouvoir placer le "FEP" que nous traitons.

Le second but visé est un équilibre des distributions mémoire et CPU sur les cartes CP. Dans ce but, nous essayons de placer chaque "FEP" sur les cartes CP lui offrant un même rapport des ressources que ce qu'il consomme. Ce critère est secondaire et uniquement pris en compte lorsque nous avons le choix entre plusieurs cartes CP sans augmenter le nombre de découpes du "FEP".

b. Détail de la recherche.

Nous expliquons dans les paragraphes suivants le détail de la recherche d'un nombre minimum de cartes CP où placer un "FEP".

Premièrement, nous examinons s'il est nécessaire d'ajouter des cartes CP vierges pour pouvoir placer le "FEP". Nous considérons l'ensemble des ressources mémoire et CPU disponibles sur les cartes CP comme un tout et en faisons la somme. D'autre part, nous faisons l'hypothèse qu'il est possible de placer le "FEP" sans le découper et

nous calculons les puissances mémoire et CPU qui lui sont nécessaires. Ensuite, nous déterminons le rapport de la mémoire nécessaire au "FEP" vis-à-vis de la mémoire disponible sur les cartes CP et le rapport du CPU nécessaire au "FEP" vis-à-vis du CPU disponible sur les cartes CP. Nous pourrions constater que soit pour la mémoire, soit pour le CPU, l'ajout d'une ou plusieurs cartes CP est indispensable. Ayant déterminé le nombre de cartes CP vierges à ajouter, nous plaçons autant de "threads" du "FEP" que possible sur celles-ci à l'exception d'une que nous gardons disponible pour le traitement suivant.

Ce calcul, illustré par l'exemple 1, est très optimiste car il se base sur deux hypothèses qui ne sont que rarement vraies : premièrement, nous supposons que les ressources disponibles se trouvent sur une même carte CP et deuxièmement, qu'il est possible de placer ce "FEP" sans le découper. Il est cependant important de calculer ceci de manière optimiste afin de n'ajouter que des cartes CP nécessaires. N'oublions pas que notre but est d'obtenir un placement sur un nombre minimum de cartes CP. Il est toutefois intéressant d'ajouter ces cartes CP vierges à priori, ceci nous permet d'éviter de scinder inutilement certains "FEP" et d'augmenter ainsi les pertes mémoire.

Nous calculons alors pour chaque carte CP le nombre de "threads" du "FEP" considéré qu'elle peut accueillir. Ce nombre est fonction des quantités mémoire et CPU encore disponibles sur les cartes CP. Ensuite nous trions les cartes CP selon l'ordre décroissant du nombre de "threads" du "FEP" qu'elles peuvent accueillir. Prenant les cartes CP dans cet ordre, nous pouvons déterminer le nombre minimum "MIN" de cartes CP nécessaires pour placer le "FEP". Notons que lorsque nous ne disposons pas de cartes CP en suffisance, nous ajoutons une carte CP vierge.

Connaissant un nombre minimum de cartes CP à utiliser, nous tentons de réaliser le second but. Il s'agit d'équilibrer les distributions des ressources sur les cartes CP. Ainsi, nous réalisons les étapes suivantes.

Nous calculons un RAPPORT DE DISPONIBILITE de chaque carte CP, qui est égal au rapport entre le CPU disponible sur la carte CP et la mémoire disponible sur cette même carte CP. Ce calcul est illustré par l'exemple 2.

Nous calculons alors un coefficient, appellé RAPPORT DE CONSOMMATION du "FEP", qui est le rapport entre la consommation CPU du "FEP" et sa consommation mémoire. Ce calcul est illustré par l'exemple 3.

Ceci fait, nous classons les cartes CP en donnant la préférence à celles dont le rapport de disponibilité est le plus proche du rapport de consommation du "FEP" à placer. A partir de ce moment, nous ne

considérons plus que les "N" premières cartes CP de cette liste; "N" étant un nombre arbitraire. Nous cherchons alors un ensemble de "MIN" cartes CP telles que la somme des "threads" pouvant être accueillis est supérieure ou égale au nombre de "threads" à placer. L'exemple 4 est l'illustration de cette étape. Si nous trouvons un ensemble de cartes CP satisfaisant, nous plaçons le "FEP" sur ces cartes CP, autrement, nous le plaçons sur les "MIN" cartes CP déterminées par l'ordre décroissant du nombre de "threads" pouvant être accueillis par les cartes CP.

c. Exemples

Exemple 1 : le calcul du nombre de cartes CP vierges à ajouter.

- Soient deux cartes CP ayant respectivement 90 kB et 110 kB de mémoire disponible; ainsi que 200 ms/s et 400 ms/s de CPU disponible.

- Soit un groupe de 5 processus de type "FEP" consommant 140 kB pour l'installation d'un premier "thread" sur une carte CP, 20 kB par "thread" supplémentaire et 100 ms/s de CPU intrinsèque et extrinsèque par "thread".

La quantité de mémoire disponible sur les cartes CP est de :

90 kB + 110 kB = 200 kB

La quantité de CPU disponible sur les cartes CP est de :

200 ms/s + 400 ms/s = 600 ms/s

La quantité de mémoire nécessaire pour placer ce "FEP" est la suivante :

(1 * 140 kB) + (4 * 20 kB) = 220 kB

La quantité de CPU nécessaire pour placer ce "FEP" est la suivante :

5 * 100 ms/s = 500 ms/s

Le rapport de la mémoire nécessaire à l'installation du "FEP" et de la mémoire disponible sur les cartes CP est le suivant :

220 /200 = 1.1

Ce rapport est supérieur à 1 et inférieur à 2; ceci implique l'ajout d'une carte CP vierge.

Le rapport du CPU nécessaire à l'installation du "FEP" et du CPU disponible sur les cartes CP est le suivant :

500 /600 = 5 /6

Ce rapport est supérieur à 0 et inférieur à 1; ceci n'implique pas l'ajout de cartes CP vierges.

Exemple 2 : le calcul du rapport de disponibilité d'une carte CP.

- Soit une carte CP ayant 200 kB de mémoire disponible et 500 ms/s de CPU disponible.

Le rapport de disponibilité de la carte CP est le suivant :

500 / 200 = 5/2.

Exemple 3 : le calcul du rapport de consommation d'un "FEP".

- Soit un groupe de 5 processus de type "FEP" consommant 40 kB pour l'installation d'un premier "thread" sur une carte CP, 15 kB par "thread" supplémentaire et 100 ms/s de CPU intrinsèque et extrinsèque par "thread".

La quantité de mémoire nécessaire pour placer ce "FEP" est la suivante :

(1 * 40 kB) + (4 * 15 kB) = 100 kB

La quantité de CPU nécessaire pour placer ce "FEP" est la suivante :

5 * 100 ms/s = 500 ms/s

Le rapport de consommation de ce "FEP", est le suivant :

500 / 100 = 5

Exemple 4 : la recherche de cartes CP où placer un "FEP".

- Soit un groupe de 5 processus de type "FEP" dont le rapport de consommation est égal à 5.

- Soient trois cartes CP dont les numéros sont 1,2 et 3. Ces cartes CP ont respectivement un rapport de disponibilité égal à 1, 1/2 et 4 et peuvent respectivement accueillir 3, 2 et 2 "threads" du FEP défini ci-dessus.

Le classement de ces cartes CP par rapport au "FEP" est le suivant :

- la carte CP 3 dont le rapport de disponibilité est égal à 4 et pouvant accueillir 2 "threads" du "FEP";
- la carte CP 1 dont le rapport de disponibilité est égal à 1 et pouvant accueillir 3 "threads" du "FEP"
- la carte CP 2 dont le rapport de disponibilité est égal à 1/2 et pouvant accueillir 2 "threads" du "FEP".

Cherchant deux cartes CP pouvant accueillir les 5 "threads" du "FEP", nous choisissons alors les cartes CP 3 et 1.

3.6 Eléments d'évaluation du placement

Dans ce paragraphe, nous définissons des éléments d'évaluation du placement que nous avons obtenu. A cette fin, nous calculons certains repères et indicateurs.

3.6.1 Repères

Les repères sont invariables quelque soit le placement réalisé. Les deux repères que nous pouvons calculer avant même d'avoir réalisé le placement sont les nombres minimum et maximum de cartes CP pour la liste des groupes à placer. Ces deux valeurs nous fournissent une échelle à l'intérieur de laquelle nous pouvons situer et ainsi évaluer la valeur de notre placement.

Le nombre minimum de cartes CP est donné par la formule suivante :

$$\text{MAXIMUM}\left(\frac{\text{la somme des consommations mémoire minimales de tous les groupes}}{\text{la capacite mémoire d'une carte CP}},\ \frac{\text{la somme des consommations CPU intrinsèque de tous les groupes}}{\text{la capacité CPU d'une carte CP}},\right.$$

le nombre de cartes CP déjà allouées,

le nombre chromatique du graphe des incompatibilités entre groupes c'est à dire le nombre minimum de cartes CP qui permet de satisfaire aux incompatibilités entre groupes
).

Le nombre maximum de cartes CP se calcule ainsi :

MAXIMUM (la somme des consommations CPU intrinsèque et extrinsèque de tous les groupes
———————————————————————————— ,
la capacité CPU d'une carte CP

la somme des consommations mémoire maximum des groupes
————————————————————————————
la capacité mémoire d'une carte CP

le nombre de cartes CP déjà allouées,

le nombre chromatique du graphe des incompatibilités entre groupes c'est à dire le nombre minimum de cartes CP qui permet de satisfaire aux incompatibilités entre groupes
).

Remarquons que nous ne calculons pas le nombre chromatique du graphe des incompatibilités car le nombre d'incompatibilités est limité et que le calcul serait trop complexe par rapport à l'utilité du résultat obtenu.

3.6.2 Indicateurs relatifs

Les indicateurs relatifs varient selon le placement réalisé. Ils sont comparables pour différents placements d'une même liste de groupes lorsque ces placements utilisent un même nombre de cartes CP. Ces indicateurs, mémoire et CPU, révèlent la part de la consommation minimum possible des ressources, vis-à-vis de la consommation réelle du placement.

Notons que ces indicateurs, notés "$ MEMOIRE" et "$ CPU", sont toujours inférieurs ou égaux à 1. Le placement est le meilleur lorsqu'ils sont égaux à 1; ils indiquent dans ce cas qu'un minimum de ressources a été utilisé. Un exemple de ces indicateurs se trouve en annexe V.

Ces indicateurs sont :

$ MEMOIRE = la somme des consommations mémoire minimales des groupes / la somme des consommations mémoire des groupes pour le placement considéré.

$ CPU = la somme des consommations CPU intrinsèque des groupes / la somme des consommations CPU intrinsèque et extrinsèque des groupes pour le placement considéré

3.7 Rétroaction

Si l'utilisateur juge les résultats insatisfaisants, il peut demander un nouveau placement. Pour l'obtenir, il est possible soit de réduire la taille maximale des noyaux, soit de modifier le rapport CPU-MEMOIRE.

Une taille réduite des noyaux permet parfois, lors de l'étape de placement des noyaux de type "NON FEP", de saturer plus facilement les ressources offertes par les cartes CP. Mais n'oublions pas que réduire la taille des noyaux signifie la perte de gains mémoire et CPU non réalisés lors de la fusion du graphe. Ensuite, seul le hasard nous permet de les réaliser lors de l'étape de placement des noyaux de type "NON FEP".

Modifier le rapport CPU-MEMOIRE influence l'étape de fusion des noyaux et permet de varier les résultats. Notons qu'après une dizaine d'expériences, il semble que le rapport CPU-MEMOIRE calculé par le système donne des résultats très satisfaisants.

4 Organigramme

La figure 4.22 reprend sous forme d'organigramme les différentes étapes de la méthode de fusion ainsi que les données du problème et les résultats demandés. A chaque donnée, étape ou résultat est associé le numéro du paragraphe où il est développé.

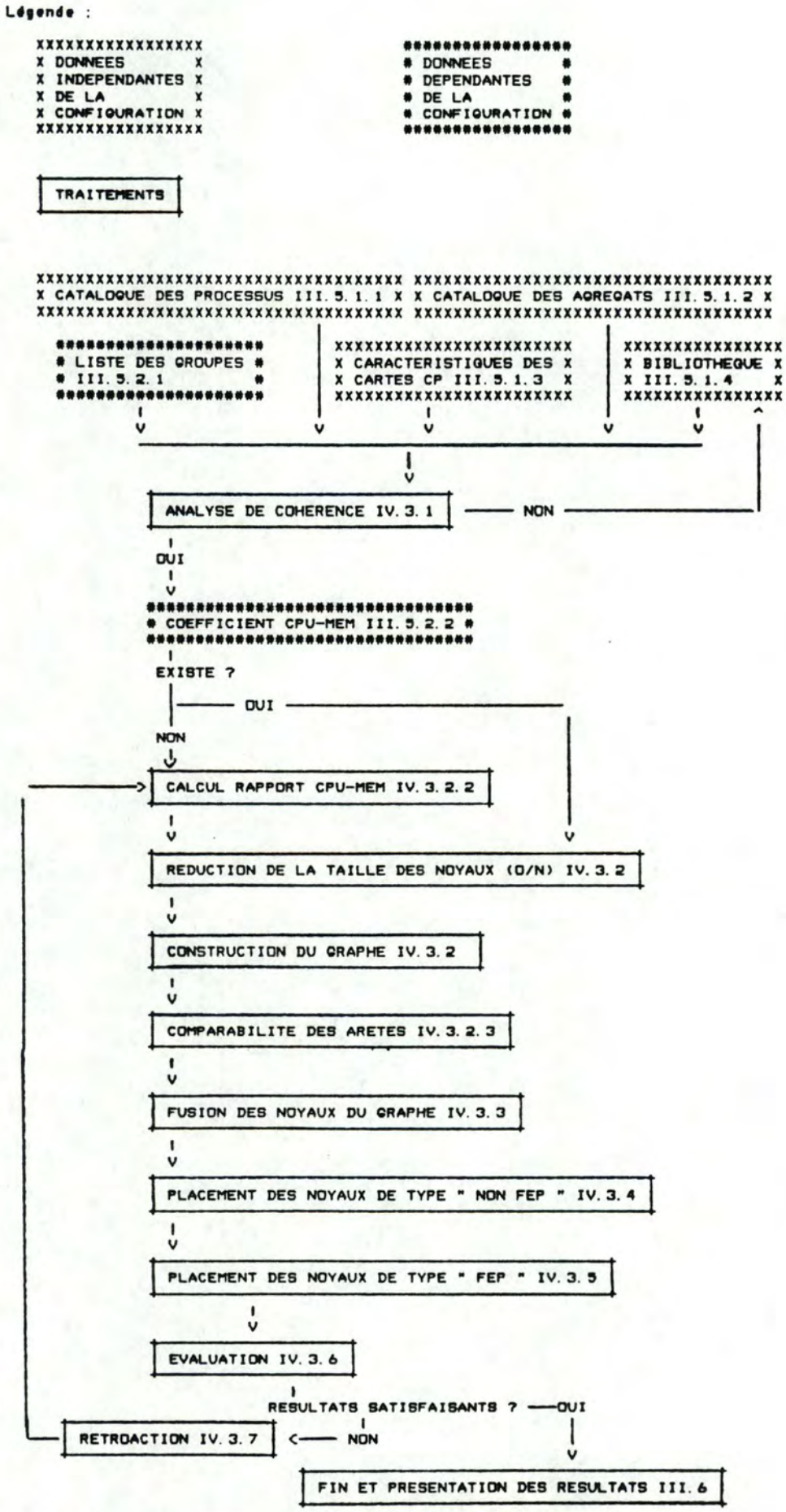


Figure 4.22 : organigramme de résolution

5 Complexité algorithmique

Nous évaluons la complexité algorithmique de la méthode de fusion en examinant le point critique de cette méthode : la recherche de l'arête maximum lors du procédé de fusion des noyaux.

Notons que dans nos formules "**" signifie "exposant".

Posons égal à "N" le nombre de groupes et donc de noyaux au départ.

Comme il y a au plus une arête de poids normalisé entre deux noyaux, le nombre maximum d'arêtes au départ est égal à :

N * (N-1)/2;

Pour simplifier les calculs, nous prenons N * N/2

Il est clair que chaque fusion réduit d'au moins une unité le nombre d'arêtes. Ainsi, au maximum, le nombre total d'arêtes à examiner au cours du procédé de fusion est donné par la formule suivante :

$$\frac{N ** 2}{2} + \frac{(N - 1) **2}{2} + \frac{(N - 2) ** 2}{2} + \ldots$$

Ce qui équivaut à :

$$\sum_{j=0}^{N-1} \frac{(N-j) ** 2}{2}$$

Ce qui équivaut à :

$$\frac{1}{2} * \frac{N}{6} (2 N ** 2 + 3 N + 1)$$

Au pire, la complexité de la méthode de fusion est donc de l'ordre de :

O (N ** 3)

6 Critiques

La méthode que nous proposons est heuristique. Affirmer que le résultat obtenu est optimal s'avère le plus souvent impossible. C'est pourquoi nous avons introduit une évaluation après chaque placement et une possibilité de rétroaction.

Dans le procédé de fusion des noyaux du graphe, nous ne mettons jamais en question la fusion des noyaux adjacents à l'arête maximum si elle est possible. La fusion de noyaux adjacents à des arêtes non maximum peut parfois donner un meilleur résultat. Mais, vu la contrainte de rapidité d'exécution à laquelle notre projet est soumis, il ne nous est pas possible de songer à des calculs plus élaborés. Nous ne devons en effet pas perdre de vue que SYANODE n'est qu'un élément d'un projet plus important : ANODE.

JUSTIFICATION DE LA METHODE DE FUSION

CHAPITRE V

1 Introduction

Dans le chapitre précédent nous avons décrit la méthode retenue pour résoudre le problème qui nous est proposé. Nous voulons ici expliciter notre démarche en montrant que ce problème est soluble et quelles furent les étapes qui nous amenèrent à opter pour la méthode de fusion. Nous avons premièrement pensé à une méthode purement aléatoire, ensuite à une méthode se basant sur l'analyse linéaire et enfin à une méthode de construction et de découpe d'un graphe. En conclusion nous expliquons en quoi ces solutions sont une progression vers la méthode de fusion et rappelons les traitements spécifiques à la méthode de fusion.

2 Caractère soluble du problème

Pour démontrer le caractère soluble du problème de placement nous devons montrer qu'il est toujours possible de placer une liste de groupes consommateurs de mémoire et de CPU, sur un nombre minimum de cartes CP offrant chacune des ressources mémoire et CPU limitées. Ceci revient à prouver que ce nombre de cartes CP a une borne supérieure finie. Nous trouvons intéressant de formaliser mathématiquement le problème et ses contraintes avant d'en montrer, très simplement, le caractère soluble.

2.1 Représentation du problème et de ses contraintes

Nous représentons les "m" groupes et "n" cartes CP à l'aide d'une matrice binaire "X" de taille (m, n). Chaque ligne de la matrice représente un groupe "i"; chaque colonne une carte CP "j". Un élément de la matrice "Xi,j" égal à "1" signifie que nous plaçons le groupe "i" sur la carte CP "j". De même, un élément égal à 0 signifie que le groupe "i" n'est pas placé sur la carte "j".

Nous présentons à la figure 5.1 un exemple de matrice montrant que dans ce cas il faut placer les groupes "1" et "3" sur la carte CP "1" et le groupe "2" sur la carte CP "2".

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Figure 5.1 : exemple de matrice de placement

Les contraintes du problème, que nous formalisons dans un système d'équation sont les suivantes :

tout groupe "i" ne peut être placé que sur une seule carte CP "j". Ceci est exprimé dans l'équation (1) de la figure 5.2.

Vu que les cartes CP sont limitées au niveau de la mémoire et du CPU disponibles, nous devons veiller à ce que les sommes des consommations de mémoire et de CPU des groupes "i", placés sur une carte CP "j" ne dépassent pas les limites de celle-ci. Pour présenter ces deux contraintes, mémoire et CPU, nous considérons comme égales à une unité de la ressource offerte, les quantités de mémoire et de CPU que toute carte CP peut offrir aux différents groupes. De plus, à chaque groupe "i" nous associons une consommation de mémoire "Mi", égale au rapport de consommation de mémoire de ce groupe vis-à-vis de la mémoire disponible d'une carte CP vierge, ainsi qu'une consommation de CPU "Pi", égale au rapport de consommation de CPU de ce groupe vis-à-vis du CPU disponible d'une carte CP vierge. Notons que tout groupe peut toujours être placé sur une carte CP. Ces deux contraintes sont respectivement exprimées dans les équations (2) et (3) de la figure 5.2.

Pour terminer, nous formulons dans l'équation (4) de la figure 5.2, que le nombre de cartes CP utilisées lors du placement doit être minimum. Ce nombre est égal au nombre de colonnes non nulles de la matrice binaire. Puisque notre but est de minimiser ce nombre, nous transformons cette fonction en fonction d'optimisation minimum.

$$\sum_{j=1}^{n} X_{i,j} = 1 \qquad \forall i \qquad (1)$$

$$\sum_{i=1}^{m} Mi \quad Xi,j \quad <= 1 \qquad \forall j \qquad (2)$$

$$\sum_{i=1}^{m} Pi \quad Xi,j \quad <= 1 \qquad \forall j \qquad (3)$$

$$\min \sum_{j=1}^{n} \left(\max_{i} (Xi,j) \right) \qquad (4)$$

Figure 5.2 : représentation du problème de placement

2.2 Démonstration du caractère soluble du problème

Le problème est soluble si nous pouvons montrer que le nombre de cartes CP à utiliser a une borne supérieure finie.

Une solution possible, sachant que tout groupe peut être placé sur une carte CP, est de placer chaque groupe sur une carte CP qui lui est propre. Dans ce cas, le nombre de cartes CP "n" est égal au nombre de groupes à placer "m", nombre fini et connu. Ce nombre est donc une borne supérieure au nombre de cartes CP à utiliser. Remarquons que cette solution est de coût maximum.

Nous présentons à la figure 5.3 une matrice unité d'ordre 4. Cette matrice est une solution de coût maximum au problème de placement de 4 groupes.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 5.3 : matrice solution de coût maximum au problème de placement de 4 groupes

3 Méthode aléatoire

Nous avons premièrement songé à une méthode d'essai aléatoire comme solution au problème de placement des groupes sur un nombre minimum de cartes CP. Il nous était en effet demandé de trouver une méthode simple et rapide pour résoudre ce problème rendu complexe par la variation, en fonction des groupes installés sur une carte CP, des consommations de mémoire et de CPU nécessaires à l'installation d'un groupe subséquent. Une méthode aléatoire très simple, répétée plusieurs fois permet d'obtenir plusieurs résultats pour une même liste de groupes et de n'en garder que le plus performant.

Nous précisons la méthode aléatoire que nous avons élaborée, nous présentons alors les tests effectués puis une comparaison à d'autres formules. Pour terminer nous montrons les lacunes de cette méthode vis-à-vis du problème posé.

3.1 Présentation de la méthode aléatoire

La méthode aléatoire se déroule en deux phases : la première consiste à trier aléatoirement la liste des groupes, la seconde consiste à traiter les groupes un à un de la manière suivante : chaque groupe est placé sur la première carte CP ayant suffisamment de mémoire et de CPU disponibles pour le recevoir. Lorsqu'un groupe ne peut être accepté sur aucune carte CP, il est permis d'en prendre une

nouvelle.

3.2 Tests effectués

Nous avons réalisé un programme de simulation afin de tester la méthode aléatoire.

Pour choisir le groupe à traiter, ce programme se base sur une formule de génération de nombres aléatoires. Cette formule appelée "SEEK" et que nous présentons à la figure 5.4, fait partie du logiciel mis à notre disposition. Le résultat de cette formule est un nombre réel uniformément distribué entre 0 inclus et 1 exclu. Notons qu'il est nécessaire d'initialiser cette formule par une valeur différente chaque fois qu'une liste de nombres aléatoires différente est désirée.

Le résultat de la formule "SEEK" étant un nombre compris entre 0 et 1, nous le transformons en un nombre compris entre 0 et N-1, par la formule "RANDOM" montrée à la figure 5.5. Nous numérotons alors successivement à partir de 0, les groupes de la liste de données et les traitons dans l'ordre défini par la formule aléatoire "RANDOM", dans laquelle nous affectons à "N" une valeur égale au nombre de groupes.

```
SEEK =  ( 69069 * SEEK + 1 ) ( modulo 2 ** 32 )
```

Figure 5.4 : la fonction aléatoire "SEEK"

```
RANDOM = trunc ( SEEK * N )
```

Figure 5.5 : la fonction aléatoire "RANDOM"

Nous avons effectué un test dit "vicieux". Ce test consiste à imaginer une liste de groupes saturant tant au niveau de la mémoire que du CPU, un nombre connu de cartes CP, puis d'y appliquer la méthode aléatoire. Remarquons que pour ce test, nous associons aux groupes des consommations de mémoire et CPU absolument fictives.

Nous avons itéré ce test cinquante fois sur une liste de soixante groupes puis sur une liste de cent cinquante groupes. Les données et les résultats de ces tests sont présentés en annexe VII. Le nombre de cartes CP déterminé par cette méthode est variable et aucun critère ne

nous permet de juger les résultats obtenus.

3.3 Comparaison à d'autres formules

Afin de pouvoir conclure sur la méthode aléatoire, nous avons besoin de la comparer à d'autres méthodes. Nous avons modifié la première phase de la méthode aléatoire en triant la liste des groupes selon différentes formules déterministes.

Les formules déterministes choisies sont les suivantes :

- l'ordre croissant selon la consommation mémoire
- l'ordre croissant selon la consommation CPU
- l'ordre croissant selon la différence mémoire-CPU en valeur absolue
- l'ordre décroissant selon la différence mémoire-CPU en valeur absolue
- l'ordre alternatif favorisant la mémoire puis le CPU, calculé selon la différence mémoire-CPU en valeur absolue

Dans le but de comparer ces formules, nous avons effectué le test dit "vicieux" sur une dizaine de listes de groupes, en les triant selon la formule aléatoire puis selon les formules déterministes. Nous montrons en annexe VII les résultats obtenus, selon les formules déterministes, pour les listes de soixante et cent cinquante groupes. Nous rappelons que ces listes sont la base du test de la méthode aléatoire, dont les résultats sont également présentés en annexe VII.

Nous constatons, par la comparaison des résultats, que le tri aléatoire, donne des résultats de moins en moins satisfaisants lorsque le nombre de groupes grandit. Les deux formules se basant sur l'ordre croissant des quantités mémoire ou CPU consommées, n'offrent dans nos exemples que rarement de bons résultats. Par contre, choisir entre les formules se basant sur une différence en valeur absolue n'est pas possible : ces trois formules fournissent régulièrement les résultats les plus performants.

Nous rappelons au lecteur que ce type de placement et ces trois dernières formules sont utilisés lors de l'étape de placement des noyaux de type "NON FEP", dans la méthode de fusion.

3.4 Lacunes de la méthode aléatoire

Dans la méthode aléatoire, nous relevons deux lacunes importantes. Premièrement, nous ne tenons jamais compte de l'intérêt à placer sur une même carte CP des groupes composés de processus homonymes ou voisins. Seul le hasard peut y contribuer. Deuxièmement, nous ne profitons en aucune manière de la possibilité de découper les groupes de type "FEP". Il semble pourtant que ceci pourrait être utile afin de saturer au mieux les cartes CP.

4 Programmation linéaire

Nous avons deuxièmement envisagé une méthode de programmation linéaire. Ceci se justifie car dans notre formalisation présentée au point 2.1, nous sommes face à des contraintes à respecter et une fonction à optimiser.

Nous avons dû très vite abandonner cette idée pour les trois raisons majeures suivantes :

la contrainte mémoire évolue de manière linéaire par parties et non pas de manière linéaire pure. Ceci a été exposé au point 2.3.3 du chapitre III.

Nous sommes face à différents types de contraintes que nous ne pouvons uniformiser : la mémoire, le CPU et l'intérêt à placer sur une même carte CP des groupes composés de processus homonymes ou voisins. Ceci n'est pas prévu dans la programmation linéaire.

Nos variables sont binaires, en effet un groupe est placé sur une carte CP ou non. Les logiciels de programmation linéaire travaillent sur des variables réelles. Ces logiciels permettent de traiter des variables entières, voire même binaires; mais dans ce cas, le traitement d'un nombre élevé de variables est très long.

Remarquons malgré tout que la programmation linéaire nous a fait penser à notre cours de "Théorie des graphes" (cfr [6]). Ceci nous a par après donné l'idée de construire un graphe pour exprimer les incompatibilités.

5 Méthode de découpe

La méthode de découpe fut la dernière étape envisagée avant la méthode de fusion. Commencer par exposer la méthode de fusion, bien qu'elle soit l'aboutissement de la méthode de découpe, nous a permis de la présenter en son tout, sans faire de redites.

Vu que les données et les résultats de la méthode de découpe sont identiques à ceux de la méthode de fusion, nous ne les présentons pas. Remarquons que seul le rapport CPU-MEMOIRE est une amélioration apportée par la méthode de fusion. Nous exposons la méthode de découpe proprement dite puis nous montrons le pourquoi de l'abandon de cette méthode.

5.1 Description de la méthode de découpe

La méthode de découpe se réalise en six étapes. Nous veillons premièrement à la cohérence de nos données. Ensuite nous construisons un graphe, représentation de la liste des groupes, puis nous le simplifions. La quatrième étape consiste à découper ce graphe en sous-graphes correspondants à des consommations de mémoire et de CPU inférieures aux quantités disponibles sur une carte CP. Nous plaçons alors les groupes compris dans ces sous-graphes sur des cartes CP et évaluons la configuration obtenue. Nous présentons pour terminer une possibilité de rétroaction puis un algorithme, représentation concrète de la méthode.

5.1.1 Analyse de cohérence

L'analyse de cohérence réalisée dans la méthode de découpe est strictement semblable à celle de la méthode de fusion. Elle a été présentée au point 3.1 du chapitre IV.

5.1.2 Construction du graphe

Un noeud du graphe correspond à un groupe de la liste des groupes et une arête représente la perte de ressources mémoire ou CPU subie si les deux groupes concordant aux noeuds qui lui sont adjacents ne sont pas placés sur une même carte CP.

De plus, nous créons des arêtes de poids maximum entre deux noeuds correspondants à des groupes que l'utilisateur a localisés sur une même carte CP. Relier deux noeuds par une arête maximum empêche qu'ils soient séparés lors d'une étape suivante.

En fin de cette étape, de même que dans la méthode de fusion, nous rendons les arêtes comparables. Ceci a été montré au point 3.2.3 du chapitre IV. Remarquons néanmoins que, commme nous l'avons signalé auparavant, dans la méthode de découpe il n'existe pas de rapport du type du rapport CPU-MEMOIRE de la méthode de fusion. Le poids normalisé des arêtes de la méthode de découpe est égal au poids relatif.

5.1.3 Simplification du graphe

Le but de cette étape est de simplifier le graphe en éliminant toutes les arêtes qui représentent des incompatibilités logiques ou physiques. Cette étape est importante car elle permet de supprimer des arêtes représentant des pertes impossibles à éviter.

Les incompatibilités logiques sont les contraintes d'incompatibilité entre deux groupes, reprises dans la bibliothèque des contraintes. Lorsque deux groupes sont incompatibles, nous supprimons toute arête reliant les noeuds dont ils font partie. Mais, vu que deux groupes incompatibles peuvent avoir un même voisin et donc être indirectement reliés, il est nécessaire au cours des étapes suivantes de veiller à ne pas placer ces groupes sur une même carte CP.

Les incompatibilités physiques sont définies par les notions de noeuds et d'arêtes non respectables présentées ci-après.

5.1.3.1 Noeuds non respectables

Les noeuds non respectables sont ceux représentant un groupe de type "FEP" qui consomme plus de mémoire ou de CPU que les quantités de ressources offertes par une carte CP vierge. Il est donc physiquement impossible de placer ce groupe sur une seule carte CP. Comme nous l'avons expliqué au point 3 du chapitre II, lorsqu'un groupe de type "FEP" n'est pas installé dans sa totalité sur une seule carte CP, il n'est plus considéré comme voisin des groupes avec lesquels il échange de l'information. Nous éliminons alors toute arête ayant ce groupe comme noeud adjacent puis éclatons ce noeud en plusieurs noeuds isolés.

5.1.3.2 Arêtes non respectables

Les arêtes non respectables sont des arêtes telles que l'addition des besoins mémoire ou CPU des deux groupes représentés par leurs noeuds adjacents, dépasse les capacités d'une seule carte CP vierge. Vu qu'il sera toujours impossible de placer ces deux groupes sur une même carte CP, nous supprimons l'arête.

Remarquons que la notion d'arête non respectable est utilisée dans la méthode de fusion sous le nom d'"arête impossible absolue", définie au point 3.3.3 du chapitre IV.

Nous présentons en un premier point le calcul proposé pour détecter les arêtes non respectables au niveau de la mémoire. En un second point, nous montrons deux possibilités de détection d'arêtes non respectables au niveau du CPU.

a) Le calcul d'arêtes non respectables au niveau de la mémoire

Le calcul d'arêtes non respectables au niveau de la mémoire est très simple : une arête mémoire est non respectable à la seule condition que les deux groupes représentés par ses noeuds adjacents consomment ensemble plus de mémoire que la mémoire disponible sur une carte CP vierge. Cette formule est donc :

$$M\_1 + M\_2 > M\_MAX$$

- où M_1 et M_2 sont les consommations de mémoire correspondant aux noeuds reliés par l'arête traitée;
- et où M_MAX est la quantité maximum de mémoire disponible sur une carte CP vierge.

b) Les calculs d'arêtes non respectables au niveau du CPU

La première formule de calcul d'arêtes non respectables au niveau du CPU est très simple : une arête CPU est non respectable à la seule condition que la somme des consommations de CPU intrinsèque des deux groupes représentés par ses noeuds adjacents soit supérieure à la quantité de CPU disponible sur une carte CP vierge. La première formule de calcul est donc :

$$CPUI\_1 + CPUI\_2 > CPU\_MAX$$

- où CPUI_1 et CPUI_2 sont les consommations de CPU intrinsèque correspondant aux noeuds reliés par l'arête traitée;

- et où CPU_MAX est la quantité maximum de CPU disponible sur une carte CP vierge.

La seconde formule de calcul d'arêtes CPU non respectables est plus précise mais se calcule en un temps plus long que la première. Dans le calcul du CPU nécessaire aux deux groupes représentés par les noyaux adjacents à l'arête, nous désirons prendre en compte les échanges d'informations de chacun avec d'autres groupes, qui ne seront certainement pas placés sur une même carte CP. En effet, il est possible que la charge CPU calculée en considérant tous les voisins de ces deux groupes installés sur la même carte CP que ceux-ci, ne dépasse pas la quantité de CPU disponible sur une carte CP vierge alors qu'en réalité, la charge est devenue trop importante suite à des pertes inévitables de voisinages. Ceci revient à calculer la charge de CPU intrinsèque des deux groupes représentés par les noeuds et d'y ajouter la charge de CPU extrinsèque qui provient de voisinages perdus, c'est-à-dire d'arêtes CPU non respectables déjà supprimées. La seconde formule est :

$$CPUI\_1 + CPUI\_2 + \sum_{i=2}^{n} CPUE(N1, Ni) + \sum_{\substack{i=1 \\ i<>2}}^{n} CPUE(N2, Ni) > CPU\_MAX$$

- où CPUI_1 et CPUI_2 sont les consommations de CPU intrinsèque correspondant aux noeuds reliés par l'arête traitée;

- où "n" est égal au nombre de noeuds;

- où CPUE (Nh, Ni) est le CPU extrinsèque impossible à éviter, correspondant aux échanges d'informations des processus représentés par les noeuds Nh et Ni;

- et où CPU_MAX est la quantité maximum de CPU disponible sur une carte CP vierge.

Remarquons qu'il est possible que ce calcul réalisé pour une arête n'entraine pas le retrait de celle-ci, alors que si elle avait été traitée suite à l'élimination d'autres arêtes non respectables, elle aurait elle aussi été qualifiée de non respectable et supprimée. Ceci nous force à réaliser un examen complet des arêtes CPU du graphe aussi longtemps que lors de cet examen, au moins l'une d'entre elles a été qualifiée de non respectable et éliminée.

5.1.4 Découpe du graphe

Le but de l'étape de découpe du graphe est de décomposer le graphe en sous-graphes tels que l'ensemble des groupes qu'ils représentent peut être placé sur une seule carte CP vierge.

Chaque sous-graphe de taille inacceptable est décomposé récursivement de la manière suivante : nous recherchons dans le sous-graphe les arêtes de poids minimum et les supprimons toutes. De plus, nous devons veiller à ce que des groupes incompatibles ne fassent pas partie d'un même sous-graphe.

Nous expliquerons lors de la présentation des raisons de l'abandon de la méthode de découpe que la décomposition du graphe n'est pas en réalité aussi simple qu'il n'apparait ici.

5.1.5 Placement des groupes

Pour cette étape de placement des groupes, nous définissons en tant qu'entité l'ensemble des groupes appartenant à un même sous-graphe.

Nous plaçons sur les cartes CP les entités définies lors de l'étape précédente. Dans ce but nous utilisons les deux phases de la méthode aléatoire, mais en pratiquant uniquement le tri des entités selon les formules déterministes se basant sur une différence en valeur absolue. Nous rappelons que cette méthode a été définie au point 3.1 de ce chapitre et que les formules de tri ont été présentées au point 3.3 de ce chapitre.

5.1.6 Evaluation

Tout comme dans la méthode de fusion, nous évaluons les taux d'occupation mémoire et CPU de chaque carte CP.

De plus, lors des étapes précédentes nous avons comptabilisé les pertes de ressources mémoire et CPU survenues chaque fois qu'une arête était supprimée. Nous connaissons ainsi avec précision le coût des pertes de ressources d'un placement et pouvons comparer ce coût pour différents placements correspondants à une même liste de groupes.

5.1.7 Rétroaction

Cette méthode de découpe étant heuristique, nous avons prévu une possibilité de rétroaction.

La rétroaction se situe au niveau de la décomposition du graphe en sous-graphes. Lorsque nous désirons modifier les résultats obtenus, nous recommençons le traitement à l'étape de découpe du graphe, en réduisant la taille maximum des sous-graphes. Ainsi, nous espérons que l'étape de placement des groupes pourra plus facilement saturer les cartes CP.

5.1.8 Algorithme

Pour terminer nous présentons à la figure 5.6 un algorithme représentant la méthode de découpe que nous venons de décrire. A chaque donnée, traitement ou résultat nous associons le numéro du paragraphe où il a été défini.

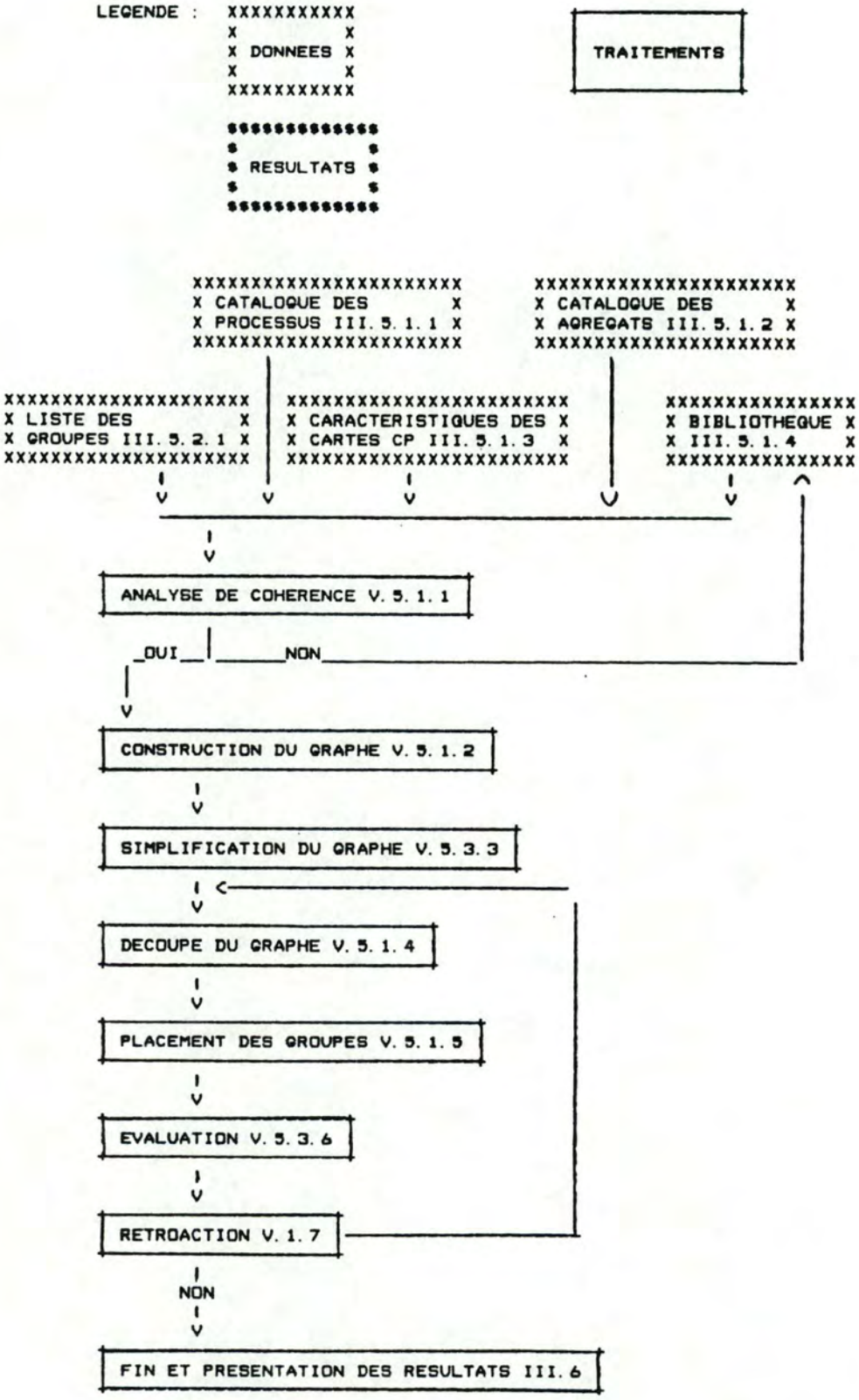


Figure 5.6 : algorithme de la méthode de découpe

5.2 Abandon de la méthode de découpe

La méthode de découpe a dû être abandonnée pour trois raisons. Les deux premières raisons, relativement peu importantes, proviennent de l'étape de simplification du graphe. Il s'agit de la difficulté à déterminer des critères pour réaliser l'éclatement d'un noeud représentant un groupe de type "FEP" et du fait que le nombre d'arêtes non respectables est peu important. La troisième raison est fondamentale : l'étape de découpe du graphe est quasi impossible à mettre en oeuvre. Nous montrons d'abord pourquoi cette étape, telle qu'elle a été définie, est irréalisable. Ensuite nous proposons deux modifications à cette étape : la première consiste à travailler sur deux graphes, la seconde à couper un ensemble d'arêtes et nous constatons qu'elles n'apportent pas de solution valable.

5.2.1 Eclatement des noeuds

Dans l'étape de simplification du graphe, il est prévu d'éclater un noeud représentant un groupe de type "FEP" lorsqu'il consomme plus de ressources que les quantités disponibles sur une carte CP vierge. Nous n'avons jusqu'à présent aucun critère permettant d'orienter la découpe de ces groupes en un certain nombre de parts et de connaitre l'importance à donner à chacune de ces parts.

C'est ainsi que nous avons décidé que les groupes de type "FEP" seraient traités comme les autres groupes jusqu'au moment où ils auraient perdu tous leurs voisins. Ceci nous permet de conserver dans la mesure du possible les voisinages existants. Les groupes de type "FEP" isolés doivent servir lors d'une étape ultérieure à saturer les cartes CP. Il nous restait à trouver une technique : nous avons élaboré celle présentée dans la méthode de fusion au point 3.5 du chapitre IV : le placement des noyaux de type "FEP".

5.2.2 Nombre d'arêtes non respectables

Nous pensions que le nombre d'arêtes non respectables serait important et que le graphe serait considérablement réduit lors de l'étape de simplification du graphe. Nous avons vite constaté, sur des exemples de graphes, qu'il y avait très peu d'arêtes non respectables au départ.

Nous rappelons que cette étape de simplification doit calculer pour chaque arête du graphe, si les groupes compris dans les noeuds qu'elle relie peuvent, au niveau de la mémoire et du CPU, être placés

sur une même carte CP. Celle-ci nous parut alors trop longue pour les maigres résultats qu'elle offrait et fut éliminée.

5.2.3 Impossibilité de la découpe du graphe

Décomposer le graphe en sous-graphes par l'élimination d'arêtes de poids minimum est impossible. En effet, une décomposition de ce genre se réalise par le retrait d'isthmes. Le graphe des groupes comporte de nombreux cycles et très peu d'isthmes qui de plus, ne sont pas toujours de poids minimum.

Nous présentons un exemple où le graphe comprend 22 groupes ayant chacun en moyenne 2 voisins. N'oublions pas qu'en réalité nous devons travailler sur des graphes d'environ 800 groupes, voisins chacun d'au plus 15 groupes.

Notons que dans cet exemple nous ne prenons pas la peine de quantifier les arêtes, en effet ces valeurs sont inutiles pour montrer l'impossibilité de la découpe du graphe.

L'exemple de graphe, présenté à la figure 5.7, comprend les groupes suivants :

- 6 groupes de 1 DRHD, chacun voisin d'un groupe de 1 HDLC.
- 6 groupes de 1 HDLC, chacun voisin d'un groupe de 1 DRHD et d'un groupe de 1 MDPER.
- 6 groupes de 1 MDPER, chacun voisin d'un groupe de 1 HDLC et d'un groupe de 1 SR.
- 3 groupes de 1 SR, chacun respectivement voisin de 1, 2 et 3 groupes de 1 MDPER et chacun voisin d'un groupe de 1 DS.
- 1 groupe de 1 DS, voisin de trois groupes de 1 SR.

Rappelons que deux groupes de processus homonymes sont reliés par une arête mémoire et que deux groupes de processus voisins sont reliés par une arête CPU. Ceci nous permet de déterminer les arêtes du graphe de l'exemple, visualisé à la figure 5.7.

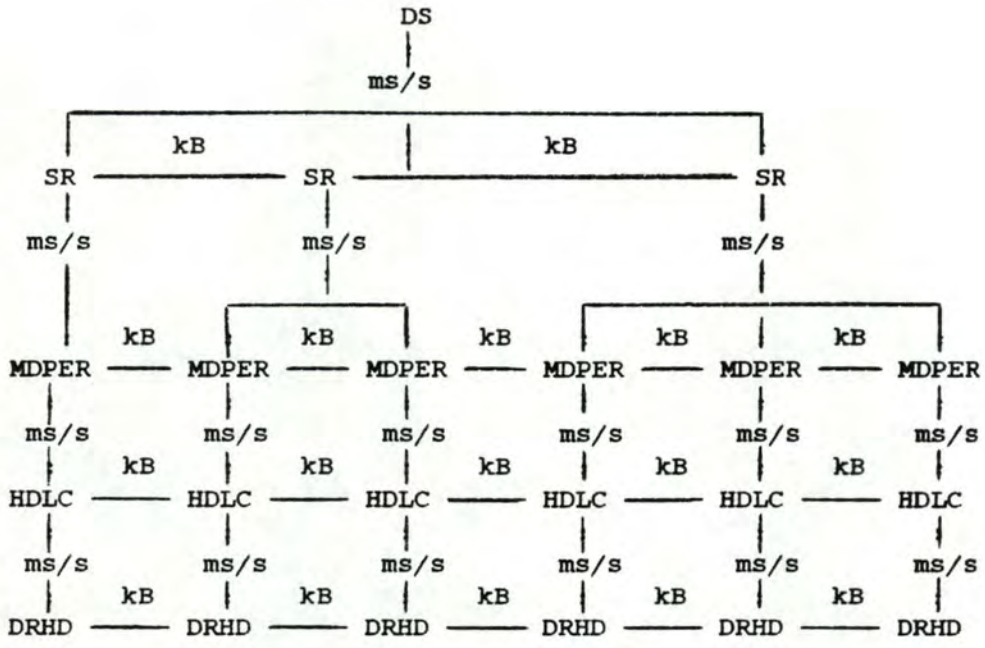


Figure 5.7 : exemple de graphe de 22 groupes

5.2.4 Première modification

La première modification apportée à l'étape de découpe du graphe consiste à décomposer parallèlement le graphe d'origine en deux graphes plus simples qui superposés, le reconstituent. Ces deux graphes sont constitués des mêmes noeuds que le graphe d'origine mais en diffèrent au point de vue des arêtes : le premier graphe ne comprend que les arêtes mémoire et le second uniquement les arêtes CPU. Ces deux graphes sont respectivement appelés graphe "mémoire" et graphe "CPU".

Les graphes "mémoire" et "CPU" issus du graphe d'origine présenté à la figure 5.7 sont respectivement visualisés aux figures 5.8 et 5.9.

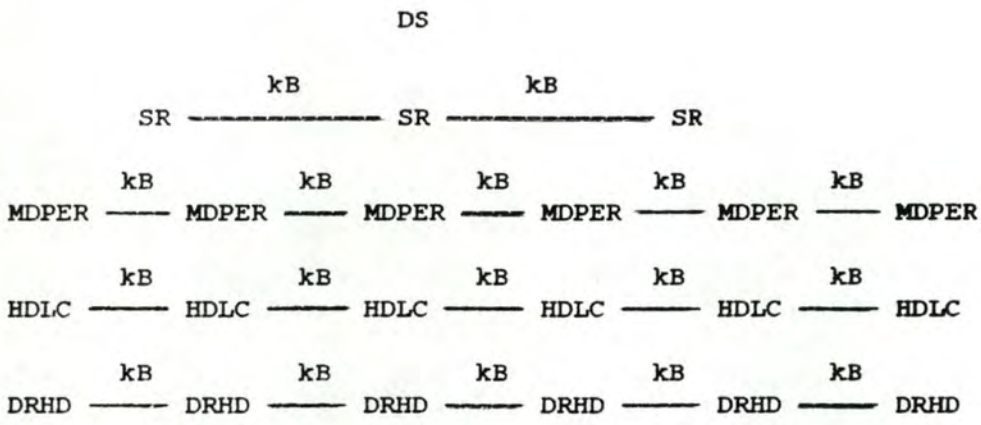


Figure 5.8 : exemple de graphe "mémoire"

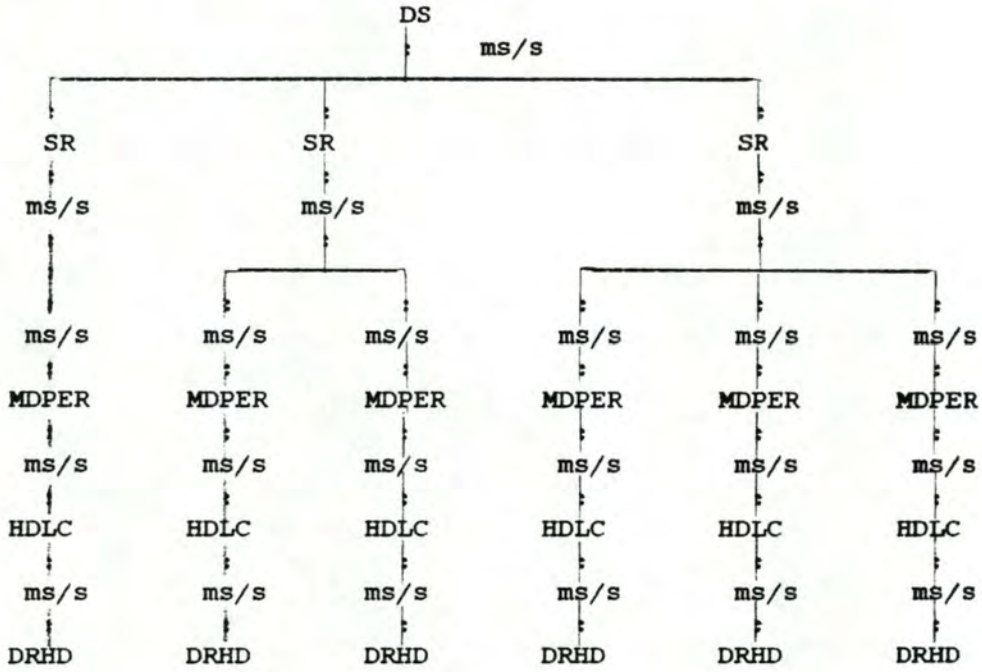


Figure 5.9 : exemple de graphe "CPU"

Décomposer chacun de ces deux graphes en sous-graphes est très simple alors que la découpe du graphe d'origine en sous-graphes est complexe vu le nombre réduit d'isthmes. Dans le cas du graphe "mémoire" il n'y a rien à faire et dans celui du graphe "CPU" il suffit d'éliminer n'importe quelle arête.

Le problème qui est de minimiser le coût de découpe du graphe en sous-graphes n'a absolument pas été réduit de cette manière. En effet toute décision prise au niveau du graphe "mémoire" doit être reportée dans le graphe "CPU" et inversément, afin de décomposer ces deux graphes en sous-graphes identiques. Minimiser la découpe dans l'un des deux graphes ne signifie pas pour autant minimiser la découpe dans l'autre : la découpe globale du graphe d'origine réalisée ainsi n'est donc pas minimum.

5.2.5 Seconde modification

La seconde modification apportée à l'étape de découpe du graphe consiste non plus à éliminer les arêtes de poids minimum mais à couper un chemin d'arêtes dont la somme des poids est minimum et qui décompose le graphe en deux sous-graphes.

La recherche d'un tel chemin est difficile et heuristique car il n'existe comme méthodes déterministes que des méthodes cherchant un isthme dans un graphe. Il est possible de trouver un exemple d'algorithme de recherche d'un isthme dans un graphe en [6].

Rechercher un chemin de deux arêtes décomposant le graphe consiste à couper aléatoirement une première arête puis à chercher un isthme. Ceci s'exprime de la manière générale suivante : chercher un chemin de "N" arêtes décomposant le graphe consiste à découper aléatoirement "N-1" arêtes puis à chercher un isthme.

Pour pouvoir affirmer que le chemin trouvé est de poids minimum alors que la recherche est principalement heuristique, il faut le comparer à tout autre chemin possible. Ceci implique que pour trouver un chemin de poids minimum décomposant le graphe en deux sous-graphes, il est nécessaire de chercher exhaustivement les chemins de "1" à "N-1" arêtes décomposant le graphe en deux sous-graphes.

Un algorithme appliquant les procédés de cette seconde modification est difficile à mettre en oeuvre et ne répond pas aux critères de simplicité et de rapidité désirés pour notre méthode. C'est alors que nous avons pensé à la méthode duale de la méthode de découpe : la méthode de fusion.

6 Conclusion

Pour conclure ce chapitre nous rappelons brièvement les apports de chaque méthode envisagée à la méthode de fusion. Nous terminons en citant les traitements spécifiques à la méthode de fusion.

La première méthode envisagée qui est la méthode aléatoire, nous a permis d'établir trois formules de tri et une méthode de placement des groupes sur des cartes CP dont les résultats sont performants. Dans la méthode de fusion, cette méthode de placement et ces formules de tri sont utilisées lors de l'étape de placement des noyaux de type "NON FEP", présentée au point 3.4 du chapitre IV.

La méthode linéaire a été très vite rejetée. C'est elle cependant qui nous a donné l'idée de construire un graphe représentant les groupes et leurs contraintes. L'étape de construction du graphe dans la méthode de fusion est exposée au point 3.2 du chapitre IV.

La méthode de découpe est la méthode duale à la méthode de fusion. C'est lors de la définition de cette méthode que nous avons constaté que les groupes de type "FEP" n'ayant plus de voisin devaient servir à saturer les cartes CP et que nous avons ébauché l'étape d'évaluation du placement. Dans la méthode de fusion, l'étape de placement des noyaux de type "FEP" est présentée au point 3.5 du chapitre IV et l'étape d'évaluation, au point 3.6 de ce même chapitre.

Les traitements spécifiques à la méthode de fusion sont les suivants : l'élaboration d'une méthode de placement des groupes de type "FEP" et l'orientation du placement grâce au rapport CPU-MEMOIRE. Ces deux étapes sont respectivement décrites aux points 3.5 et 3.2.2 du chapitre IV.

COMPARAISONS BIBLIOGRAPHIQUES

CHAPITRE VI

1 Introduction

L'objectif de ce sixième chapitre est de comparer avec notre méthode de fusion quelques modèles développés par des auteurs dans le cadre d'applications réalisées sur des systèmes à multiprocesseurs et de décrire un algorithme alternatif à la phase de placement des noyaux de type "NON FEP" de notre méthode.

Nous étudions plus particulièrement le modèle proposé par Stone et Indurkhya dans [11] et celui proposé par Huang dans [10]. Ces modèles ont pour objectif de minimiser l'usage des ressources qu'un système à multiprocesseurs peut offrir. Après avoir décrit chacun de ces modèles, nous analysons dans quelle mesure ils sont applicables pour résoudre notre problème et si c'est le cas, dans quelle mesure nous pouvons en tirer des enseignements pour notre propre méthode de résolution.

Nous étudions également un algorithme de "bin packing" ou "remplissage de coffres" proposé par Chung dans [5] qui présente des points de ressemblance non négligeables avec la phase de placement des noyaux de type "NON FEP" de la méthode de fusion. Après avoir présenté cette méthode, nous la critiquons en en dégageant les aspects convergents et divergents par rapport à notre méthode.

2 Modèle de Stone et Indurkhya

2.1 Introduction

Stone et Indurkhya proposent un modèle d'assignation d'un ensemble de tâches issues de la décomposition d'un programme aux processeurs d'une architecture à multiprocesseurs. Les auteurs définissent une tâche comme une unité d'exécution nécessitant la réquisition de ressources mémoire et CPU. Les tâches peuvent s'échanger des informations via une voie de communication reliant les processeurs. L'objectif de leur modèle est d'arriver à une assignation optimale des tâches aux processeurs. L'optimalité d'une assignation se mesure par un temps total d'exécution des tâches qui doit être le plus petit possible. Toutefois, dans une architecture à N processeurs reliés par une voie de communication, un dilemme se présente : d'un côté il est intéressant de distribuer le plus possible les tâches entre les processeurs de façon à avoir un parallélisme maximum entre eux et d'un autre côté, il est tout aussi intéressant d'éviter un ralentissement global des tâches lié à l'utilisation d'une voie de communication plus ou moins lente; pour limiter ce ralentissement, la tendance est alors plutôt à concentrer les tâches sur le même processeur.

Stone et Indurkhya se basent sur une formalisation en graphe des tâches et de leurs échanges : c'est la notion de "graphe aléatoire". Les conclusions auxquelles arrivent les auteurs sont assez surprenantes dans la mesure où elles sont extrêmes : le temps d'exécution total des tâches est minimal soit par un partage égal des

tâches entre les processeurs, soit par une concentration de toutes les tâches sur un seul processeur. Nous montrons que cela ne choque pas vraiment notre intuition, mais que nous devons limiter l'étendue de ces conclusions dans le cadre d'un système réel en général et de SOPHO-NET en particulier. Ces limites sont d'ailleurs reconnues par les auteurs qui souhaitent un approfondissement de leur modèle.

2.2 Hypothèses et objectif du modèle de Stone et Indurkhya

Les hypothèses de départ du modèle de Stone et Indurkhya sont :

- hypothèse 1 : le programme à distribuer entre les processeurs est décomposé en un ensemble de tâches. Ces tâches constituent les noeuds d'un graphe. Sur chacun de ces noeuds est repris le temps d'exécution de la tâche qu'il représente. Les noeuds du graphe sont reliés par des arcs qui représentent les échanges entre ces tâches. A chaque arc est associé un poids qui indique le nombre de bits échangés entre les tâches adjacentes à cet arc. Ce graphe ne représente qu'un moyen de schématiser les tâches et les échanges entre celles-ci;

- hypothèse 2 : les poids des arcs sont supposés être indépendants les uns par rapport aux autres. Le poids de ces arcs est une variable aléatoire ayant une moyenne et une variance finies. Une probabilité d'existence est également associée à chaque arc;

- hypothèse 3 : le modèle suppose aussi une indépendance entre les temps d'exécution des tâches. Le temps d'exécution d'une tâche est également une variable aléatoire de moyenne et de variance finies. Du fait que les temps d'exécution des tâches et les poids des arcs sont des variables aléatoires, le graphe des tâches est appelé "graphe aléatoire";

- hypothèse 4 : le modèle suppose N (N>=2) processeurs homogènes, c'est à dire de même performance. Les auteurs ont étendu leur modèle au cas de processeurs hétérogènes mais nous ne considérons pas ce cas ici;

- hypothèse 5 : la voie de communication entre les processeurs est caractérisée par une certaine vitesse exprimée en bits par seconde;

- hypothèse 6 : le temps de communication entre deux tâches est considéré comme nul si ces tâches sont assignées au même processeur. Le temps de communication d'une tâche avec une autre est déduit du volume d'informations échangé entre ces deux tâches et de la vitesse de la voie de communication entre les processeurs exécutant ces tâches.

Le calcul du temps total d'exécution d'une tâche i s'effectue de la manière suivante :

temps d'exécution de la tâche i + temps de communication de la tâche i avec les autres tâches

Si nous disposons de N tâches à distribuer entre P (P>1) processeurs, alors l'objectif est de trouver une assignation de ces N tâches aux processeurs telle que le temps d'exécution de l'ensemble des tâches soit minimum. Cette assignation est l'assignation optimale des N tâches.

2.2.1 Cas de deux processeurs homogènes

L'hypothèse supplémentaire ici est donc que nous ne disposons que de deux processeurs à performances équivalentes et de N tâches à répartir entre ces deux processeurs. La figure 6.1 montre un schéma d'assignation tout à fait général où :

- k tâches sont assignées au premier processeur et donc N-k tâches au second;
- k(N-k) communications sont possibles au maximum entre les tâches séparées.

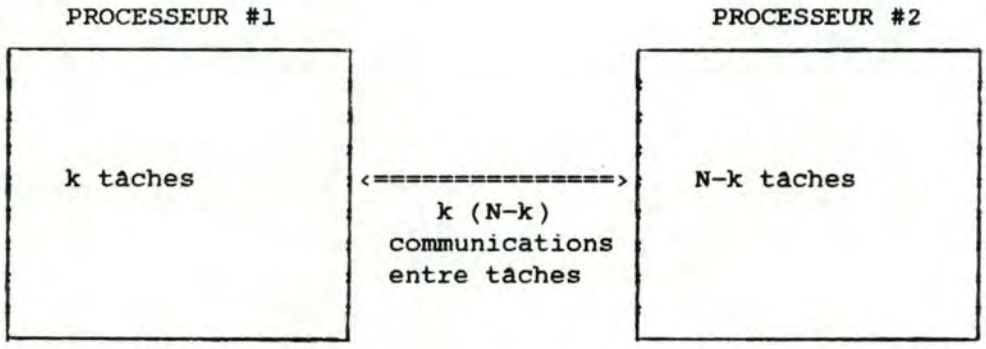


Figure 6.1 : schéma d'assignation de N tâches entre deux processeurs

Dans la suite de l'étude, nous prenons comme notation que :

- ri est le temps d'exécution de la tâche i;
- cj est le temps nécessaire au jème des k (N-k) temps de communication;
- E est le symbole de moyenne;
- Texe(k) est le temps d'exécution total des N tâches avec k tâches sur le premier processeur.

Puisque le système permet un parallélisme entre les deux processeurs, le temps total d'exécution des N tâches avec k tâches sur le premier processeur et N-k tâches sur le second, est égal au maximum des temps totaux d'exécution des tâches sur chaque processeur, auquel s'additionne le total des temps de communication entre tâches :

$$Texe(k) = MAX \left\{ \sum_{i=1}^{k} (ri), \sum_{i=k+1}^{N} (ri) \right\} + \sum_{i=1}^{k(N-k)} (ci)$$

si le calcul est effectué en moyenne, la formule devient :

$$E\,[\,Texe(k)] = E\,[\,MAX \left\{ \sum_{i=1}^{k} (ri), \sum_{i=k+1}^{N} (ri) \right\}] + E\,[\sum_{i=1}^{k(N-k)} (ci)]$$

Les auteurs démontrent dans [11] que l'approximation suivante est valable.

$$E\,[\,Texe(k)] = MAX \left\{ E\,[\sum_{i=1}^{k} (ri)], E\,[\sum_{i=k+1}^{N} (ri)] \right\} + E\,[\sum_{i=1}^{k(N-k)} (ci)]$$

sous les hypothèses 2 et 3 définies au point 2.2, la moyenne d'une somme est égale à la somme des moyennes; c'est le théorème de la limite centrale.

$$E\,[\,Texe(k)] = MAX \left\{ \sum_{i=1}^{k} E\,[(ri)], \sum_{i=k+1}^{N} E\,[(ri)] \right\} + \sum_{i=1}^{k(N-k)} E\,[\,(ci)]$$

soit $E\,[(ri)] = R$ et $E\,[(ci)] = C$

on a alors

$$E\,[\,Texe(k)\,] = MAX\,\{\,kR,\ (N-k)R\,\} + k\,(N-k)\,C$$

supposons que kR est supérieur à (N-k) R ce qui signifie que k est supérieur à N/2,

on a alors que :

$$E\,[\,Texe(k)\,] = kR + k\,(N-k)\,C \qquad (1)$$

par une mise en évidence de C

$$E\ [\ Texe(k)\ ] = C\ [\ k(N-k) + kR/C\ ] \qquad (2)$$

par transformation sous forme d'une différence de deux carrés

$$E\ [\ Texe(k)\ ] = C\ [\ 1/4\ (\ N + R/C\ )^2 - (k-(N+R/C)/2)^2\ ] \qquad (3)$$

Nous pouvons représenter la fonction (3) par le graphique de la figure 6.2 (fonction croissante) ou 6.3 (fonction décroissante) selon la valeur de R et de C :

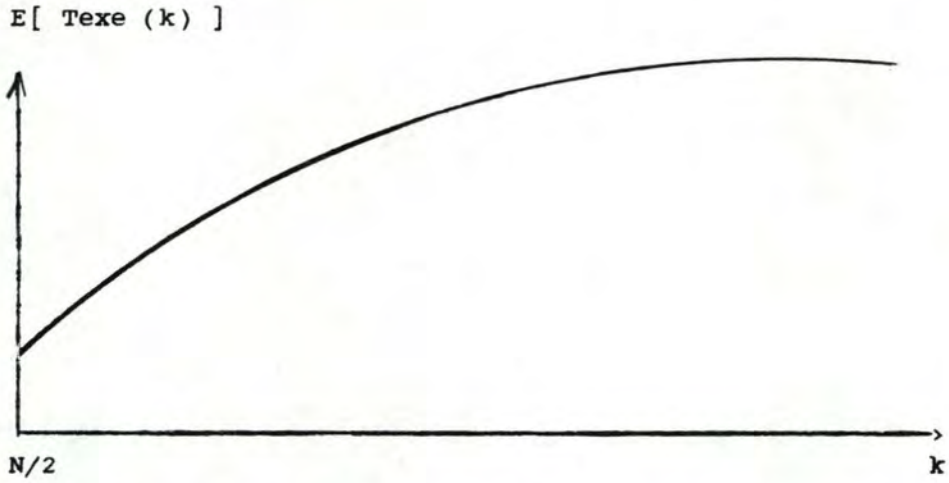


Figure 6.2 : graphique d'évolution croissant du temps moyen total d'exécution des tâches en fonction du nombre de tâches assignées au premier processeur

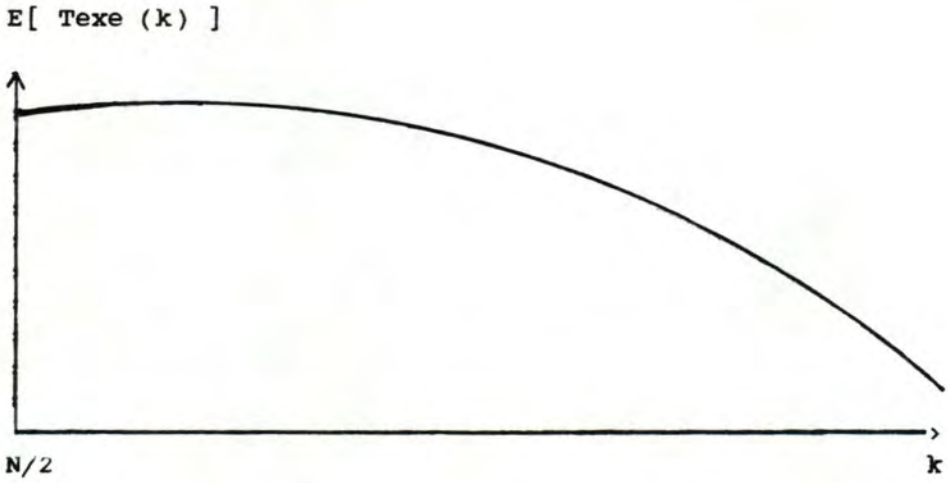


Figure 6.3 : graphique d'évolution décroissant du temps moyen total d'exécution des tâches en fonction du nombre de tâches assignées au premier processeur

La fonction (3) présente un point critique (par annulation de la dérivée première) en k=(N+R/C)/2. Ce point correspond à un maximum car la dérivée seconde de (3) est négative. Ce maximum dans l'intervalle [N/2,N] se trouve soit en k=N (figure 6.2), soit en k=N/2 (figure 6.3) selon les valeurs de R et de C.

Nous pouvons trouver le point de maximum par l'inéquation suivante :

E [Texe(N)] >= E [Texe(N/2)] (4)

qui est équivalent à :

N/2 <= R/C (5)

Cette équivalence est montrée en remplacant k par N, puis k par N/2 dans la fonction (3).

Le MAXIMUM de la fonction (3) est donc atteint pour k=N si la proposition (5) est vraie. Cela signifie que le temps total d'exécution des tâches est MINIMUM lorsque nous partageons les N tâches de manière égale entre les deux processeurs.

Au contraire, le MAXIMUM de la fonction (3) est atteint pour k=N/2 si la proposition (5) est fausse et donc que N/2 est supérieur à R/C. Dans ce cas, l'assignation optimale des tâches correspond à une concentration de celles-ci sur le même processeur.

Ces résultats sont extrêmes dans la mesure où ils impliquent une assignation complète des tâches sur un seul processeur ou un partage égal des tâches entre les deux processeurs.

Ces résultats sont intuitivement corrects : dans (5), R/C a d'autant plus de chance d'être supérieur ou égal à N/2 que C est faible. Ce qui signifie que la voie de communication est rapide. Dans ce cas, l'optimum est de répartir les tâches entre les processeurs et de bénéficier ainsi au plus du parallélisme. Par contre, si C est élevé et donc que la voie de communication est lente, il est préférable d'assigner les tâches au même processeur et de réduire ainsi les communications entre ces tâches.

2.2.2 Cas de N (N>2) processeurs homogènes

Dans le cas de N (N>2) processeurs homogènes, il est possible de généraliser le modèle et les démonstrations vues dans le point précédent.

Ainsi, par le même raisonnement, les auteurs aboutissent à la conclusion que :

si R/C >= N/2, alors le plus avantageux est d'assigner de manière égale les tâches entre les N processeurs; sinon, il est plus intéressant d'assigner les N tâches au même processeur. Les résultats sont encore extrêmes dans ce cas.

2.3 Limites de ce modèle par rapport à notre problème

Ce modèle présente des limites dans son application à un système réel en général et pour la conception d'un noeud SOPHO-NET en particulier; en effet, nous pouvons constater que :

- l'objectif pour SOPHO-NET n'est pas le même que celui qui est poursuivi dans ce modèle : le modèle de Stone et Indurkhya veut permettre la terminaison le plus tôt possible des tâches exécutées par les processeurs. Un des moyens pour atteindre cet objectif est de s'assurer que les communications ne sont pas trop importantes et ce, d'autant plus que la voie de communication est lente. Par contre, pour SOPHO-NET, l'objectif est d'avoir un minimum de processeurs et donc de cartes CP; un des moyens d'atteindre cet objectif est de maximiser l'intégration verticale et donc de limiter les communications sur le bus CP. Les objectifs pousuivis dans les deux cas sont donc fondamentalement différents, mais le moyen d'y arriver reste sensiblement le même;

- le modèle de Stone et Indurkhya suppose au départ un nombre fixe de processeurs et il propose une façon de répartir les tâches sur ces processeurs, or la nouvelle philosophie de SOPHO-NET rejette cette possibilité; elle veut avant tout minimiser le nombre de processeurs utilisés;

- nous ne sommes pas toujours certains d'avoir un graphe aléatoire et de pouvoir respecter les hypothèses de départ. Dans le cas de SOPHO-NET, cette possibilité est exclue puisque il y a dépendance entre les temps d'échange et entre les temps d'exécution des tâches, qui correspondent aux processus de SOPHO-NET (les hypothèses 2 et 3 sautent donc). De plus, la notion de probabilité de présence des arêtes n'est pas intéressante puisque nos échanges sont connus exactement. Dans le modèle de Stone et Indurkhya, cette probabilité sert au calcul des moyennes C et R;

- les auteurs ne font jamais référence à des limites de puissance CPU des processeurs. Cela est lié au fait que le modèle s'intéresse essentiellement au temps d'exécution des tâches et non à leur proportion d'utilisation du processeur comme dans notre cas;

- le volume des échanges d'information est l'unique critère de regroupement des tâches sur un même processeur. Nous pouvons comparer ce critère à notre critère d'intégration verticale. Par contre, la notion d'intégration horizontale liée à un gain mémoire n'apparait jamais dans ce modèle dans la mesure où il ne tient compte que de processeurs et jamais d'une capacité mémoire.

2.4 Enseignements de ce modèle pour notre problème

le modèle décrit peut nous apporter certains éléments intéressants pour notre problème de C.A.O. :

- la notion de graphe aléatoire présentée est intéressante pour visualiser les échanges entre tâches qui, nous le rappelons, peuvent être assimilées aux processus dans notre cas. Nous avons d'ailleurs utilisé ce mode de représentation en l'adaptant à nos besoins;

- les poids sur les arcs du graphe aléatoire représentent les volumes d'informations en bits échangés entre tâches. Dans les calculs, les auteurs ont traduit ces volumes d'informations échangées en temps moyen de communication entre tâches. Dans le cadre de SOPHO-NET, nous avons repris directement cette idée mais en associant aux poids des arêtes des consommations CPU en fonction du volume des échanges et du nombre d'activations moyen du processus par seconde.

3 Modèle de Huang

3.1 Introduction

Huang propose un modèle de "partitionnement" d'une application en tâches dans le cadre de système à multiprocesseurs en temps réel. Nous rappelons qu'un système en temps réel est, par définition, un système qui doit fournir les résultats en réponse à une demande dans un temps prédéterminé avec une très forte probabilité. La contrainte du temps d'exécution de l'application est donc fondamentale dans ce cas et l'objectif est de l'optimiser au maximum c'est à dire de rendre ce temps le plus petit possible.

Nous verrons que la modélisation proposée présente un grand intérêt par rapport à notre méthode de résolution mais que certains problèmes subsistent malgré tout quant à sa possibilité d'application pour résoudre notre problème.

3.2 Description du modèle de Huang

3.2.1 Graphe des modules

Huang modélise tout d'abord une application sous forme d'un graphe. Une application est décomposée en un ensemble de modules logiques (par découpe fonctionnelle selon la définition de Parnas [14]). Ces modules peuvent s'échanger des informations. Les modules et les échanges d'informations entre ces modules sont représentés dans un graphe orienté : les modules en sont les noeuds et les échanges sont représentés par les arcs. La sémantique de ces arcs est la suivante :

- un arc d'un module A vers un module B signifie que le module A doit être exécuté avant le module B : le module B attend une information du module A pour être activé. Dans ce cas, le module A est dit PRECEDENT du module B ou encore le module B est dit SUCCESSEUR du module A;

- il est bien sûr possible que la relation soit réciproque et qu'un module soit en même temps successeur et précédent d'un autre. Sur le graphe nous avons alors un arc de A vers B et de B vers A;

- enfin, un module A est parallèle à un module B s'ils peuvent être exécutés indépendamment. Il n'y a pas d'arc entre eux dans ce cas.

Grâce à ces quelques définitions, il est possible de construire un graphe des modules.

Il est possible de définir sur ce graphe une certaine hiérarchie en utilisant quatre règles :

- règle 1 : un module sans précédent est de niveau 1;

- règle 2 : un module avec un seul précédent a le niveau hiérarchique de ce précédent incrémenté d'une unité;

- règle 3 : un module avec plusieurs précédents a le niveau hiérarchique le plus élevé de ses précédents incrémenté d'une unité;

- règle 4 : deux modules qui se précèdent et se succèdent en même temps ont le même niveau hiérarchique : c'est le niveau le plus élevé calculé en appliquant les règles de 1 à 3 ci-dessus à chacun des modules séparément.

Par application successive de ces règles, il est possible d'obtenir un graphe hiérarchisé des modules dans le sens où chaque module est associé à un niveau de hiérarchie.

3.2.2 Modèle de "partitionnement" de Huang

Sur le graphe des modules, Huang affirme qu'il est possible de calquer un graphe des tâches; c'est ce qu'il app elle le "partitionnement". Une tâche représente la véritable matérialisation exécutable des modules avec les réquisitions de ressources mémoire et CPU que cela implique. Chaque tâche est ainsi associée à un coût CPU direct (le coût de son exécution), et à un coût CPU indirect lié à l'échange d'informations de cette tâche avec d'autres tâches, en supposant que les coûts de communication dans une tâche sont négligeables. Huang appelle ce coût indirect le "surplus". L'objectif du "partitionnement" est de réaliser une correspondance optimale entre modules et tâches; c'est à dire un "partitionnement" qui minimise ce "surplus". Cette minimisation permet une terminaison au plus tôt de l'ensemble des tâches et donc un plus grand respect de la contrainte de temps dans le cadre d'un système en temps réel.

Les tâches issues d'un "partitionnement" sont assignées à des noeuds processeurs. Ces noeuds processeurs sont caractérisés par une certaine puissance CPU et une quantité mémoire disponibles. Ces ressources sont identiques pour tous les noeuds processeurs. Les ressources des noeuds processeurs représentent une contrainte importante dans le processus de "partitionnement" puisqu'elles ne peuvent être dépassées par les tâches.

Une seconde contrainte est que le "partitionnement" du graphe des modules doit respecter les relations de précédence qui ont été décrites plus haut. De cette façon, le "partitionnement" évite qu'une tâche s'interrompe suite à l'indisponibilité d'une information qui doit lui être fournie par une tâche qui n'est pas encore activée ou qui n'a pas encore pu produire cette information. Cette situation peut provoquer des retards nocifs dans l'exécution des tâches d'une application en temps réel. L'auteur veut donc arriver à un "partitionnement" où chaque tâche est ce qu'il appelle une tâche "convenablement activée" : c'est à dire une tâche dont tous les modules disposent de leurs informations d'entrée dès l'activation de la tâche.

A partir de ces contraintes, l'auteur déduit son modèle de "partitionnement" :

soient :

- M = { M1,M2,...Mm }, l'ensemble des modules à "partitionner";
- T = { T1,T2,...,Tn }, l'ensemble des tâches résultant du "partitionnement";
- m (Tk), la consommation mémoire de la tâche k;
- Tmax, le temps CPU maximal accordé à une tâche;
- Mmax, la capacité mémoire maximale d'un noeud processeur;
- E (Tk), la consommation CPU directe de la tâche k;
- O (Tk), la consommation CPU indirecte de la tâche Tk, ou le "surplus" de la tâche k.

Un "partitionnement" doit satisfaire aux contraintes suivantes pour être valide :

- contrainte 1 : l'union des modules composant les tâches du "partitionnement" doit reprendre tous les modules sans exception;
- contrainte 2 : aucune tâche ne peut exiger l'exécution par seconde d'un nombre d'instructions plus élevé que ce que le processeur ne peut exécuter par seconde;
- contrainte 3 : m(Tk) <= Mmax pour toutes les tâches : aucune tâche ne consomme plus de mémoire que ce que le noeud processeur ne peut fournir;

- contrainte 4 : E (Tk) + O (Tk) <= Tmax: le temps CPU total de l'exécution de la tâche ne peut excéder le temps maximal d'exécution accordé à une tâche;

- contrainte 5 : deux modules qui se succèdent et se précèdent en même temps doivent être inclus dans la même tâche; cela évite qu'une tâche ne fasse attendre une autre;

- contrainte 6 : pour qu'un module Pn soit inclus dans une tâche Tk, il faut :

  - contrainte 6.1 : soit que Pn ait le niveau hiérarchique le plus bas dans Tk;

  - contrainte 6.2 : soit que Pn ait tous ses modules précédents inclus dans Tk;

  - contrainte 6.3 : soit que Pn n'ait pas tous ses modules précédents dans la même tâche, mais alors, ces modules sont inclus dans une autre tâche Ti qui contient un module précédent celui de niveau le plus bas dans Tk.

Remarquons que les contraintes 5 et 6 sont aussi appelées contraintes de précédence et que l'application de ces six contraintes pour un même graphe de modules peut aboutir à des "partitionnements" différents.

Pour chaque "partitionnement" un indicateur d'efficacité "@" est calculé; cet indicateur est obtenu par le rapport suivant :

$$@ = \frac{\sum_{k=1}^{n} (E (Tk))}{\sum_{k=1}^{n} (E (Tk) + O (Tk))}$$

Cet indicateur reflète un meilleur "partitionnement" quand il tend vers 1, c'est à dire quand le "surplus" O (Tk) au dénominateur tend vers 0. Entre plusieurs "partitionnements" d'un même graphe de modules, il ne faut retenir que celui pour lequel "@" est maximum.

3.3 Enseignements et limites du modèle de Huang

Le modèle de Huang présente des points de ressemblance non négligeables par rapport à notre problème; il nous a d'ailleurs apporté certains éléments très utiles dont nous nous sommes inspirés dans notre méthode de fusion.

Ainsi, il est clair que si nous remplaçons "module" par "processus" et "tâche" par "noyau" dans le modèle décrit dans le point 3.2, nous obtenons une formalisation en graphe très proche de celle que nous avons adoptée dans notre méthode de fusion. Les arcs du graphe des modules représentent les communications entre modules, nous en avons fait de même dans notre graphe des groupes.

De plus, l'auteur ne retient que les coûts de communication entre tâches; il considère ceux-ci comme négligeables pour les relations entre les modules réunis dans une même tâche. Cela ressemble aux puissances extrinsèques que nous comptions à la fin du placement pour les noyaux qui n'étaient pas placés sur la même carte CP. Le coût de "surplus" que Huang retient est celui lié aux communications entre tâches; nous n'avons pas fait différemment pour notre contrainte CPU : cela correspond en fait à notre intégration verticale.

Enfin l'indicateur "@" de Huang a été repris dans notre méthode, aussi bien pour le calcul de l'efficacité de notre placement sur le plan du CPU que sur le plan de la consommation mémoire puisque nous pouvons avoir un "surplus" dans les deux cas.

Notons enfin que les six contraintes du modèle de Huang ont été en grande partie reprises et développées dans notre méthode : nous avons respecté les quatre premières contraintes; quant aux deux dernières, il n'est pas évident que nous ayons pu les respecter puisque nous avions deux types d'arêtes faisant évoluer nos décisions de fusion dans un sens qui ne respectait pas nécessairement ces contraintes de précédence. En fait, notre optimalité est à deux dimensions (CPU et mémoire), tandis que celle de Huang n'a qu'une seule dimension (CPU). C'est là le point de divergence fondamental entre les deux modèles.

Remarquons que notre méthode de fusion a été imaginée avant la lecture de l'article. Nous n'en avons vraiment repris que l'indicateur d'efficacité "@".

4 Algorithme de"remplissage de coffres"de Chung

4.1 Introduction

L'algorithme présenté par Chung est un algorithme de "remplissage de coffres" (bin packing) à deux dimensions : il consiste à placer un nombre "L" de rectangles de longueur et largeur définies dans un nombre minimum de "coffres" de longueur et largeur définies également.

Cet algorithme nous intéresse dans la mesure où il implique la considération de deux dimensions pour le placement d'objets. Ce cas semble similaire à celui de notre placement de noyaux de type "NON FEP" dans la méthode de fusion; chaque noyau est en effet caractérisé par deux dimensions : sa taille CPU et mémoire. D'autre part, les "coffres" à remplir semblent correspondre à nos cartes CP que nous devons remplir avec les groupes.

Nous présentons ici les points importants de l'algorithme, et nous étudions ensuite ses limites et ses enseignements pour notre phase de placement des noyaux de type "NON FEP" dans la méthode de fusion.

Pour rappel, la phase de placement des noyaux de type "NON FEP" dans la méthode de fusion utilise trois techniques de tri des noyaux :

- l'ordre croissant selon la différence en valeur absolue entre les tailles mémoire et CPU des noyaux;
- l'ordre décroissant selon la différence en valeur absolue entre les tailles mémoire et CPU des noyaux;
- l'ordre alternatif favorisant la taille mémoire puis la taille CPU du noyau selon leur différence en valeur absolue.

Notre phase de placement place ensuite les noyaux triés sur les cartes CP selon la technique du "FIRST FIT"; cette technique consiste à placer chaque noyau sur la première carte CP déjà entamée ayant assez de place sur le plan de la mémoire et du CPU pour l'accueillir. Si aucune carte CP ne convient, une nouvelle carte CP vierge est créée.

4.2 Présentation de l'algorithme de Chung

4.2.1 Hypothèses

Soient "L" rectangles { R1,R2,...,RL } caractérisés chacun par leur longueur et leur largeur.

Les longueur et largeur des "coffres" sont supposées identiques et constantes.

Un placement de ces "L" rectangles dans P "coffres" est valide si et seulement si :

- chaque rectangle est entièrement compris dans un "coffre";
- les rectangles ne se chevauchent pas dans les "coffres";
- la largeur (longueur) de chaque rectangle est parallèle à la largeur (longueur) du "coffre" qui le contient; il n'y a donc pas de rotation possible des rectangles dans les "coffres".

4.2.2 Efficacité d'un algorithme de placement

Si P "coffres" ont été obtenus par un algorithme de placement et si OPT(L) est le nombre optimal de "coffres" pour la liste des "L" rectangles à placer, alors P doit être le plus proche possible de OPT(L).

Chung présente une mesure plus formelle d'efficacité d'un algorithme appelée "mesure au pire des cas".

Soit A(L) qui correspond au nombre de "coffres" obtenus par application d'un algorithme de placement "A" à une liste de "L" rectangles, soit OPT(L) ayant le même sens que défini ci-dessus,

soit Ra(L) = A(L) / OPT(L)

et Ra^{n} = max { Ra(L) avec OPT(L)=n } le pire des placements avec n "coffres" comme optimum

on calcule lim Ra^{n} pour n tendant vers l'infini.

Cette limite fournit le rapport entre le nombre de "coffres" issu du pire des placements fournis par un algorithme "A" appliqué à une liste de "L" rectangles et le placement optimum de ces rectangles dans n "coffres" avec n tendant vers l'infini. Cette limite est appelée "mesure au pire des cas".

Plus cette limite est proche de 1 et meilleur est l'algorithme de placement "A".

4.2.3 Algorithme du "FIRST FIT"

Chung part de l'algorithme du "FIRST FIT". Cet algorithme place chaque rectangle dans le premier "coffre" pouvant l'accueillir. Rien n'est imposé quant à l'ordre dans lequel l'algorithme doit prendre les rectangles pour les placer et le nombre de "coffres" disponibles est supposé illimité.

La mesure au pire des cas est de 1.7 pour le "FIRST FIT".

4.2.4 Algorithme du "FIRST FIT DECREASING HEIGHT"

Il est possible d'améliorer les résultats obtenus par le "FIRST FIT" en triant au préalable les rectangles selon un certain ordre. Chung choisi l'ordre décroissant selon la largeur des rectangles. Le nouvel algorithme de placement obtenu est appelé "FIRST FIT DECREASING HEIGHT" (FFDH).

Cet algorithme fonctionne comme suit :

- il trie les rectangles sur l'ordre décroissant de leur largeur;
- il ne considère qu'un seul "coffre" de longueur constante finie mais de largeur infinie;
- le "coffre" est décomposé en blocs ayant chacun la largeur du plus haut rectangle qu'il contient et la longueur du "coffre";
- l'algorithme place chaque rectangle dans le bloc le plus bas possible dans le "coffre", et le plus à gauche dans ce bloc pour autant qu'il y ait suffisamment de place dans ce bloc et que la longueur du rectangle touche entièrement la longueur du bloc. Si un rectangle ne respecte pas ces deux conditions pour les blocs existant déjà dans le "coffre", il est placé au sommet du bloc le plus élevé du "coffre", et sa largeur détermine la largeur d'un nouveau bloc. Les rectangles étant eux-mêmes triés selon leur largeur au départ, nous en déduisons que les blocs sont de largeur décroissante lorque nous montons dans le "coffre".

La mesure au pire des cas est de 1.22 pour le "FIRST FIT DECREASING HEIGHT".

4.2.5 Algorithme "hybride" de Chung

L'algorithme hybride proposé par Chung comporte deux phases :

- la première phase consiste à appliquer l'algorithme du FFDH sur les "L" rectangles à placer, il construit ainsi des blocs à partir des rectangles; comme ces blocs ont même longueur, le problème à deux dimensions est ramené à un problème à une seule dimension;
- au cours de la seconde phase, ces blocs sont pris en ordre décroissant de leur largeur et sont placés selon la technique du "FIRST FIT" dans des "coffres" de même largeur et de même longueur mais où cette fois, la largeur de chaque "coffre" est finie. La longueur de ces "coffres" reste la même que celle du "coffre" de largeur infinie décrite dans le point 4.2.4.

La mesure au pire des cas de l'algorithme hybride de Chung est comprise entre 2.02 et 2.125. Selon Chung, ce résultat est bon pour un placement d'objets à deux dimensions. Cependant, il lui semble qu'il peut encore être amélioré.

4.3 Limites et enseignements pour notre méthode

Par rapport à notre algorithme de placement des noyaux, l'algorithme hybride de Chung présente un point de divergence fondamental quant à la signification des dimensions :

Considérons le "coffre" de la figure 6.4, rempli selon l'algorithme hybride de Chung par trois rectangles appelés R1, R2, et R3.

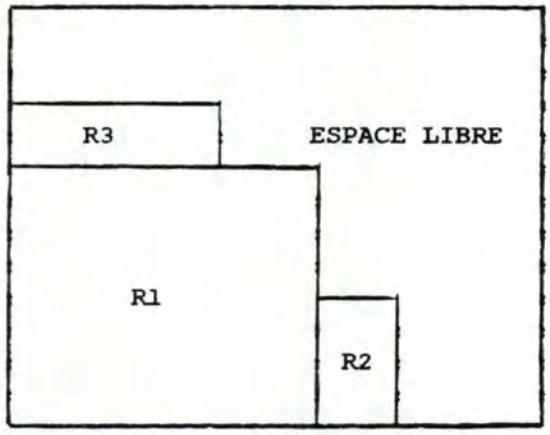


Figure 6.4 : représentation d'un "coffre" rempli selon l'algorithme hybride de Chung

La surface libre pour d'autres rectangles à placer dans ce "coffre" est la surface totale du "coffre" moins la somme des surfaces des rectangles R1, R2 et R3.

Considérons maintenant un placement de rectangles dans le cas de SYANODE. Les rectangles R1, R2 et R3 correspondent respectivement aux noyaux N1, N2 et N3. La longueur et largeur des rectangles correspondent respectivement aux consommations mémoire et CPU des noyaux. Le "coffre" représente une carte CP. Les longueur et largeur du "coffre" correspondent respectivement aux quantités mémoire et CPU conseillées sur la carte CP. Un placement possible de ces noyaux est visualisable à la figure 6.5.

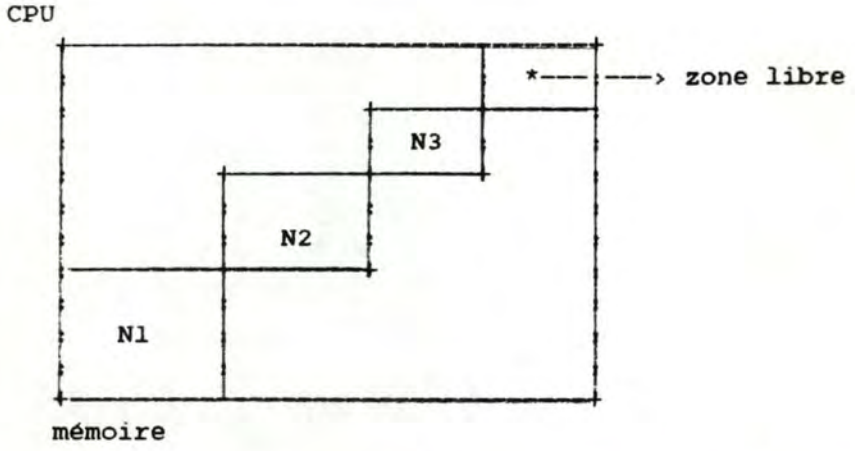


Figure 6.5 : représentation des noyaux dans une carte CP selon un schéma à deux dimensions

Dans ce cas, les longueurs et largeurs des noyaux sont additives et ne peuvent se recouvrir. La surface libre du "coffre" (et donc les ressources encore disponibles sur la carte CP) est donc dépendante des sommes des longueurs et des largeurs des rectangles.

Nous pouvons, dans certains cas, accepter des chevauchements de rectangles. Ces chevauchements nous permettent de représenter les gains CPU et mémoire induits par les intégrations verticale et horizontale. Ceci peut être visualisé à la figure 6.6. Un tel chevauchement viole cependant l'une des hypothèses de Chung.

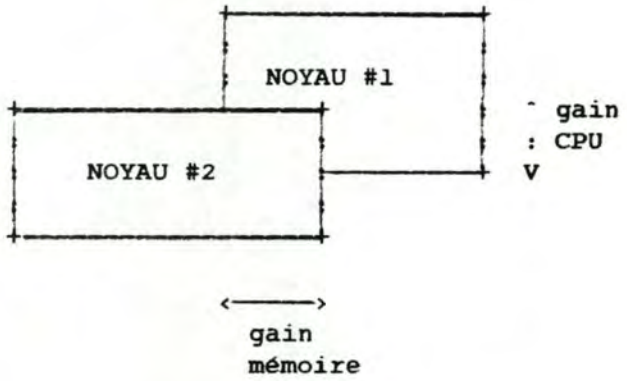


Figure 6.6 : chevauchement des noyaux lors de gains mémoire et CPU

Notre problème de placement de noyaux sur des cartes CP apparait donc comme quelque chose de fondamentalement différent d'un placement d'objets à deux dimensions sur une surface.

Les autres limites et enseignements de l'algorithme hybride de Chung sont que :

- l'algorithme de Chung trie les rectangles selon une seule des dimensions de ceux-ci; nous avons envisagé dans notre méthode de fusion des tris de noyaux sur la mémoire ou le CPU uniquement mais les résultats étaient très défavorables;

- l'une des idées de l'algorithme hybride qui nous semble intéressante est celle qui prévoit la définition d'entités de placement intermédiaires entre le rectangle et le "coffre". Remarquons que la notion de noyau que nous avons utilisée dans la méthode de fusion est un niveau intermédiaire entre le groupe et la carte CP.

Remarque : il est possible de trouver d'autres algorithmes de "remplissage de coffres" dans [1], [2], [4], [9] et [13].

DOSSIER DE PROGRAMMATION

CHAPITRE VII

1 Introduction

Le chapitre VII a pour objectif de décrire, dans un premier temps, un modèle conceptuel de structuration des informations : le modèle entité association et de réaliser son application aux informations manipulées par SYANODE. Ensuite, nous définissons la notion de type abstrait qui fut notre guide lors de la réalisation des structures de données et de la modularisation du programme SYANODE. Cette modularisation est décrite au point 6.

2 Modèle entité association

2.1 Introduction

Il existe de nombreux modèles conceptuels de structuration des informations. Dans notre méthode de fusion, nous avons retenu le modèle entité association développé dans [3]. Ce modèle présente une grande capacité de communication et un bon moyen de représenter les informations du réel perçu; d'autre part, ce modèle est celui que nous maitrisons le mieux de par notre formation.

Le modèle entité association propose de modéliser les informations du réel perçu à partir de trois concepts élémentaires : ENTITE, ASSOCIATION et ATTRIBUT. Ces concepts sont décrits dans les points suivants; nous appliquons ensuite ce modèle pour modéliser les informations manipulées par le programme SYANODE.

2.2 Concepts de base

2.2.1 Concept d'entité

L'entité est définie comme une chose concrète ou abstraite appartenant au réel perçu et à propos de laquelle nous voulons enregistrer des informations. Une entité n'existe en tant que telle que par rapport à un individu ou un groupe qui la considère comme un tout, lui confère une existence autonome et la distingue d'autres entités et de son environnement. Une entité peut posséder des attributs (cfr point 2.2.3).

Les entités sont réunies en type d'entités; celui-ci est défini par son nom, sa définition et l'ensemble des entités qui le composent. Une entité individuelle est aussi qualifiée d'occurrence d'un type d'entité.

Un type d'entité est représenté graphiquement de la manière suivante :

Figure 7.1 : exemple de type d'entité

Conventionnellement, le type d'entité est donc représenté par un rectangle comportant une cartouche où figure le nom du type d'entité. Sur la figure 7.1, nous avons illustré le concept de type d'entité par le type d'entité "carte CP" que nous estimons être une chose concrète, identifiable du réel perçu et à propos de laquelle nous désirons enregistrer des informations.

2.2.2 Concept d'association

Une association est définie par une correspondance entre deux ou plusieurs entités (non nécessairement distinctes) où chacune assume un rôle donné. Il est possible d'enregistrer des informations relatives à cette correspondance. Une association peut posséder un ou plusieurs attributs (voir point 2.2.3). De la définition, il suit que l'existence d'un association est contingente à l'existence des entités qu'elle met en correspondance.

De la même façon que pour une entité, les associations peuvent être réunies en type d'association et une association individuelle est qualifiée d'occurrence d'un type d'association.

Une propriété de base fondamentale de l'association est sa connectivité; elle résume les propriétés de multiplicité et d'existence d'une association :

- la propriété de multiplicité indique le nombre maximum d'occurrences de l'association auxquelles peut participer une même entité. Ce nombre est typiquement 1 ou N (N ayant la signification de l'infini);

- la propriété d'existence indique le nombre minimum d'occurrences de l'association auxquelles doit participer une entité; ce nombre est 0 s'il n'est jamais imposé à une entité de participer à une telle association, il est égal à 1 dans le cas contraire.

La connectivité permet de synthétiser ces deux propriétés par deux valeurs entre chaque type d'association et chaque type d'entité.

Un type d'association entre deux types d'entité et leurs connectivités sont représentés graphiquement de la manière suivante :

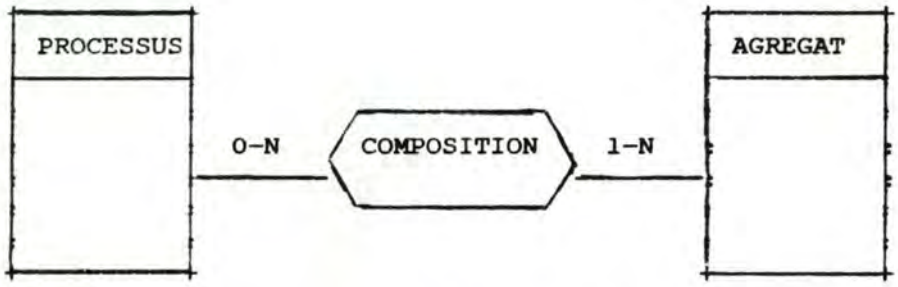


Figure 7.2 : exemple de type d'association entre deux types d'entité

La figure 7.2 illustre le cas d'un type d'association de nom "COMPOSITION" mettant en correspondance les types d'entité de nom "PROCESSUS" et " AGREGAT". La connectivité entre "COMPOSITION" et "PROCESSUS" (0-N) indique que : une entité de type "PROCESSUS" ne doit pas obligatoirement appartenir à une association de type "COMPOSITION" (ce qui explique le 0) mais qu'elle peut appartenir au maximum à un nombre illimité d'une telle association (ce qui explique le N). De façon similaire, la connectivité entre le type d'entité "AGREGAT" et le type d'association "COMPOSITION" (1-N) indique que : une entité de type "AGREGAT" doit appartenir au minimum à une association de type "COMPOSITION" (explication du 1) et qu'elle peut appartenir à un nombre maximum infini d'une telle association (explication du N).

Par rapport au réel perçu, ces deux connectivités indiquent que d'une part un processus ne doit pas toujours entrer dans la composition d'un agrégat mais qu'il peut appartenir à plusieurs agrégats, et d'autre part, qu'un agrégat doit être composé d'au moins un processus et d'au maximum un nombre N.

2.2.3 Concept d'attribut

Un attribut est défini comme une caractéristique ou qualité d'un type d'entité ou d'association. Elle peut prendre une ou plusieurs valeurs parmi un ensemble de valeurs. Les propriétés de base d'un attribut sont :

- un attribut peut être simple ou répétitif selon qu'une entité ou une association ayant cet attribut peut prendre une ou plusieurs valeurs du type de cet attribut;

- un attribut peut être obligatoire ou facultatif selon qu'une entité ou une association ayant cet attribut doit ou ne doit pas posséder une valeur pour cet attribut;

- un attribut peut être identifiant ou non selon que l'entité ou l'association qui contient cet attribut est identifié par chaque valeur de cet attribut.

Conventionnellement, les attributs sont représentés par leur nom à l'intérieur des rectangles des entités auxquelles ils appartiennent;

un exemple est présenté à la figure 7.3.

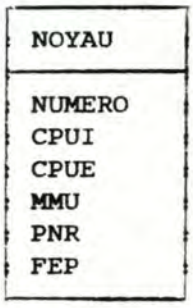


Figure 7.3 : exemple d'attributs dans un type d'entité

La figure 7.3 illustre la situation du type d'entité "NOYAU" dont les attributs et leur sémantique sont :

- NUMERO : le numéro du noyau. Cet attribut est simple, obligatoire et identifiant; il prend ses valeurs parmi les entiers positifs;

- CPUI : la consommation CPU intrinsèque du noyau. Cet attribut est simple, obligatoire et non identifiant; il prend ses valeurs parmi les réels positifs.

- CPUE : la consommation CPU extrinsèque du noyau. Cet attribut est simple, obligatoire et non identifiant; il prend ses valeurs parmi les réels positifs.

- MMU : la consommation en nombre de "MMU Banks" du noyau. Cet attribut est simple, obligatoire et non identifiant; il prend ses valeurs parmi les entiers positifs.

- PNR : la consommation en nombre de PNR du noyau. Cet attribut est simple, obligatoire et non identifiant; il prend ses valeurs parmi les entiers positifs.

- FEP : un attribut booléen indiquant si un noyau est de type "FEP" ou non. Cet attribut est simple, obligatoire et non identifiant.

2.3 Application du modèle entité association à SYANODE

2.3.1 Modèle entité association

Le modèle des informations que nous présentons à la figure 7.4 ne reprend que les informations d'entrée de notre problème de placement. Les notions de travail utilisées par SYANODE (noyau, FEP etc...) sont donc absentes du modèle.

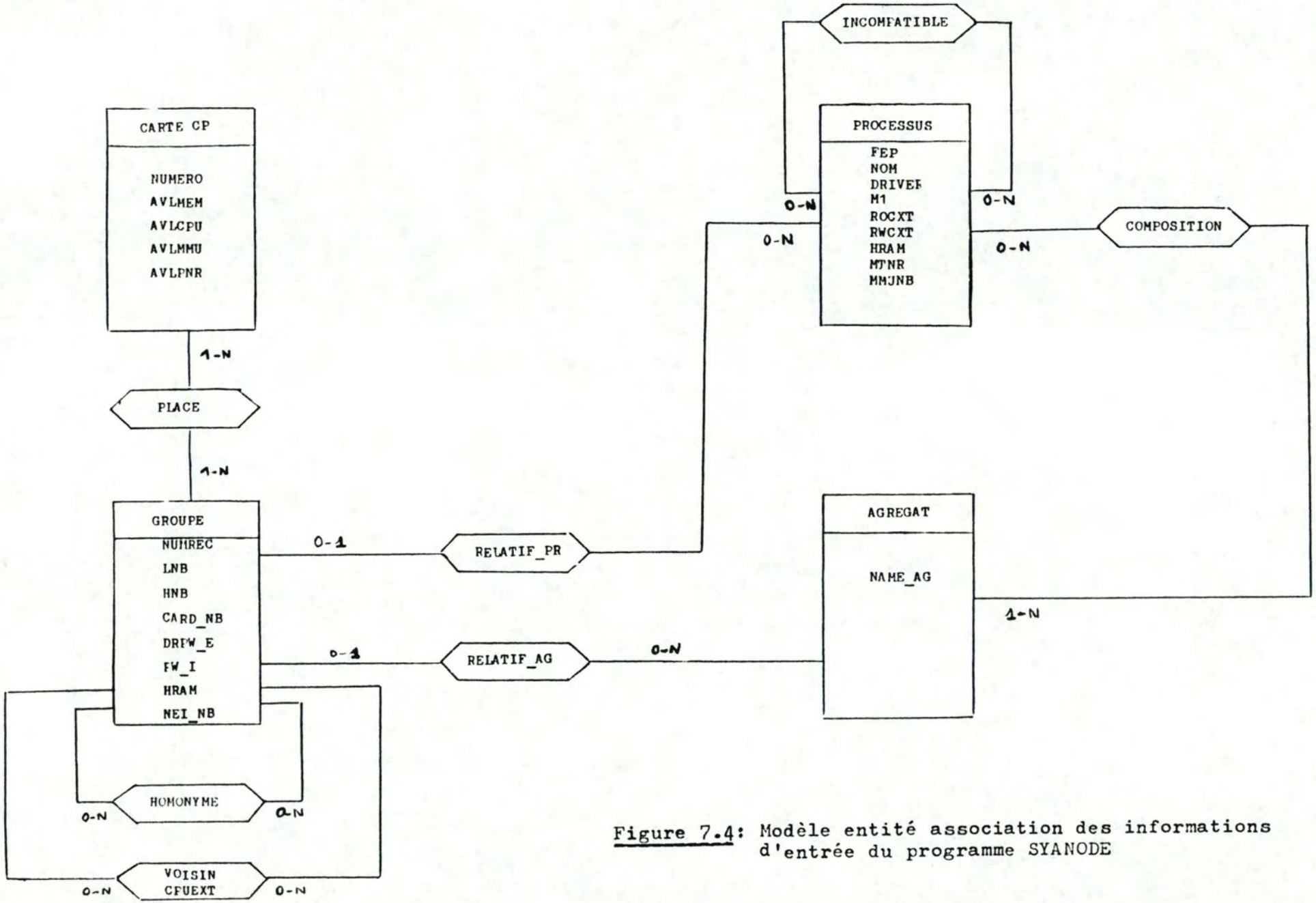


Figure 7.4: Modèle entité association des informations d'entrée du programme SYANODE

2.3.2 Sémantique des entités

entité AGREGAT

Le type d'entité AGREGAT correspond à la notion d'agrégat telle qu'elle a été définie dans le chapitre III. Les attributs de ce type d'entité sont :

- NAME_AG : le nom de l'agrégat commençant conventionnellement par le caractère "@". Cet attribut est l' identifiant de l'agrégat;

entité CARTE CP

Le type d'entité carte CP correspond à la carte CP du réel perçu telle qu'elle a été étudiée dans le chapitre I. Les attributs de ce type d'entité sont :

- NUMERO : le numéro logique de la carte CP, identifiant de la carte CP;
- AVLMEM : capacité mémoire résiduelle de la carte CP en kB;
- AVLCPU : capacité CPU résiduelle de la carte CP en ms/s;
- AVLMMU : capacité MMU résiduelle de la carte CP en "MMU banks";
- AVLPNR : capacité PNR résiduelle de la carte CP en nombre de pnr.

entité GROUPE

Le type d'entité GROUPE correspond a la notion de groupe telle qu'elle a été définie dans le chapitre III. Ses attributs sont :

- NUMREC : le numéro identifiant du groupe;
- LNB : la borne inférieure du groupe. Cette borne inférieure et le nom du groupe trouvé via l'association RELATIF_PR ou RELATIF_AG est un second identifiant du groupe;
- HNB : la borne supérieure du groupe;
- CARD_NB : le numéro de carte CP d'accueil du groupe;
- DRPW_E : la consommation CPU driver extrinsèque, en ms/s, du groupe en tenant compte des drivers placés sur la même carte CP;
- PW_I : la consommation CPU intrinsèque du groupe en ms/s;
- HRAM : la consommation HRAM effective du groupe en kB;
- NEI_NB : le nombre de groupes voisins du groupe.

entité PROCESSUS

Le type d'entité PROCESSUS correspond à la notion de processus telle qu'elle a été définie dans le chapitre III. Ses attributs sont :

- NOM : le nom identifiant du processus;
- DRIVER : cet attribut indique que le processus est de type driver ou non;
- M1 : la consommation mémoire du premier "thread" du processus en kB;
- ROCXT : la consommation mémoire du read-only context du processus en kB;
- RWCXT : la consommation mémoire du read-write context du processus en kB;
- HRAM : la consommation mémoire du HRAM maximum du processus en kB;
- MTNR : le nombre maximum de "threads" acceptés sous un même PNR;
- MMUNB : la consommation en MMU du processus en nombre de "MMU banks";
- NPNR : la consommation en PNR du processus en nombre de pnr;
- FEP : cet attribut indique que le processus est de type "FEP" ou non.

2.3.3 Sémantique des associations

association COMPOSITION

L'association de type COMPOSITION entre les types d'entité AGREGAT et PROCESSUS exprime qu'un agrégat est composé d'un ou de plusieurs processus et qu'un processus peut être composant d'un ou de plusieurs agrégats.

association HOMONYME

L'association récursive de type HOMONYME entre entités de type GROUPE exprime une relation d'homonymie entre groupes de mêmes nom.

association INCOMPATIBLE

L'association récursive de type INCOMPATIBLE du type d'entité PROCESSUS vers lui-même exprime qu'un processus peut être incompatible avec un autre (éventuellement non distinct).

association PLACE

L'association de type PLACE entre les types d'entité GROUPE et CARTE CP exprime qu'un groupe est placé sur une ou plusieurs cartes CP

et qu'une carte CP est le lieu de placement d'un ou de plusieurs groupes.

association RELATIF_AG

L'association de type RELATIF_AG entre les types d'entité AGREGAT et GROUPE exprime qu'un groupe peut être relatif à un agrégat et qu'un agrégat peut être relatif à un ou plusieurs groupes.

association RELATIF_PR

L'association de type RELATIF_PR entre les types d'entité PROCESSUS et GROUPE exprime qu'un groupe peut être relatif à un processus et qu'un processus peut être relatif à un ou plusieurs groupes.

association VOISIN

L'association récursive de type VOISIN entre le type d'entité GROUPE et lui-même exprime qu'un groupe peut être voisin d'un autre groupe. Ce type d'association a les attributs suivants :

- CPUE : la consommation CPU extrinsèque nécessaire aux échanges entre les groupes voisins en ms/s.

2.3.4 Contraintes sémantiques du modèle

Les contraintes sémantiques du modèle entité association sont des informations qui ne peuvent être exprimées dans le formalisme utilisé par ce modèle. Ces contraintes doivent être vérifiées à tout moment. Il s'agit de :

- AVL_MEM d'une entité de type CARTE CP est la différence entre la constante AVL_MEM_CP et la somme des MEM des entités de type GROUPE reliées à CARTE CP par une association de type PLACE;

- AVL_CPU d'une entité de type CARTE CP est la différence entre la constante AVL_PW_CP et la somme des CPUI des entités de type GROUPE reliées à CARTE CP par une association de type PLACE;

- AVL_PNR d'une entité de type CARTE CP est la différence entre la constante AVL_PNR_CP et la somme des PNR des entités de type GROUPE reliées à CARTE CP par une association de type PLACE;

- AVL_MMU d'une entité de type CARTE CP est la différence entre la constante AVL_MUB_CP et la somme des MMU des entités de type GROUPE reliées à CARTE CP par une association de type PLACE;

- le HNB d'une entité de type GROUPE est supérieur ou égal à son LNB;

- une entité de type GROUPE ne peut être reliée qu'à une seule association de type RELATIF_PR ou RELATIF_AG;

- les entités de type GROUPE origine et l'extrémité d'une association de type VOISIN ne peuvent être les mêmes;

- le HRAM d'une entité GROUPE est inférieur ou égal au HRAM de l'entité de type PROCESSUS à laquelle elle est reliée par une association de type RELATIF_PR, ou à la somme des HRAM des entités de type PROCESSUS reliées par l'association de type COMPOSITION à un AGREGAT auquel ce GROUPE est relié par une association de type RELATIF_AG.

3 Mise en oeuvre du modèle

Le modèle entité association décrit dans les points précédents donne lieu à une découpe des données en tables, fichiers et structures de données dynamiques. Ceux-ci font l'objet d'une étude approfondie au point 5.

4 Notion de type abstrait

Le type abstrait est une structure de données associée à un concept de base du problème à résoudre. Le type abstrait est caractérisé et visible par :

- les propriétés de la structure:
  - le nombre d'éléments de la structure;
  - la loi de structuration des éléments;
  - la structure des éléments;
- la liste des opérations qu'il est possible d'effectuer sur la structure. Il existe une relation biunivoque entre la structure de données et ces opérations;
- la spécification de chaque opération.

Cette caractérisation de la structure de données est indépendante d'un choix particulier pour sa représentation; on reste donc à un niveau logique.

Exemple :

Le module MOD_AG (cfr point 6.6.8) réalise un type abstrait :

- la structure de données impliquée dans ce module est la table globale des agrégats (AGR_LIST) dont la définition et la structure ont été étudiées au point 5.4.1.
- les opérations possibles sur cette structure et leur spécification sont : la vérification de l'existence d'un nom d'agrégat (opération EXIST_AG), la recherche de la composition d'un agrégat (opération SEARCH_COMP_AG) et l'accès séquentiel sur les agrégats (opération GET_FIRST_AG et GET_NEXT_AG).

Cette façon de concevoir et de définir les structures de données et les opérations sur ces structures a été suivie lors de la mise en oeuvre de la méthode de fusion et a donné lieu à la découpe en modules telle qu'elle est décrite dans le point 6.

5 Présentation des structures de données.

5.1 Introduction

Nous présentons les structures de données globales définies dans notre programme. Il s'agit des structures représentant les bibliothèques formelle et effective des incompatibilités, le catalogue des agrégats, le catalogue des processus, la liste des arêtes, la liste des cartes, la liste des groupes, la liste des noyaux et la liste des noyaux de type "FEP".

5.2 Bibliothèque effective des incompatibilités

5.2.1 Description physique

La bibliothèque effective des incompatibilités est reprise dans la "BIBLI_LIST". Elle est représentée par un tableau d'incompatibilités de nom BIBLI et de type BIB.

Chaque incompatibilité (de type BIB_EL) est caractérisée par :

- PROC1 : le numéro du premier groupe impliqué dans l'incompatibilité. Son type est NUM_NUC;
- PROC2 : le numéro du second groupe impliqué dans l'incompatibilité. Son type est NUM_NUC;

Un schéma de cette structure de données est présenté à la figure 7.5.

5.2.2 Justification de la structure employée

Nous avons opté pour une structure statique de type tableau pour les raisons suivantes :

- le nombre d'incompatibilités est faible;
- l'accès à ces données est fréquent et doit donc être rapide;
- chaque élément est statique et uniquement composé de deux nombres.

Figure 7.5 : SCHEMA DE LA " BIBLI_LIST "

BIBLI

PROC1	PROC2

5.3 Bibliothèque formelle des incompatibilités

5.3.1 Description physique

Un fichier des incompatibilités formelles reprend, sous forme de record, les incompatibilités formelles entre processus. Ces incompatibilités sont reprises dans la bibliothèque des incompatibilités formelles telle qu'elle a été définie dans le chapitre III relatif à la description du projet de C.A.O. Les records d'incompatibilité, appelés "INCOMP", sont construits directement à partir du fichier TEXT de nom "INPUT.INCOMP"; ce fichier TEXT est décrit dans le manuel d'utilisateur en annexe VI. Les caractéristiques du fichier des incompatibilités sont :

- son nom : "INCOMP.LIST";
- son type : INCOMP_LIST, file of INCOMP (voir point 5.3.3);
- son organisation : ce fichier est purement séquentiel.

5.3.2 Justification de la structure employée

Il est moins évident, dans le cas des incompatibilités, de justifier l'emploi d'une structure de données plutôt qu'une autre étant donné que :

- pour l'instant le nombre d'incompatibilités est faible (ce qui est un argument pour l'usage d'une table) mais à terme ce nombre peut évoluer;
- l'accès sur les incompatibilités est surtout séquentiel, mais le nombre d'accès est très limité;
- la traduction, à chaque exécution de SYANODE, du fichier TEXT des incompatibilités en fichier de records est très courte vu la taille de ce fichier TEXT.

5.3.3 Description de l'INCOMP

Une "INCOMP" est le record de base du fichier des INCOMP. Nous pouvons voir à la figure 7.6 que ce record est constitué des champs suivants :

- NUM_INC : ce champ contient le numéro identifiant de l'incompatibilité formelle. Ce champ est de type entier strictement positif;
- PR_OR : ce champ contient le nom du premier des deux processus de l'incompatibilité. Ce champ est une chaine de caractères de longueur maximale constante LMAXPR (PROCESS_NAME);
- PR_DEST : ce champ contient le nom du second processus de l'incompatibilité. Ce champ est de type PROCESS_NAME. Le couple

(PR_OR,PR_DEST) constitue également un identifiant de la contrainte formelle.

NUM_INC		identifiant de l'INCOMP
PR_OR	PR_DEST	processus composant de l'INCOMP

Figure 7.6 : structure d'un record INCOMP

5.4 Catalogue des agrégats

5.4.1 Description physique

La table des agrégats reprend pour chacun de ceux-ci la liste des processus qui le composent. Cette table des agrégats est appelée "catalogue des agrégats" dans le chapitre III relatif à la description du projet de C.A.O. Cette table de records est construite à partir d'un fichier TEXT de nom "INPUT.AGNAM" fourni par l'utilisateur; ce fichier TEXT est décrit dans le manuel d'utilisateur en annexe VI. Les caractéristiques de la table des agrégats sont :

- son nom : "AGR_LIST";

- son type : AGREGAT_LIST (array [1.. MAXAG] of AGREGAT); la structure AGREGAT est définie au point 5.4.3.

5.4.2 Justification de la structure employée

Il n'est pas nécessaire de réaliser le catalogue des agrégats sous forme de fichier car :

- le nombre d'agrégats est très limité.

5.4.3 Description de l'AGREGAT

Un record de type AGREGAT (voir figure 7.7) composant la table est construit directement à partir des agrégats fournis à l'entrée. Chaque AGREGAT reprend les champs suivants :

- NAME_AG : ce champ contient le nom de l'agrégat. Pour rappel, ce nom conventionnellement commence toujours par le caractère "@". Ce champ est une chaîne de caractères de longueur maximale constante LMAXAG (AGR_NAME);

- TP_AG : ce champ booléen est positionné à vrai si un processus de type driver est repris parmi les processus composant l'agrégat;

- COMPOSITION : ce champ contient les noms des processus contenus dans l'agrégat. Ces noms sont séparés par un caractère blanc. Ce champ est une chaine de caractères de longueur maximale constante LMAXCOMPO (COMPOSTYP).

NAME_AG	identifiant de l'AGREGAT
TYP_AG : COMPOSITION	qualité et composition de l'AGREGAT

Figure 7.7 : structure d'un record AGREGAT

5.5 Catalogue des processus

5.5.1 Description physique

Un fichier des processus reprend l'ensemble des caractéristiques des processus qui sont indépendantes de la configuration. Ce fichier est appelé "catalogue des processus" dans le chapitre III relatif à la description du projet de C.A.O. Ce fichier de record est construit à partir d'un fichier TEXT appelé "INPUT.PNAM" décrit dans le manuel d'utilisateur en annexe VI. Les caractéristiques du fichier des processus sont :

- son nom : "PNAM.LIST";
- son type : PROCESS_LIST, file of PROCESS;
- ses clés : le fichier des processus a deux clés (voir figure 7.8), la première est basée sur le nom du processus. Cette clé est la clé [0], elle est identifiante. La deuxième clé est celle se basant sur le numéro du processus. Cette clé est également identifiante et porte le numéro [1].
- son organisation : séquentielle indexée sur la clé [0].

5.5.2 Justification de la structure employée

Il ne nous a pas semblé intéressant d'utiliser une table pour la mémorisation de la description des processus lors de l'exécution de SYANODE car :

- nous désirons maintenir plusieurs clés d'accès sur l'ensemble des processus;
- il aurait été nécessaire de traduire une structure TEXT sous forme de table à chaque exécution de SYANODE alors que le nombre de processus à terme peut devenir très important. Nous avons donc prévu, dans notre programmme, de ne traduire le fichier TEXT que si celui-ci avait été modifié par l'utilisateur.

5.5.3 Description du PROCESS

Un record PROCESS (voir figure 7.8) est construit directement à partir des processus du fichier TEXT d'entrée. Il se compose des champs suivants :

- NAME : ce champ contient le nom du processus. Son type est PROCESS_NAME;

- NPR : ce champ contient le numéro identifiant du processus. Il correspond à l'ordre de lecture des processus à partir du fichier TEXT. Le type de ce champ est NUM_PR;

- DRIVER : ce champ booléen est positionné à vrai si le processus est de type driver et à faux dans le cas contraire;

- FEP : ce champ booléen est positionné à vrai si le processus est de type "FEP" et à faux dans le cas contraire;

- MMUNB : ce champ contient une information relative à la consommation en "MMU banks" du processus. La valeur contenue dans ce champ est en fait l'exposant de 2 permettant de calculer cette consommation. Pour la carte CP de type "CP512", sa valeur est 0, 1 ou 2. La consommation en "MMU banks" du processus est donc soit de 1, 2 ou 4 "MMU bank(s)". La valeur de ce champ est donc de type entier (INTEGER);

- M1 : ce champ contient la consommation mémoire initiale du processus. Celle-ci comprend la taille du code et des données communes pour un programme "multithread" plus la taille du contexte du premier "thread" de ce programme. La valeur de ce champ est de type MEM_QTY;

- RWCXT : ce champ contient la taille du "read-write context" du processus. Cette valeur est de type MEM_QTY;

- ROCXT : ce champ contient la taille du "read-only context" du processus. La valeur de ce champ est de type MEM_QTY;

- MTNR : ce champ contient le nombre maximum de "threads" d'un programme pouvant avoir le même PNR. La valeur de ce champ est de type entier strictement positif (INTEGER);

- HRAM : ce champ contient la taille du HRAM maximum pour un processus. La valeur de ce champ est de type MEM_QTY.

NAME [CLE (0)]		NPR [CLE (1)]				identifiants du PROCESS
DRIVER		FEP				qualités du PROCESS
MMUNB	M1	RWCXT	ROCXT	MTNR	HRAM	consommations mémoire du PROCESS

Figure 7.8 : structure d'un record PROCESS

5.6 Liste des arêtes

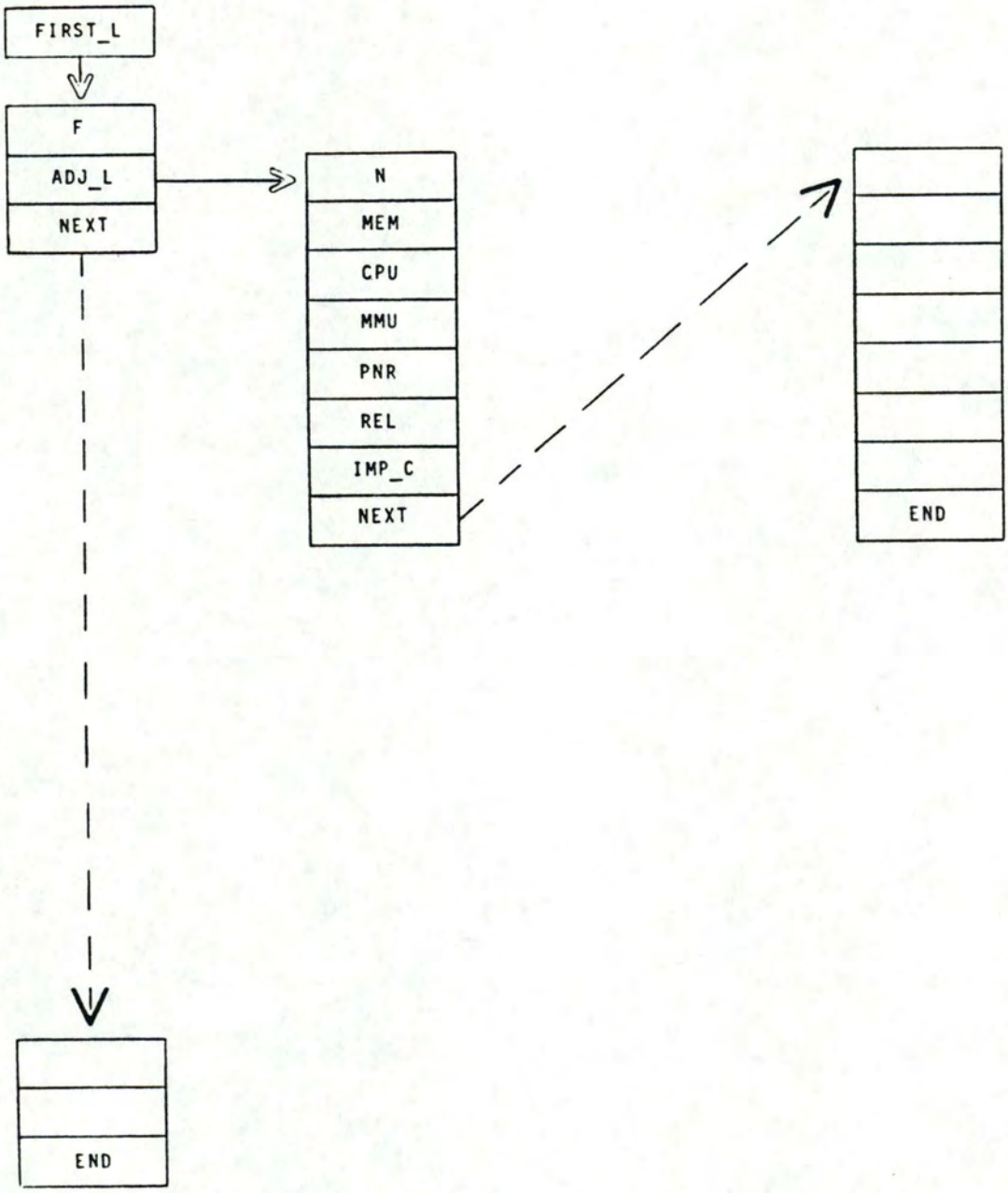
5.6.1 Description physique

Les arêtes du graphe sont reprises dans la "EDGE_LIST". Chaque arête est représentée par son noyau d'origine et son noyau de destination. Par convention, le noyau d'origine est celui ayant le plus petit numéro, le noyau de destination ayant le plus grand numéro. De cette sorte une arête entre les noyaux 7 et 1 est représentée par une arête partant du noyau 1 vers le noyau 7.

La "EDGE_LIST", qui est une liste chaînée, est composée d'un ensemble de noyaux d'origine. Chaque noyau d'origine est caractérisé par :

- F : un numéro de type NUM_NUC;
- ADJ_L : la liste des noyaux de destination lui correspondant, cette liste est de type PADJ. Cette liste est un ensemble d'éléments composés de :
  - N : un numéro de noyau de type NUM_NUC;
  - MEM : la quantité correspondant au gain mémoire de cette arête. Son type est MEM_QTY;
  - CPU : la quantité correspondant au gain CPU de cette arête. Son type est type CPU_QTY;
  - MMU : la quantité correspondant au gain MMU de cette arête. Son type est MMU_QTY;
  - PNR : la quantité correspondant au gain PNR de cette arête. Son type est PNR_QTY;
  - REL : le poids normalisé de cette arête. Son type est REL_QTY;
  - IMP_CEIL : le fait que l'arête est impossible suite au limites des tailles mémoire et CPU des noyaux. Son type est BOOLEAN;

Figure 7.9 : SCHEMA DE LA " EDGE_LIST "



- NEXT : la référence au noyau de destination suivant. Son type est PADJ;

- NEXT : la référence au noyau d'origine suivant. Son type est PFIRST.

La référence au premier noyau d'origine se trouve dans la variable globale : FIRST_L.

Un schéma de cette structure de données est présenté à la figure 7.9

5.6.2 Justification de la structure employée

Nous avons choisi une structure de données dynanique pour représenter les arêtes car le nombre d'arêtes dépend du nombre de groupes. De plus, comme une matrice représentant les échanges entre noyaux serait creuse, il est inutile de choisir une représentation matricielle.

5.7 Liste des cartes

5.7.1 Description physique

L'ensemble des cartes entamées et donc encore utilisables est repris dans la "CARD_LIST". La "FULL_CARD_LIST" représente, en cours de traitement, les cartes saturées et en fin de programme, l'ensemble de toutes les cartes utilisées pour le placement. Ces deux listes ont évidement une même structure.

Chaque carte CP est caractérisée par :

- NUM : un numéro de type NUM_CARD;

- AVLMEM : la quantité de mémoire encore disponible sur cette carte CP. Son type est MEM_QTY;

- AVLCPU : la quantité de CPU encore disponible sur cette carte CP. Son type est CPU_QTY;

- AVLMMU : la quantité de MMU encore disponible sur cette carte CP. Son type est MMU_QTY;

- AVLPNR : le nombre de PNR encore disponibles sur cette carte CP. Son type est PNR_QTY;

- WELCOM : le nombre de "threads ", d'un "FEP" considéré, qu'elle peut accueillir. Son type est INTEGER;

- RAP_DISPO : le rapport de disponibilité qui lui correspond. Son type est REAL;

- ASS_NUC_L : la liste des numéros des noyaux statiques placés sur cette carte CP. Son type est P_ASS_NUC. Cette liste est un

ensemble d'éléments composés de :

- N : un numéro de noyau de type NUM_NUC;

- NEXT : une référence à l'élément suivant. Son type est P_ASS_NUC;

- ASS_FEP_L : la liste des numéros de noyaux dynamiques placés sur cette carte CP. Son type est P_ASS_FEP. Cette liste est un ensemble d'éléments composés de :

  - N : un numéro de noyau de type NUM_NUC;

  - NEXT : une référence à l'élément suivant. Son type est P_ASS_FEP;

- NEXT : une référence à la carte CP suivante. Son type est PCARD.

La référence à la "CARD_LIST" se trouve dans la variable globale CARD_L (PCARD) et la référence à la "FULL_CARD_LIST" se trouve dans la variable globale FULL_CARD_L (PCARD).

Un schéma de cette structure de données est présenté à la figure 7.10

5.7.2 Justification de la structure employée

Nous avons opté pour une structure de données dynamique car le nombre de cartes CP nécessaires varie fortement selon le nombre et les consommations de mémoire et de CPU des groupes à placer.

Nous avons également opté pour des structures dynamiques afin de représenter la liste des noyaux statiques ainsi que la liste des noyaux dynamiques associés à une carte CP. Ce choix provient du fait que les nombres de noyaux statiques ou dynamiques associés à une carte CP varient fortement d'une carte CP à l'autre.

5.8 Liste des groupes

5.8.1 Description physique

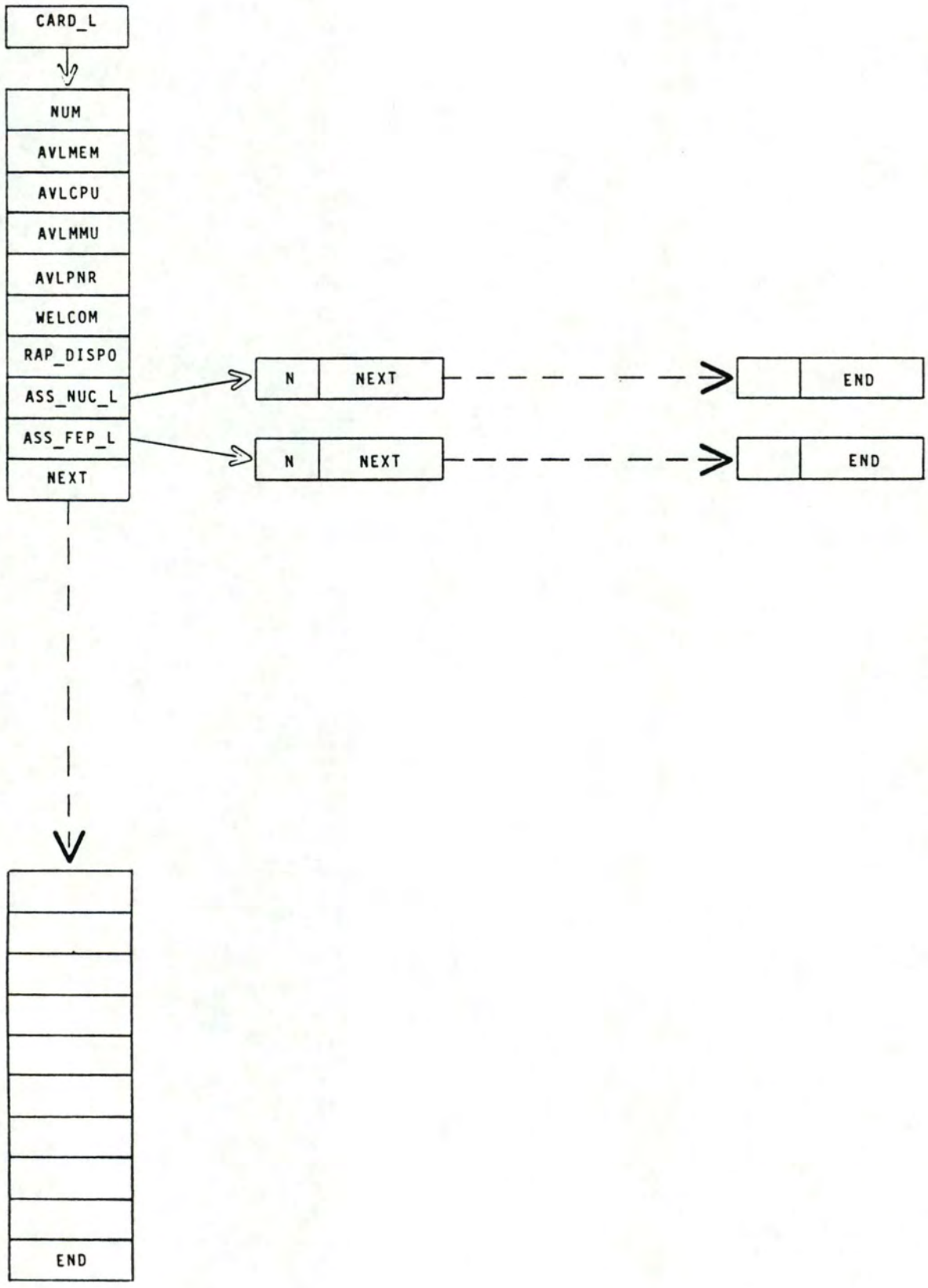
Un fichier des groupes reprend, sous forme de records de type GROUP, la liste des groupes d'entrée à placer sur les cartes CP d'un noeud SOPHO-NET. Ce fichier de record est construit à partir du fichier TEXT de nom "INPUT.GRP" fourni par l'utilisateur; ce fichier TEXT est décrit dans le manuel d'utilisateur en annexe VI. Les caractéristiques du fichier des groupes sont :

- son nom : "GRP.LIST";

- son type : GROUP_LIST, file of GROUP (voir point 5.8.3);

- ses clés d'accès : le fichier des groupes en a trois (voir figure 7.11), la première de ces clés porte sur le couple (nom

Figure 7.10 : SCHEMA DE LA " CARD_LIST "



du GROUP, borne inférieure du GROUP). Cette clé est la clé numéro [0], elle est identifiante. La deuxième clé se base sur le nom du groupe, cette clé est non identifiante et porte le numéro [1]. Enfin, la troisième clé est basée sur l'ordre de lecture des groupes à partir du fichier TEXT d'entrée de l'utilisateur. Cette clé est identifiante et porte le numéro [2];

- son organisation : séquentielle indexée sur la clé [0].

5.8.2 Justification de la structure employée

Le fichier des groupes n'aurait pas pu raisonnablement être réalisé sous forme d'une table car :

- le nombre de groupes peut être très important (MAXGR groupes);
- plusieurs clés d'accès différentes sont demandées sur l'ensemble des groupes; d'où la nécessité de définir plusieurs clés.

5.8.3 Description du GROUP

Le GROUP (voir figure 7.11) est le record de base du fichier des groupes défini dans le point précédent. Il correspond de manière biunivoque aux groupes fournis dans le fichier TEXT d'entrée par l'utilisateur. Le record GROUP, visualisable à la figure 7.11 est constitué des champs suivants :

- NUMREC : ce champ correspond à l'ordre de lecture du groupe dans le fichier TEXT initial de l'utilisateur. C'est donc un entier positif (INTEGER) dont la valeur est inférieure ou égale à la constante MAXGR. Ce champ est clé d'accès identifiante numéro 2;
- IDREC : ce champ est lui-même un record contenant l'identifiant (clé 0) du groupe. Il est constitué des champs suivants :
  - GR_NAME : ce champ est le nom du groupe; c'est la clé d'accès non identifiante numéro 1. GR_NAME une chaine de caractères de longueur maximum constante MAXGRNAME (GROUP_NAME);
  - LN_GR : ce champ reprend la borne inférieure des "threads" ou agrégats qui constituent le groupe, sa valeur est de type entier positif (INTEGER);
- HNB : ce champ contient la borne supérieure des "threads" ou agrégats réunis dans le groupe. Ce champ est un entier positif (INTEGER);
- CARD_NB : ce champ contient le numéro logique de la carte CP sur laquelle l'utilisateur impose que le groupe soit placé. Ce numéro est en général indéfini, il a alors la valeur constante UNDEFINED. Ce champ est un entier compris entre UNDEFINED et le numéro de carte CP maximum par noeud : la constante MAXCARDNB;

- PW_I : ce champ contient la consommation CPU intrinsèque du groupe. C'est donc une valeur de type CPU_QTY;

- DRPW_E : ce champ contient la consommation driver CPU extrinsèque. Il a la valeur constante 0 si le groupe n'est pas de type driver. Il a une valeur non nulle de type CPU_QTY dans le cas contraire;

- HRAM : ce champ contient la valeur du HRAM pour le groupe considéré. La valeur de ce champ est de type MEM_QTY;

- XCPU : ce champ contient la somme des consommations CPU extrinsèques du groupe avec des groupes voisins indéfinis, ou que l'utilisateur ne veut pas spécifier explicitement; la valeur de ce champ est de type CPU_QTY;

- OVF : ce champ contient une valeur booléene positionnée à vrai si le groupe qui la contient est en situation d'overflow mémoire ou CPU dès sa lecture, à partir du fichier TEXT. Il est positionné à faux dans le cas contraire;

- NEI_NB : ce champ reprend le nombre de groupes voisins du groupe considéré. Sa valeur est entière positive et inférieure ou égale à la constante MAXNEINB;

- NEI_LIST : ce champ est un tableau d'au plus MAXNEINB NEIGHBOUR, un "NEIGHBOUR" est également un record dont la structure est la suivante :

  - NUM : ce champ est un entier contenant le numéro du groupe voisin; ce champ a le même sens que NUMREC mais pour le groupe voisin;

  - NAME : ce champ contient le nom du groupe voisin; il est donc de type GROUP_NAME;

  - LNG : ce champ contient la borne inférieure du groupe voisin. Il est de type entier. Ce champ associé au champ précédent est en fait l'identifiant du NEIGHBOUR;

  - PW_E : ce champ contient la consommation CPU extrinsèque du groupe avec le groupe voisin identifié par le couple (NAME,LNG). Sa valeur est de type CPU_QTY;

  - SEEN : ce champ booléen est positionné à vrai lorsque l'étape de construction du graphe a déjà construit l'arête CPU entre le groupe considéré et son groupe voisin. Il est positionné à faux dans le cas contraire. Ce champ sert essentiellement à éviter, lors de l'étape de construction du graphe, de devoir créer chaque arête CPU par deux accès au fichier des groupes.

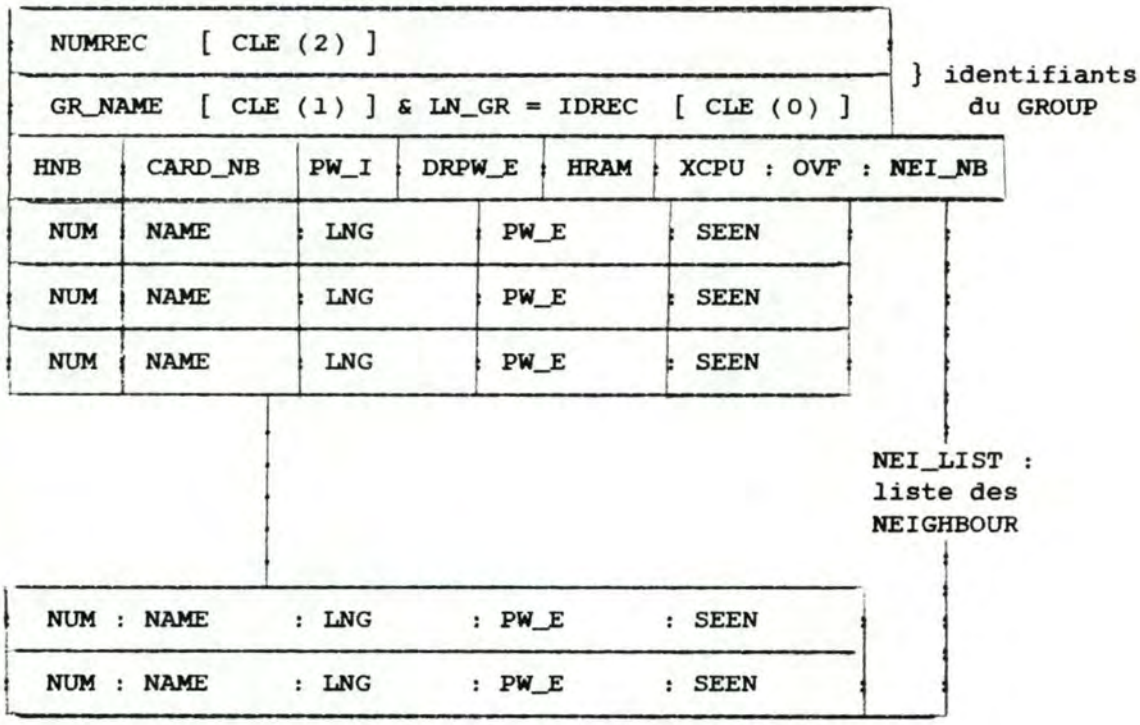


Figure 7.11 : structure d'un record GROUP

5.9 Liste des noyaux

5.9.1 Description physique

Durant les étapes de construction et de fusion du graphe, tous les noyaux sont repris dans la "NUCLEUS_LIST". Avant l'étape de placement des noyaux de type "NON FEP", tous les noyaux de type "FEP" sont transférés dans la structure que nous décrirons au point suivant : la "FEP_LIST".

La "NUCLEUS_LIST" est un ensemble d'éléments caractérisés par :

- N : un numéro de type NUM_NUC;
- CPUI : la quantité de CPU intrinsèque nécessaire au noyau. Son type est CPU_QTY;
- CPUE : la quantité de CPU extrinsèque nécessaire au noyau. Son type est CPU_QTY;
- MEM : la quantité de mémoire nécessaire au noyau. Son type est MEM_QTY
- NUMCARD : le numéro de la carte CP où le noyau doit être placé. Son type est NUM_CARD;

- FEP : le fait que le noyau est de type "FEP" ou non. Son type est BOOLEAN;

- CONSTR : le fait que le noyau est ou non impliqué dans une ou plusieurs incompatibilités. Son type est BOOLEAN;

- MMU : la quantité de MMU nécessaire au noyau. Son type est MMU_QTY;

- PNR : le nombre de PNR dont le noyau a besoin. Son type est PNR_QTY;

- GRCOMP : la liste des numéros des groupes le composant. Son type est PCOMP. Cette liste est un ensemble d'éléments composés de :

  - GR : le numéro d'un groupe faisant partie du noyau considéré. Son type est INTEGER;

  - NEXT : une référence à l'élément suivant. Son type est PCOMP;

- PRCOMP : la liste des numéros des processus correspondants aux groupes le composant. Son type est PCOMP. Cette liste est un ensemble d'éléments composés de :

  - PR : le numéro d'un processus faisant partie du noyau considéré. Son type est INTEGER;

  - NEXT : une référence à l'élément suivant. Son type est PCOMP;

- NEXT : la référence au noyau suivant. Son type est PNUC.

La référence au premier noyau se trouve dans la variable globale NUC_L (PNUC).

Notons que nous avons associé à un noyau la liste des numéros des processus correspondants aux groupes qui le composent. Ceci dans le but, lors de l'étape de fusion des noyaux, de vérifier si deux arêtes de gain mémoire représentent un gain pour un même processus ou non.

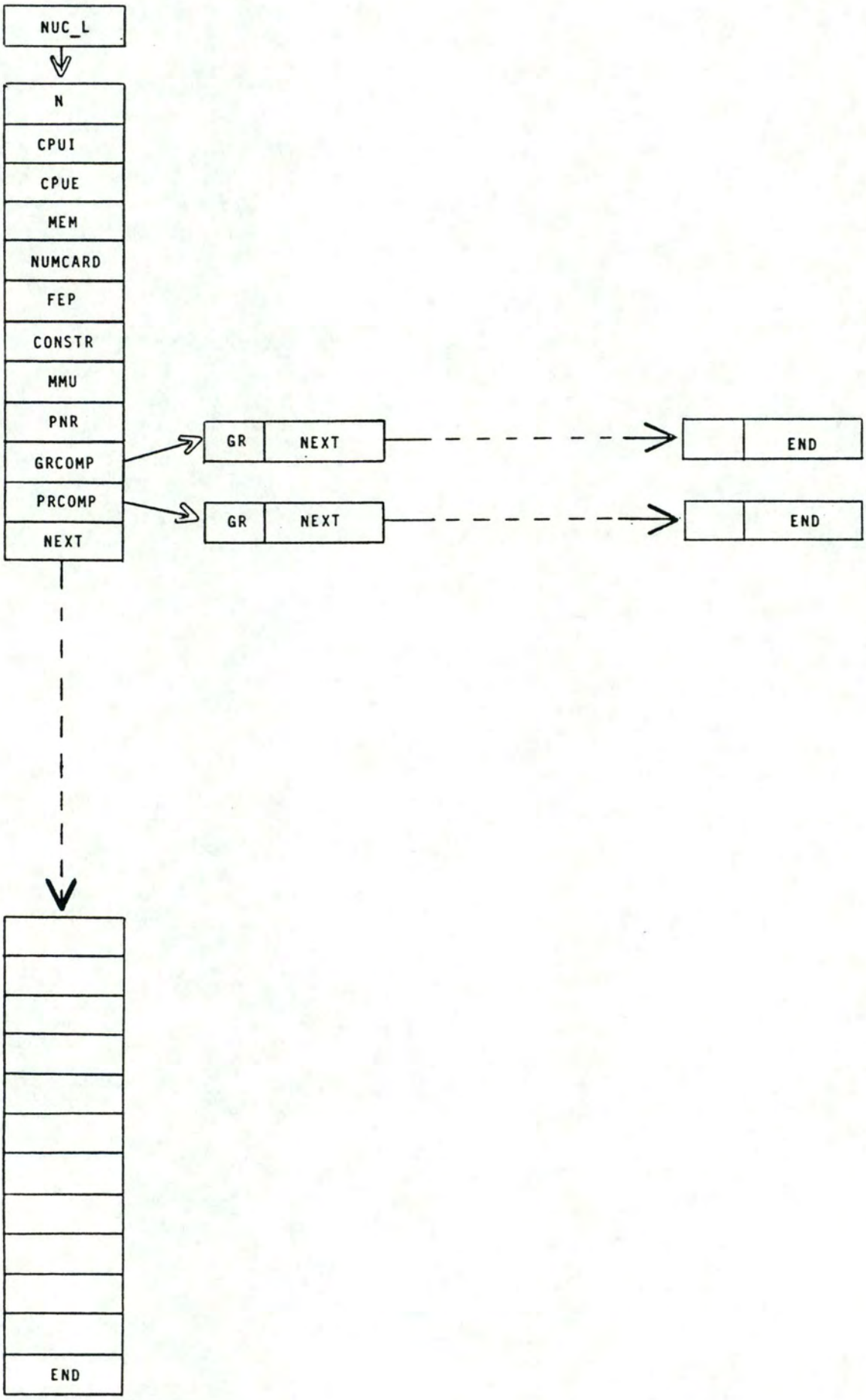
Un schéma de cette structure de données est présenté à la figure 7.12

5.9.2 Justification de la structure employée

Nous avons choisi une structure de données dynamique pour représenter les noyaux car le nombre de noyaux dépend strictement du nombre de groupes à placer ainsi que de leurs consommations mémoire et CPU. De plus le nombre de noyaux dépend des tailles limites mémoire et CPU définies pour un noyau. Il est donc impossible de définir une borne supérieure au nombre de noyaux.

Nous avons opté pour des structures dynamiques dans le but de représenter la liste des numéros des groupes correspondants aux noyaux

Figure 7.12 : SCHEMA DE LA " NUCLEUS_LIST "



ainsi que la liste des numéros de processus correspondants à ces groupes car la taille de ces listes est variable pour chaque noyau.

5.10 Liste des noyaux de type "FEP"

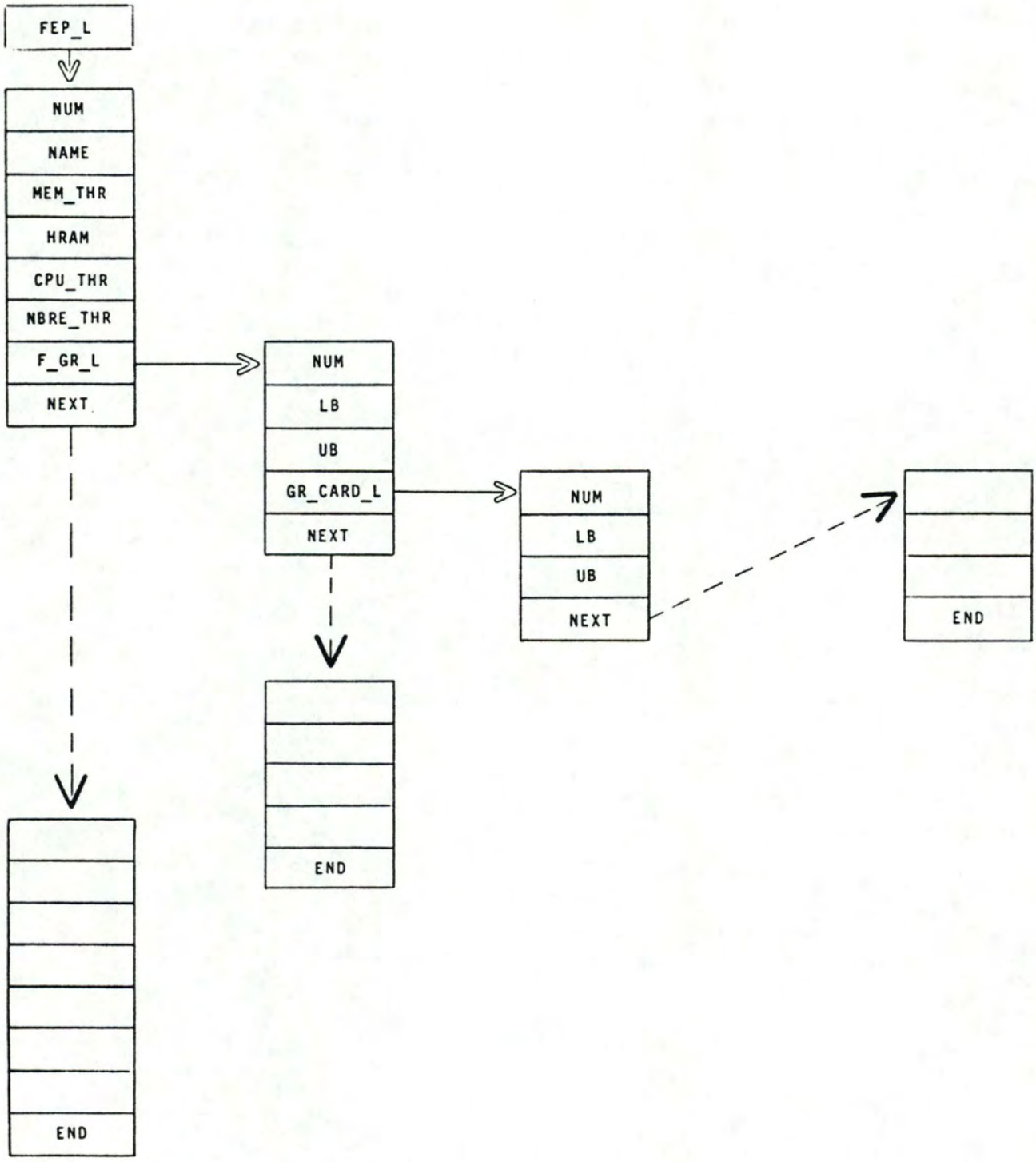
5.10.1 Description physique

Les noyaux de type "FEP" sont repris dans la "FEP_LIST" qui est utilisée lors de l'étape de placement des noyaux de type "FEP".

La "FEP_LIST" est un ensemble d'éléments caractérisés par :

- NUM : un numéro de type NUM_FEP;
- FEP_NAME : un nom de type FEP_NAME;
- MEM_THR : la quantité de mémoire nécessaire à un "thread" du noyau de type "FEP" considéré . Son type est MEM_QTY;
- HRAM : la quantité HRAM correspondant au noyau de type "FEP" considéré. Son type est MEM_QTY;
- CPU_THR : la quantité de CPU nécessaire à un "thread" du noyau de type "FEP" considéré. Son type est CPU_QTY;
- NBRE_THR : le nombre de "threads" du noyau de type "FEP" considéré. Son type est INTEGER;
- F_GR_L : la liste des numéros de groupes composant le noyau de type "FEP" considéré. Son type est PF_GR. Cette liste est un ensemble d'éléments composés de :
  - NUM : un numéro de groupe; de type NUM_GR;
  - LB : la borne inférieure des "threads" de ce groupe. Son type est INTEGER;
  - UB : la borne supérieure des "threads" de ce groupe. Son type est INTEGER;
  - GR_CARD_L : la liste des cartes CP sur lesquelles ce groupe a été placé. Son type est PGR_CARD. Cette liste est un ensemble d'éléments composés de :
    - NUM : un numéro de carte CP; de type NUM_CARD;
    - LB : la borne inférieure des "threads" du groupe placés sur cette carte CP. Son type est INTEGER;
    - UB : la borne supérieure des "threads" du groupe placés sur cette carte CP. Son type est INTEGER;
    - NEXT : une référence à une autre carte CP où a été placé ce groupe. Son type est P_GR_CARD;

Figure 7.13 : SCHEMA DE LA " FEP_LIST "



- NEXT : une référence au groupe suivant. Son type est PF_GR;

- NEXT : une référence au "FEP" suivant. Son type est PFEP.

La référence au premier noyau de type "FEP" se trouve dans la variable globale : FEP_L.

Un schéma de cette structure de données est présenté à la figure 7.13.

5.10.2 Justification de la structure employée

La justification d'une structure dynamique pour représenter les noyaux de type "FEP" est similaire à celle des noyaux. En effet, il est impossible d'évaluer la borne supérieure du nombre des noyaux de type "FEP".

La liste des numéros des groupes composant un noyau ainsi que la liste des cartes CP sur lesquelles un groupe est placé sont des structures dynamiques car la taille de ces listes varie pour chaque noyau.

6 Présentation de la découpe en modules

6.1 Introduction

Nous définissons premièrement le concept de module, ensuite le concept de hiérarchie et plus particulièrement celui de hiérarchie "utilise". C'est selon cette hiérarchie que notre programme et nos modules ont été construits. Nous présentons alors un schéma de l'architecture globale de nos modules. Ensuite nous expliquons le formalisme adopté dans le but de schématiser chaque module en particulier. En effet, les procédures comprises dans un module sont également définies selon la hiérarchie "utilise". Pour terminer, nous détaillons chaque module en décrivant sa fonction, les structures de données globales qu'il utilise, ses particularités au point de vue des structures de données ou de programmation, la liste alphabétique des procédures qu'il comprend et leur numéro de référence dans le fichier source correspondant au module ainsi qu'un schéma montrant l'agencement des procédures selon la hiérarchie "utilise".

Notons que les spécifications des procédures sont reprises dans le programme source. Chaque procédure a une fonction qui lui est propre et nous avons veillé à ce que la taille limite du code d'une procédure soit une page. Ceci facilite la correction et les modifications des procédures.

6.2 Définition d'un module

Un module comprend toutes les procédures susceptibles d'être changées lors d'une même modification importante. Vis-à-vis d'autres modules, un module doit tenir secret le choix d'un mode de représentation des données, les décisions de conception, les schémas d'accès aux données et les modes de réalisation des traitements. Construire des modules permet de réaliser plus facilement les modifications et adaptations futures du programme.

6.3 Définitions de hiérarchie et hiérarchie "utilise"

Une structure est hiérarchisée s'il existe une relation "R" entre ses composants. Cette relation doit permettre de définir des niveaux tels que le niveau 0 soit l'ensemble des composants A pour lesquels il n'existe aucun composant B tel que R (A, B). Il s'agit d'une structure arborescente.

La hiérarchie "utilise" est définie de la manière suivante : le composant A utilise le composant B si le fonctionnement correct de A dépend de la disponibilité d'une version correcte de B.

6.4 Architecture globale des modules

Nous présentons l'architecture globale des modules sous forme de schéma à la figure 7.14. Chaque module y est représenté par une boite et lorsqu'un module en utilise un autre, un trait les relie.

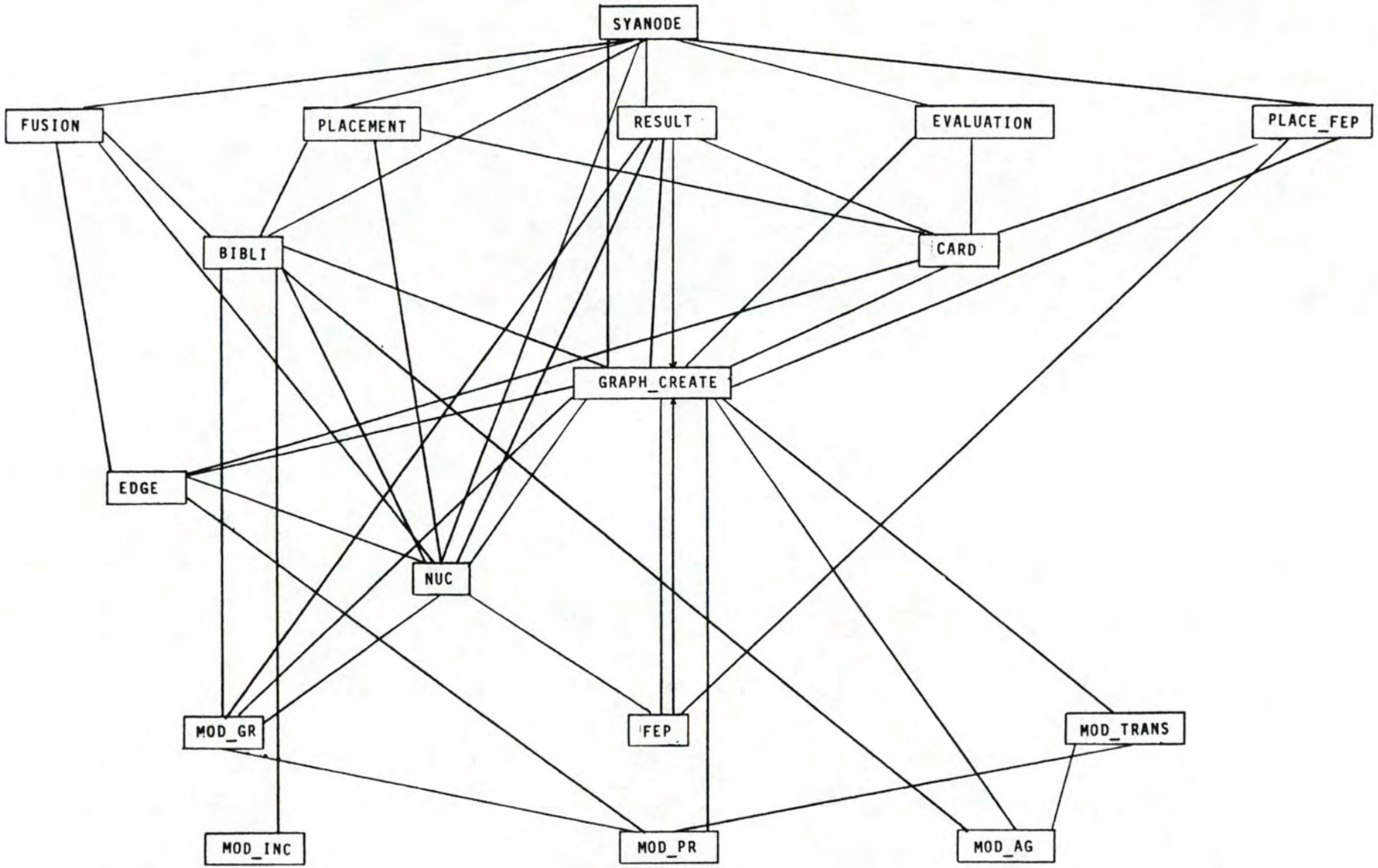
6.5 Formalisme de schématisation d'un module

Le formalisme de schématisation d'un module est décrit dans les paragraphes suivants.

Nous avons représenté les procédures propres au module décrit par une boite en traits continus. Par contre, les procédures définies dans d'autres modules, mais utilisées dans le module décrit sont représentées par des boites en traits discontinus. Les structures de données impliquées dans le module décrit sont représentées par une boite en doubles traits continus.

Lorsqu'une procédure d'un module en utilise une autre, les deux boites représentant ces deux procédures sont reliées par une arête. Notons que lorsqu'une arête doit pour atteindre une boite, en traverser d'autres, nous mettons des flèches sur cette arête à l'endroit des coupures. Ceci signifie que l'arête coupée continue au-delà de la boite qu'elle a rencontrée.

Figure 7.14 : ARCHITECTURE GLOBALE DES MODULES



6.6 Description des modules

6.6.1 Module BIBLI

6.6.1.1 Spécifications

Le module BIBLI a pour fonction de créer et de gérer la bibliothèque effective des contraintes.

6.6.1.2 Structures de données globales gérées

Le module BIBLI gère la structure de données "BIBLI_L"; cette structure est celle reprenant les contraintes effectives.

6.6.1.3 Particularités

Le module BIBLI utilise un type de structure de données qui lui est propre; il s'agit du type "TLIST" qui est un tableau de nombres entiers positifs. Nous mémorisons dans ce type de tableaux les numéros des groupes correspondants à un nom de processus donné. Ceci nous permet de traduire les incompatibilités exprimées en termes de processus dans la bibliothèque formelle en incompatibilités exprimées en termes de groupes dans la bibliothèque effective.

6.6.1.4 Liste des procédures et fonctions

Nous rappelons que les numéros à coté des procédures correspondent aux numéros de référence dans le fichier source.

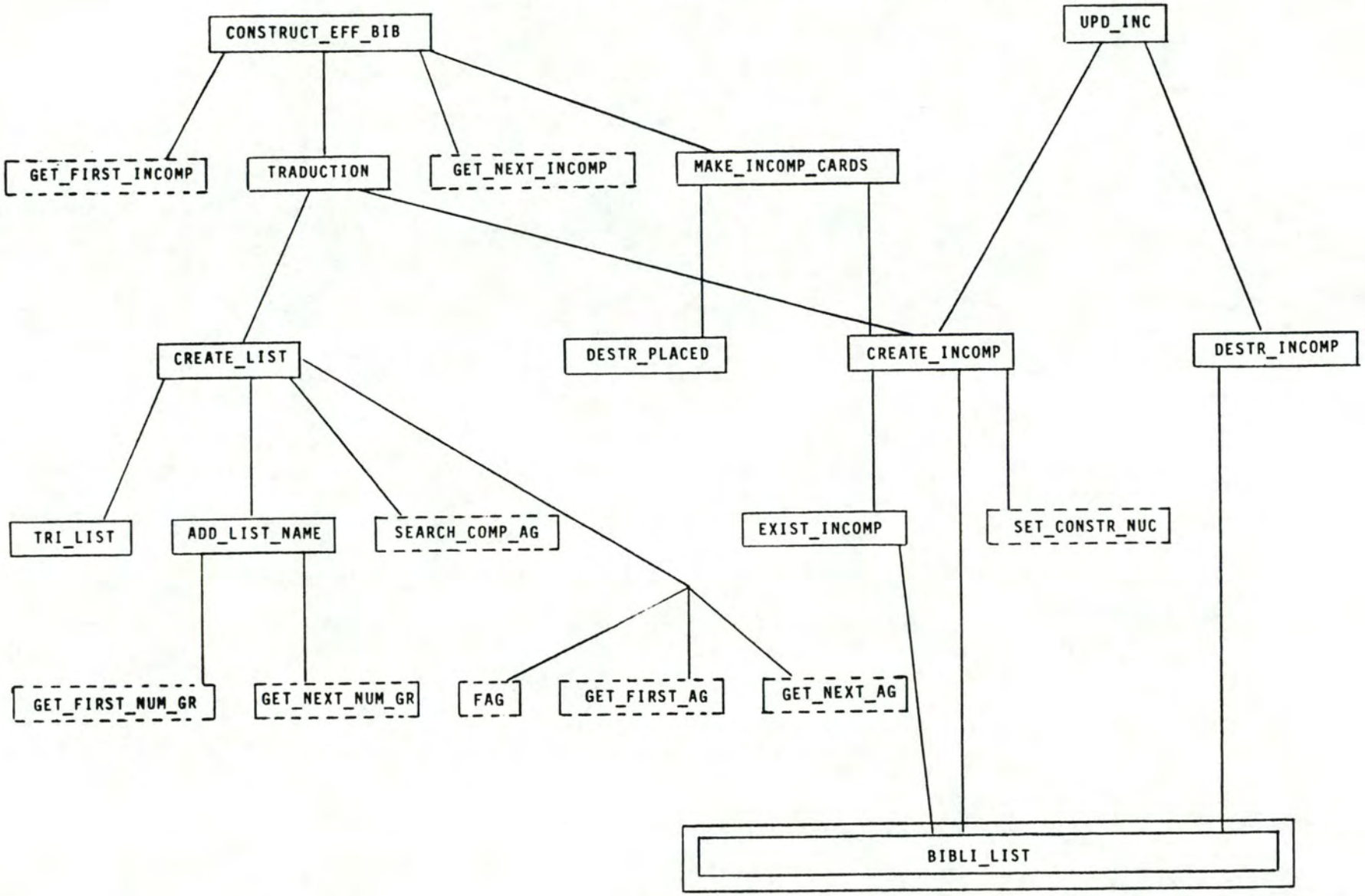
- ADD_LIST_NAME 9
- CREATE_INCOMP 2
- CREATE_LIST 10
- CONSTRUCT_EFF_BIB 12
- DESTR_INCOMP 3
- DESTR_PLACED 7
- EXIST_INCOMP 1
- MAKE_INCOMP_CARD 8
- PRINT_INCOMP 4
- TRADUCTION 11
- TRI_LIST 5
- UPD_INC 6

6.6.1.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.15.

6.6.2 Module CARD

Figure 7.15 : ARCHITECTURE DU MODULE BIBLI



6.6.2.1 Spécifications

Le module CARD, qui est un module d'accès, gère deux structures de données identiques. Il s'agit de la "CARD_L" et de la "FULL_CARD_L". La "CARD_L" représente, en cours de traitement, l'ensemble des cartes CP entamées tandis que la "FULL_CARD_L" représente les cartes saturées. En fin de traitement, toutes les cartes utilisées sont reprises dans la "FULL_CARD_L".

6.6.2.2 Structures de données globales gérées

Le module CARD gère les structures de données "CARD_L" et "FULL_CARD_L".

6.6.2.3 Particularités

Ce module permet aux autres modules d'accéder aux structures de données "CARD_L" ET "FULL_CARD_L" sans en connaitre la structure physique.

6.6.2.4 Liste des procédures et fonctions

- ATTACH_LIST 44
- CALC_WELC_CARDS 38
- CREATE_AF 22
- CREATE_AN 16
- CREATE_NEW_CARD 10
- CUT_LIST 43
- EXIST_CARD 7
- EXIST_FULL_CARD 8
- FIND_AF 21
- FIND_AN 15
- FIND_CARD 6
- GAIN 30
- GET_AVL_Q_CARD 34
- GET_FIRST_ASS_FEP 31
- GET_FIRST_ASS_NUC 28
- GET_FIRST_CARD 26
- GET_NEXT_ASS_FEP 32
- GET_NEXT_ASS_NUC 29
- GET_NEXT_CARD 27
- GET_REF_AF 20
- GET_REF_AN 14
- GET_REF_CARD 3
- MAKE_CARD 9
- NUMBER_CARD 5
- PRINT_CARD 33
- PUT_FEP_CARD 25
- PUT_FULL_CARD 35
- PUT_NUC_CARD 24
- PUT_PROCENT_WELCOM 37
- RAP_AVL_CARD 36
- SEARCH_MIN_CARD_NUM 40
- SEARCH_MIN_CARD_REP 39

- SEARCH_MIN_CARD_WELC 41
- SUIV_AF 19
- SUIV_AN 13
- SUIV-CARD 2
- TRI_CARD 42
- UPD_AVL_CARD 23
- VIDE_AF 17
- VIDE_ASS_FEP_L 18
- VIDE_AN 11
- VIDE_ASS_NUC_L 12
- VIDE_CARD 1
- WELCOM_CARD 4

6.6.2.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.16.

6.6.3 Module EDGE_

6.6.3.1 Spécifications

Le module EDGE, qui est un module d'accès, est chargé de la gestion de la structure de données représentant les arêtes entre deux noyaux.

6.6.3.2 Structures de données globales gérées

Le module EDGE gère la structure de données "EDGE_L".

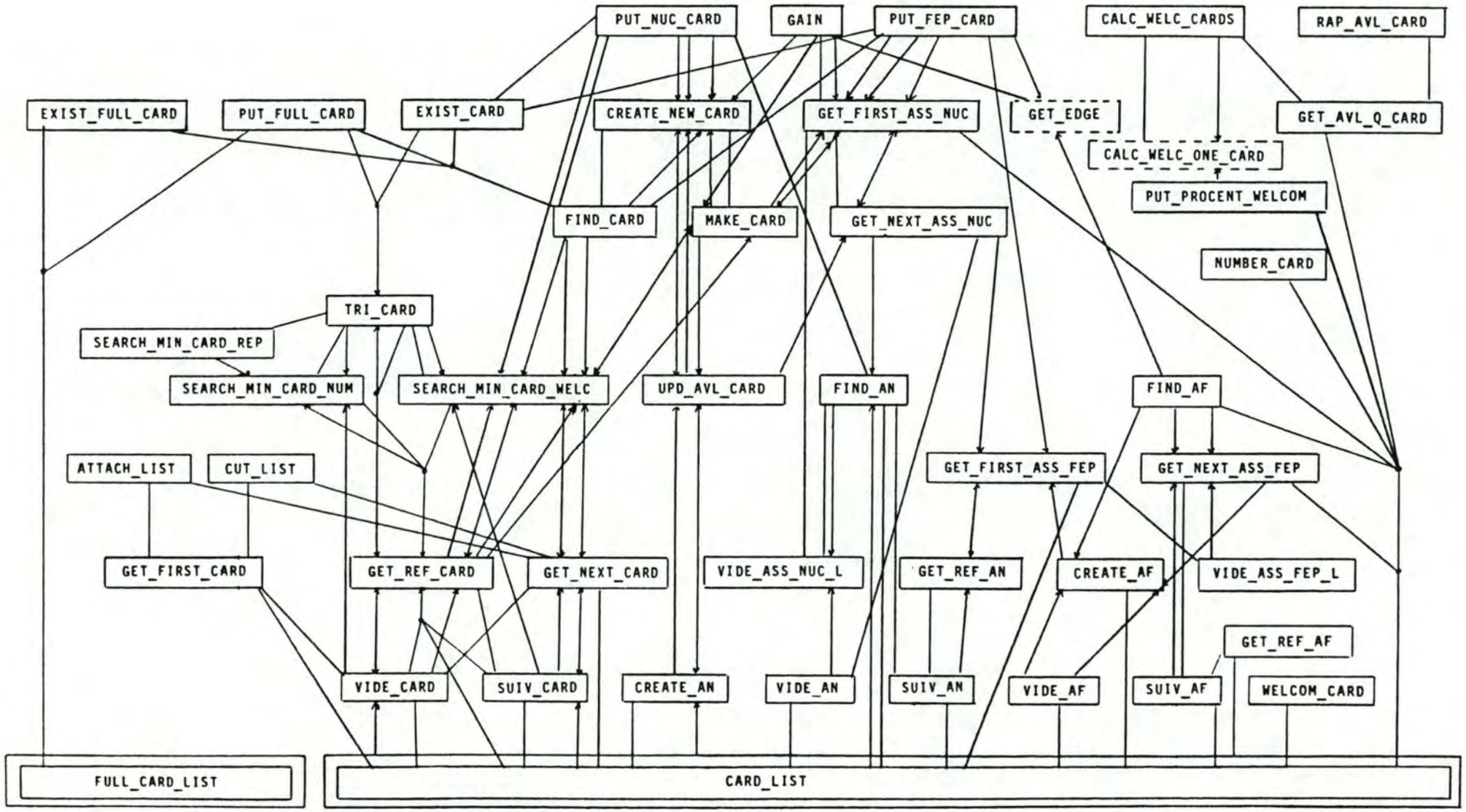
6.6.3.3 Particularités

Ce module permet aux autres modules d'accéder à la structure de données "EDGE_L" sans en connaitre la structure physique.

6.6.3.4 Liste des procédures et fonctions

- ADD_EDGE 14
- CALC_REL_WEIGH 27
- CREATE_ADJ 12
- CREATE_EDGE 15
- CREATE_FIRST 5
- DESTR_ADJ 13
- DESTR_EDGE 16
- DESTR_FIRST 6
- FIND_ADJ 11
- FIND_FIRST 4
- FUSI_EDGE 21
- GAIN_MEMORY 19
- GET_EDGE 18
- GET_EDGE_QUANT 17
- GET_FIRST_ADJ 23
- GET_FIRST_EDGE 25
- GET_NEXT_ADJ 22
- GET_NEXT_EDGE 26

Figure 7.16 : ARCHITECTURE DU MODULE CARD



- GET_REF_ADJ 10
- GET_REF_FIRST 3
- MODIFY_CEIL_EDGE 24
- MODIFY_REL_WEIGH 28
- PRINT_EDGE 20
- SUIV_ADJ 9
- SUIV_FIRST 2
- VIDE_ADJ 7
- VIDE_ADJ_L 8
- VIDE_FIRST 1

6.6.3.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique des procédures est montré à la figure 7.17.

6.6.4 Module EVALUATION

6.6.4.1 Spécifications

Le module EVALUATION a pour but de réaliser les fonctions d'évaluation décrites au point 3.6 du chapitre IV. Il s'agit des distributions et moyennes mémoire et CPU ainsi que des indicateurs relatifs.

6.6.4.2 Structures de données globales gérées

Le module évaluation ne gère aucune structure de données globale

6.6.4.3 Particularités

Aucune particularité n'est à remarquer pour ce module.

6.6.4.4 Liste des procédures et fonctions

- ABSOLUTE_INDICATORS 7
- CAL_AVL 6
- CPU_TOT 1
- DISTRIBUTION 8
- EVAL 9
- GET_USED_QTY_CARDS 2
- MAX_CARD 4
- MIN_CARD 5
- RELATIVE_INDICATORS 3

6.6.4.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.18.

6.6.5 Module FEP

Figure 7.17 : ARCHITECTURE DU MODULE EDGE

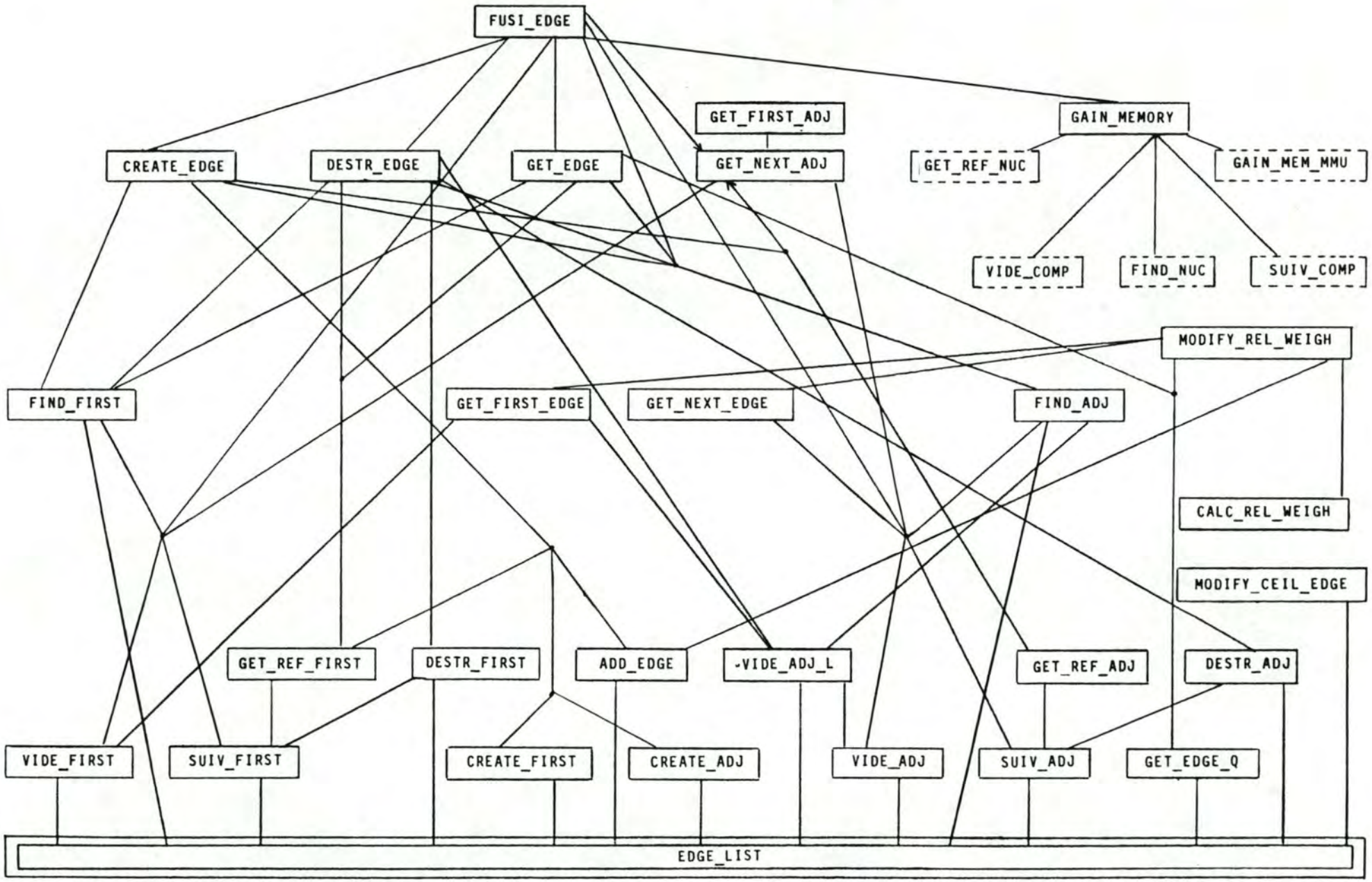
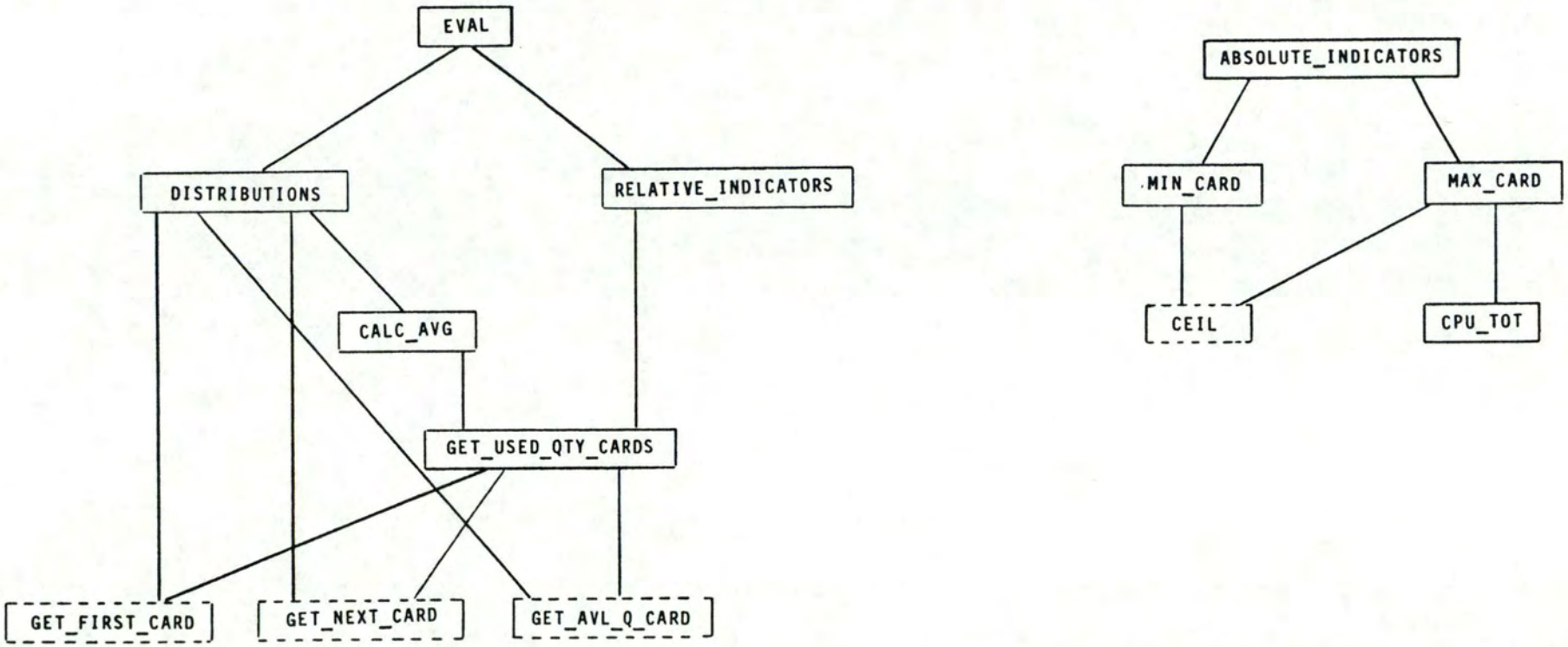


Figure 7.18 : ARCHITECTURE DU MODULE EVALUATION



6.6.5.1 Spécifications

Le module FEP, qui est un module d'accès, a pour fonction la gestion de la structure de données représentant les noyaux de type "FEP".

6.6.5.2 Structures de données globales gérées

Le module FEP gère la structure de données "FEP_L".

6.6.5.3 Particularités

Ce module permet aux autres modules d'accéder à la structure de données "FEP_L" sans en connaitre la structure physique.

6.6.5.4 Liste des procédures et fonctions

- ADD_F_GR	18
- CALC_NB_THR_GR	7
- CAL_CPU_FEP	11
- CAL_RAP_CONS_FEP	31
- CPU_THREAD	12
- CREATE_FEP	20
- CREATE_GR_CARD	25
- FIND_PLACE_FEP	4
- GET_BOUND_CARD	29
- GET_BOUND_GROUP	28
- GET_FIRST_CARD_NAMEFEP_NGR	32
- GET_FIRST_FEP	5
- GET_FIRST_F_GR	16
- GET_FIRST_GR_CARD	26
- GET_FIRST_GR_FEPNUM_CARDNUM	34
- GET_NEXT_CARD_NAMEFEP_NGR	33
- GET_NEXT_FEP	6
- GET_NEXT_F_GR	17
- GET_NEXT_GR_CARD	27
- GET_NEXT_GR_FEPNUM_CARDNUM	35
- GET_QHRAM	13
- GET_REF_FEP	3
- GET_REF_GR_CARD	24
- LOW_B	8
- MAKE_FEP	19
- NUMBER_FEP	9
- NUMBER_THR_FEP	10
- PRINT_FEP	30
- SUIV_FEP	2
- SUIV_F_GR	15
- SUIV_GR_CARD	23
- VIDE_FEP	1
- VIDE_F_GR	14
- VIDE_GR_CARD	21
- VIDE_GR_CARD_L	22

6.6.5.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.19.

6.6.6 Module FUSION

6.6.6.1 Spécifications

Le module FUSION a pour fonction de réaliser les fusions de noyaux à l'intérieur du graphe des noyaux. Cette fusion s'opère conformément à ce qui a été décrit dans la méthode de fusion du chapitre IV; pour rappel, nous fusionnons les noyaux en tenant compte des poids des arêtes adjacentes aux noyaux, de la taille limite courante d'un noyau, des incompatibilités entre noyaux et du numéro de carte CP éventuellement déjà défini des noyaux. Ce processus de fusion s'arrête lorsque il n'y a plus d'arêtes dans le graphe, ou s'il ne reste que des arêtes "impossibles temporaires".

6.6.6.2 Structures de données globales gérées

Le module FUSION ne gère pas de structures de données globales à SYANODE.

6.6.6.3 Particularités

Les deux structures de données propres au module FUSION sont

- TYPEDGE: un tableau de longueur variable avec un maximum de LMAXTYPEDGE caractères. Cette structure de données sert à décrire le type d'une arête. Les valeurs constantes possibles sont: "IMPOSSIBLE ABSOLUTE", "IMPOSSIBLE TEMPORARY", "POSSIBLE" conformément aux types d'arête défini dans la méthode de fusion. Nous prévoyons également un type d'arête "IMPOSSIBLE" signifiant que l'arête est impossible sur base de la taille limite courante du noyau;

- la procédure NUCL_GR_AG utilise une structure de données particulière lui permettant de se comporter de façon performante : un tableau de booléens indicé de 1 à MAXGR et initialisé à faux. Ce tableau est utilisé lors de la fusion des numéros des groupes dont le numéro de carte CP a été défini au préalable par l'utilisateur.

  Nous savons que la liste des groupes ayant ce paramètre déjà défini et son numéro de carte CP se trouvent dans un tableau appelé MAT2 de longueur constante LMAT2. Le jème élément du tableau de booléens est positionné à vrai lorsque le numéro de carte CP se trouvant dans le jème élément du tableau à deux dimensions MAT2 a déjà été traité; ce qui revient à dire que tous les groupes ayant ce numéro de carte CP ont déjà fait l'objet d'une fusion. Cette astuce permet, avec une petite structure de données booléenne, de réaliser la fusion des groupes avec numéro de carte CP défini en une seule passe sur le tableau MAT2.

Figure 7.19 : ARCHITECTURE DU MODULE FEP

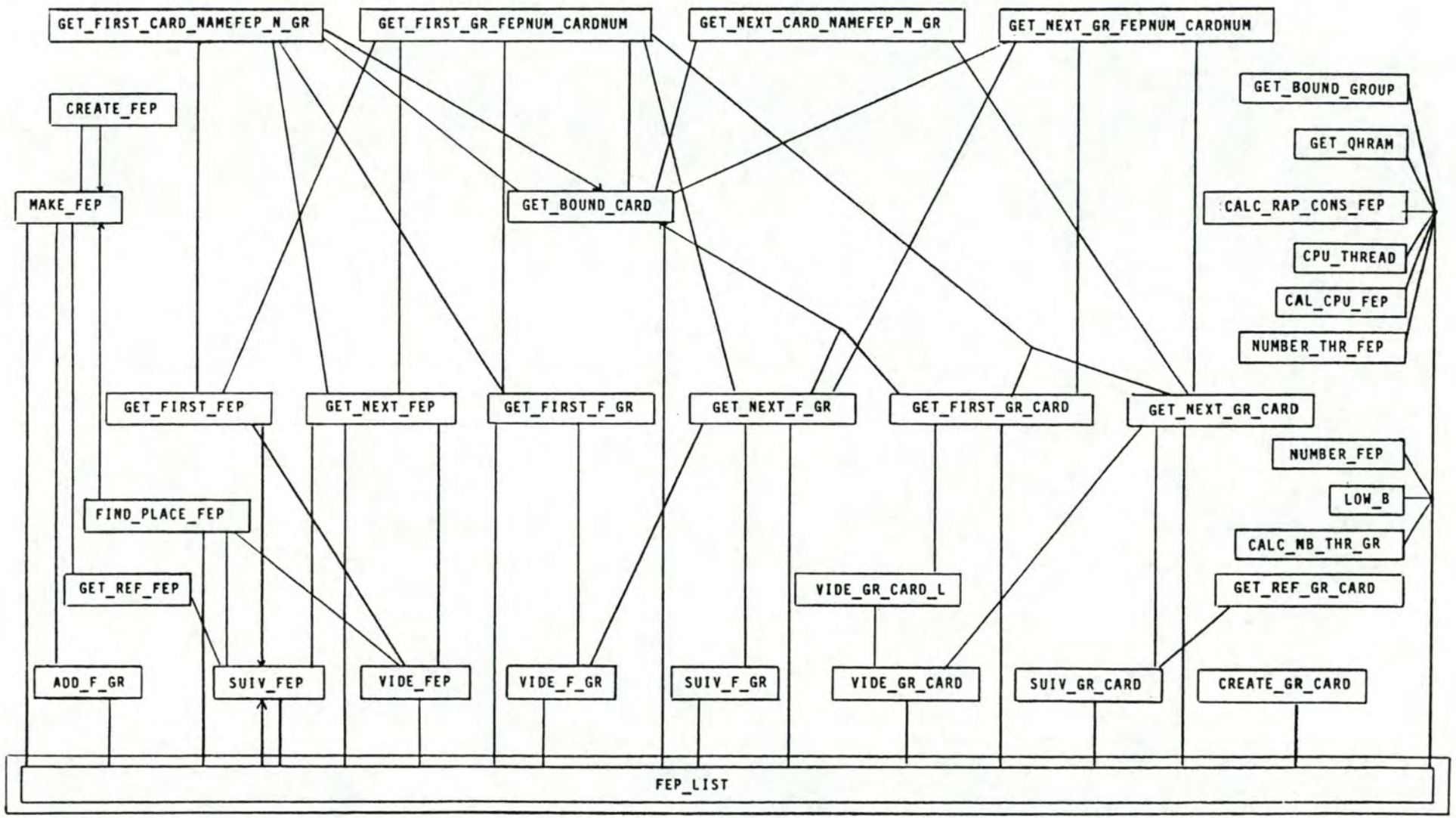
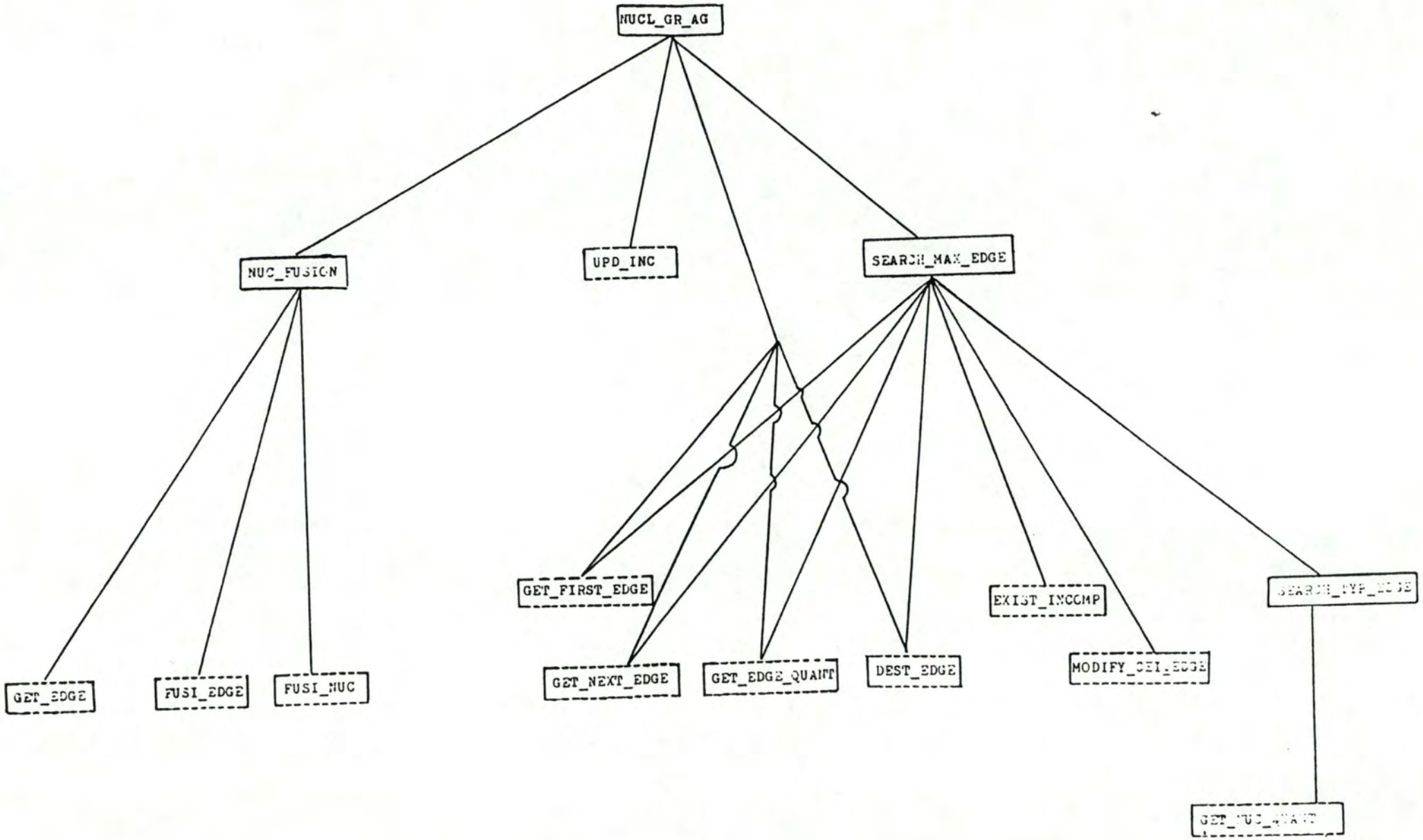


Figure 7.20 : ARCHITECTURE DU MODULE FUSION



6.6.6.4 Liste des procédures et fonctions

- NUC_FUSION 1
- NUC_GR_AG 4
- SEARCH_MAX_EDGE 3
- SEARCH_TYPE_EDGE 2

6.6.6.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.20.

6.6.7 Module GRAPH_CREATE

6.6.7.1 Spécifications

Le module GRAPH_CREATE a deux fonctions principales : celle de traduire le fichier TEXT des groupes d'entrée en un fichier de records GROUP correct d'une part et de construire le graphe des noyaux d'autre part.

6.6.7.2 Structures de données globales gérées

Le module GRAPH_CREATE crée le fichier de records global des groupes GRP.LIST ainsi que la table globale MAT2. Cette dernière est constituée d'éléments ayant deux parties : pour chaque groupe ayant à l'entrée son numéro de carte CP défini, son numéro de groupe et le numéro de carte CP. La longueur de ce tableau est LMAT2.

GRAPH_CREATE construit un fichier global d'erreur ERROR_LIST quand cela est nécessaire. Les différents diagnostics d'erreurs sont repris dans le manuel d'utilisateur; un diagnostic d'erreur (ER_DIA) est une chaine de caractères de longueur variable avec une borne supérieure constante ERTPL.

6.6.7.3 Particularités

Le module GRAPH_CREATE utilise deux structures de données particulières :

- ACCKEY (voir figure 7.21); il s'agit d'un record correspondant à la clé d'accès sur la GROUP_LIST. L'ACCKEY comprend deux champs: un nom de groupe (GR_N de type GROUP_NAME) et la borne inférieure du groupe (LN_G de type integer);

GR_N	LN_G

Figure 7.21 : structure d'une ACCKEY

- NODE_IDENT (voir figure 7.22): il s'agit de la description générale du noeud SOPHO-NET sur lequel s'applique SYANODE. Cette description est reprise dans un record constitué des champs

suivants : NETWORK (de type NET_NAME) contient le nom du réseau auquel appartient le noeud à configurer, NODE (de type NODE_N) est l'identification du noeud à configurer, VERS (de type integer) le numéro de version et DATE (de type DATE_N), une date de référence. Cette description est trouvée dans le fichier TEXT de nom "INPUT.GRP".

NETWORK
NODE
VERS
DATE

Figure 7.22 : structure d'un NODE_IDENT

6.6.7.4 Liste des procédures et fonctions

- CALC_MEM_CPU_GR 6
- CALC_MEM_FEP 12
- CALC_MEM_MMU_GR 5
- CALC_WELC_ONE_CARD 14
- CEIL F2
- CORRECT_MEM_MIN 9
- CREATE_GRAPH 13
- DETECT_EXIST 4
- DETECT_OVF 7
- FAG F1
- FIRST_PASS 10
- GET_CP_CAR 8
- INTO F3
- RD_GR 1
- REC_ERR 2
- TEST_FOR 3
- TEST_NEI 11

6.6.7.5 Schéma hiérarchique de type "utilise"

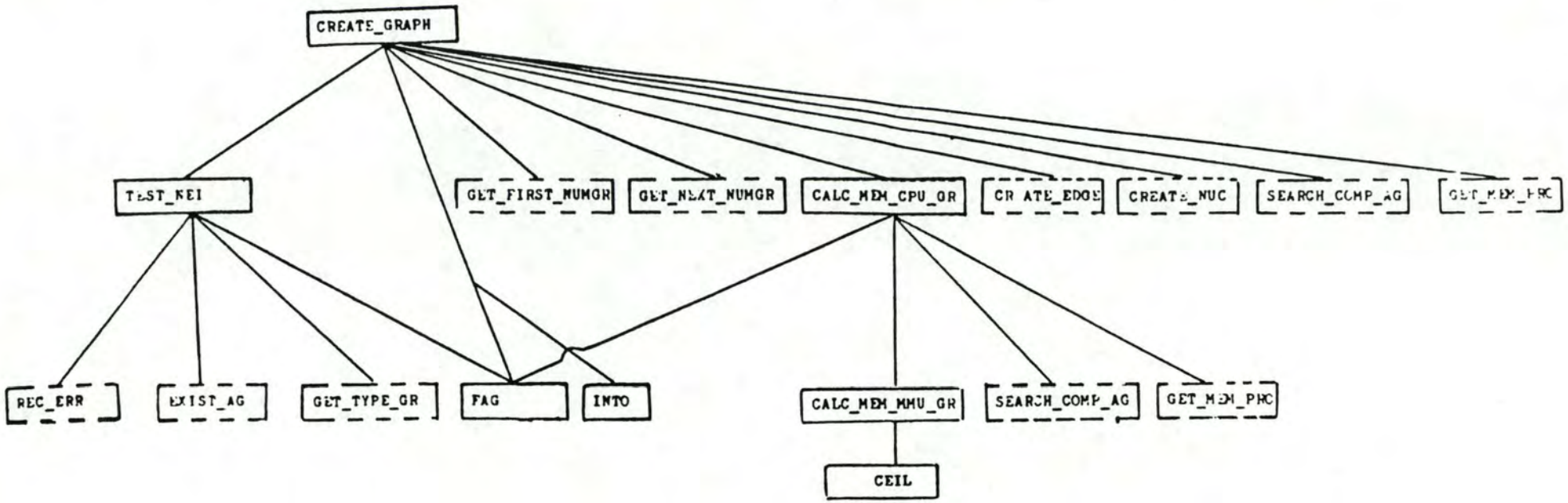
Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.23.

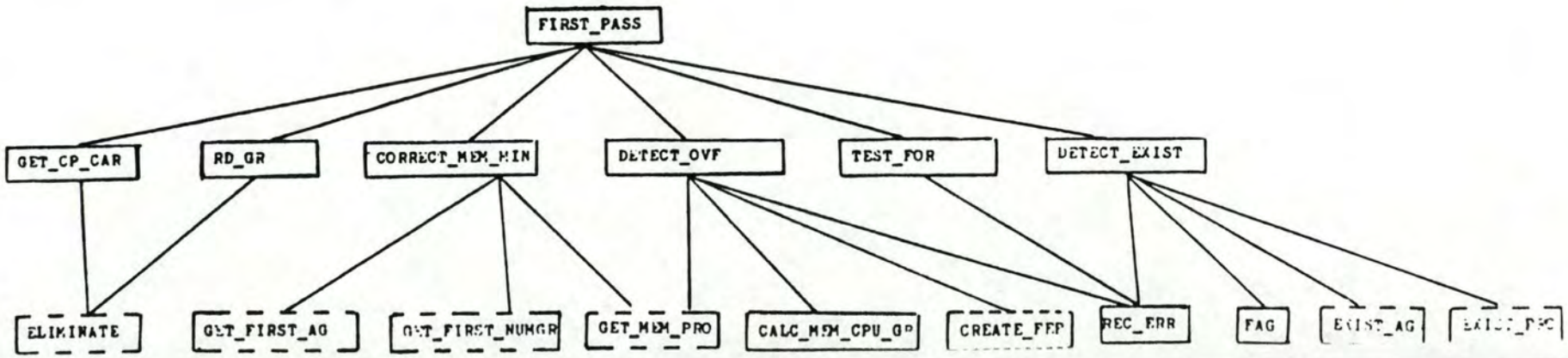
6.6.8 Module MOD_AG

6.6.8.1 Spécifications

Le module MOD_AG rassemble les procédures d'accès à la table des agrégats. Il est donc la mise en oeuvre d'un module d'accès à cette table. Il permet l'accès séquentiel aux agrégats, la détection de l'existence d'un agrégat et la recherche de sa composition.

Figure 7.23: ARCHITECTURE DU MODULE GRAPH_CREATE





6.6.8.2 Structures de données globales gérées

Le module MOD_AG gère la table globale des agrégats AGR_LIST.

6.6.8.3 Particularités

Aucune particularité n'est à remarquer pour le module MOD_AG.

6.6.8.4 Liste des procédures et fonctions

- EXIST_AG F1
- GET_FIRST_AG 2
- GET_NEXT_AG 3
- SEARCH_COMP_AG 1

6.6.8.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.24.

6.6.9 Module MOD_GR

6.6.9.1 Spécifications

Le module MOD_GR rassemble les fonctions et procédures d'accès au fichier de records des groupes. Il est donc la mise en oeuvre d'un module d'accès à ce fichier. Il permet la détection de l'existence d'un groupe dans ce fichier, la recherche du numéro d'un groupe de nom et borne inférieure donnés, l'accès au premier et au suivant des groupes de nom donné, le calcul des pourcentages d'accueil des "FEP" sur une carte CP et la recherche des caractéristiques mémoire et CPU d'un groupe de type "FEP" de numéro donné.

6.6.9.2 Structures de données globales gérées

Le module MOD_GR gère le fichier global des records groupes GRP.LIST.

6.6.9.3 Particularités

Aucune particularité n'est à remarquer pour le module MOD_GR.

6.6.9.4 Liste des procédures et fonctions

- EXIST_GR F1
- GET_FEP_ATT 4
- GET_FIRST_NUMGR 1
- GET_NEXT_NUMGR 2
- NUM_REC_GR F2

6.6.9.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.25.

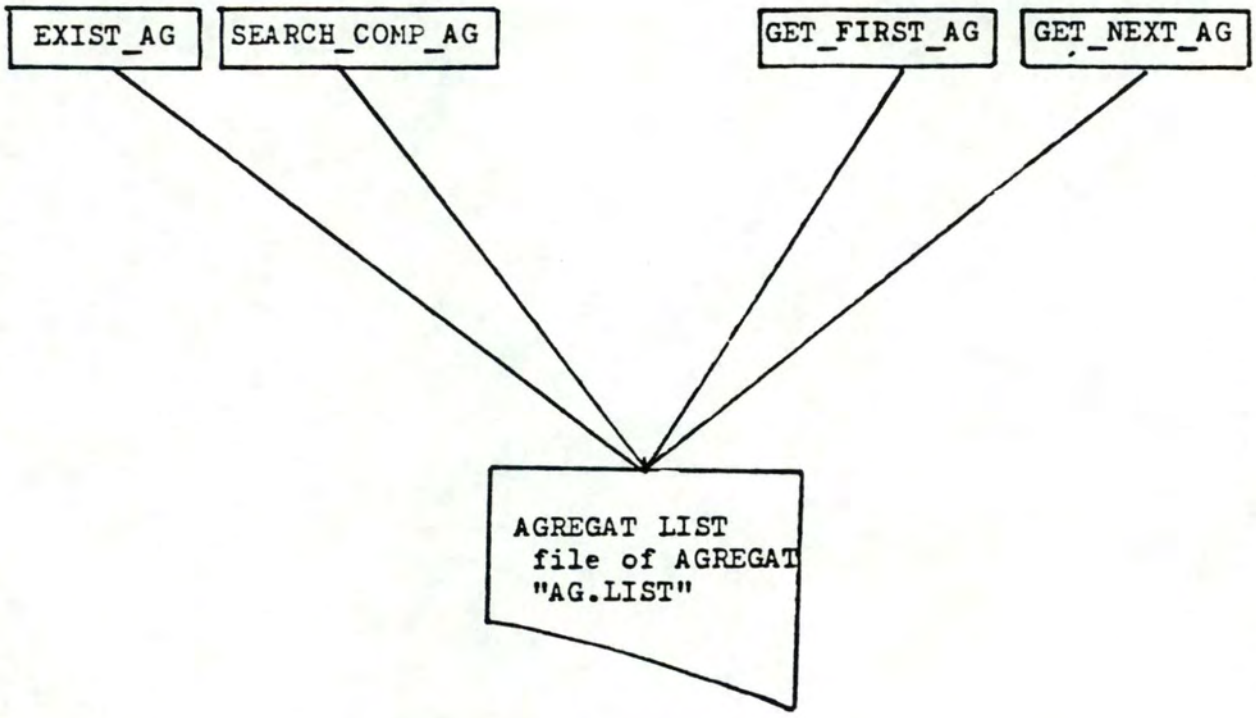


Figure 7.24 : ARCHITECTURE DU MODULE MOD_AG

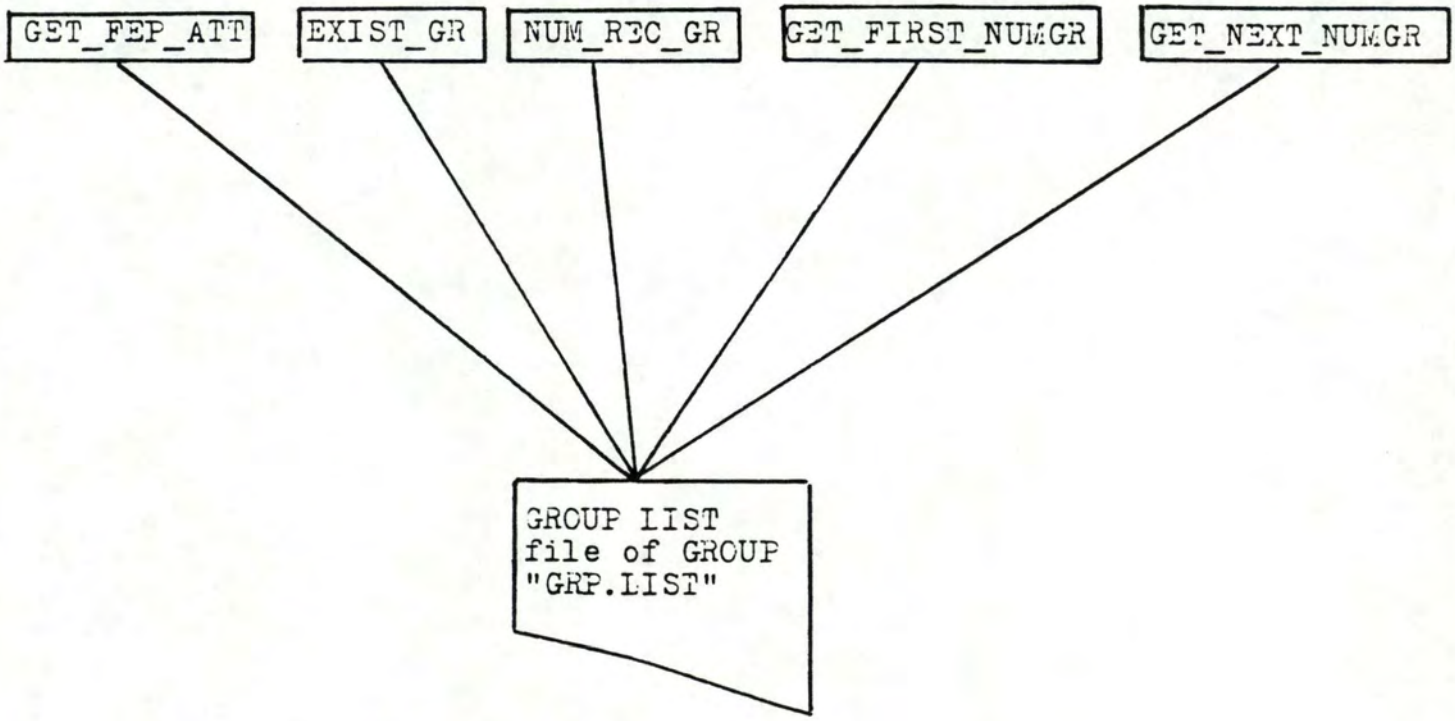


Figure 7.25 : ARCHITECTURE DU MODULE MOD_GR

6.6.10 Module MOD_INC

6.6.10.1 Spécifications

Le module MOD_INC offre les procédures d'accès séquentiel sur le fichier de records des incompatibilités (accès à la première et à la suivante des incompatibilité); il représente la mise en oeuvre d'un type abstrait.

6.6.10.2 Structures de données globales gérées

Le module MOD_INC gère le fichier global INCOMP.LIST.

6.6.10.3 Particularités

Aucune particularité n'est à remarquer pour le module MOD_INC.

6.6.10.4 Liste des procédures et fonctions

- GET_FIRST_INC 1
- GET_NEXT_INC 2

6.6.10.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.26.

6.6.11 Module MOD_PR

6.6.11.1 Spécifications

Le module MOD_PR rassemble les procédures d'accès au fichier de records des processus. Il est donc la mise en oeuvre d'un module d'accès à ce fichier. Il permet la détection de l'existence d'un nom de processus dans ce fichier, la recherche du numéro d'un processus de nom donné, la recherche des gains à placer des processus homonymes sur une même carte CP, la recherche des consommations mémoire globale d'un processus, la recherche du type d'un processus, la recherche des caractéristiques mémoire d'un processus de type "FEP".

6.6.11.2 Structures de données globales gérées

Le module MOD_PR gère le fichier global des processus PNAM.LIST.

6.6.11.3 Particularités

Aucune particularité n'est à remarquer pour le module MOD_PR.

6.6.11.4 Liste des procédures et fonctions

- EXIST_PRO F1
- GAIN_MEM_MMU 2
- GET_FEP_CAR 5
- GET_MEM_PRO 3
- GET_NUM_PR 1

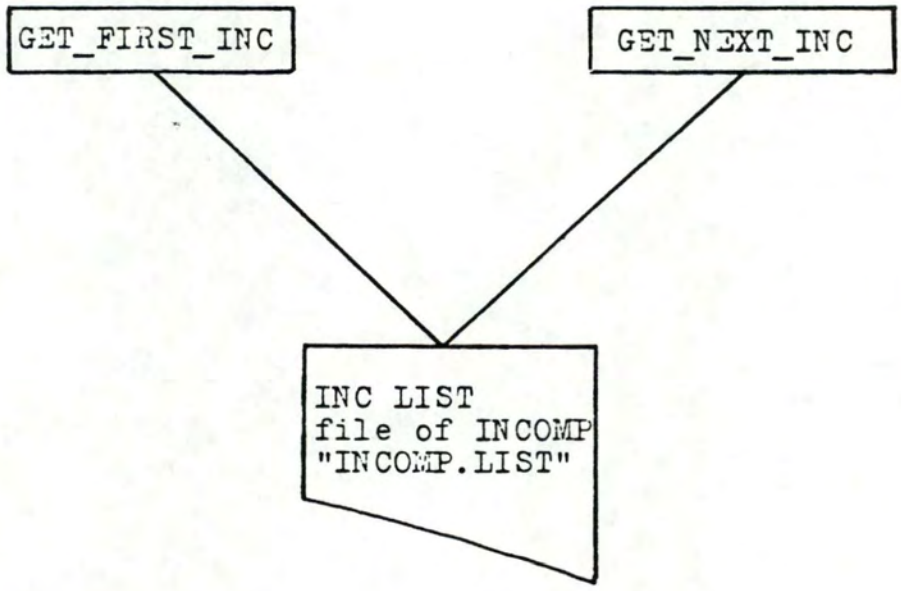


Figure 7.26 : ARCHITECTURE DU MODULE MOD_INC

- GET_TYPE_GR 4

6.6.11.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.27.

6.6.12 Module MOD_TRANS

6.6.12.1 Spécifications

Le module MOD_TRANS a pour fonction de traduire les fichiers TEXT de l'utilisateur (fichiers des processus, incompatibilités et agrégats) sous une forme de fichiers de records (pour les deux premiers) ou sous forme d'une table (pour les agrégats).

6.6.12.2 Structures de données globales gérées

Le module MOD_TRANS crée les fichiers globaux PNAM.LIST, INCOMP.LIST et la table globale AGR_LIST.

6.6.12.3 Particularités

Aucune particularité n'est à remarquer pour le module MOD_TRANS.

6.6.12.4 Liste des procédures et fonctions

- ELIMINATE 1
- TRANS_AG_LIST 2
- TRANS_INC_LIST 4
- TRANS_PR_LIST 3

6.6.12.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.28.

6.6.13 Module NUC

6.6.13.1 Spécifications

Le module NUC, qui est un module d'accès, a comme fonction la création et la gestion de la structure de données représentant les noyaux; il s'agit de la "NUC_L".

6.6.13.2 Particularités

Ce module permet aux autres modules d'accéder à la structure de données "NUC_L" sans en connaitre la structure physique.

6.6.13.3 Structures de données globales gérées

Le module NUC gère la structure de données "NUC_L".

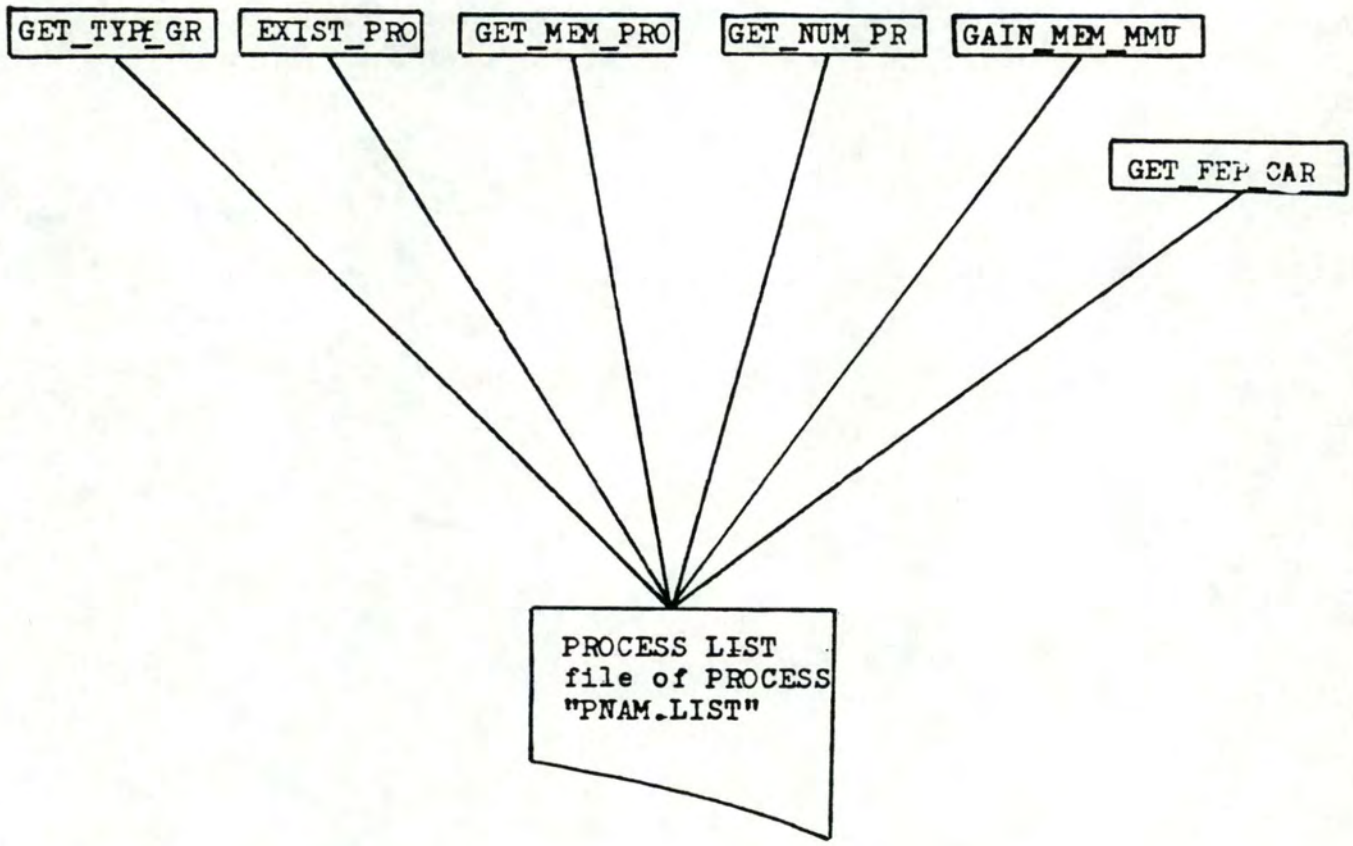


Figure 7.27 : ARCHITECTURE DU MODULE MOD_PR

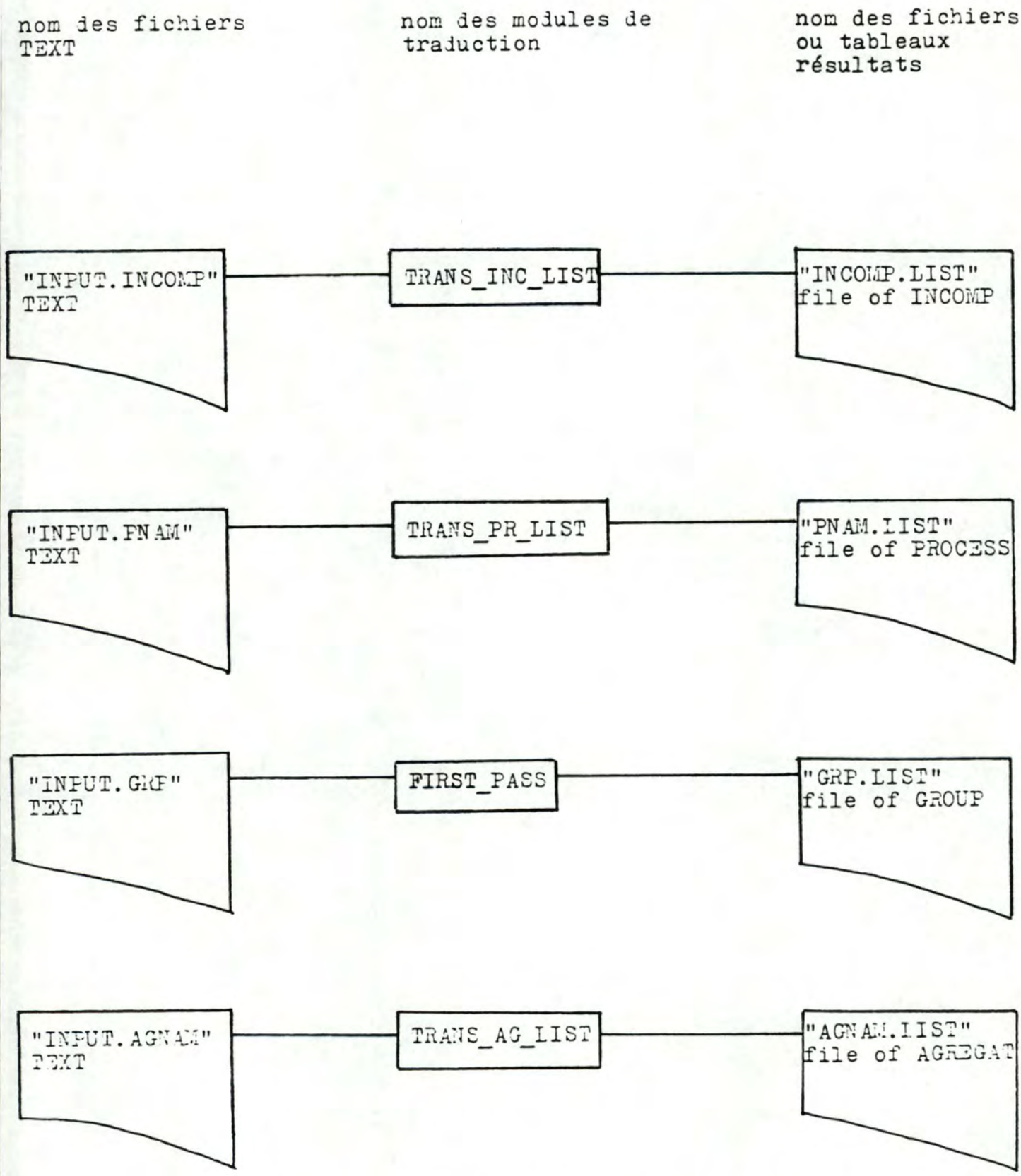


Figure 7.28 : ARCHITECTURE DU MODULE MOD_TRANS

6.6.13.4 Liste des procédures et fonctions

- CREATE_NUC 2
- DESTR_NUC 17
- DESTROY_NUC 7
- EXIST_CONSTR_NUC 20
- FEP_NUC 22
- FIND_NUC 6
- FUSI_COMP 13
- FUSI_NUC 16
- GET_FIRST_GRCOMP 10
- GET_FIRST_NUC 4
- GET_FIRST_PRCOMP 11
- GET_NEXT_COMP 12
- GET_NEXT_NUC 5
- GET_NUC_QUANT 18
- GET_REF_NUC 14
- PREPARE_FUSI_PR_COMP 15
- PRINT_NUC 21
- SET_CONSTR_NUC 19
- TRANSFORM_NUC_FEP 23
- SUIV_COMP 9
- SUIV_NUC 3
- VIDE_COMP 8
- VIDE_NUC 1

6.6.13.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.29.

6.6.14 Module PLACEMENT

6.6.14.1 Spécifications

Le module PLACEMENT réalise le placement des noyaux de type "NON FEP" sur les cartes CP après avoir trié ces noyaux de type "NON FEP" selon trois critères. Les trois méthodes de tri retenues actuellement sont :

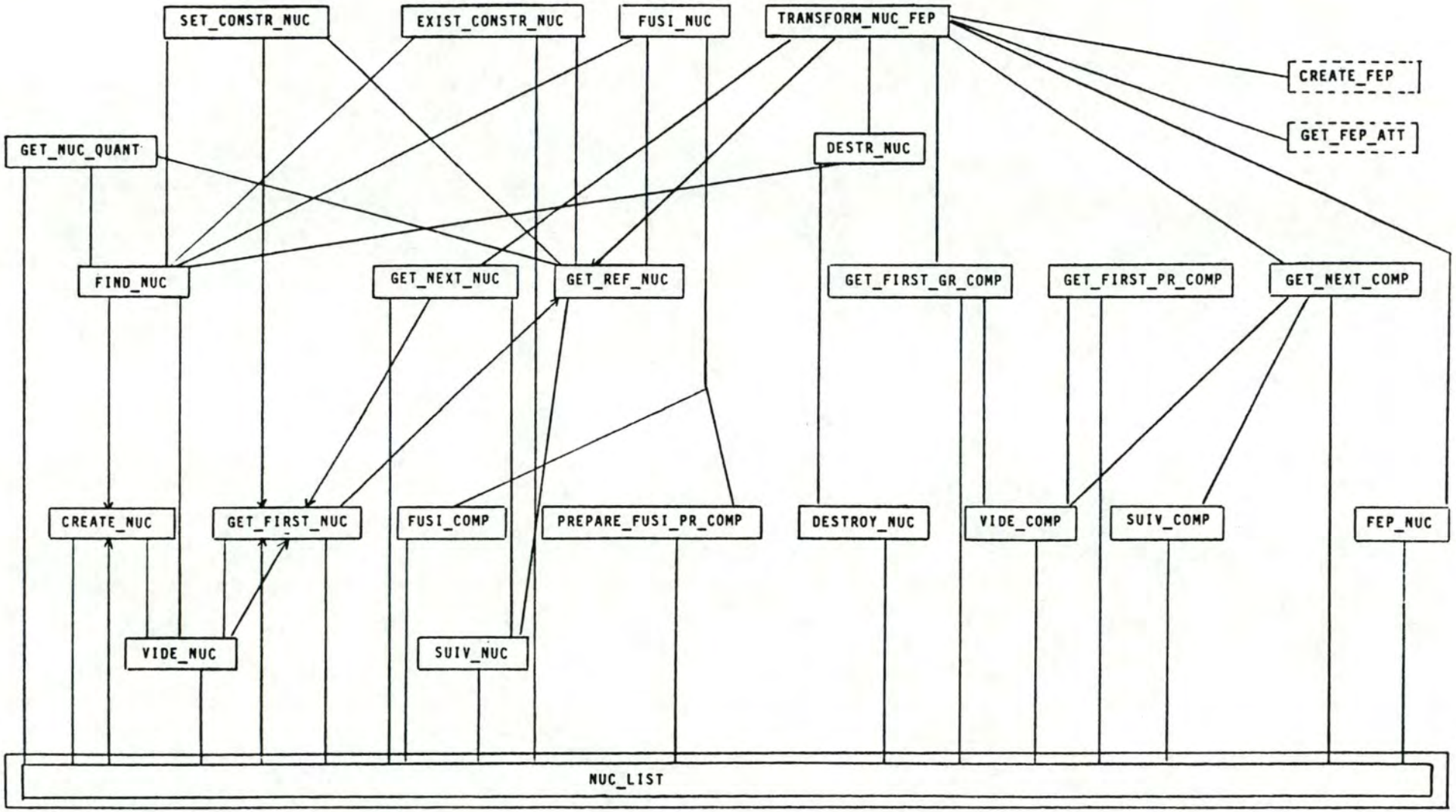
- ordre croissant selon la différence en valeur absolue entre la consommation mémoire et CPU des noyaux de type "NON FEP";

- ordre décroissant selon la différence en valeur absolue entre la consommation mémoire et CPU des noyaux de type "NON FEP";

- ordre alternif selon la différence en valeur absolue en favorisant la taille CPU puis la taille mémoire des noyaux de type "NON FEP".

A l'issue du placement des noyaux de type "NON FEP", SYANODE retient celui ayant le plus petit nombre de cartes CP utilisées et à nombre de cartes CP égal, il retient celui ayant les distributions

Figure 7.29 : ARCHITECTURE DU MODULE NUC



mémoire et CPU les plus faibles.

6.6.14.2 Structures de données globales gérées

Le module PLACEMENT ne gère pas de structures de données globales à SYANODE.

6.6.14.3 Particularités

Le module PLACEMENT utilise trois structures de données propres : des tableaux globaux de nom GLO_AR_PLGR, NUCL_AR et SCARD_L.

Structure GLO_AR_PLGR

Il s'agit d'un tableau d'entiers dont le ième élément contient le numéro de carte CP du groupe numéro i, si ce groupe n'est pas de type "FEP"; il contient la constante UNDEFINED dans le cas contraire. Cette structure est construite à partir de la CARD_LIST et évite lors de la génération des résultats d'accéder au fichier des groupes.

Structure NUCL_AR

La structure NUCL_AR est un tableau indicé de 1 à MAXNUC de NUC_CAR où NUC_CAR est une structure de données de la forme suivante :

NUMNUC
NUMCARD
CPU
MMU
MEM
DIF
NPNR

Figure 7.30 : structure d'un NUC_CAR

La figure 7.30 illustre la structure d'un NUC_CAR dans laquelle nous distinguons les champs suivants :

- NUMNUC : le numéro du noyau;
- NUMCARD : le numéro du noyau, la constante UNDEFINED en général;
- CPU : la consommation CPU totale du noyau;
- MMU : la consommation en nombre de "MMU bank" du noyau;

- MEM : la consommation mémoire totale du noyau;

- DIF : la différence entre la consommation mémoire du noyau et sa consommation CPU multipliée par le coefficient de rapport CPU-MEMOIRE;

- NPNR : la consommation en PNR du noyau.

Le tableau de type NUC_CAR est construit lors de l'entrée dans le module PLACEMENT à partir des noyaux de la NUC_LIST. Toutes les procédures du module placement travailleront donc sur ce tableau.

Il est possible de justifier l'usage d'un tableau plutôt que d'une structure dynamique des noyaux pour les raisons suivantes :

- la rapidité dans les tris dont le nombre actuel est trois mais qui pourrait évoluer;

- l'argument de rapidité joue aussi dans le sens où le nombre d'éléments à trier peut devenir important. La complexité du tri est donc multipliée; une structure dynamique provoquerait un dégradation des performances;

- il est inutile de recréer une structure dynamique alors que la NUC_LIST en est déjà une. D'autre part, le module PLACEMENT ne peut utiliser la NUC_LIST qui ne contient pas toutes les informations nécessaires au placement; celui-ci recrée donc une structure de données statique en fonction de ses besoins;

- la taille d'une structure NUC_CAR est limitée. Le gaspillage dans la réservation de place pour le tableau est donc relativement faible même lorque le nombre de noyaux à trier est petit.

Structure SCARD_L

La structure SCARD_L est un tableau d'éléments de type SCARD indicé de 1 à MAXCARDNB et servant au placement des noyaux sur les cartes CP. Une SCARD a la forme suivante :

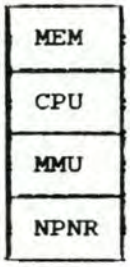


Figure 7.31 : structure d'une SCARD

La figure 7.31 nous permet de distinguer les champs suivants d'une SCARD :

- MEM : contient la quantité mémoire disponible sur la carte CP;
- CPU : contient la quantité CPU disponible sur la carte CP;
- MMU : contient le nombre de "MMU banks" disponibles sur la carte CP;
- NPNR : contient le nombre de PNR disponibles sur la carte CP.

Le tableau de type SCARD_L sert à comptabiliser les disponibilités sur les cartes CP au cours de la phase de placement des noyaux.

Remarquons qu'ici non plus, nous n'avons pas utilisé de structure dynamique pour représenter les cartes CP; cela tient essentiellement au fait que la structure de carte SCARD utilisée est de taille très réduite et qu'un gaspillage de place mémoire serait donc négligeable lors de l'usage d'un tableau.

6.6.14.4 Liste des procédures et fonctions

- BETTER_NUC_PLAC F1
- CHANGE 3
- DIFFERENCE 5
- INVERT 4
- LOAD_NUC_AR 1
- NUC_PLACE 7
- PLNUC 8
- PL_NUC 2
- SORT_NUC 6

6.6.14.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.32.

6.6.15 Module PLACE_FEP

6.6.15.1 Spécifications

Le module PLACE_FEP a pour fonction de placer les noyaux de type "FEP" sur des cartes, selon la méthode définie au point 3.5 du chapitre IV.

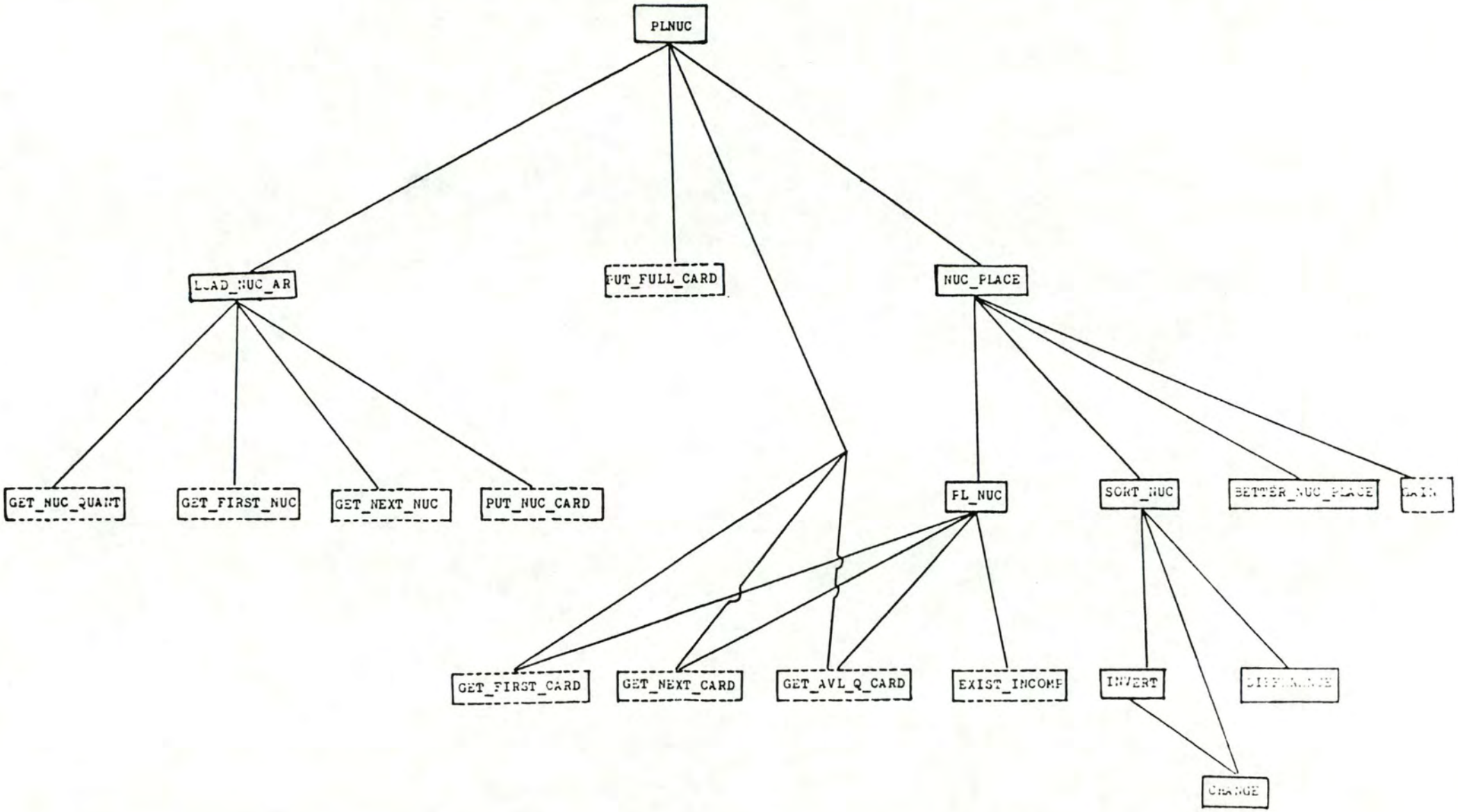
6.6.15.2 Structures de données globales gérées

Le module PLACE_FEP ne gère aucune structure de données globale.

6.6.15.3 Particularités du module

Ce module utilise un type de structure de données qui lui est propre : "TAB_P_CARD" qui est un tableau de références aux cartes CP. Ce type de tableau est utile dans différents cas. Premièrement, dans le but de mémoriser les références des cartes CP sur lesquelles un "FEP" de même nom a été placé. Deuxièmement, pour rechercher les cartes d'accueil d'un noyau de type "FEP".

Figure 7.32 : ARCHITECTURE DU MODULE PLACEMENT



6.6.15.4 Liste des procédures et fonctions

- CALC_WELC 3
- CALCUL_RAP_AVL_CARDS 1
- CHANGE 4
- CREATE_WELC_CARD 7
- MAKE_FIRST_PREC_L 12
- MAJ_PREC_L 13
- MAKE_TAB 5
- PLACE_FEPS 16
- PLACER_FEP 10
- PLACER_NEW_CARDS 15
- PUT_PREC_L 14
- SEARCH_NB_PART 9
- SEE_ADD_WELC_CARDS 8
- SELECT_CARD 6
- SORT_TAB 11
- TAKE_NBRE_CARD 2

6.6.15.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.33.

6.6.16 Module RESULT

6.6.16.1 Spécifications

Le module RESULT a pour fonction d'écrire dans les fichiers TEXT de noms "OUTPUT.CRD", "OUTPUT.GRP" et "OUTPUT.EXC" les résultats du meilleur placement des groupes sur les cartes CP obtenu par SYANODE. Nous distinguons trois types de résultats :

- la liste des groupes placés sur chaque carte CP (dans "OUTPUT.CRD");
- la liste des groupes comme à l'entrée avec spécification du numéro de carte CP (dans "OUTPUT.GRP");
- les échanges CPU entre chaque paire de cartes CP (dans "OUTPUT.EXC").

6.6.16.2 Structures de données globales gérées

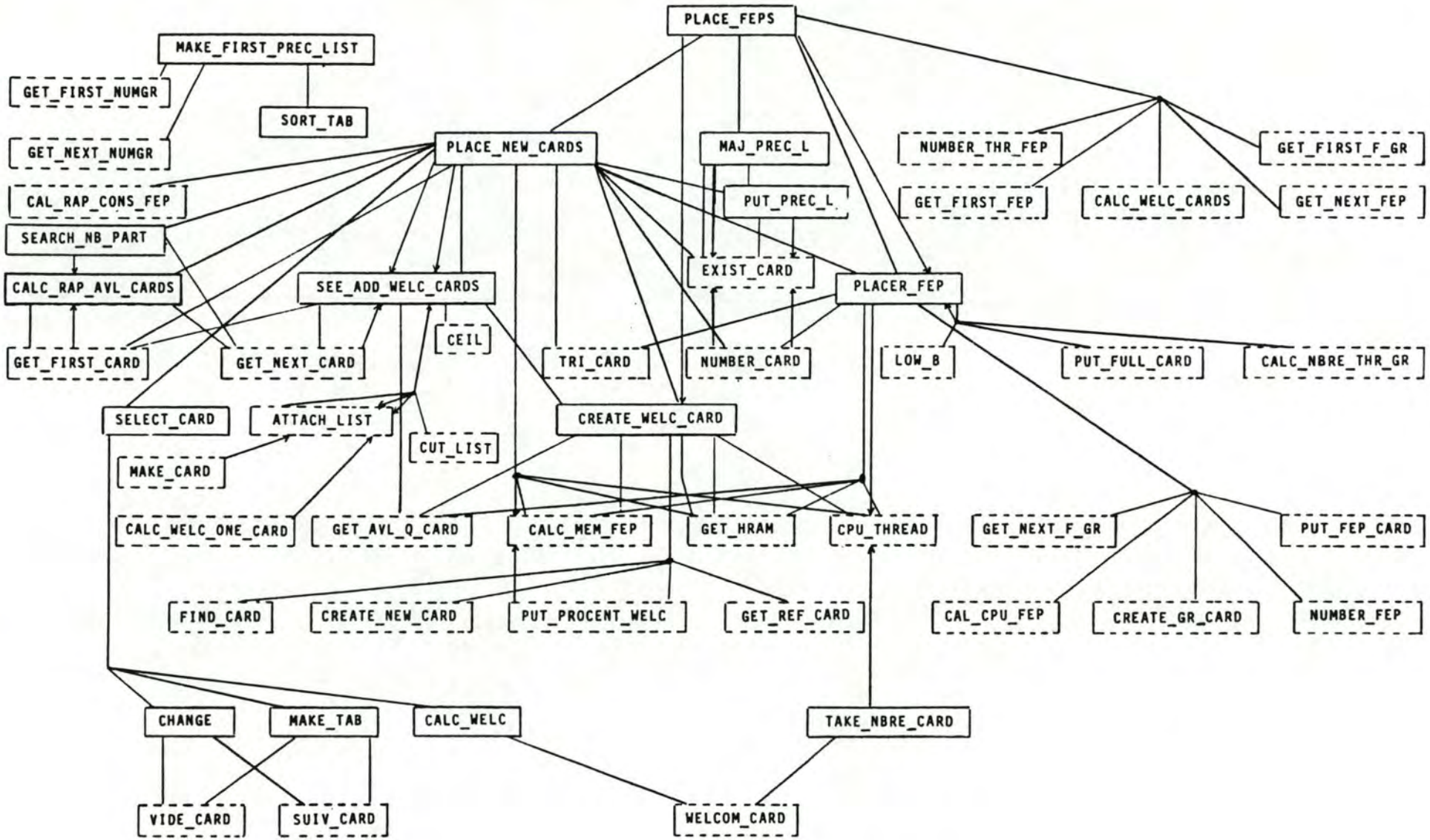
Le module RESULT ne gère pas de structures de données à SYANODE.

6.6.16.3 Particularités

Le module RESULT utilise une seule structure de données propre :

- La structure matricielle M est une matrice de dimension MAXGR X MAXGR et dont chaque élément est un réel représentant le gain CPU réalisé entre les groupes dont les numéros sont ceux de la ligne

Figure 7.33 : ARCHITECTURE DU MODULE PLACE_FEP



et la colonne de la matrice. De cette matrice, nous n'avons utilisé que la partie supérieure (indice de ligne supérieur à l'indice de colonne); il n'y a aucun intérêt à distinguer un sens de gain CPU.

6.6.16.4 Liste des procédures et fonctions

- PRINT_NODE 1
- RES_OUT_CD 3
- RES_OUT_GR 2

6.6.16.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.34.

6.6.17 Module SYANODE

6.6.17.1 Spécifications

Le module SYANODE est le programme principal du programme de même nom. Il a pour fonction de réaliser les opérations d'entrées sorties avec l'utilisateur notamment pour connaitre le coefficient de rapport CPU-MEMOIRE et pour savoir s'il faut traduire les fichiers TEXT indépendants de la configuration. C'est à partir de ce module que les autres modules sont appelés. La description des messages d'entrées sorties et des réponses possibles de l'utilisateur se trouvent dans le manuel d'utilisateur en annexe VI.

6.6.17.2 Structures de données globales gérées

Le module SYANODE ne gère pas de structures globales à SYANODE.

6.6.17.3 Particularités

Le module SYANODE n'utilise aucune structure de données propre.

6.6.17.4 Liste des procédures et fonctions

- BETTER_PL F2
- CALC_RAP_CONS 3
- CH_INC F3
- CH_NUC F4
- CH_PR F1
- EVAL_NUC_SIZE 1
- GET_RAP_CONS 2
- INIT 4

6.6.17.5 Schéma hiérarchique de type "utilise"

Un schéma hiérarchique de type "utilise" des procédures est montré à la figure 7.35.

Figure 7.34 : ARCHITECTURE DU MODULE RESULT

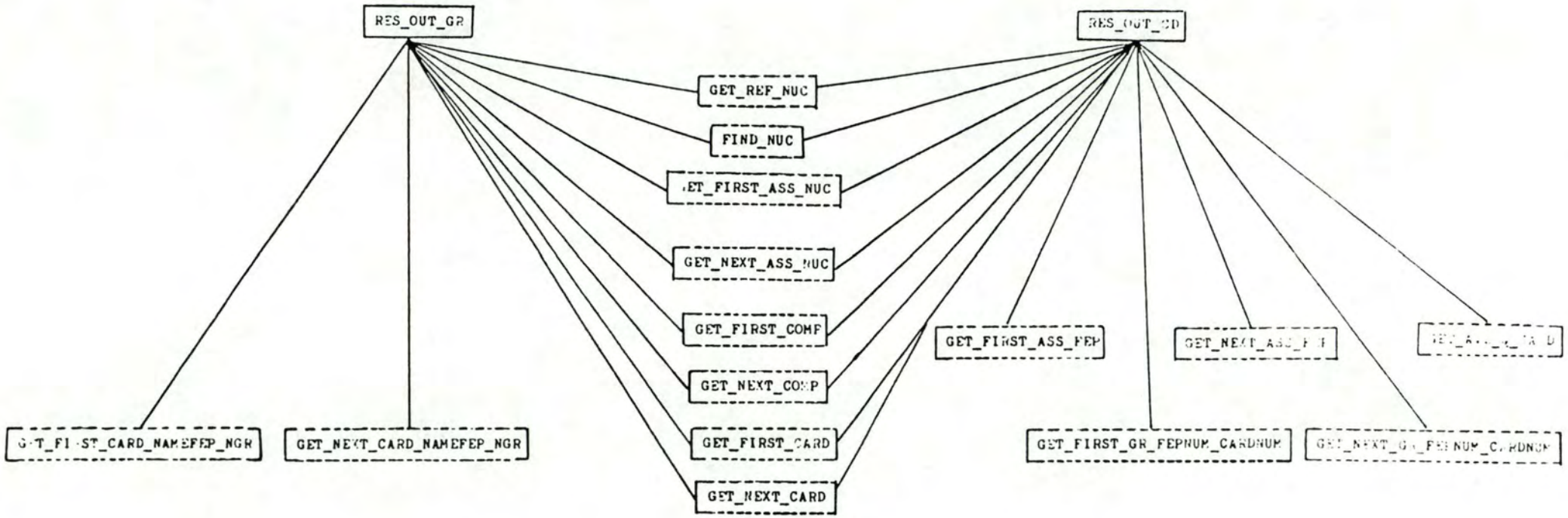
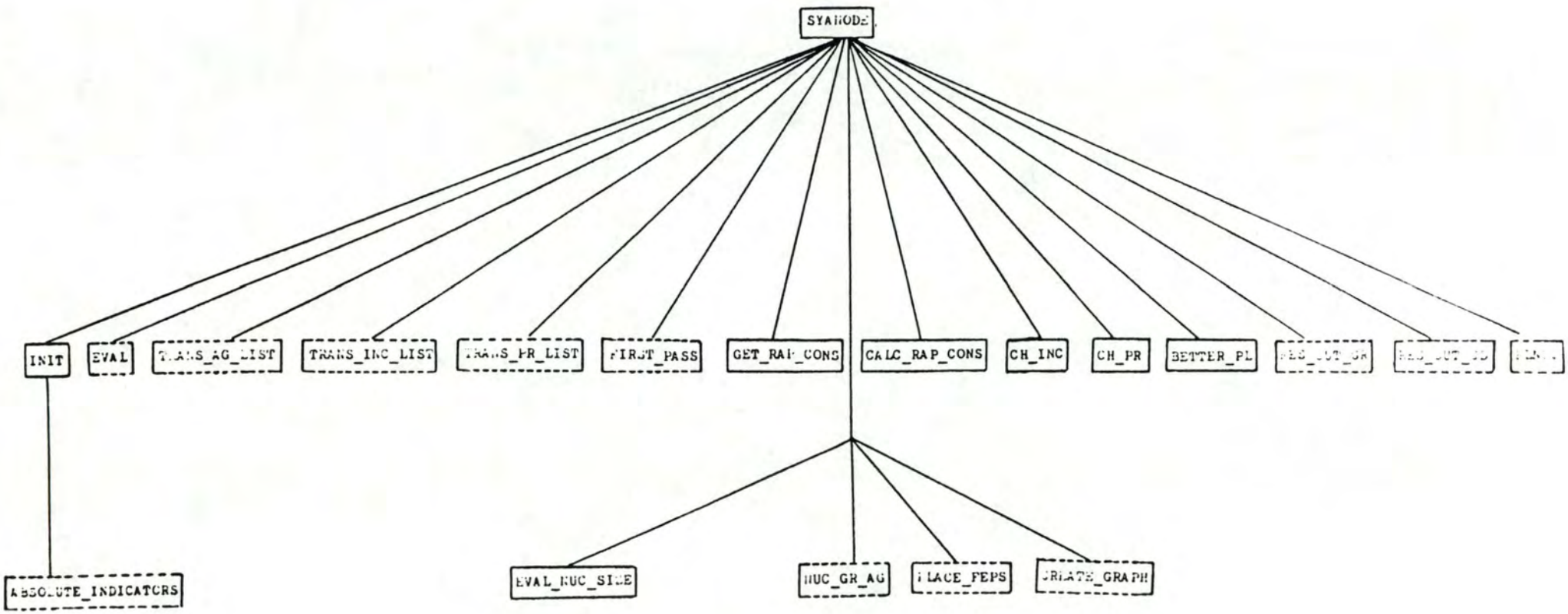


Figure 7.35 : ARCHITECTURE DU MODULE SYANODE



CONCLUSION GENERALE

Pour conclure ce mémoire, nous en présentons une brève synthèse. Ensuite, nous en montrons les limites et nous proposons quelques extensions possibles.

1 Synthèse

Nous avons décrit de manière générale les structures matérielle et logicielle d'un réseau SOPHO-NET. Ensuite, nous avons présenté les étapes nécessaires à la conception d'un noeud. Ceci nous a permis de préciser les lacunes de l'étape de la réalisation d'un fichier de chargement et de proposer un projet de conception assistée par ordinateur pour y remédier. Nous avons alors élaboré la méthode de fusion et montré en quoi elle est la prolongation d'autres méthodes. Pour terminer, nous avons comparé le problème qui nous était proposé à des problèmes similaires traités dans la littérature.

2 Limites

2.1 Complexité algorithmique de la méthode de fusion

Nous avons calculé au pire la complexité algorithmique de la méthode de fusion. Une analyse du nombre moyen d'arêtes de poids normalisé entre deux noyaux s'impose pour pouvoir la déterminer avec plus de justesse.

2.2 Utilisation du langage Pascal

Philips nous proposait d'utiliser soit le langage Pascal, soit le langage Fortran.Nous avons opté pour le langage Pascal car notre application n'est pas, comme le langage Fortran, orientée vers les mathématiques et impliquait l'utilisation de variables dynamiques. De plus, commme le logiciel devait être développé assez rapidement, utiliser le langage Pascal qui est notre langage d'école, nous permettait d'éviter l'étape d'initiation à la pratique d'un nouveau langage.

Notons que notre application devra probablement être adaptée à un micro ordinateur. Dans ce cas, Philips semble décidé à opter pour le langage C.

2.3 Tests

Le programme SYANODE a été complètement écrit conformément à la méthode décrite dans le chapitre IV. Chaque procédure ainsi que chaque module a été testé.

Les tests que nous avons réalisés ne sont cependant pas encore suffisants pour pouvoir évaluer le temps d'exécution du logiciel en fonction du nombre de groupes. Des tests supplémentaires devront également déterminer la valeur de la méthode que nous avons élaborée. Remarquons également que les exemples qui ont servi aux tests ne sont pas vraiment représentatifs de la réalité car des processus forts consommateurs en mémoire sont encore en développement.

La table 1 permet une première estimation des performances de SYANODE sur base de trois exemples; elle indique pour un nombre de groupes (NB GROUPES), les minimum et maximum de carte CP512 possibles (MINIMUM et MAXIMUM CP), le nombre de cartes CP512 effectivement utilisées, et le temps total d'exécution du programme en ms (TEMPS CPU TOT). Remarquons que le deuxième exemple est identique au premier mais utilise des agrégats.

NB GROUPES	MINIMUM CP	MAXIMUM CP	NB CP	TEMPS (ms) CPU TOT
54	6	26	8	62760
25	6	26	8	26620
58	17	131	23	77690

Table 1 : exemples de résultats de placement de groupes sur des cartes CP512

2.4 Méthodologie de développement de logiciel

Nous avions pour objectif de suivre la méthode de développement de logiciel et de structuration des données telle que proposée par nos professeurs au cours de nos études de licence. Cette méthode est décrite dans [3], [8] et [16].

Cet objectif n'a été atteint qu'en majeure partie car le délai fixé avant la remise d'une première version exécutable du programme SYANODE n'était pas suffisant.

3 Extensions possibles

3.1 Placement des groupes de type "driver"

Nous avons écarté le problème du placement des groupes de type "driver". Celui-ci pourrait aussi être automatisé.

3.2 Contraintes d'incompatibilité formelle

Dans notre méthode, pour définir une incompatibilité formelle entre différents processus, il faut en citer les noms de manière précise. Pouvoir formuler des incompatibilités entre différentes catégories de processus pourrait être utile. Par exemple, tel processus est incompatible avec tout driver.

3.3 Contrainte des bus CP

Nous avons expliqué, dans le premier chapitre, que deux cartes CP communiquent via un ou plusieurs bus CP. Une hypothèse importante de notre méthode est qu'il n'y a jamais de limite au débit d'un bus CP. Il serait peut être intéressant de définir un degré de voisinage des groupes en fonction du nombre de bus CP par lesquels les informations doivent transiter lors d'un échange. En effet, plus le nombre de bus à traverser est grand et plus la consommation de CPU nécessaire à la gestion de l'échange est importante. Il est prévu à l'étape suivante de ANODE d'optimiser le placement des cartes CP sur les bus CP en fonction des échanges entre cartes CP.

3.4 Localisation des fusions

Suite aux tests réalisés, il semble que le noyau issu de la dernière fusion est souvent impliqué dans la fusion suivante. C'est dans ce sens que nous pensons à une localisation des fusions. Profiter de la localisation des fusions réduirait probablement d'un degré, la complexité algorithmique de la méthode que nous avons utilisée.

3.5 Projet d'automatisation

Le projet de C.A.O. que nous avons défini ne permet une intervention humaine qu'à trois niveaux : la détermination de la taille des noyaux, celle du rapport CPU-MEMOIRE et la décision de rétroaction. Définir à ces trois niveaux des normes ou calculs précis mènerait à un projet d'automatisation.

3.6 Utilisation des cartes CP2000

Nous avons défini au cours du premier chapitre, la carte CP512. Philips a récemment introduit dans l'architecture de ses réseaux, un nouveau type de carte CP : la carte CP2000. Cette carte CP est comparable à la carte CP512 si ce n'est qu'elle possède une mémoire RAM de 2000 kB.

L'utilisation de ce type de carte CP fait surgir de nouveaux problèmes; en particulier l'ajout d'une carte CP vierge à une configuration matérielle existante implique le choix entre une carte CP512 et une carte CP2000. Notons qu'un tel choix doit prendre en compte les coûts de ces cartes CP, les consommations de mémoire et CPU des processus qu'il faut encore placer ainsi que les caractéristiques de la configuration matérielle déjà réalisée: en effet, il semble qu'il sera interdit d'installer des cartes CP de types différents sur un même étagère.

Notons que nous avons, par curiosité, répété les tests mentionnés en 2.3 en utilisant des cartes de type CP2000 uniquement. L'utilisation des cartes CP2000 ne semble pas réduire le nombre de cartes CP nécessaires. La contrainte sur le CPU en est, à notre avis, responsable.

La table 2 permet une estimation des performances de SYANODE sur base de trois exemples identiques à ceux de la première table, mais où la carte CP est de type CP2000; la table indique pour un nombre de groupes (NB GROUPES), les minimum et maximum de carte CP2000 possibles (MINIMUM et MAXIMUM CP), le nombre de cartes CP2000 effectivement utilisées (NB CP), et le temps total d'exécution du programme en ms (TEMPS CPU TOT). Remarquons que le deuxième exemple est identique au premier mais utilise des agrégats. Notons aussi que le nombre de cartes CP utilisées ainsi que le temps d'exécution du programme n'ont pas été réduits de manière significative par l'utilisation de cartes CP2000 plutôt que par des cartes CP512.

NB GROUPES	MINIMUM CP	MAXIMUM CP	NB CP	TEMPS (ms) CPU TOT
54	6	9	7	60650
25	6	9	7	24870
58	17	23	23	66870

Table 2 : exemples de résultats de placement de groupes sur des cartes CP2000

3.7 Décomposition des agrégats

La notion d'agrégat ne peut être directement intégrée dans le fichier de chargement effectif. Il serait intéressant, dans les résultats de notre projet, de décomposer les agrégats fournis en entrée en les processus les composant. Cette décomposition pose problème car les numéros de "thread" d'un même processus ne peuvent être dupliciés.

3.8 Système "intelligent"

Une dernière extension consisterait à définir un système "intelligent" qui se baserait sur son expérience de création de fichiers de chargement pour définir lui même des agrégats. Ceci réduirait dès le départ, le nombre de noyaux à traiter et donc le temps total d'exécution du programme.

BIBLIOGRAPHIE

[1] AIELLO A., BURATTINI E, MASSAROTTI A., & VENTRIGLIA F.
"A nonstandard evaluation method for bin
packing approximation algorithm"
6th Symposium On Operations Research, Augsburg,
Germany 7-9 Sept. 1981, NO 43, P. 175

[2] BELLMAN R. & S.E DREYFUS
"Applied dynamic programming"
Princeton University Press, Princeton, N.J. 1962

[3] F.BODART & Y. PIGNEUR
"Conception assistée des applications informatiques
1. Etude d'opportunité et analyse conceptuelle"
Masson, Presses Universitaires De Namur, 1983

[4] BRENDA S. BAKER, E.G. COFFMAN,JR & RONALD L.RIVEST
"Orthogonal packings in two dimensions"
SIAM J. Computer (USA), vol.9, NO 4, P.846-855, 1980

[5] F.R.K CHUNG, M.R. GAREY & D.S. JOHNSON
"On packing two-dimensional bins"
SIAM J. Algebraic & Discrete Methods (USA), vol.3, NO 1, P.66-76
Mars 1982

[6] FICHEFET J.
"Théorie des graphes"
Syllabus de première licence, FNDP Namur, 1984

[7] M.R. GAREY, R.L. GRAHAM, D.S. JOHNSON & A.C. YAO
"Ressource constrained scheduling as generalized bin packing"
J. of Combinatorial Th., vol. 21, NO 3, P. 257-297,
Novembre 1976

[8] J.L. HAINAUT
"2. conception de la base de données"
Masson, Presses Universitaires De Namur, 1985

[9] E.HOROWITZ & S.SAHNI
"Fundamentals of computer algorithms"
FITMAN, 1978

[10] JAMES P. HUANG
"Modeling of software partition for distributed
real-time applications"
IEEE Transactions On Software Engineering. vol.SE-11, NO. 10,
P. 1113-1125, Octobre 1985

[11] BIPIN INDURKHYA, HAROLD S.STONE & LU XI-CHENG
"Optimal partioning of randomly generated distributed programs"
IEEE Transactions On Software Engineering. Vol. SE-12, NO. 3,
P. 483-495, Mars 1986

[12] J.-C. LIENARD, Philips Telesoft, Brussels
"SOPHO-NET, a versatile private network"
Philips Telecommunication Review, vol.41, NO 3, P. 202-224, 1983

[13] R. MARCOGLIESE & R. NOVARESE
"Module and data allocation methods in distributed system"
The Second International Conference On Distributed Computing Systems
P. 50-59, PARIS, FRANCE 8-10 APRIL 1981

[14] D. PARNAS
"On the criteria to be used in decomposing systems into modules"
Communications Of The ACM, vol.15, NO 12, P. 1053-1058

[15] Philips Telecommunications
"SOPHO-NET product description, business communications"
1983

[16] A. VAN LAMBSWEERDE & J.L. HAINAUT
"Conception assistée des applications informatiques
2. Conception et modularisation de la base de données
et des programmes"
Masson, Presses Universitaires De Namur, 1985

TABLE DES MATIERES

Chapitre I Introduction au réseau SOPHO-NET

Chapitre II Conception logicielle d'un noeud SOPHO-NET

Facultés Universitaires N. D. de la Paix Namur

Institut d'Informatique

CONCEPTION ASSISTEE PAR ORDINATEUR D'UN NOEUD SOPHO-NET

ANNEXES

Dazard Olivier
Mahiat Cécile

Promoteur : Philippe Van Bastelaer

Mémoire présenté
en vue de l'obtention
du titre de
Licencié et Maître
en Informatique

Année Académique 1985 - 1986

LISTE DES ABREVIATIONS

ANNEXE I

1 Liste des abréviations generales

ANODE	Automated NOde DEsign
BAU	Bus Arbitration Unit
BSC	Binary Synchronous Communication
CAO	Conception Assistée par Ordinateur
CD	Communication Domain
CP	Processing card
CPU	Central Processing Unit
DMA	Direct Memory Access
FEP	Functionnaly Equivalent Processes
FTC	Fast Transfer Card
ID	Information Domain
I/O	Input Output
ISO	International Standard Organisation
ISR	Interrupt Service Routine
kB	1024 bytes
MARC	Multi ARrangement for Communication
MEM	Memoire
MIN	Minimun
MMU	Memory Management Unit
ms	Milliseconde
ms/s	MilliSeconde par seconde
NBRE	NomBRE
NMC	Network Management Center
OS	Operating System
PROM	Programmable Read Only Memory
PCT	Program Control Table
PNR	Program Number
RAM	Random Access Memory
ROM	Read Only Memory
RTCP	Reseau Telephonique Commute Public
SOPHO-NET	Synergetic Open PHilips NETwork
SYANODE	SYstem Aid for NOde DEsign
TD	Transfer Domain
TrD	Transmission Domain
TCT	Thread Control Table
VCP	Virtual Channel Path

2 Noms des processus cités dans les exemples

DRHD	Driver Half Duplex
DS	Distributor
DSM	Distributor Manager
EPST	Emulator Process Synchronous Terminal
FMGR	Flow Manager
HDLC	High Level Data Link Control
IDA	Information Domain Administrator
IR	Internal Routing
LNKA	LiNK Asynchronous
LUC	Logical Unit Controller
MDPER	Masking Device Permanent lines
SR	Sender Receiver
SRM	Sender Receiver Manager
VMGR	VCP Manager

LISTE DES DEFINITIONS

ANNEXE II

1 Agrégat

L'agrégat est une collection d'au moins deux processus de noms différents rassemblés en un ensemble indécomposable. Les processus constituant un agrégat sont en général caractérisés par des échanges intensifs entre eux. L'agrégat a son nom propre commencant conventionnellement par le caractère " @ ". L'objectif de l'agrégation est de favoriser l'intégration verticale, d'introduire des conventions de placement ou d'imposer le placement d'un ensemble de processus sur une même carte CP.

2 Arête de gain CPU

Une arête de gain CPU entre deux noyaux représente le gain CPU réalisé lorsque ces noyaux sont placés sur une même carte CP.

3 Arête de gain mémoire

Une arête de gain mémoire entre deux noyaux représente le gain mémoire réalisé lorsque ces deux noyaux sont placés sur une même carte CP.

4 Arête impossible absolue

Une arête est dite " impossible absolue " dès qu'une des quatre propositions suivantes est vraie :

- il existe une incompatibilité entre les groupes inclus dans les noyaux qui lui sont adjacents;
- différents numéros de carte CP définis sont associés aux noyaux qui lui sont adjacents;
- la somme des besoins CPU intrinsèques des deux noyaux qui lui sont adjacents dépasse la quantité CPU limite d'un noyau;
- la somme des besoins mémoire des deux noyaux qui lui sont adjacents, dont est soustraite la quantité représentée par l'arête, dépasse la capacité limite mémoire d'un noyau.

5 Arête impossible temporaire

Une arête est dite " impossible temporaire " si elle n'est ni " possible " ni " impossible absolue ". En ce sens qu'il est à ce moment impossible de fusionner les noyaux qui lui sont adjacents mais que, suite à d'autres fusions, ils pourraient être réunis. Seule une arête CPU peut être " impossible temporaire ".

6 Arête possible

Une arête est dite " possible " si les quatre conditions suivantes sont vérifiées :

- il n'y a pas d'incompatibilité entre les groupes correspondant aux deux noyaux qui lui sont adjacents;
- lorsque les deux noyaux qui lui sont adjacents ont un numéro de carte CP défini, ces numéros sont les mêmes;
- la somme des besoins CPU intrinsèques et CPU extrinsèques des deux noyaux qui lui sont adjacents moins la quantité de CPU extrinsèque qu'elle représente ne dépasse pas la capacité CPU limite d'un noyau;
- la somme des besoins mémoire des deux noyaux qui lui sont adjacents moins la quantité mémoire qu'elle représente ne dépasse pas la limite mémoire d'un noyau.

7 Bloc de communication

Le bloc de communication est un message structuré qui est émis d'un processus vers un autre, que ceux-ci soient placés sur une même carte CP ou non. C'est sous le contrôle de l'OS de la carte CP contenant le processus destinataire que le bloc de communication est placé dans un des fichiers d'entrée du processus destinataire.

8 Bus arbitration unit

La bus arbitration unit ou BAU est la carte chargée de gérer les conflits d'accès au bus CP. La carte BAU reçoit des cartes CP les demandes d'utilisation du bus CP. Le BAU avertit une de ces cartes CP quand le bus CP est libre.

9 Bus CP

Le bus CP est un moyen de communication très rapide utilisé par les cartes CP pour pouvoir s'échanger des informations. L'accès à ce bus est géré par une bus arbitration unit. Un bus CP ne peut relier qu'un nombre limité de cartes CP.

10 Bus I/O

Le bus I/O est le moyen de communication permettant le passage de caractères entre les cartes CP et les cartes I/O connectées à ce bus et controlées par cette carte CP. Un bus I/O ne peut relier qu'un nombre limité de cartes I/O.

11 Carte CP

La carte CP ou " Processing Card " est l'élément moteur d'un noeud SOPHO-NET. La carte CP est le lieu d'activation des processus localisés sur un noeud. Il est possible d'en trouver jusqu'à 256 par noeud. Les cartes CP sont reliées entre elles par un ou plusieurs bus CP.

12 Carte FTC

La carte FTC ou Fast Transfer Card est la carte permettant la liaison entre deux bus CP qui sont physiquement separés dans un noeud.

13 Carte I/O

La carte I/O est la carte sur laquelle sont connectées les lignes externes de transmission du noeud. La carte I/O reçoit les caractères de ces lignes externes et les transmet via le bus I/O, à la carte CP qui la gère. La carte I/O possède un buffer d'un caractère pour chaque ligne externe qui lui est connectée.

14 Consommation CPU extrinsèque d'un driver

La consommation CPU extrinsèque d'un driver est la puissance CPU qui, lors du traitement des interruptions externes, est nécessaire par seconde au sauvetage et à la restitution du contexte du processus interrompu. L'unité utilisée pour mesurer cette consommation CPU est la ms/s.

15 Consommation CPU extrinsèque d'un processus

La consommation CPU extrinsèque d'un processus est la puissance CPU nécessaire par seconde à l'échange d'informations entre ce processus et un autre. Cette consommation CPU est considerée comme nulle si les deux processus sont placés sur une même carte CP mais est non nulle lorsqu'ils sont placés sur des cartes CP différentes. L'unité utilisée pour la mesure de cette consommation CPU est la ms/s.

16 Consommation CPU intrinsèque d'un processus

La consommation CPU intrinsèque d'un processus est la puissance CPU nécessaire par seconde à l'exécution de ce processus. L'unité utilisée pour mesurer cette consommation est la ms/s.

17 Coût de découpe d'un " FEP "

Le coût de découpe d'un " FEP " est égal au supplément de mémoire nécessaire chaque fois qu'il faut installer un premier " thread " du " FEP " sur une nouvelle carte CP.

18 Direct Memory Acces

La DMA ou Direct Memory Access est un processeur d'échange localisé dans le système d'interface de la carte CP avec le bus CP. La DMA peut accéder directement à la mémoire RAM de la carte CP pour y lire ou y écrire des informations. La DMA est activée par le microprocesseur de la carte CP. L'utilité de la DMA réside dans le fait qu'elle peut décharger le microprocesseur de la carte CP de toutes les opérations d'échange généralement coûteuses en temps CPU, avec les autres cartes CP.

19 Domaine

Le domaine est le nom donné aux couches logiques d'un noeud SOPHO-NET. SOPHO-NET distingue quatre domaines dans un noeud :

- le domaine d'information ou Information Domain;
- le domaine de communication ou Communication Domain;
- le domaine de transfert ou Transfer Domain;
- le domaine de transmission ou TRansmission Domain qui est lui-même décomposé en trois sous-domaines :
  - le domaine de circuit ou circuit domain;
  - le domaine de liaison de données ou link domain;

- le domaine de paquet ou packet domain.

20 Driver

Le driver est un processus particulier qui gère l'émission et la réception de caractères vers ou à partir de la carte I/O. Il a comme caractéristiques supplémentaires par rapport au processus non driver, un certain débit de caractères, un temps de traitement par caractère et une puissance driver extrinsèque proportionnelle au débit de la ligne qu'il gère. Son numéro de carte CP est toujours précisé.

21 Elément

La notion d'élément est purement commerciale. L'élément correspond à une unité logicielle qui est caractérisé par son nom, sa fonction dans le noeud et son prix.

22 FEP

Le " FEP " (Functionnaly Equivalent Processes) est une collection de processus de même nom et même fonction. Tout usage de la fonction du " FEP " entraine l'allocation d'un des processus de ce " FEP " choisi arbitrairement parmi les processus libres de cette collection. Les " FEP " sont aussi appelés " processus dynamiques ".

23 Groupe

Un groupe est composé soit d'un ou plusieurs processus homonymes, soit d'un ou plusieurs agrégats de même nom. Un groupe a la qualité de " FEP " ou non. S'il est de type " FEP ", alors il est décomposable et ne doit donc pas être nécessairement placé sur une même carte CP. Le groupe " non FEP " est lui indécomposable, il doit être placé entièrement sur la même carte CP. Un groupe dynamique est un groupe de type " FEP ". A l'opposé, un groupe statique est un groupe de type " non FEP ". L'objectif du groupement est de favoriser l'intégration horizontale, de contraindre le placement de plusieurs processus sur une même carte CP ou d'accélérer le placement de ces processus.

24 Intégration horizontale

L'intégration horizontale est une technique consistant à placer sur une même carte CP des processus homonymes ou des agrégats de même nom afin de profiter du partage des pages de code et de données communes et de limiter ainsi la consommation mémoire de cet ensemble de processus. Pour les drivers, cette intégration horizontale peut donner lieu à une économie de puissance CPU.

25 Intégration verticale

L'intégration verticale est une technique consistant à placer des processus échangeant une quantité importante d'informations sur une même carte CP en vue d'annuler la consommation CPU extrinsèque nécessaire à leurs échanges. Ces processus sont généralement de noms différents.

26 Memory Management Unit

La MMU ou memory management unit est une unité de gestion de la mémoire qui a pour mission de réaliser une conversion d'adresse logique fournie par le processeur en adresse physique. Cette conversion est faite à partir de l'adresse logique fournie par le processeur de la carte CP, du PNR et du contenu des " locations " dans les " MMU banks ".

27 Niveau de découpe N

Le niveau de découpe N est égal au nombre " N " de parts en lesquelles est décomposé le " FEP ".

28 Noyau

Un noyau, qui au départ correspond à un groupe, résulte de la fusion d'un ou plusieurs noyaux. Ses attributs sont la liste des groupes le composant, ses consommations mémoire et CPU intrinsèque et extrinsèque. Un noyau issu d'une fusion ne peut jamais requérir plus de mémoire ou de CPU que ce qui a été fixé comme limites mémoire et CPU d'un noyau. Un noyau est également caractérisé par une consommation de " MMU banks " et de " PNR ". A chaque noyau est associé un numéro de carte CP qui est soit indéfini, soit égal au numéro de carte CP imposé par l'utilisateur, pour le placement d'un ou plusieurs groupes qui font partie de ce noyau.

De même que les groupes, tout noyau est de type " FEP " ou non. Au départ, le noyau est de même type que le groupe qu'il représente. Le noyau résultant de la fusion d'un noyau de type " FEP " et d'un noyau de type " NON FEP " est de type " NON FEP ". Seuls les noyaux de type " FEP " sont par définition décomposables.

29 Processus

Le processus est une unité d'exécution logicielle caractérisée par son nom, sa fonction, sa consommation mémoire, sa consommation CPU intrinsèque et extrinsèque vis à vis de ses voisins, une consommation de " MMU banks " et éventuellement un numéro de carte CP. Le processus est matérialisé par un code ou programme.

30 Programme

Le programme est le code qui est exécuté par les processus. Un programme est partageable par tous les processus homonymes placés sur une même carte CP.

31 Rapport CPU-MEMOIRE

Le rapport CPU-MEMOIRE est un rapport fourni par l'utilisateur ou calculé par le système et reflétant la direction à donner au processus de fusion; soit dans le sens d'une plus grande intégration verticale, soit dans le sens d'une plus grande intégration horizontale.

32 Rapport de consommation d'un " FEP "

Le rapport de consommation d'un " FEP " est le rapport entre sa consommation CPU et sa consommation mémoire. Ce rapport est fonction du niveau de découpe du " FEP ".

33 Rapport de disponibilité d'une carte CP

Le rapport de disponibilité d'une carte CP est le rapport entre le CPU disponible sur cette carte CP et la mémoire disponible sur cette même carte CP.

34 Voisins d'un processus

Les voisins d'un processus sont les processus avec lesquels il échange des blocs de communication.

QUESTIONNAIRE SOPHO-NET

ANNEXE III

GUIDELINES FOR COMPLETING THE NETWORK CONFIGURATION QUESTIONNAIRE - PART 1.

A. Composition of the questionnaire.

- Heading page

  Contains general information

- Network description page set
  Numbered as 0/XXX, where XXX is a serial number inside the network description page set (starting from 1)

- Node description page set
  Numbered as YYY/ZZZ, where YYY is the node number, as defined in the network description set and ZZZ is the serial number inside that node description set (starting from 1).

B. EXPLANATION OF TERMS

Heading page

Network name :
- identifies the network
- maximum 9 characters : alphanumeric or the underscore sign(_).

Version number :
- identifies the version of the questionnaire
- each time a questionnaire is revised due to a change of one item, the version number increases by one.

Date :
- date of questionnaire - part 1
- format is day/month/year

Number of pages :

- total number of pages of questionnaire - part 1, including the heading page.

Originator :

- name, company name, address and telephone number of the person who completed the questionnaire.

Project :

- Your name of the project/prospect to which the network relates.

Network description page set.

Page nbr : numbered as 0/xxx , where xxx is a serial number inside the network description page set (starting from 1)

Node number :

- sequence number of the node in this network
- not to be confused with node identification numbers, as used by the routing process.

Node name :

- unique name for node identification in this network.
- maximum of 9 characters : alphanumeric or the underscore sign(_).

Location :

- address of the location of the node

Node description page set.

Page nbr : numbered as YYY/ZZZ, where YYY is the node number, as defined in the network description set and ZZZ is the serial number inside that node description set (Starting from 1).

Node name :
- One of the node names as given in the network description.

Preferred engine type :

- choice between basic, mini or micro engine

Mains power supply voltage :
- voltage of the national grid to which the engine is to be connected.

Redundancy :
- redundant CP : one redundant CP per shelf
  redundant disk : for each disk one redundant disk
  redundant power supply : one redundant power unit per cabinet.

Maximum number of simultaneous call end-points :
- defines the number of call end-points which may be needed in this node.
- a call is equivalent to a communication between two users (or two devices).
- a call with a user connected to another node implies a call end-point in this node.
- a call between two users on the same node implies two call end-points.

Network Management Centre

- indicates if an NMC is to be connected on this node.
- If yes, then a port with "NMC" protocol should be filled in as well.

Administrative Unit

- indicates if an AU has to be installed in this node.
- It is advisable to define an AU in a node, if there is a high concentration of end-users around that node.
- Currently only one AU per network is permitted.

Port(s) group identification

Identifies a group of one or more ports with the same characteristics, as specified in the row(s) of this table.
Several rows may be used for a port group description (e.g. in case of several protocols).

Quantity (QTY)

- Defines the number of ports, with the same characteristics.

Speed :

- Defines the line speed which will be used on the ports, expressed in bits per second.
- In the case of the autobaud facility on a port, the highest allowed speed has to be taken into account.

Protocol(s)

- Defines the protocol(s) on the port(s) group
- Names as defined in the price list should be used.
- Protocols, which do not appear in the price list should be given the names : NEW1, NEW2.... These protocols should be described on separate sheets
- Only one protocol per row.

Number of devices

- Defines the number of devices (or device images) which are simultaneously connectable through this port(s) group (Total for all the ports).
- If group hunting is requested then the number should be followed by H".
- Is irrelevant for trunk ports

Number of calls

- defines the number of calls (VCP's and/or flows) which can exist simultaneously for the port(s) group.
- for X.25T, it defines the number of virtual circuits
- for X.25DCE or X.25DTE, it defines the number of calls which is taken equal to the number of virtual circuits.

Messages (from devices) to nodes

Number per hour(nbr/hr)
the number of messages, which would be sent by the devices of this port group to the node in a one hour period, if the peak rate, were measured on a few seconds period, to continue for the whole hour.

Length :
The average number of characters of these messages.

Message (from node) to devices.

Similar to the above, but in the other direction.

Destination(s) :

Node :
The node number of the partner port(s) group(s), which communicate with this port(s) group.

% :
The percentage of the number of messages of this port(s) which group go to the partner port(s) group node. Is irrelevant for trunk ports.

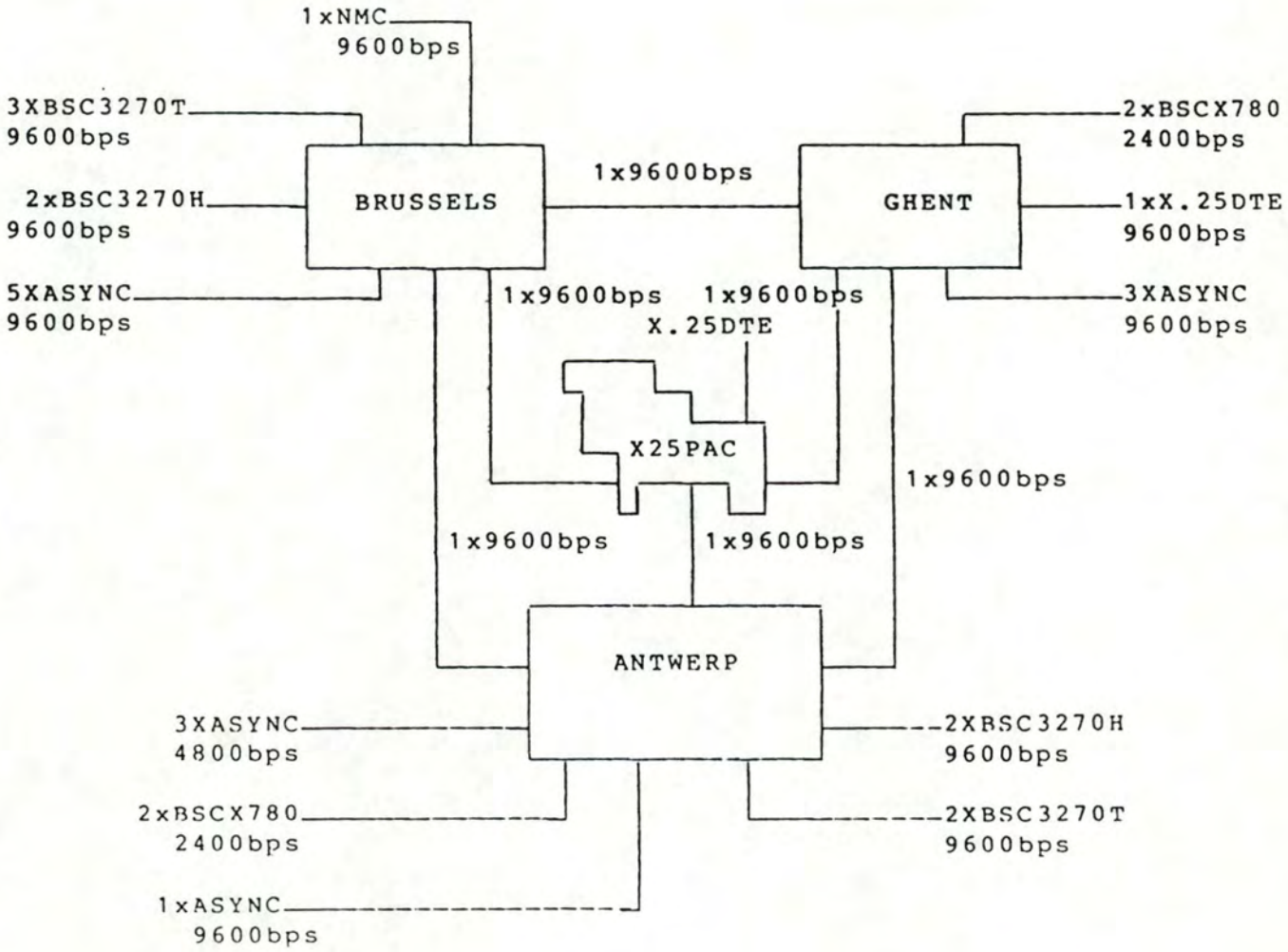
Several destinations may be given.

Comments :

For information, which does not fit in the standard format.

EXAMPLE : SOPHO-NET

1. NETWORK OVERVIEW



2. COMMUNICATION MATRIX

- The BSC3270T terminals can go to all the local BSC3270H hosts as well as all to the remote ones. It is assumed that during the busy period half are local, half are remote.

- Async GHENT goes to async BRUSSELS

- Async ANTWERP goes to async BRUSSELS

- X.25DTE locally connected to GHENT goes to the X.25DTE connected via X.25PAC, the public X.25 packet switching network.

- BSCX780 GHENT goes to BSCX780 ANTWERP

- X.25PAC is used as back-up for trunk lines.

3. NUMBER OF DEVICES AND THEIR USE.

- On each BSC3270T port, 16 terminals are connected. At most 12 of these will have one call (VCP) simultanesously. During the busy period, all these 12 terminals exchange messages, as follows :
  - Every terminal receives a new screen every 3 minutes. Screen contents = 300 characters.
  - The user enters about 50 characters.

- On each BSC3270H port, 16 terminal images have to be foreseen. They all can have a call (VCP) simultaneously. Group hunting is asked.

- To each async port, one async device is connected, In BRUSSELS, on host side, group hunting is asked.

- In Antwerp, all async terminals are in communication during the busy period and they are used in line mode. A command line (mean length 25 characters) is entered on the terminal. Every 10 seconds, such a line is sent to the BRUSSELS node. After 5 command lines, the BRUSSELS host answers by a message of 200 characters.

- In GHENT, all async terminals are in communication during the busy period and they are used in character mode, accessing a screen editing application. Every character entered on the terminal, is sent through the network to Brussels, where it is interpreted by the host. The host returns most of the time the character plus part of the current screen line (assume mean length is 20 characters).
  We assume that only 2 out of 3 terminals are effectively working during the busy period, and that on a working terminal 6 characters per second are entered.
  Remark : Be aware that this way of working puts a heavy load on the network!

- The BSCX780 ports are used for file transfer. Transfer happens in both directions, but never simultaneously. Every night (so out of the busy period), 500 kBytes are sent from ANTWERP to GHENT, and 100 kBytes in the opposite direction.

- The X.25DTE's have two independent data flows between them. During the busy period, on each flow every 5 seconds a 1000 character message is sent in each direction.

4. SOME EXPLANATIONS.

For a couple of cases, the definition of the figures about message numbers per hour, is explained here.
The reference is node number, followed by port group identification.

2A Every 10 secs a line is entered at the terminal side, so 1 line/10 sec = 360 lines/hr.
The host answers back after 5 lines have been entered, so 360/5 = 72 messages/hr.

3A Two out of 3 terminals are working, although all 3 have a call.
6 chars/sec x 3600 x 2 = 43.200 messages per hour

1A and 1B :
As the number of lines is lower than the sum of the corresponding lines in Ghent, (3A) and Antwerp (2A), the highest load is taken.

1C and 1E :
Per terminal : 360/180 = 20 messages per hour.
For 36 terminals : 720 messages per hour.
For 32 terminals : 640 messages per hour.
Half of them are local. So there is an incoherence between the 18 at terminal side and the 16 at host side, which is neglected here.

V.O.F. Philips Networks International S.N.C.

issue b

- Example 5 -

NETWORK CONFIGURATION QUESTIONNAIRE - PART 1

Heading page

NETWORK NAME : S O P H O _ N E T

VERSION NUMBER : 1

DATE : d 01 m 10 y 1985

NUMBER OF PAGES : 0 0 8

ORIGINATOR : NAME: P H I L I P S

COMPANY : P N I

ADDRESS: B V D A N S P A C H 1

B R U S S E L S

B E L G I U M

tel nbr: country 32 city 02 tel. number 21190995

PROJECT : C O N F . E X .

Network description

NODE NBR	NODE NAME	LOCATION
1	BRUSSELS	1000 BRUSSELS BVD ANSPACH 1
2	ANTWERP	2000 ANTWERP HAVENSTRAAT 12
3	GHENT	9000 GHENT MARKT 1

Node description

- Node name :

B	R	U	S	S	E	L	S	

- Preferred engine type

BASIC	MINI	MICRO
X		

- Mains power supply voltage :220 VOLT

- Redundancy

	YES	NO
Redundant CP	X	
Redundant disk	X	
Redundant Power Unit	X	

- Maximum number of simultaneous call end-points 73

- Network Management Centre

YES	NO
X	

- Administrative Unit

YES	NO
X	

Node description

Port(s) group	QTY	Speed (b/s)	Protocol(s)	Nbr of devices	Nbr of calls	Messages to node		Messages to device		Destination(s)		Comments
						Nbr/hr	length	Nbr/hr	length	Node	%	
A	5	9600	ASYNC	5H	3	43200	20	43200	1	3	100	
B					2	144	200	720	25	2	100	
C	3	9600	BSC3270T	48	36	720	50	720	300	1	50	
D										2	50	
E	2	9600	BSC3270H	32H	32	640	300	640	50	1	50	
F										2	50	
G	1	9600	HDLCT	-						2		
H	1	9600	HDLCT	-						3		
I	1	9600	X.25T	-	2					2,3		
J	1	9600	NMC									
K												
L												
M												

V.O.F. Philips Networks International S.N.C.

Node description

- Node name :

A	N	T	W	E	R	P		

- Preferred engine type

BASIC	MINI	MICRO
X		

- Mains power supply voltage :220VOLTS

- Redundancy

	YES	NO
Redundant CP	X	
Redundant disk	X	
Redundant Power Unit	X	

- Maximum number of simultaneous call end-points 66

- Network Management Centre

YES	NO
	X

- Administrative Unit

YES	NO
	X

Node description

Port(s) group	QTY	Speed (b/s)	Protocol(s)	Nbr of devices	Nbr of calls	Messages to node Nbr/hr	Messages to node length	Messages to device Nbr/hr	Messages to device length	Destination(s) Node	Destination(s) %	Comments
A	1	9600	ASYNC	1	1	360	25	72	200	1	100	
B	3	4800	ASYNC	3	3	1080	25	216	200	1	100	
C	2	2400	BSCX780	2	2		500K		100K	3	100	NOT IN BUSY HOURS
D	2	9600	BSC3270T	32	24	480	50	480	300	2	50	
E										1	50	
F	2	9600	BSC3270H	36	36H	640	300	640	50	2	50	
G										1	50	
H	1	9600	HDLCT							1		
I	1	9600	HDLCT							3		
J	1	9600	X.25T	-	2					1,3		
K												
L												
M												

V.O.F. Philips Networks International S.N.C.

Node description

- Node name :

G	H	E	N	T				

- Preferred engine type

BASIC	MINI	MICRO
X		

- Mains power supply voltage :220VOLTS

- Redundancy

	YES	NO
Redundant CP	X	
Redundant disk	X	
Redundant Power Unit	X	

- Maximum number of simultaneous call end-points 9

- Network Management Centre

YES	NO
	X

- Administrative Unit

YES	NO
	X

Node description

Port(s) group	QTY	Speed (b/s)	Protocol(s)	Nbr of devices	Nbr of calls	Messages to node		Messages to device		Destination(s)		Comments
						Nbr/hr	length	Nbr/hr	length	Node	%	
A	3	9600	ASYNC	3	3	43200	1	43200	20	1	100	
B	1	9600	X.25DCE	1	2	1440	1000	1440	1000	3	100	FDX
C	2	2400	BSCX780	2	2		100K		500K	2	100	NOT IN BUSY HOURS
D	1	9600	HDLCT							1		
E	1	9600	HDLCT							2		
F	1	9600	X.25T	-	2					1,2		
G			X.25DTE	1	2	1440	1000	1440	1000	3	100	FDX
H												
I												
J												
K												
L												
M												

V.O.F. Philips Networks International S.N.C.

CATALOGUES

ANNEXE IV

```
IIIIII    NN      NN  PPPPPPPP    UU      UU  TTTTTTTTTT
IIIIII    NN      NN  PPPPPPPP    UU      UU  TTTTTTTTTT
  II      NN      NN  PP      PP  UU      UU      TT
  II      NN      NN  PP      PP  UU      UU      TT
  II      NNNN    NN  PP      PP  UU      UU      TT
  II      NNNN    NN  PP      PP  UU      UU      TT
  II      NN  NN  NN  PPPPPPPP    UU      UU      TT
  II      NN  NN  NN  PPPPPPPP    UU      UU      TT
  II      NN    NNNN  PP          UU      UU      TT
  II      NN    NNNN  PP          UU      UU      TT
  II      NN      NN  PP          UU      UU      TT
  II      NN      NN  PP          UU      UU      TT
IIIIII    NN      NN  PP          UUUUUUUUUU      TT
IIIIII    NN      NN  PP          UUUUUUUUUU      TT

  AAAAAA      GGGGGGGG  NN      NN    AAAAAA    MM      MM
  AAAAAA      GGGGGGGG  NN      NN    AAAAAA    MM      MM
AA      AA  GG          NN      NN  AA      AA  MMMM  MMMM
AA      AA  GG          NN      NN  AA      AA  MMMM  MMMM
AA      AA  GG          NNNN    NN  AA      AA  MM  MM  MM
AA      AA  GG          NNNN    NN  AA      AA  MM  MM  MM
AA      AA  GG          NN  NN  NN  AA      AA  MM      MM
AA      AA  GG          NN  NN  NN  AA      AA  MM      MM
AAAAAAAAAA  GG  GGGGGG  NN    NNNN  AAAAAAAAAA  MM      MM
AAAAAAAAAA  GG  GGGGGG  NN    NNNN  AAAAAAAAAA  MM      MM
AA      AA  GG      GG  NN      NN  AA      AA  MM      MM
AA      AA  GG      GG  NN      NN  AA      AA  MM      MM
AA      AA    GGGGGG    NN      NN  AA      AA  MM      MM
AA      AA    GGGGGG    NN      NN  AA      AA  MM      MM
```

FICHIER TEXT DES AGREGATS

```
@HD    y  DRHD   HDLC   MDPER
@RSRM  n  ROUTER SRM
@ID    n  IDA    SPM    POM
```

```
IIIIII    NN      NN  PPPPPPPP        11
IIIIII    NN      NN  PPPPPPPP        11
  II      NN      NN  PP      PP    1111
  II      NN      NN  PP      PP    1111
  II      NNNN    NN  PP      PP      11
  II      NNNN    NN  PP      PP      11
  II      NN  NN  NN  PPPPPPPP        11
  II      NN  NN  NN  PPPPPPPP        11
  II      NN    NNNN  PP              11
  II      NN    NNNN  PP              11
  II      NN      NN  PP              11
  II      NN      NN  PP              11
IIIIII    NN      NN  PP            111111
IIIIII    NN      NN  PP            111111
```

```
AAAAAA
AAAAAA
AA      AA
AA      AA
AA      AA
AA      AA
AA      AA
AA      AA
AAAAAAAAAA
AAAAAAAAAA
AA      AA
AA      AA
AA      AA
AA      AA
```

```
  GGGGGGGG  RRRRRRRR    PPPPPPPP
  GGGGGGGG  RRRRRRRR    PPPPPPPP
GG          RR      RR  PP      PP
GG          RR      RR  PP      PP
GG          RR      RR  PP      PP
GG          RR      RR  PP      PP
GG          RRRRRRRR    PPPPPPPP
GG          RRRRRRRR    PPPPPPPP
GG  GGGGGG  RR  RR      PP
GG  GGGGGG  RR  RR      PP
GG      GG  RR    RR    PP
GG      GG  RR    RR    PP
  GGGGGG    RR      RR  PP
  GGGGGG    RR      RR  PP
```

FICHIER TEXT DES GROUPES D'ENTREE

```
NETWORK ANODE_STEP3_TEST
NODE INP1_A.GRP
VERS 2
DATE 18-AUG-1986

CPTYPE          CP512
AVL_MEM_CP        342
AVL_MUB_CP        120
AVL_PNR_CP        120   60   30   0

AVL_PW_CP         750


@HD      1     5     0    465.40    0    0    2
                                                SR     1   12.25
                                                SR     4   49.00

@HD      6    10     1    465.40    0    0    2
                                                SR     2   24.50
                                                SR     3   36.75

DRHD    11    11     2    651.68    0    0    1
                                                HDLC  11   61.25

HDLC    11    11    -1    56.4      0    0    2
                                                DRHD  11   49.0
                                                MDPACK 1   61.25

MDPACK   1     1    -1    75.74     0    0    4
                                                HDLC  11   61.25
                                                FUN    1   12.25
                                                LCC    1   24.5
                                                LCH    1   49.00

FUN      1     1    -1    19.82     0    0    2
                                                MDPACK 1   12.25
                                                LCH    1   24.5

LCC      1     1    -1    13.5      0    0    1
                                                MDPACK 1   24.5

LCH      1     4    -1    100.2     0    0    6
                                                MDPACK 1   49.0
                                                FUN    1   24.5
                                                SR     1    6.2
                                                SR     2   12.4
                                                SR     3   18.6
                                                SR     4   24.8

SR       1     1    -1    23.1      0    0    6
                                                @HD    1   9.8
                                                LCH    1   5.0
                                                IR     1   5.0
                                                XXX    0   5.5
                                                DS     1   7.4
                                                DSM    1   2.0

SR       2     2    -1    46.2      0    0    6
                                                @HD    6   19.6
                                                LCH    1   10.0
                                                IR     1   10.0
                                                XXX    0   11.0
                                                DS     1   14.8
```

```
                                                    DSM     1 4.0

SR      3    3    -1    69.3      0     0     6
                                                    @HD     6 29.4
                                                    LCH     1 15.0
                                                    IR      1 15.0
                                                    XXX     0 16.5
                                                    DS      1 22.2
                                                    DSM     1 6.0

SR      4    4    -1    92.4      0     0     6
                                                    @HD     1 39.2
                                                    LCH     1 20.0
                                                    IR      1 20.0
                                                    XXX     0 22.0
                                                    DS      1 29.6
                                                    DSM     1 8.0

@RSRM   1    1    -1    13        0    16     5
                                                    DS      1 7.4
                                                    SR      1 7.4
                                                    SR      2 7.4
                                                    SR      3 7.4
                                                    SR      4 7.4

DSM     1    1    -1    43.26     0     0     6
                                                    DS      1 46.0
                                                    SR      1  2
                                                    SR      2  4
                                                    SR      3  6
                                                    SR      4  8
                                                    XXX     0 27.6

DS      1    1    -1    595.21    0     0     6
                                                    DSM     1 46.0
                                                    SR      1 7.5
                                                    SR      2 15.0
                                                    SR      3 22.5
                                                    SR      4 30.0
                                                    XXX     0 82.7

IR      1    1    -1    15        0     0     4
                                                    SR      1 5
                                                    SR      2 10
                                                    SR      3 15
                                                    SR      4 20

PUC     1    1    -1    1         0     0     0
VMGR    1   15    -1    15        0     0     0
FMGR    1   30    -1    30        0     0     0

LUC     1    1    -1    1         0     0     0
VMGR 1001 1030    -1    30        0     0     0
FMGR 1001 1030    -1    500       0     0     1
                                                    XXX     0 250

LUC     2    2    -1    1         0     0     0
VMGR 2001 2030    -1    30        0     0     0
FMGR 2001 2030    -1    500       0     0     1
                                                    XXX     0 250
```

```
IIIIII    NN      NN  PPPPPPPP    UU      UU  TTTTTTTTTT
IIIIII    NN      NN  PPPPPPPP    UU      UU  TTTTTTTTTT
  II      NN      NN  PP      PP  UU      UU      TT
  II      NN      NN  PP      PP  UU      UU      TT
  II      NNNN    NN  PP      PP  UU      UU      TT
  II      NNNN    NN  PP      PP  UU      UU      TT
  II      NN  NN  NN  PPPPPPPP    UU      UU      TT
  II      NN  NN  NN  PPPPPPPP    UU      UU      TT
  II      NN    NNNN  PP          UU      UU      TT
  II      NN    NNNN  PP          UU      UU      TT
  II      NN      NN  PP          UU      UU      TT
  II      NN      NN  PP          UU      UU      TT
IIIIII    NN      NN  PP          UUUUUUUUUU      TT
IIIIII    NN      NN  PP          UUUUUUUUUU      TT

IIIIII    NN      NN    CCCCCCCC    OOOOOO    MM      MM  PPPPPPPP
IIIIII    NN      NN    CCCCCCCC    OOOOOO    MM      MM  PPPPPPPP
  II      NN      NN  CC          OO      OO  MMMM  MMMM  PP      PP
  II      NN      NN  CC          OO      OO  MMMM  MMMM  PP      PP
  II      NNNN    NN  CC          OO      OO  MM  MM  MM  PP      PP
  II      NNNN    NN  CC          OO      OO  MM  MM  MM  PP      PP
  II      NN  NN  NN  CC          OO      OO  MM      MM  PPPPPPPP
  II      NN  NN  NN  CC          OO      OO  MM      MM  PPPPPPPP
  II      NN    NNNN  CC          OO      OO  MM      MM  PP
  II      NN    NNNN  CC          OO      OO  MM      MM  PP
  II      NN      NN  CC          OO      OO  MM      MM  PP
  II      NN      NN  CC          OO      OO  MM      MM  PP
IIIIII    NN      NN    CCCCCCCC    OOOOOO    MM      MM  PP
IIIIII    NN      NN    CCCCCCCC    OOOOOO    MM      MM  PP
```

FICHIER TEXT DES INCOMPATIBILITES FORMELLES

LUC / LUC
EPST / LNKA
EPST / DRHD
EPST / DRVB
SMHP / LNKA
SMHP / DRHD
SMHP / DRVB
DS / DS

```
IIIIII     NN       NN  PPPPPPPP     UU       UU  TTTTTTTTTT
IIIIII     NN       NN  PPPPPPPP     UU       UU  TTTTTTTTTT
  II       NN       NN  PP      PP   UU       UU      TT
  II       NN       NN  PP      PP   UU       UU      TT
  II       NNNN     NN  PP      PP   UU       UU      TT
  II       NNNN     NN  PP      PP   UU       UU      TT
  II       NN  NN   NN  PPPPPPPP     UU       UU      TT
  II       NN  NN   NN  PPPPPPPP     UU       UU      TT
  II       NN     NNNN  PP           UU       UU      TT
  II       NN     NNNN  PP           UU       UU      TT
  II       NN       NN  PP           UU       UU      TT
  II       NN       NN  PP           UU       UU      TT
IIIIII     NN       NN  PP           UUUUUUUUUU       TT
IIIIII     NN       NN  PP           UUUUUUUUUU       TT

PPPPPPPP   NN       NN    AAAAAA     MM       MM
PPPPPPPP   NN       NN    AAAAAA     MM       MM
PP      PP NN       NN  AA      AA   MMMM   MMMM
PP      PP NN       NN  AA      AA   MMMM   MMMM
PP      PP NNNN     NN  AA      AA   MM  MM  MM
PP      PP NNNN     NN  AA      AA   MM  MM  MM
PPPPPPPP   NN  NN   NN  AA      AA   MM      MM
PPPPPPPP   NN  NN   NN  AA      AA   MM      MM
PP         NN     NNNN  AAAAAAAAAA   MM      MM
PP         NN     NNNN  AAAAAAAAAA   MM      MM
PP         NN       NN  AA      AA   MM      MM
PP         NN       NN  AA      AA   MM      MM
PP         NN       NN  AA      AA   MM      MM
PP         NN       NN  AA      AA   MM      MM
```

FICHIER TEXT DES PROCESSUS

DRVB	y	y	1	30	1824	3008	1	0
DLMH	n	y	2	39	710	736	27	5
DLMT	n	y	2	40	814	768	27	5
SESH	n	n	2	35	92	384	45	0
SEST	n	n	2	42	128	384	30	5
LNKA	y	y	2	42	1698	1216	1	0
SESA	n	n	1	27	176	384	48	0
SMHP	n	n	2	39	526	448	27	5
EPST	n	n	2	34	452	256	13	20
DRHD	y	y	0	12	5006	736	1	0
HDLC	n	y	1	15	94	320	45	0
MDPER	n	y	1	15	64	256	60	0
MDPACK	n	y	0	11	22	224	48	0
FUN	n	y	0	7	12	128	48	0
LCC	n	y	1	13	736	576	45	0
LCH	n	n	2	30	346	416	63	0
IDA	n	y	2	60	16716	1024	15	0
SPM	n	y	1	19	10548	896	22	0
POM	n	y	0	8	514	1600	43	0
EPM	n	y	1	19	10548	896	20	0
VMGR	n	n	2	52	3478	928	32	0
FMGR	n	n	2	42	2926	768	36	0
IR	n	y	2	51	0	20288	1	0
LUC	n	y	2	60	0	2048	1	0
PUC	n	y	2	52	0	2144	1	0
ROUTER	n	y	1	24	15316	352	16	64
DS	n	y	1	16	5018	256	32	0
DSM	n	y	0	7	64	192	51	0
SR	n	y	1	24	1120	3946	30	0
SRM	n	y	0	7	148	288	45	0

EXEMPLE DE RESULTATS

ANNEXE V

```
  OOOOOO    UU      UU  TTTTTTTTTT      11                    AAAAAA
  OOOOOO    UU      UU  TTTTTTTTTT      11                    AAAAAA
OO      OO  UU      UU      TT        1111                  AA      AA
OO      OO  UU      UU      TT        1111                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AAAAAAAAAA
OO      OO  UU      UU      TT          11                  AAAAAAAAAA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
  OOOOOO    UUUUUUUUUU      TT        111111    ________    AA      AA
  OOOOOO    UUUUUUUUUU      TT        111111    ________    AA      AA

            CCCCCCCC  RRRRRRRR    DDDDDDDD
            CCCCCCCC  RRRRRRRR    DDDDDDDD
          CC          RR      RR  DD      DD
          CC          RR      RR  DD      DD
          CC          RR      RR  DD      DD
          CC          RR      RR  DD      DD
          CC          RRRRRRRR    DD      DD
          CC          RRRRRRRR    DD      DD
          CC          RR  RR      DD      DD
          CC          RR  RR      DD      DD
....      CC          RR    RR    DD      DD
....      CC          RR    RR    DD      DD
....        CCCCCCCC  RR      RR  DDDDDDDD
....        CCCCCCCC  RR      RR  DDDDDDDD
```

FICHIER TEXT DES RESULTATS PAR CARTE CP

NETWORK: ANODE_STEP3_TEST
NODE: INP1_A.GRP
VERSION= 2
DATE: 18-AUG-1986

---------- CP_CARD CHARACTERISTICS ----------

Type : CP512
Available MEMORY (ko) = 342.0
Available PROCESSING POWER (ms/s) = 750.0
Available MMU BANKS (#) = 120
Available PNRs (# # # #) = 120 60 30 0

---------- PLACEMENT CHARACTERISTICS ----------

Allocated	MEMORY		PROCESSING POWER (PW)	
	Ko	%CP	ms/s	%CP
Average	176.5	51.6	672.9	89.7
Standard Deviation	91.5	26.8	173.9	23.2

Minimum Number of CP_CARD(s)= 6
Number of USED CP_CARD(s) = 8
Maximum Number of CP_CARD(s)= 26

Used MEM/PW ratio (RAP) = 2.03126E-01

PW OVERHEAD = 28.4 %
MEM OVERHEAD = 18.3 %

```
-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :            0

MEMORY CONSUMPTION IS :            185 Kb
PW     CONSUMPTION IS :            696 ms/s
MMU    CONSUMPTION IS :             17 mmubank(s)
PNR    CONSUMPTION IS :              9 pnr

@HD                  1              5
HDLC                11             11
PUC                  1              1
LUC                  2              2

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :            1

MEMORY CONSUMPTION IS :            338 Kb
PW     CONSUMPTION IS :            700 ms/s
MMU    CONSUMPTION IS :             18 mmubank(s)
PNR    CONSUMPTION IS :             10 pnr

@HD                  6             10
DSM                  1              1
LUC                  1              1
VMGR                 1             15
VMGR              1001           1020

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :            2

MEMORY CONSUMPTION IS :            159 Kb
PW     CONSUMPTION IS :            743 ms/s
MMU    CONSUMPTION IS :              5 mmubank(s)
PNR    CONSUMPTION IS :              2 pnr

DRHD                11             11
FMGR                 1             30

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :            3

! Card over-allocated

MEMORY CONSUMPTION IS :             16 Kb
PW     CONSUMPTION IS :            799 ms/s
MMU    CONSUMPTION IS :              2 mmubank(s)
PNR    CONSUMPTION IS :              1 pnr

DS                   1              1

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :            4

MEMORY CONSUMPTION IS :            176 Kb
PW     CONSUMPTION IS :            695 ms/s
MMU    CONSUMPTION IS :             14 mmubank(s)
PNR    CONSUMPTION IS :              6 pnr

FUN                  1              1
LCH                  1              4
SR                   1              1
```

```
SR                    2          2
SR                    3          3
SR                    4          4
@RSRM                 1          1
IR                    1          1

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :          5

MEMORY CONSUMPTION IS :          245 Kb
PW     CONSUMPTION IS :          252 ms/s
MMU    CONSUMPTION IS :            7 mmubank(s)
PNR    CONSUMPTION IS :            3 pnr

MDPACK                1          1
LCC                   1          1
VMGR               1021       1030
VMGR               2001       2030

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :          6

MEMORY CONSUMPTION IS :          147 Kb
PW     CONSUMPTION IS :          750 ms/s
MMU    CONSUMPTION IS :            4 mmubank(s)
PNR    CONSUMPTION IS :            1 pnr

FMGR               1001       1030

-----------------------------------------------------------------------------

GROUPS PLACED UPON CARD NUMBER :          7

MEMORY CONSUMPTION IS :          147 Kb
PW     CONSUMPTION IS :          750 ms/s
MMU    CONSUMPTION IS :            4 mmubank(s)
PNR    CONSUMPTION IS :            1 pnr

FMGR               2001       2030

-----------------------------------------------------------------------------

! WARNING: card(s) over-allocated
-----------------------------------------------------------------------------
```

```
  OOOOOO    UU      UU  TTTTTTTTTT      11                    AAAAAA
  OOOOOO    UU      UU  TTTTTTTTTT      11                    AAAAAA
OO      OO  UU      UU      TT        1111                  AA      AA
OO      OO  UU      UU      TT        1111                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AAAAAAAAAA
OO      OO  UU      UU      TT          11                  AAAAAAAAAA
OO      OO  UU      UU      TT          11                  AA      AA
OO      OO  UU      UU      TT          11                  AA      AA
  OOOOOO    UUUUUUUUUU      TT        111111    __________  AA      AA
  OOOOOO    UUUUUUUUUU      TT        111111    __________  AA      AA
```

```
          GGGGGGGG  RRRRRRRR    PPPPPPPP
          GGGGGGGG  RRRRRRRR    PPPPPPPP
        GG          RR      RR  PP      PP
        GG          RR      RR  PP      PP
        GG          RR      RR  PP      PP
        GG          RR      RR  PP      PP
        GG          RRRRRRRR    PPPPPPPP
        GG          RRRRRRRR    PPPPPPPP
        GG  GGGGGG  RR  RR      PP
        GG  GGGGGG  RR  RR      PP
        GG      GG  RR    RR    PP
        GG      GG  RR    RR    PP
          GGGGGG    RR      RR  PP
          GGGGGG    RR      RR  PP
```

FICHIER TEXT DES RESULTATS PAR GROUPE

NETWORK: ANODE_STEP3_TEST
NODE: INP1_A GRP
VERSION= 2
DATE: 18-AUG-1986

@HD	1	5	0
@HD	6	10	1
DRHD	11	11	2
HDLC	11	11	0
MDPACK	1	1	5
FUN	1	1	4
LCC	1	1	5
LCH	1	4	4
SR	1	1	4
SR	2	2	4
SR	3	3	4
SR	4	4	4
@RSRM	1	1	4
DSM	1	1	1
DS	1	1	3
IR	1	1	4
PUC	1	1	0
VMGR	1	15	1
FMGR	1	30	2
LUC	1	1	1
VMGR	1001	1020	1
VMGR	1021	1030	5
FMGR	1001	1030	6
LUC	2	2	0
VMGR	2001	2030	5
FMGR	2001	2030	7

```
  OOOOOO     UU       UU  TTTTTTTTTT       11                    AAAAAA
  OOOOOO     UU       UU  TTTTTTTTTT       11                    AAAAAA
OO      OO   UU       UU      TT         1111                  AA      AA
OO      OO   UU       UU      TT         1111                  AA      AA
OO      OO   UU       UU      TT           11                  AA      AA
OO      OO   UU       UU      TT           11                  AA      AA
OO      OO   UU       UU      TT           11                  AA      AA
OO      OO   UU       UU      TT           11                  AA      AA
OO      OO   UU       UU      TT           11                  AAAAAAAAAA
OO      OO   UU       UU      TT           11                  AAAAAAAAAA
OO      OO   UU       UU      TT           11                  AA      AA
OO      OO   UU       UU      TT           11                  AA      AA
  OOOOOO     UUUUUUUUUU      TT         111111    __________  AA      AA
  OOOOOO     UUUUUUUUUU      TT         111111    __________  AA      AA

             EEEEEEEEEE  XX      XX    CCCCCCCC
             EEEEEEEEEE  XX      XX    CCCCCCCC
             EE          XX      XX  CC
             EE          XX      XX  CC
             EE            XX  XX    CC
             EE            XX  XX    CC
             EEEEEEEE        XX      CC
             EEEEEEEE        XX      CC
             EE            XX  XX    CC
             EE            XX  XX    CC
    ....     EE          XX      XX  CC
    ....     EE          XX      XX  CC
    ....     EEEEEEEEEE  XX      XX    CCCCCCCC
    ....     EEEEEEEEEE  XX      XX    CCCCCCCC
```

FICHIER TEXT DES RESULTATS PAR PAIRE DE CARTES CP

NETWORK: ANODE_STEP3_TEST
NODE: INP1_A.GRP
VERSION= 2
DATE: 18-AUG-1986

EXT.PW between CP_CARDs	0	&	2	is	110.3
EXT.PW between CP_CARDs	0	&	4	is	110.3
EXT.PW between CP_CARDs	0	&	5	is	122.5
EXT.PW between CP_CARDs	1	&	3	is	92.0
EXT.PW between CP_CARDs	1	&	4	is	150.3
EXT.PW between CP_CARDs	3	&	4	is	156.4
EXT.PW between CP_CARDs	4	&	5	is	122.5

```
  OOOOOO     UU      UU  TTTTTTTTTT      11                    AAAAAA
  OOOOOO     UU      UU  TTTTTTTTTT      11                    AAAAAA
OO      OO   UU      UU      TT        1111                  AA      AA
OO      OO   UU      UU      TT        1111                  AA      AA
OO      OO   UU      UU      TT          11                  AA      AA
OO      OO   UU      UU      TT          11                  AA      AA
OO      OO   UU      UU      TT          11                  AA      AA
OO      OO   UU      UU      TT          11                  AA      AA
OO      OO   UU      UU      TT          11                  AAAAAAAAAA
OO      OO   UU      UU      TT          11                  AAAAAAAAAA
OO      OO   UU      UU      TT          11                  AA      AA
OO      OO   UU      UU      TT          11                  AA      AA
  OOOOOO     UUUUUUUUUU      TT        111111    ________    AA      AA
  OOOOOO     UUUUUUUUUU      TT        111111    ________    AA      AA

             EEEEEEEEEE  RRRRRRRR    RRRRRRRR
             EEEEEEEEEE  RRRRRRRR    RRRRRRRR
             EE          RR      RR  RR      RR
             EE          RR      RR  RR      RR
             EE          RR      RR  RR      RR
             EE          RR      RR  RR      RR
             EEEEEEEE    RRRRRRRR    RRRRRRRR
             EEEEEEEE    RRRRRRRR    RRRRRRRR
             EE          RR  RR      RR  RR
             EE          RR  RR      RR  RR
    ....     EE          RR    RR    RR    RR
    ....     EE          RR    RR    RR    RR
    ....     EEEEEEEEEE  RR      RR  RR      RR
    ....     EEEEEEEEEE  RR      RR  RR      RR
```

FICHIER TEXT D'ERREUR

NETWORK: ANODE_STEP3_TEST
NODE: INP1_A.GRP
VERSION= 2
DATE: 18-AUG-1986

```
  OOOOOO     UU       UU  TTTTTTTTTT      11                    AAAAAA
  OOOOOO     UU       UU  TTTTTTTTTT      11                    AAAAAA
OO      OO   UU       UU      TT       1111                  AA      AA
OO      OO   UU       UU      TT       1111                  AA      AA
OO      OO   UU       UU      TT         11                  AA      AA
OO      OO   UU       UU      TT         11                  AA      AA
OO      OO   UU       UU      TT         11                  AA      AA
OO      OO   UU       UU      TT         11                  AA      AA
OO      OO   UU       UU      TT         11                  AAAAAAAAAA
OO      OO   UU       UU      TT         11                  AAAAAAAAAA
OO      OO   UU       UU      TT         11                  AA      AA
OO      OO   UU       UU      TT         11                  AA      AA
  OOOOOO     UUUUUUUUUU      TT       111111   ____________  AA      AA
  OOOOOO     UUUUUUUUUU      TT       111111   ____________  AA      AA

             LL            OOOOOO      GGGGGGGG
             LL            OOOOOO      GGGGGGGG
             LL          OO      OO  GG
             LL          OO      OO  GG
             LL          OO      OO  GG
             LL          OO      OO  GG
             LL          OO      OO  GG
             LL          OO      OO  GG
             LL          OO      OO  GG  GGGGGG
             LL          OO      OO  GG  GGGGGG
....         LL          OO      OO  GG      GG
....         LL          OO      OO  GG      GG
....         LLLLLLLLLL    OOOOOO      GGGGGG
....         LLLLLLLLLL    OOOOOO      GGGGGG
```

EXEMPLE D'EXECUTION DE SYANODE

```
initialisation completed..
was pr_list updated since last execution (y/n) ?> N
was inc_list updated since last execution (y/n) ?> N
first pass started...
first pass completed..

NETWORK:  ANODE_STEP3_TEST
NODE:     INP1_A GRP
VERSION=          2
DATE:     18-AUG-1986

 computing rap cpu memory...
estimate of CPU/MEM demands (rap) is: 2.03126E-01
PLEASE GIVE YOUR rap VALUE ( 0 IF ABOVE VALUE AGREED
#### VALUE IS ?  > 0
 computing rap cpu memory...
VALUE OF RAP IS  2.03126E-01
graph creation started ...              8970
graph_creation completed...            16440
constructing effective library ...
construction over..
nucleus size reduction (y/n) ?> N
graph agglomeration started...
graph agglomeration completed...
nucleus placement has started...
nucleus placement completed...
fep placement has started...           19680
fep placement completed...             21700
min & max nr of cards.          6         26
better evaluation..
mem SDV :   9.15302E+01
cpu SDV :   1.73917E+02

number of card(s) is :          8
$CPU :   7.15877E-01
$mem :   8.17138E-01
writing all output files ....
....output writing is completed
SYANODE: elapsed cpu time (ms):        28730
```

MANUEL D'UTILISATEUR DU
PROGRAMME SYANODE

ANNEXE VI

VERSION 21 août 1986

1 Introduction

Le manuel d'utilisateur du programme SYANODE a pour objectif d'aider ses utilisateurs à :

- construire les fichiers TEXT d'entrées du programme;
- comprendre les fichiers TEXT résultats du programme;
- intervenir lors de l'apparition d'un fichier TEXT d'erreur;
- pouvoir comprendre et le cas échéant répondre aux messages fournis par SYANODE lors de son exécution;
- modifier certaines valeurs constantes du programme.

Dans un premier point, nous présentons, par des exemples, la syntaxe utilisée dans les différents fichiers TEXT d'entrée. Dans un second point, nous montrons également par des exemples la facon de lire les fichiers TEXT résultats. Dans un troisième point, nous passons en revue les différents messages de SYANODE et montrons comment y répondre le cas échéant. L'avant dernier point permet l'analyse du fichier TEXT d'erreurs et des diagnostics d'erreur et montre comment agir dans ces cas. Le dernier point analyse par ordre alphabétique. La sémantique de chaque constante du programme.

2 Analyse des fichiers TEXT d'entrée

2.1 Fichier TEXT des agrégats

Le fichier TEXT des agrégats a pour nom "**INPUT.AGNAM**"; il contient la description des agrégats. Ce fichier peut être créé par un éditeur de texte quelconque. La syntaxe d'un agrégat est la suivante :

‹AGREGAT› ‹DRIVER› ‹PROCESSUS 1› ... ‹PROCESSUS N›

Explications :

- AGREGAT est le nom d'un agrégat, c'est une chaine de caractères de longueur maximum constante **LMAXAGNAM** et commençant toujours par le caractère "@", ce caractère est le délimiteur de chaque description d'agrégat;

- DRIVER est un caractère indiquant que l'agrégat est composé ou non d'au moins un processus de type driver. Dans l'affirmative l'utilisateur indique un caractère "Y" ou "y" sinon, il indique un caractère "N" ou "n". Remarquons que cette information est redondante par rapport au fichier TEXT des processus où est déjà indiqué qu'un processus est de type driver ou non;

- PROCESSUS i est le ième nom de processus composant l'agrégat; ce nom doit exister dans la liste des processus. Le nombre de processus composants l'agrégat est limité à la constante MAXNCOMPO; i est donc compris entre 1 et **MAXNCOMPO**;

- l'utilisateur est libre de choisir le nombre de blancs ou de RETURN en place des symboles "›" et "‹". Nous conseillons néanmoins à l'utilisateur de limiter chaque description d'agrégat à une ligne dans le fichier TEXT; ceci permet une plus grande lisibilité.

Sur l'exemple de la figure VI.1, nous observons que l'agrégat décrit est de nom "@HD", qu'il est composé d'au moins un processus de type driver (à cause du "y") et que ses processus composant sont de nom "DRHD", "HDLC" et "MDPER".

```
@HD   y   DRHD  HDLC  MDPER
```

Figure VI.1 : exemple d'un agrégat du fichier TEXT des agrégats

Remarque : un exemple plus complet de fichier TEXT des agrégats est présenté en annexe IV.

2.2 Fichier TEXT des groupes

Le fichier TEXT des groupes a pour nom "INPUT.GRP"; il peut être créé par un éditeur de texte quelconque et il contient la description du noeud à configurer et de chacun des groupes.

2.2.1 Description du noeud à configurer

La description du noeud à configurer se trouve dans l'entête du fichier TEXT des groupes, sa syntaxe est la suivante :

```
<"NETWORK"> <NOM DU RESEAU> <CR>
<"NODE">    <NOM DU NOEUD>  <CR>
<"VERS">    <NUMERO DE VERSION> <CR>
<"DATE">    <DATE> <CR>
<CR>
<"CPTYPE">      <TYPE DE CARTE CP> <CR>
<"AVL_MEM_CP">  <MEMOIRE DISPONIBLE> <CR>
<"AVL_MUB_CP">  <MMU BANKS DISPONIBLE> <CR>
<"AVL_PNR_CP">  <PNR DISPONIBLE> <CR>
<"AVL_PW_CP">   <CPU DISPONIBLE> <CR>
```

Explications :

les quatre premières informations sont relatives à la description du noeud :

- au point de vue de la notation, les informations entre "" doivent être écrites comme telles dans le fichier TEXT des groupes et le <CR> signifie un passage à la ligne;

- NOM DU RESEAU est une chaine de caractères de longueur maximale constante NETL indiquant le nom du réseau auquel appartient le noeud à configurer;

- NOM DU NOEUD est une chaine de caractères de longueur maximale constante LNODE indiquant l'identification du noeud de réseau à configurer;

- NUMERO DE VERSION est un entier positif indiquant le numéro de version de la configuration;

- DATE est une chaine de caractères de longueur maximale constante LDATE, le format de la date est libre et n'est pas contrôlé.

L'ordre d'écriture de ces quatres informations est libre.

Les cinq informations suivantes sont relatives à la description de la carte CP du noeud à configurer elles sont séparées des quatre premières par un <CR> :

- TYPE DE CARTE CP est une chaine de caractères de longueur maximale constante LCTP contenant le type de la carte CP utilisée pour la configuration du noeud; aucun contrôle n'est effectué sur cette chaine de caractères par SYANODE;

- MEMOIRE DISPONIBLE est un entier positif indiquant le nombre de kB disponibles sur la carte CP;

- CPU DISPONIBLE est un réel positif indiquant le nombre de ms/s disponibles sur la carte CP;

- MMU BANK DISPONIBLE est un entier positif indiquant le nombre de "MMU banks" disponibles sur la carte CP;

- PNR DISPONIBLE sont quatre entiers positifs représentant le nombre de PNR disponibles sur la carte CP pour des programmes petits, moyens, larges et extra-larges.

  L'ordre d'écriture de ces informations est libre.

La figure VI.2 montre un exemple de description de noeud à configurer : le nom du réseau est "ABCDEF", le nom du noeud est "A", le numéro de la version est 1 et la date le 3-juillet-1986. En ce qui concerne la carte CP, son type est "CP512", il y a des disponibilités de 342 kB, 120 "MMU banks", 750 ms/s et respectivement 120, 60, 30 et 0 PNR pour les programmes petits, moyens, larges et extra-larges sur la carte CP.

```
NETWORK   ABCDEF
NODE      A
VERS      1
DATE      3-juillet-1986

CPTYPE       CP512
AVL_MEM_CP   342
AVL_MUB_CP   120
AVL_PNR_CP   120 60 30 0
AVL_PW_CP    750
```

Figure VI.2 : exemple de description de noeud à configurer dans l'entête du fichier TEXT des groupes

2.2.2 Description du groupe

La syntaxe d'un groupe dans le fichier TEXT des groupes est la suivante :

```
<GROUPE><BI><BS><CARTE CP><PW INTRINSEQUE><PW DRIVER EXTRINSEQUE><HRAM><NBRE
VOISINS> <CR>
<NOM VOISIN 1> <BORNE INFERIEURE> <PW EXTRINSEQUE> <CR>...
<NOM VOISIN N> <BORNE INFERIEURE> <PW EXTRINSEQUE>
```

Explications :

- le ‹CR› signifie un passage à la ligne;

- GROUPE est une chaîne de caractères de longueur maximale constante LMAXGR représentant un nom de processus ou d'agrégat du groupe à placer;

- BI est un entier positif indiquant le numéro inférieur des processus ou agrégats du groupe à placer;

- BS est un entier positif et supérieur ou égal à BI, indiquant le numéro supérieur des processus ou agrégats du groupe à placer;

- CARTE CP est un entier indiquant le numéro de carte CP sur lequel l'utilisateur impose que le groupe soit placé. Sa valeur est constante UNDEFINED si ce numéro n'est pas imposé.

- PW INTRINSEQUE est le réel positif indiquant la consommation CPU intrinsèque en ms/s du groupe à placer;

- PW DRIVER EXTRINSEQUE est le réel positif indiquant la consommation CPU driver extrinsèque du groupe de type driver, en ms/s, nécessaire aux temps header et trailer du driver; si le groupe à placer n'est pas de type driver, cette valeur est nulle;

- HRAM est le réel positif indiquant la consommation mémoire HRAM (en kB) du groupe à placer; ce nombre doit être inférieur ou égal à la valeur HRAM maximale du processus se trouvent dans le fichier TEXT des processus (voir point 2.4);

- NBRE VOISINS est un entier positif indiquant le nombre de voisins du groupe à placer; ce nombre ne peut dépasser la constante MAXNEINB;

- NOM DE VOISIN i est le nom de groupe du voisin du groupe à placer; ce nom à le même sens que le nom de groupe défini ci-dessus. Ce nom doit se trouver dans le fichier TEXT des groupes à placer. La chaîne de caractères constante UNKNOWNNEI signifie que l'utilisateur ne s'intéresse pas au nom du voisin. Ce nom peut être utilisé plusieurs fois dans la liste des voisins d'un même groupe. La borne inférieure n'a pas d'importance;

- BORNE INFERIEURE est un entier strictement positif représentant la borne inférieure du groupe voisin, cette borne ET le nom du voisin décrit juste avant DOIVENT se trouver ensemble dans le fichier TEXT des groupes à placer;

- PW EXTRINSEQUE est le réel positif indiquant la consommation CPU en ms/s du groupe à placer avec le groupe voisin.

La figure VI.3 illustre un exemple de groupe; nous y observons que le groupe de nom "HDLC" a une borne inférieure de 1, une borne supérieure de 4, que son numéro de carte CP est indéfini (valeur UNDEFINED = -1), que sa consommation CPU intrinsèque est de 56.5 ms/s et sa consommation driver extrinsèque est de 0 ms/s (ce groupe n'est

en fait pas de type driver), que sa consommation HRAM est de 0 kB, et qu'il a deux voisins : les groupes de nom "DRHD" et "MDPER" avec leur borne inférieure qui sont respectivement 1 et 1 et la consommation CPU extrinsèque du groupe "HDLC" avec ces deux voisins sont respectivement de 12.25 et 12.35 ms/s.

Remarquons que les descriptions des voisins sont séparées par des ‹CR› et que en dehors du ‹CR› seul le blanc est admis comme séparateur. Pour des raisons de lisibilité, il est conseillé à l'utilisateur de cadrer à droite les noms de voisins ainsi que leur borne inférieure.

```
HDLC 1   4    -1    56.5    0   0   2
                                     DRHD    1   12.25
                                     MDPER   1   12.35
```

Figure VI.3 : exemple de groupe du fichier TEXT des groupes

Remarque : un exemple plus complet de fichier TEXT des groupes est présenté en annexe IV.

2.3 Fichier TEXT des incompatibilités formelles

Le fichier TEXT des incompatibilités formelles a pour nom "INPUT.INCOMP"; il peut être créé par un éditeur de texte quelconque et il contient la description des incompatibilités formelles. La syntaxe d'une incompatibilité formelle est la suivante :

‹NOM 1› ‹"/"› ‹NOM 2› ‹CR›

Explications :

au point de vue de la notation, les informations entre "" doivent être écrites comme telles dans le fichier TEXT des groupes et le ‹CR› signifie un passage à la ligne;

NOM 1 et NOM 2 sont des noms de processus ou d'agrégats éventuellement identiques; ces noms sont séparés par le caractère "/". Ces processus ou agrégats sont incompatibles entre eux. L'utilisateur veillera à garder chaque incompatibilité sur une ligne.

La figure VI.4 illustre une incompatibilité formelle entre le processus de nom "LNKA" et celui de nom "EPST".

LNKA / EPST

Figure VI.4 : exemple d'incompatibilité du fichier TEXT des incompatibilités formelles

Remarque : un exemple plus complet de fichier TEXT des incompatibilités est présenté en annexe IV.

2.4 Fichier TEXT des processus

Le fichier TEXT des processus a pour nom "INPUT.PNAM"; il contient la description des processus. La syntaxe d'un processus est la suivante :

<PROCESSUS><DRIVER><FEP><CLASSE><M1><ROCXT><RWCXT><MTNR><HRAM MAX><CR>

Explications :

- le <CR> indique un passage à la ligne;
- PROCESSUS est une chaine de caractères de longueur maximale LMAXPR contenant le nom du processus;
- DRIVER est un caractère indiquant que le processus est ou non de type driver. Dans l'affirmative l'utilisateur indique un caractère "Y" ou "y" sinon, il indique un caractère "N" ou "n".
- FEP est un caractère indiquant que le processus est ou non de type "FEP". Dans l'affirmative l'utilisateur indique un caractère"N" ou "n" sinon, il indique un caractère "y" ou "Y".
- CLASSE est un entier positif indiquant la classe du processus : 0 pour petit, 1 pour moyen, 2 pour large et 3 pour extra-large;
- M1 est un entier positif indiquant le nombre de kB consommés pour la mémoire initiale du processus;
- ROCXT est un réel positif indiquant la consommation en BYTES du processus pour son contexte read-only;
- RWCXT est un réel positif indiquant la consommation en BYTES du processus pour son contexte read-write;
- MTNR est un réel indiquant le nombre maximum de "threads" acceptés sous un même PNR pour le processus;
- HRAM MAX est un entier positif indiquant le nombre de kB de HRAM maximum consommés par le processus;
- entre ces informations, l'utilisateur placera autant de caractères blancs qu'il désire tout en conservant la description du processus sur une seul ligne.

La figure VI.5 montre un exemple de description de processus : il s'agit du processus de nom "FMGR" qui n'est pas de type driver mais bien de type "FEP", c'est un processus large (2), sa mémoire

initiale est de 42 kB, ses tailles de contexte read-only et read-write sont respectivement de 2926 et 768 bytes, son MTNR est de 36 et son HRAM maximum est de 0 kB.

```
FMGR    n    n    2    42    2926    768    36    0
```

Figure VI.5 : exemple de processus

Remarque : un exemple plus complet de fichier TEXT des processus est présenté en annexe IV.

3 Analyse des fichiers TEXT de résultats

3.1 Fichier TEXT des résultats par groupe

Le fichier TEXT des résultats par groupe, de nom "OUTPUT.GRP", contient le résultat du placement des groupes réalisé par le programme SYANODE. Ces groupes se trouvent dans l'ordre dans lequel ils ont été fournis dans le fichier TEXT des groupes d'entrée. Ce fichier TEXT des résultats par groupe contient une partie de l'entête du fichier TEXT d'entrée des groupes et l'identification de chaque groupe avec son numéro de carte CP d'accueil.

3.1.1 Syntaxe de l'entête

La syntaxe de l'entête du fichier TEXT résultat par groupe est :

```
<"NETWORK:"> <NOM DU RESEAU> <CR>
<"NODE:">    <NOM DU NOEUD> <CR>
<"VERSION="> <NUMERO DE VERSION> <CR>
<"DATE:">    <DATE> <CR>
```

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT des résultats par groupe et le <CR> signifie un passage à la ligne;

- NOM DU RESEAU est une chaine de caractères de longueur maximale constante NETL indiquant le nom du réseau auquel appartient le noeud à configurer;

- NOM DU NOEUD est une chaine de caractères de longueur maximale constante LNODE indiquant l'identification du noeud de réseau à configurer;

- NUMERO DE VERSION est un entier positif indiquant le numéro de version de la configuration;

- DATE est une chaine de caractères de longueur maximale constante LDATE, le format de la date est libre et n'est pas contrôlé par SYANODE.

La figure VI.6 montre un exemple d'entête de description de noeud à configurer : le nom du réseau est "ABCDEF", le nom du noeud est "A", le numéro de version de la configuration est 1 et la date le 3-juillet-1986.

```
NETWORK:  ABCDEF
NODE:     A
VERSION=  1
DATE:     3-juillet-1986
```

Figure VI.6 : exemple d'entête de description de noeud à configurer du fichier TEXT des résultats par groupe

3.1.2 Syntaxe des groupes

L'explication de la syntaxe des groupes nécessite une distinction entre d'une part les groupes de type non "FEP" et d'autre part ceux de type "FEP" :

les groupes de type non "FEP" ont la syntaxe suivante :

‹NOM DU GROUPE› ‹BI› ‹BS› ‹CARTE CP› ‹CR›

Explications :

- le ‹CR› indique un passage à la ligne;
- NOM DE GROUPE est une chaine de caractères de longueur maximale constante LMAXGR représentant un nom de processus ou d'agrégat du groupe placé;
- BI est un entier positif indiquant le numéro inférieur des processus ou agrégats du groupe placé;
- BS est un entier positif et supérieur ou égal à la borne inférieure indiquant le numéro supérieur des processus ou agrégats du groupe placé;
- CARTE CP est un entier positif indiquant le numéro de carte CP sur lequel le programme SYANODE a placé le groupe dans son entièreté.

La figure VI.7 illustre un exemple de groupe non "FEP" en sortie, ce groupe a le nom "SR" ses bornes inférieure et supérieure sont 3 et 4 et son numéro de carte CP de placement est la carte numéro 1.

```
SR   3  4  1
```

Figure VI.7 : exemple de groupe non "FEP" après placement

les groupes "FEP" ont la syntaxe suivante :

‹CR›
‹NOM DU GROUPE› ‹BI› ‹BS› ‹1ère CARTE CP› ‹CR› ...
‹NOM DU GROUPE› ‹BI› ‹BS› ‹Nème CARTE CP› ‹CR›

Explications :

- le ‹CR› indique un passage à la ligne;
- NOM DU GROUPE est une chaine de caractères de longueur maximale constante LMAXGR représentant le nom de processus ou d'agrégat du groupe placé; ce nom apparait autant de fois que le groupe "FEP" a été décomposé;
- BI est un entier positif indiquant le numéro inférieur des processus ou agrégats de la partie du groupe "FEP" placé sur cette ième carte CP;
- BS est un entier positif et supérieur ou égal à BI, indiquant le numéro supérieur des processus ou agrégats de la partie du groupe "FEP" placé sur cette ième carte CP;
- ième CARTE CP est un entier positif indiquant le ième numéro de carte CP sur lequel le programme SYANODE a placé une partie du groupe "FEP".

La figure VI.8 illustre le placement d'un groupe de nom "FMGR" placé sur la carte CP numéro 2 pour la borne 1 à 10 et sur la carte CP numéro 3 pour la borne 11 à 30.

```
FMGR     1      10     2
FMGR     11     30     3
```

Figure VI.8 : exemple de groupe de type "FEP" du fichier TEXT de résultats par groupe

Remarque : un exemple plus complet de fichier TEXT résultat par groupe est présenté en annexe V.

3.2 Fichier TEXT des résultats par carte CP

Le fichier TEXT des résultats par carte CP, de nom "OUTPUT.CRD", contient le résultat du placement des groupes réalisé par le programme SYANODE selon le numéro croissant de carte CP. Ce fichier de résultats contient les résultats globaux du placement, une partie de l'entête du fichier TEXT d'entrée des groupes, l'identification de chaque carte CP avec ce qui a été consommé en mémoire, CPU, MMU et PNR sur cette carte CP et la liste des groupes placés sur chaque carte CP.

3.2.1 Syntaxe de l'entête

La syntaxe de l'entête du fichier TEXT résultat par carte CP est :

```
<"NETWORK:"> <NOM DU RESEAU> <CR>
<"NODE:">    <NOM DU NOEUD> <CR>
<"VERSION="> <NUMERO DE VERSION> <CR>
<"DATE:">    <DATE> <CR>
<CR>
```

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT des résultats par carte CP et le <CR> signifie un passage à la ligne;

- NOM DU RESEAU est une chaine de caractères de longueur maximale constante NETL indiquant le nom du réseau auquel appartient le noeud à configurer;

- NOM DU NOEUD est une chaine de caractères de longueur maximale constante LNODE indiquant l'identification du noeud de réseau à configurer;

- NUMERO DE VERSION est un entier positif indiquant le numéro de version de la configuration;

- DATE est une chaine de caractères de longueur maximale constante LDATE, le format de la date est libre et n'est pas contrôlé par SYANODE.

La figure VI.9 montre un exemple d'entête de description de noeud à configurer : le nom du réseau est "ABCDEF", le nom du noeud est "A", le numéro de la version est 1 et la date le 3-juillet-1986.

```
NETWORK:  ABCDEF
NODE:     A
VERSION=  1
DATE:     3-juillet-1986
```

Figure VI.9 : exemple d'entête de description de noeud à configurer du fichier TEXT résultat par carte CP

3.2.2 Syntaxe des caractéristiques communes aux cartes CP

La syntaxe des caractéristiques communes aux cartes CP est la suivante :

```
<" -----------------CP_CARDS CHARACTERISTICS----------">
<CR>
<"Type : "><TYPE DE CARTE CP> <CR>
<"Available MEMORY (ko) =        "> <MEM DISPONIBLE>  <CR>
<"Available PROCESSING POWER =   "> <CPU DISPONIBLE>  <CR>
<"Available MMUBANKS (#) =       "> <MMUB DISPONIBLE> <CR>
<"Available PNR(s) ( # # # # ) = "> <PNR DISPONIBLE>  <CR>
```

Explications :

- TYPE DE CARTE CP est une chaine de caractères de longueur maximale constante LCTP contenant le type de la carte CP utilisée pour la configuration du noeud;

- MEM DISPONIBLE est un entier positif indiquant le nombre de kB disponibles sur la carte CP;

- CPU DISPONIBLE est un réel positif indiquant le nombre de ms/s disponibles sur la carte CP;

- MMUB DISPONIBLE est un entier positif indiquant le nombre de "MMU banks" disponibles sur la carte CP;

- PNR DISPONIBLE sont quatre entiers positifs représentant le nombre de PNR disponibles sur la carte CP pour des programmes petits, moyens, larges et extra-larges.

La figure VI.10 montre un exemple de carte CP, son type est "CP512", il y a des disponibilités de 342 kB, 120 "MMU banks", 750 ms/s et respectivement 120, 60, 30 et 0 PNR pour les programmes petits, moyens, larges et extra-larges sur la carte CP.

```
------------------CP_CARDS CHARACTERISTICS----------------

Type :       CP512
Available MEMORY (ko) =              342
Available PROCESSING POWER (ms/s) = 750
Available MMUBANKS (#) =             120
Available PNRs ( # # # # ) =         120 60 30 0
```

Figure VI.10 : exemple de description des caractéristiques communes des cartes CP

3.2.3 Syntaxe d'une carte CP

La syntaxe d'une description de carte CP du fichier TEXT de résultats par carte CP avec ses groupes est la suivante :

```
<"-------------------------------------------------------------">‹CR›
<"GROUPS PLACED UPON CARD NUMBER :"> ‹CARTE CP› ‹CR›
<"MEMORY CONSUMPTION IS :"> ‹CONS MEM› ‹"kB"›  ‹CR›
<"PW     CONSUMPTION IS :"> ‹CONS CPU› ‹"ms/s"› ‹CR›
<"MMU    CONSUMPTION IS :"> ‹CONS MMU› ‹"mmu bank(s)"› ‹CR›
<"PNR    CONSUMPTION IS :"> ‹CONS PNR› ‹"pnr"› ‹CR›
‹CR›
‹1er NOM DU GROUPE›  ‹BI› ‹BS› ‹CR›...
‹Nème NOM DU GROUPE›  ‹BI› ‹BS› ‹CR›
<"-------------------------------------------------------------">‹CR›
```

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT des résultats par carte CP et le ‹CR› signifie un passage à la ligne;

- CARTE CP est un entier positif représentant le numéro de la carte CP;

- CONS MEM est un entier positif représentant le nombre de kB consommés sur la carte CP;

- CONS CPU est un réel positif représentant le nombre de ms/s consommés sur la carte CP;

- CONS MMU est un entier positif représentant le nombre de "MMU bank(s)" consommés sur la carte CP;

- CONS PNR est un entier positif représentant le nombre de PNR consommés sur la carte CP;

- ième NOM DE GROUPE est une chaine de caractères représentant le nom du ième groupe placé sur cette carte CP il s'agit d'un nom de processus ou d'agrégat de longueur maximale constante LMAXGR;

- BI est un entier positif indiquant le numéro inférieur des processus ou agrégats du ième groupe placé sur cette carte CP;

- BS est un entier positif et supérieur ou égal à BI, indiquant le numéro supérieur des processus ou agrégats du ième groupe placé sur cette carte CP.

Remarque : un message "! card over-allocated " apparait si la consommation CPU sur la carte CP dépasse la constante AVL_PW_CP dans le fichier TEXT des résultats par carte CP.

La figure VI.11 montre un exemple de carte CP numéro 0 où les consommations mémoire, CPU, MMU et PNR sont respectivement de 60 kB, 5 ms/s, 8 mmu bank(s) et 2 pnr. Les groupes placés sur cette carte CP sont de noms "LUC" et "FMGR" et de bornes inférieures et supérieures 1 à 1 et 1 à 2.

```
------------------------------------------------------------
GROUPS PLACED UPON CARD NUMBER :  0

MEMORY CONSUMPTION IS : 60 kB
PW     CONSUMPTION IS :  5 ms/s
MMU    CONSUMPTION IS :  8 mmu bank(s)
PNR    CONSUMPTION IS :  2 pnr

LUC    1    1
FMGR   1    2
------------------------------------------------------------
```

Figure VI.11 : exemple de carte CP dans le fichier TEXT de résultats par carte CP

3.2.4 Syntaxe des résultats globaux du placement

A la fin du fichier "OUTPUT.CRD" se trouvent les résultats globaux du placement; la syntaxe de ces résultats est la suivante :

```
<"--------------PLACEMENT CHARACTERISTICS-----------------">‹CR›
‹CR›
‹" Allocated        MEMORY          PROCESSING POWER (PW) ">‹CR›
‹"                 ko       %CP     ms/s        %CP       ">‹CR›
‹"Average">        ‹MM›     ‹MMR›    ‹MPW›       ‹MCR› ‹CR›
‹"Standard Deviation"> ‹DM› ‹DMR›   ‹DPW›       ‹DCR› ‹CR›
‹CR›
‹"Minimum  Number of CP_CARDS =">  ‹MINIMUM DE CARTES CP› ‹CR›
‹"Number of USED CARD(S)      =">  ‹CARTES CP UTILISEES › ‹CR›
‹"Maximum  Number of CP_CARDS =">  ‹MAXIMUM DE CARTES CP› ‹CR›
‹CR›
‹"USED MEM-CPU RATIO (rap) =">  ‹RAPPORT CPU-MEMOIRE›
‹CR›
‹"PW  OVERHEAD =">  ‹SURPLUS CPU›      ‹"%"›  ‹CR›
‹"MEM OVERHEAD =">  ‹SURPLUS MEMOIRE›  ‹"%"›  ‹CR›
‹"-------------------------------------------------------">‹CR›
```

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT des résultats par carte CP et le ‹CR› signifie un passage à la ligne;

- MM est un réel positif représentant la moyenne d'occupation mémoire des cartes CP en kB;

- MMR est un réel positif représentant la moyenne d'occupation mémoire des cartes CP par rapport à la taille mémoire d'une carte CP;

- MPW est un réel positif représentant la moyenne d'occupation CPU des cartes CP en ms/s;

- MCR est un réel positif représentant la moyenne d'occupation CPU des cartes CP par rapport à la taille CPU d'une carte CP;

- DM est un réel positif représentant la distribution de la consommation mémoire sur toutes les cartes CP utilisées;

- DMR est un réel positif représentant la distribution de la consommation mémoire sur toutes les cartes CP utilisées, relativisée par rapport à la taille mémoire d'une carte CP;

- DPW est un réel positif représentant la distribution de la consommation CPU sur toutes les cartes CP utilisées;

- DCR est un réel positif représentant la distribution de la consommation CPU sur toutes les cartes CP utilisées, relativisée

par rapport à la taille CPU d'une carte CP;

- MINIMUM DE CARTES CP est un entier positif représentant le nombre minimu de cartes CP qui idéalement auraient dû être consommées;

- CARTES CP UTILISEES est un entier positif représentant le nombre de cartes CP qui ont effectivement été consommées;

- MAXIMUM DE CARTES CP est un entier positif représentant le nombre maximum de cartes CP qui, au pire pourraient être consommées;

- RAPPORT CPU-MEMOIRE est un réel positif indiquant la valeur du rapport CPU-MEMOIRE utilisé pour le placement des groupes;

- SURPLUS MEMOIRE est un entier positif représentant (en %) le surplus de place mémoire dû au non respect complet de l'intégration horizontale lors du placement des groupes;

- SURPLUS CPU est un entier positif représentant (en %) le surplus de consommation CPU dû au non respect complet de l'intégration verticale lors du placement des groupes.

Remarque : en cas de dépassement de la capacité CPU d'au moins une des cartes CP (la constante AVL_PW_CP) un message "! warning card(s) over-allocated" apparait à la fin du fichier TEXT des résultats par carte CP.

La figure VI.12 illustre les caractéristiques globales d'un placement où les moyennes de consommation mémoire et CPU sur les cartes CP sont respectivement de 200 kB et 720 ms/s, les moyennes mémoires et CPU relativisées sont respectivement de 63 et 82 %, les distributions mémoire et CPU sont respectivement de 3.2 et 5.7, les distributions mémoire et CPU relativisées sont respectivement de 0.3 et 0.2, les nombres minimum et maximum de cartes CP sont respectivement 1 et 3, le nombre de cartes CP effectivement utilisées est de 2, le rapport CPU-memoire est de 2.5 et les pourcentages de surplus mémoire et CPU sont respectivement de 0 et 36 %.

```
-------------------PLACEMENT CHARACTERISTICS----------------

Allocated                MEMORY          PROCESSING POWER (PW)
                        ko    %CP        ms/s        %CP
Average                200    63         720         82
Standard Deviation     3.2    0.3          5.7        0.2

 Minimum Number of CP_CARDS IS : 1
 Number of USED CP_CARD(S) IS : 2
 Maximum Number of CP_CARDS IS : 3

Used MEM-CPU ratio (rap) IS : 2.5

 PW  OVERHEAD = 36 %
 MEM OVERHEAD =  0 %
```

Figure VI.12 : exemple de résultats globaux d'un placement dans le fichier TEXT de résultats par carte CP

Remarque : un exemple plus complet de fichier TEXT résultat par carte CP est présenté en annexe V.

3.3 Fichier TEXT des résultats par paire de cartes CP

Le fichier TEXT des résultats par paire de cartes CP, de nom "OUTPUT.EXC" contient le volume des échanges CPU entre chaque paire de cartes CP ayant de tels échanges entre elles suite au placement des groupes effectué par le programme SYANODE.

3.3.1 Syntaxe de l'entête

La syntaxe de l'entête du fichier TEXT résultat par paire de cartes CP :

```
<"NETWORK:"> <NOM DU RESEAU> <CR>
<"NODE:">    <NOM DU NOEUD> <CR>
<"VERSION="> <NUMERO DE VERSION> <CR>
<"DATE:">    <DATE> <CR>
```

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT des résultats par paire de cartes CP et le <CR> signifie un passage à la ligne;

- NOM DU RESEAU est une chaine de caractères de longueur maximale constante NETL indiquant le nom du réseau auquel appartient le noeud à configurer;

- NOM DU NOEUD est une chaine de caractères de longueur maximale constante LNODE indiquant l'identification du noeud de réseau à configurer;

- NUMERO DE VERSION est un entier positif indiquant le numéro de version de la configuration;

- DATE est une chaine de caractères de longueur maximale constante LDATE, le format de la date est libre et n'est pas contrôlé par SYANODE.

La figure VI.13 montre un exemple d'entête de description de noeud à configurer : le nom du réseau est "ABCDEF", le nom du noeud est "A", le numéro de version de la configuration est 1 et la date le 3-juillet-1986.

```
NETWORK:  ABCDEF
NODE:     A
VERSION=  1
DATE:     3-juillet-1986
```

Figure VI.13 : exemple d'entête de description de noeud à configurer du fichier TEXT des résultats par paire de cartes CP

3.3.2 syntaxe des résultats par paire de cartes CP

La syntaxe des résultats par paire de cartes CP est :

<" EXT.PW between CP_CARDs"><CARTE CP ORIGINE><"&"><CARTE CP DESTINATAIRE><"IS"><CPU EXTRINSEQUE> <CR>

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT des résultats par paire de cartes CP et le <CR> signifie un passage à la ligne;

- CARTE CP ORIGINE est un entier positif représentant le numéro de la première carte CP de la paire;

- CARTE CP DESTINATAIRE est un entier positif représentant le numéro de la seconde carte CP de la paire;

- CPU EXTRINSEQUE est un réel strictement positif représentant la quantité de CPU consommée pour les échanges entre les deux cartes CP précitées.

La figure VI.14 illustre un exemple de paire de cartes CP de numéro 1 et 2 qui consomment 30m/s pour leurs échanges entre elles.

EXT.PW between CP_CARDs 1 & 2 is 30 ms/s

Figure VI.14 : exemple de paire de cartes CP
---------- avec échanges CPU

Remarque : un exemple plus complet de fichier TEXT des résultats par paire de cartes CP est présenté en annexe V.

4 Description des messages de SYANODE

Le programme SYANODE se caractérise par la faiblesse de l'intervention humaine; à terme, celle-ci devra devenir nulle.

Actuellement, en cours d'exécution de SYANODE, des messages apparaissent à l'écran et peuvent requérir une réponse de l'utilisateur. En annexe V, le lecteur peut trouver un exemple d'exécution de SYANODE avec ses entrées et sorties. Dans l'ordre chronologique, les messages fournis lors de l'exécution de SYANODE sont les suivants :

- le message "was pr_list updated since last execution (y/n) ? >" permet de demander à l'utilisateur s'il faut traduire le fichier TEXT des processus ("INPUT.PNAM"). L'utilisateur répond "y" ou "Y" s'il a mis à jour ce fichier TEXT depuis la dernière exécution de SYANODE, il répond un "n" ou "N" dans le cas contraire. Tout autre caractère introduit signifie une négation;

- le message "was inc_list updated since last execution (y/n) ? >" permet de demander à l'utilisateur s'il faut traduire le fichier TEXT des incompatibilités formelles ("INPUT.INCOMP"). L'utilisateur répond "y" ou "Y" s'il a mis à jour cette liste depuis la dernière exécution de SYANODE, il répond un "n" ou "N" dans le cas contraire. Tout autre caractère introduit signifie une négation;

- les messages "first pass started..." et "first pass completed" indiquent le début et la fin de la phase d'analyse de cohérence et de traduction du fichier TEXT des groupes. Pas d'intervention de l'utilisateur;

- à l'issue de la première passe, la description du noeud apparait à l'écran; celle-ci a le même format et la même signification que l'entête du fichier TEXT des groupes (cfr point 2.2);

- le message "computing rap cpu memory..." indique que SYANODE calcule le rapport CPU-MEMOIRE. Pas d'intervention de l'utilisateur; nous rappelons ici que ce rapport indique une mesure de la ressource critique : une valeur inférieure à 1 indique que c'est la mémoire; le CPU est la ressource critique dans le cas contraire.

- le rapport CPU-MEMOIRE apparait ensuite avec le message "estimate of CPU/MEM demands (rap) is :". Pas d'intervention de l'utilisateur;

- par le message "PLEASE GIVE YOUR rap VALUE (-1 IF ABOVE VALUE AGREED) ### VALUE OF RAP IS ?" le système demande ici la valeur d'un rapport CPU-MEMOIRE de l'utilisateur. Une valeur 0 ou négative donnée par l'utilisateur aura pour effet de provoquer le calcul de ce rapport par le système;

- suite au message "value of RAP is " le système renvoie la valeur du rapport donné par l'utilisateur ou calculé par lui-même. Pas d'intervention de l'utilisateur;

- le message "nucleus size reevaluation (Y/N) ? " permet de savoir si l'utilisateur veut une taille de noyau réduite. Celle-ci est fixée par défaut à la taille mémoire, cpu, mmu et pnr d'une carte CP. Une réponse "Y" ou "y" de l'utilisateur aura pour effet de réduire cette taille de 10%; une réponse "n" ou "N" ou autre signifie une négation.

- les messages "graph creation started..." et "graph creation completed" indiquent le début et la fin de la phase de création du graphe. Pas d'intervention de l'utilisateur;

- les messages "constructing effective library..." et "construction over" indiquent le début et la fin de la phase de construction de la bibliothèque effective. Pas d'intervention de l'utilisateur;

- les messages "graph agglomeration has started..." et "graph agglomeration completed" indiquent le début et la fin de la phase d'agglomération du graphe. Pas d'intervention de l'utilisateur;

- les messages "nucleus placement has started..." et "nucleus placement completed" indiquent le début et la fin de la phase de placement des noyaux. Pas d'intervention de l'utilisateur;

- les messages "FEP placement has started..." et "FEP placement completed" indiquent le début et la fin de la phase de placement des "FEP". Pas d'intervention de l'utilisateur;

- le message "min & max cards :" est suivi de deux chiffres indiquant les nombres minimum et maximum de cartes CP calculé par SYANODE pour le placement des groupes d'entrée. Pas d'intervention de l'utilisateur;

- les sorties suivantes à l'écran sont les mêmes que celles qui ont été décrites au point 3.2.4, relatives aux résultats globaux du placement mais en moins complet. SYANODE sort les distributions mémoire ("mem SDV") et CPU ("cpu SDV") de la configuration, le nombre de cartes CP effectivement utilisées pour le placement ("number of card(s) is :") et les surplus mémoire ("$MEM") et CPU ("$CPU"). Ces informations peuvent guider l'utilisateur pour qu'il donne un rapport CPU-MEMOIRE lors d'une nouvelle exécution de SYANODE. Ces sorties n'apparaissent que si le placement obtenu est meilleur : le nombre de cartes CP est inférieur au nombre obtenu précédemment ou bien le nombre de cartes CP utilisé est resté le même mais les distributions sont plus faibles. Un message "better evaluation" sera apparu pour indiquer à l'utilisateur que SYANODE a trouvé un meilleur placement que le précédent;

- à la fin des sorties précédentes, SYANODE s'arrêtera et imprimera le message : "please press ‹RETURN› for output files". Ce message permet à l'utilisateur de consulter les résultats du placement avant de continuer. Après sa consultation, l'utilisateur pressera

‹RETURN› et verra ensuite les messages "...writing all output files" et "output writing completed" indiquant le début et la fin de la phase d'écriture des résultats par carte CP, par groupe et par paire de cartes CP dans les fichiers TEXT correspondants;

- le message "continue? (y/n)" apparait uniquement lorsque le nombre de cartes CP obtenu pour le placement n'est pas égal au minimum possible. L'utilisateur a alors la possibilité soit d'arrêter SYANODE (en appuyant "n" ou "N") et de consulter, via un éditeur quelconque, les fichiers TEXT de résultats soit en appuyant sur "y" ou "Y", de continuer et de retourner au début du programme et ainsi de tenter d'améliorer les résultats;

- en cas d'erreur découverte par le système au cours d'une des phases d'exécution, le programme SYANODE s'interrompt et imprime à l'écran le message "GENERAL ERROR PLEASE CONSULT FILE OUTPUT.ERR". L'utilisateur peut consulter ce fichier TEXT avec l'éditeur de son choix et prendre l'action appropriée sur cette base. L'étude de ce fichier TEXT est développée dans le point suivant;

- le dernier message de SYANODE avant terminaison est "SYANODE elapsed cpu time (ms)" suivi de la valeur du temps cpu total de l'exécution du programme.

5 Analyse du fichier TEXT d'erreur

Le fichier TEXT d'erreur, dont le nom est "OUTPUT.ERR", contient les erreurs trouvées dans les fichiers TEXT des groupes ou des incompatibilités formelles lors de l'exécution du programme. Ce fichier peut être consulté par un éditeur de texte quelconque.

5.1 Syntaxe de l'entête

La syntaxe de l'entête du fichier TEXT d'erreur résultat est :

```
<"NETWORK:"> <NOM DU RESEAU> <CR>
<"NODE:">    <NOM DU NOEUD> <CR>
<"VERSION="> <NUMERO DE VERSION> <CR>
<"DATE:">    <DATE> <CR>
<CR>
```

Explications :

- au point de vue de la notation, les informations entre "" sont écrites comme telles dans le fichier TEXT d'erreur et le <CR> signifie un passage à la ligne;
- NOM DU RESEAU est une chaine de caractères de longueur maximale constante NETL indiquant le nom du réseau auquel appartient le noeud à configurer;
- NOM DU NOEUD est une chaine de caractères de longueur maximale constante LNODE indiquant l'identification du noeud de réseau à configurer;
- NUMERO DE VERSION est un entier positif indiquant le numéro de version de la configuration;
- DATE est une chaine de caractères de longueur maximale constante LDATE, le format de la date est libre et n'est pas contrôlé par SYANODE.

La figure VI.15 montre un exemple d'entête de description de noeud à configurer : le nom du réseau est "ABCDEF", le nom du noeud est "A", le numéro de la version est 1 et la date le 3-juillet-1986.

```
NETWORK:  ABCDEF
NODE:     A
VERSION=  1
DATE:     3-juillet-1986
```

Figure VI.15 : exemple d'entête de description de noeud à configurer du fichier TEXT résultat d'erreur

5.2 Syntaxe d'une erreur

La syntaxe d'une erreur est la suivante :

```
<"-------------------------------------------------------------">
<"GR NAME"> <NOM DU GROUPE> <"GNUMBER"> <BI> <CR>
<"ERROR IDENTIFICATION"> <TYPE ERREUR> <CR>
<"ERROR DIAGNOSTIC IS"> <DIAGNOSTIC D'ERREUR><CR>
<"ACTION EXPECTED"> <ACTION><CR>
<"-------------------------------------------------------------">
```

Explications :

- NOM DU GROUPE est une chaine de caractères de longueur maximale LMAXGR représentant le nom de processus ou d'agrégat du groupe erroné;

- BI est un entier positif indiquant le numéro inférieur des processus ou agrégats du groupe erroné;

- TYPE ERREUR est un message pouvant être soit "urgency is WARNING" soit "urgency is FATAL" selon que l'erreur détectée n'est qu'un avertissement ou une erreur fatale;

- ACTION est un message pouvant être soit "file modification advised" soit "file modification required" selon le type d'erreur;

- DIAGNOSTIC D'ERREUR est le code de diagnostic de l'erreur rencontrée. Ces codes peuvent être :

  - "ERR EXIST", ce type d'erreur apparait quand un nom de groupe est inexistant dans le fichier TEXT des processus ou dans le fichier TEXT des agrégats; cette erreur est fatale, elle provoque l'arrêt de SYANODE et requiert une intervention humaine obligatoire dans le fichier TEXT des groupes pour le corriger ou dans le fichier TEXT des processus ou des agrégats pour décrire ce nom;

  - "ERR FORMAT", ce type d'erreur apparait quand une erreur de format est détectée dans un groupe; cette erreur est fatale, elle provoque l'arrêt de SYANODE et requiert une intervention humaine obligatoire dans le fichier TEXT des groupes afin de la corriger; les raisons possibles de ce type d'erreur sont :

    - la consommation CPU intrinsèque est négative ou supérieure à la constante AVL_PW_CP;

    - la borne inférieure du groupe n'est pas inférieure ou égale à sa borne supérieure;

    - le nombre de voisins est négatif ou supérieur à la constante MAXNEINB;

- la quantité HRAM est négative ou supérieure à la constante AVL_MEM_CP;

- la puissance driver extrinsèque n'est pas positive;

- la consommation CPU extrinsèque avec un ou plusieurs groupes voisins n'est pas positive.

- "ERR OVF CPU", la consommation CPU intrinsèque du groupe est supérieure à la constante AVL_PW_CP et le groupe n'est pas de type "FEP"; cela produit un message d'avertissement uniquement dans le fichier TEXT d'erreur et sur le fichier des résultats par carte CP. Suite à cette erreur, le programme ne s'arrête pas, mais enregistre un message d'avertissement dans le fichier TEXT des erreurs. Il est conseillé à l'utilisateur de vérifier et éventuellement de modifier la valeur coupable;

- "ERR OVF MEM", la consommation mémoire du groupe est supérieure à la constante AVL_MEM_CP et le groupe n'est pas de type "FEP"; ce type d'erreur est fatal et requiert une intervention de l'utilisateur dans le fichier TEXT des groupes pour modifier la valeur en cause;

- "ERR OVF MMU", la consommation MMU du groupe est supérieure à la constante AVL_MUB_CP et le groupe n'est pas de type "FEP"; ce type d'erreur est fatal et requiert une intervention de l'utilisateur dans le fichier TEXT des groupes pour modifier la valeur en cause;

- "ERR INC", ce type d'erreur apparait quand une incompatibilité est détectée dans un groupe; ce type d'erreur est fatal et requiert une intervention de l'utilisateur dans le fichier TEXT des groupes afin de modifier ce groupe;

- "ERR FOR INC", ce type d'erreur signifie qu'une erreur de format a été détectée dans le fichier TEXT des incompatibilités formelles; en général, il s'agit d'un oubli du caractère "/" entre les noms de processus ou d'agrégats incompatibles, ou l'oubli d'un des deux membres de l'incompatibilité formelle. Ce type d'erreur est fatal et requiert une intervention de l'utilisateur dans le fichier TEXT des incompatibilités formelles;

- "ERR EX INC", ce type d'erreur indique la présence d'un nom de processus ou d'agrégat non reconnu dans une incompatibilité formelle; ce type d'erreur est fatal et requiert une intervention de l'utilisateur soit dans le fichier TEXT des incompatibilités formelles, dans celui des processus ou dans celui des agrégats;

- "ERR NEIB", ce type d'erreur signifie qu'un voisin n'a pas été reconnu soit à cause de son nom ou à cause du nom et de la borne inférieure qui ne sont pas présents dans le fichier TEXT des groupes. Cette erreur est fatale et requiert une

modification par l'utilisateur du fichier TEXT des groupes;

- "ERR SAME NAME", ce type d'erreur signifie qu'un groupe est de même nom et borne inférieure qu'un de ses voisins. Cette erreur est fatale et requiert une modification dans le fichier TEXT des groupes.

La figure VI.16 illustre un exemple d'erreur relative au groupe de nom "DDLC" et de borne inférieure 1, l'erreur est fatale et requiert une intervention de l'utilisateur dans le fichier TEXT des groupes, le diagnostic d'erreur (ERR EXIST) indique que le nom du groupe est erroné.

```
------------------------------------------------------------
GR NAME : DDLC    GNUMBER : 1
ERROR IDENTIFICATION : urgency is FATAL
error diagnostic : ERR EXIST
ACTION EXPECTED : file modification required
------------------------------------------------------------
```

Figure VI.16 : exemple d'erreur du fichier TEXT d'erreur

6 Description des constantes de SYANODE

Le fichier TEXT de nom "SYANODE.CONS" contient les valeurs des constantes utilisées dans le programme SYANODE. Par éditeur de texte il est possible de modifier ces valeurs. Les constantes sont classées par ordre alphabétique. Leur nom et sémantique sont :

- ERR_EX = 'ERR EXIST' { voir point 5.2 pour description }
- ERR_FOR = 'ERR FORMAT' { voir point 5.2 pour description }
- ERR_FOR_INC = 'ERR FOR INC' { voir point 5.2 pour description }
- ERR_EX_INC = 'ERR EX INC' { voir point 5.2 pour description }
- ERR_INC = 'ERR INCO' { voir point 5.2 pour description }
- ERR_NEI = 'ERR NEIB' { voir point 5.2 pour description }
- ERR_NEI_SAME = 'ERR SAME NAME' { voir point 5.2 pour description }
- ERR_OVF_CPU = 'ERR OVF CPU' { voir point 5.2 pour description }
- ERR_OVF_MEM = 'ERR OVF MEM' { voir point 5.2 pour description }
- ERR_OVF_MMU = 'ERR OVF MMU' { voir point 5.2 pour description }
- ERTPL = 13 { longueur maximale d'une chaine de caractères pour un diagnostic d'erreur }
- FATAL_ERR = true { identification d'un diagnostic d'erreur fatal }
- IMP_ABS = 'IMPOSSIBLE ABSOLUTE' { constante pour une arête de type impossible absolue }
- IMP_TEMP = 'IMPOSSIBLE TEMPORARY' { constante pour une arête de type impossible temporaire }
- IMP = 'IMPOSSIBLE' { constante pour une arête de type impossible }
- POSSIBLE = 'POSSIBLE' { constante pour une arête de type possible }
- LTCP = 10 { longueur maximale, en caractères, d'un nom de type de carte CP }
- LDATE = 30 { longueur maximale, en caractères, d'une date }
- LIDSTR = 4 + LMAXGR { longueur, en caractères, de l'identifiant IDSTR d'un record GROUP }

- LNODE = 30 { longueur maximale, en caractères, d'un nom du noeud de réseau à configurer }

- LMAXAGNAM = LMAXGR { longueur maximale, en caractères, d'un nom d'agrégat AGREGAT_NAME }

- LMAXCOMP = (MAXNCOMPO * (LMAXGR + 1)) -1 { longueur maximale, en caractères, d'une composition d'agrégat : le nombre de type de processus multiplié par la longeur en caractères maximale d'un processus et un blanc entre chaque nom de processus }

- LMAXGR = 12 { longueur maximale, en caractères, d'un nom de groupe GROUP_NAME }

- LMAXPR = LMAXGR { longueur maximale, en caractères, d'un nom de processus PROCESS_NAME }

- LMAXTYPEDGE = 20 { longueur maximale, en caractères, d'un type d'arête TYPEDGE }

- MAXAG = 10 { nombre maximum de type d'agrégat, nombre de lignes dans le fichier TEXT "INPUT.AGNAM" }

- MAXBIB = 50 { nombre maximum d'incompatibilités effectives }

- MAXCARDNB = 255 { numéro maximum d'une carte CP }

- MAXFILENAME = 20 { longueur maximale, en caractères, d'un nom de fichier }

- MAXGR = 800 { nombre maximum de groupes d'entrée dans le fichier TEXT "INPUT.GRP"}

- MAXINC = 50 { nombre maximum d'incompatibilités formelles, nombre de lignes du fichier TEXT "INPUT.INC" }

- MAXNEINB = 15 { nombre maximum de voisins par groupes dans le fichier TEXT "INPUT.GRP" }

- MAXNCOMPO = 6 { nombre maximum de types de processus pouvant composer un agrégat }

- MAXNUC = 800 { nombre maximum de noyaux après fusion }

- MAXPART = 10 { nombre maximum en lequel un "FEP" peut être décomposé }

- MAXPREC = 10 { nombre maximum de cartes CP avec un "FEP" déjà placé du type de celui traité MAXPREC doit être au plus égal à MAXPART }

- MIN_CPU_QTY = 0 { capacité CPU résiduelle d'une carte CP avant qu'elle ne soit considérée comme pleine }

- MIN_MEM_QTY = 0 { capacité mémoire résiduelle d'une carte CP avant qu'elle ne soit considérée comme pleine }

- MIN_MMU_QTY = 0 { capacité MMU résiduelle d'une carte CP avant qu'elle ne soit considérée comme pleine }

- MIN_PNR_QTY = 0 { capacité en PNR résiduelle d'une carte CP avant qu'elle ne soit considérée comme pleine }

- NB_SORT = 4 { nombre actuel de méthodes de tri des noyaux avant le placement des noyaux sur les cartes }

- NETL = 30 { longueur maximale d'un nom de réseau dans le fichier TEXT des groupes "INPUT.GRP" }

- PCTL = 1 { longueur de la TCT en page RAM de 1 kB }

- PGL = 1024 { longueur d'une page RAM en BYTES }

- UNDEFINED = -1 { valeur standard de ce qui est indéfini }

- UNKNOWNEI = 'XXX' { chaine de caractères correspondant à un voisin indéfini }

- WARNING = false { constante de diagnostic d'erreur d'avertissement }

DONNEES ET RESULTATS DE LA
METHODE ALEATOIRE

ANNEXE VII

LISTE DE CENT CINQUANTE GROUPES

NUM	MEM	CPU
1	40	25
2	18	20
3	2	12
4	5	13
5	4	3
6	6	5
7	12	9
8	6	4
9	4	5
10	3	4
11	25	40
12	20	18
13	12	2
14	13	5
15	3	4
16	5	6
17	9	12
18	4	6
19	5	4
20	4	3
21	13	3
22	14	9
23	15	1
24	16	44
25	2	12
26	10	4
27	5	9
28	5	10
29	10	5
30	10	3
31	3	13
32	9	14
33	1	15
34	44	16
35	12	2
36	4	10
37	9	5
38	10	5
39	5	10
40	3	10
41	7	40
42	5	30
43	9	20
44	18	5
45	22	3
46	12	0
47	2	0
48	17	1
49	3	0
50	5	1
51	40	7
52	30	5
53	20	9
54	5	18
55	3	22
56	0	12
57	0	2
58	1	17
59	0	3
60	1	5
61	80	20
62	10	20
63	1	20
64	2	20
65	3	20
66	1	0
67	3	0
68	20	80
69	20	10
70	20	1

71	20	2
72	20	3
73	0	1
74	0	3
75	7	8
76	17	9
77	12	23
78	15	19
79	20	2
80	8	15
81	10	10
82	2	10
83	5	2
84	4	2
85	8	7
86	9	17
87	3	12
88	19	15
89	2	20
90	15	8
91	10	10
92	10	2
93	22	5
94	2	4
95	83	69
96	2	3
97	1	1
98	1	23
99	12	2
100	1	2
101	19	30
102	4	9
103	21	18
104	1	3
105	28	17
106	13	10
107	6	7
108	5	2
109	2	2
110	1	2
111	39	47
112	20	26
113	7	5
114	6	1
115	4	1
116	1	10
117	8	2
118	15	8
119	2	24
120	3	19
121	2	8
122	1	4
123	1	1
124	1	7
125	1	2
126	37	7
127	10	1
128	5	7
129	29	6
130	3	4
131	3	1
132	1	7
133	1	2
134	19	24
135	4	23
136	21	13
137	1	13
138	28	5
139	13	1
140	6	1
141	5	10
142	2	2
143	1	8
144	18	34
145	23	35
146	10	3
147	9	1
148	15	23
149	12	2
150	13	2

LISTE DE SOIXANTE GROUPES

NUM	MEM	CPU
1	30	54
2	24	12
3	2	4
4	4	4
5	10	1
6	7	11
7	2	5
8	1	6
9	1	1
10	1	2
11	54	30
12	12	24
13	4	2
14	4	4
15	1	10
16	11	7
17	5	2
18	6	1
19	1	1
20	2	1
21	80	23
22	4	34
23	2	21
24	3	14
25	1	3
26	0	2
27	1	2
28	0	1
29	1	0
30	0	0
31	23	80
32	34	4
33	21	2
34	14	3
35	3	1
36	2	0
37	2	1
38	1	0
39	0	1
40	0	0
41	4	11
42	43	42
43	14	14
44	20	13
45	2	15
46	2	2
47	2	2
48	1	1
49	3	0
50	0	0
51	11	4
52	42	43
53	14	14
54	13	20
55	15	2
56	2	2
57	2	2
58	1	1
59	0	3
60	0	0

RESULTATS DE LA LISTE DE SOIXANTE GROUPES

CLASSE DE TRI	MEM	CPU	NB CARTES	VAR. MEM.	VAR. CPU
ORDRE DONNE	1.00000E+02	1.00000E+02	6	0.00000E+00	0.00000E+00
ALEATOIRE	7.50000E+01	7.50000E+01	8	4.59429E+02	8.42857E+02
MEMOIRE DECR.	7.50000E+01	7.50000E+01	8	1.45686E+03	1.09600E+03
CPU DECR.	7.50000E+01	7.50000E+01	8	8.41429E+02	1.68429E+03
DECR.ABS	8.57143E+01	8.57143E+01	7	1.02290E+03	1.02290E+03
CROISS.ABS	8.57143E+01	8.57143E+01	7	2.30571E+02	7.77238E+02
1 MEM, 1 CPU	8.57143E+01	8.57143E+01	7	1.36290E+03	1.36290E+03

CLASSE DE TRI	MEM	CPU	NB CARTES	VAR. MEM.	VAR. CPU
1 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.12514E+03	1.34200E+03
2 ALEATOIRE	8.57143E+01	8.57143E+01	7	2.48905E+02	8.12905E+02
3 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.34571E+02	4.30905E+02
4 ALEATOIRE	7.50000E+01	7.50000E+01	8	5.30286E+02	6.95429E+02
5 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.89905E+02	4.17905E+02
6 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.65238E+02	4.71905E+02
7 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.51238E+02	3.70571E+02
8 ALEATOIRE	8.57143E+01	8.57143E+01	7	2.38571E+02	7.00905E+02
9 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.76905E+02	4.82905E+02
10 ALEATOIRE	7.50000E+01	7.50000E+01	8	8.10857E+02	4.64000E+02
11 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.19857E+03	2.92286E+02
12 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.58571E+02	4.93238E+02
13 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.31238E+02	4.04905E+02

14 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.28429E+03	9.33143E+02
15 ALEATOIRE	8.57143E+01	8.57143E+01	7	1.02571E+02	4.58905E+02
16 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.75571E+02	5.31238E+02
17 ALEATOIRE	8.57143E+01	8.57143E+01	7	4.05571E+02	5.86905E+02
18 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.06486E+03	9.54857E+02
19 ALEATOIRE	7.50000E+01	7.50000E+01	8	7.82000E+02	6.51429E+02
20 ALEATOIRE	8.57143E+01	8.57143E+01	7	6.55905E+02	2.82238E+02
21 ALEATOIRE	8.57143E+01	8.57143E+01	7	1.17905E+02	7.91905E+02
22 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.04905E+02	1.20957E+03
23 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.90571E+02	3.48905E+02
24 ALEATOIRE	8.57143E+01	8.57143E+01	7	4.93905E+02	3.74238E+02
25 ALEATOIRE	8.57143E+01	8.57143E+01	7	2.59238E+02	5.76571E+02
26 ALEATOIRE	7.50000E+01	7.50000E+01	8	6.24000E+02	9.39143E+02
27 ALEATOIRE	7.50000E+01	7.50000E+01	8	9.13143E+02	1.58714E+03
28 ALEATOIRE	8.57143E+01	8.57143E+01	7	4.33571E+02	5.31905E+02
29 ALEATOIRE	8.57143E+01	8.57143E+01	7	4.29905E+02	4.45571E+02
30 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.57905E+02	5.96238E+02
31 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.27229E+03	7.20000E+02
32 ALEATOIRE	8.57143E+01	8.57143E+01	7	2.78571E+02	4.75571E+02
33 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.02229E+03	1.32857E+03

34 ALEATOIRE	8.57143E+01	8.57143E+01	7	4.28238E+02	3.10238E+02
35 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.33029E+03	8.42571E+02
36 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.57905E+02	8.05571E+02
37 ALEATOIRE	7.50000E+01	7.50000E+01	8	8.26857E+02	7.72000E+02
38 ALEATOIRE	8.57143E+01	8.57143E+01	7	4.45905E+02	1.65905E+02
39 ALEATOIRE	8.57143E+01	8.57143E+01	7	5.95571E+02	3.64905E+02
40 ALEATOIRE	8.57143E+01	8.57143E+01	7	6.53905E+02	3.55905E+02
41 ALEATOIRE	7.50000E+01	7.50000E+01	8	8.46000E+02	8.53429E+02
42 ALEATOIRE	8.57143E+01	8.57143E+01	7	2.42905E+02	3.31238E+02
43 ALEATOIRE	8.57143E+01	8.57143E+01	7	2.91571E+02	3.93571E+02
44 ALEATOIRE	7.50000E+01	7.50000E+01	8	4.86000E+02	9.79429E+02
45 ALEATOIRE	7.50000E+01	7.50000E+01	8	1.14314E+03	1.26514E+03
46 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.68238E+02	2.21571E+02
47 ALEATOIRE	8.57143E+01	8.57143E+01	7	8.08238E+02	2.98571E+02
48 ALEATOIRE	8.57143E+01	8.57143E+01	7	7.75714E+01	7.76905E+02
49 ALEATOIRE	8.57143E+01	8.57143E+01	7	3.72238E+02	4.46238E+02
50 ALEATOIRE	8.57143E+01	8.57143E+01	7	1.21905E+02	4.14905E+02

RESULTATS DE LA LISTE DE CENT CINQUANTE GROUPES

CLASSE DE TRI	MEM	CPU	NB CARTES	VAR. MEM.	VAR. CPU
ORDRE DONNE	1.00000E+02	1.00000E+02	16	0.00000E+00	0.00000E+00
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.58575E+02	4.19281E+02
MEMOIRE DECR.	7.61905E+01	7.61905E+01	21	1.41276E+03	9.55862E+02
CPU DECR.	7.61905E+01	7.61905E+01	21	6.51562E+02	1.29276E+03
DECR.ABS	9.41176E+01	9.41176E+01	17	3.68610E+02	3.72360E+02
CROISS.ABS	8.88889E+01	8.88889E+01	18	1.75869E+02	3.75046E+02
1 MEM, 1 CPU	9.41176E+01	9.41176E+01	17	3.40610E+02	3.31360E+02

CLASSE DE TRI	MEM	CPU	NB CARTES	VAR. MEM.	VAR. CPU
1					
ALEATOIRE	8.88889E+01	8.88889E+01	18	1.86222E+02	6.99281E+02
2					
ALEATOIRE	9.41176E+01	9.41176E+01	17	1.59735E+02	1.75485E+02
3					
ALEATOIRE	8.88889E+01	8.88889E+01	18	5.79399E+02	4.93752E+02
4					
ALEATOIRE	8.42105E+01	8.42105E+01	19	6.05287E+02	3.61620E+02
5					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.42105E+02	6.15046E+02
6					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.65869E+02	1.64575E+02
7					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.47869E+02	4.77752E+02
8					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.02222E+02	3.74222E+02
9					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.01163E+02	4.02340E+02
10					
ALEATOIRE	8.42105E+01	8.42105E+01	19	4.87398E+02	7.14509E+02
11					
ALEATOIRE	8.42105E+01	8.42105E+01	19	3.65064E+02	5.64398E+02
12					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.62340E+02	2.29987E+02
13					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.92105E+02	4.33752E+02
14					
ALEATOIRE	9.41176E+01	9.41176E+01	17	1.61735E+02	5.47353E+01
15					
ALEATOIRE	8.88889E+01	8.88889E+01	18	1.72693E+02	5.06340E+02
16					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.37634E+02	5.30340E+02
17					
ALEATOIRE	8.42105E+01	8.42105E+01	19	5.99953E+02	6.15398E+02
18					
ALEATOIRE	8.42105E+01	8.42105E+01	19	6.35398E+02	7.48509E+02
19					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.44340E+02	4.40222E+02
20					
ALEATOIRE	9.41176E+01	9.41176E+01	17	1.40735E+02	2.05235E+02
21					
ALEATOIRE	9.41176E+01	9.41176E+01	17	9.29853E+01	3.92353E+01
22					
ALEATOIRE	9.41176E+01	9.41176E+01	17	1.37110E+02	1.32360E+02

23					
ALEATOIRE	9.41176E+01	9.41176E+01	17	2.27110E+02	1.73610E+02
24					
ALEATOIRE	8.88889E+01	8.88889E+01	18	2.87399E+02	5.60810E+02
25					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.47869E+02	3.14272E+02
26					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.07516E+02	4.61516E+02
27					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.06810E+02	5.19281E+02
28					
ALEATOIRE	9.41176E+01	9.41176E+01	17	2.68360E+02	1.18610E+02
29					
ALEATOIRE	8.88889E+01	8.88889E+01	18	2.48458E+02	3.96458E+02
30					
ALEATOIRE	9.41176E+01	9.41176E+01	17	5.72353E+01	1.07485E+02
31					
ALEATOIRE	8.88889E+01	8.88889E+01	18	6.03634E+02	6.02810E+02
32					
Aleatoire	8.88889e+01	8.88889e+01	18	3.88272e+02	3.39516e+02
33					
ALEATOIRE	8.88889E+01	8.88889E+01	18	5.49869E+02	2.63046E+02
34					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.18575E+02	6.01046E+02
35					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.48340E+02	4.47046E+02
36					
ALEATOIRE	8.42105E+01	8.42105E+01	19	8.22509E+02	6.85509E+02
37					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.06458E+02	3.72928E+02
38					
ALEATOIRE	8.88889E+01	8.88889E+01	18	5.93869E+02	4.19399E+02
39					
ALEATOIRE	8.42105E+01	8.42105E+01	19	7.76509E+02	9.06176E+02
40					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.59163E+02	3.34693E+02
41					
ALEATOIRE	8.88889E+01	8.88889E+01	18	2.60575E+02	4.63516E+02
42					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.32575E+02	3.82575E+02
43					
ALEATOIRE	8.88889E+01	8.88889E+01	18	3.37987E+02	5.21281E+02
45					
ALEATOIRE	9.41176E+01	9.41176E+01	17	1.01110E+02	1.55110E+02
46					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.25516E+02	2.31046E+02
47					
ALEATOIRE	9.41176E+01	9.41176E+01	17	7.91103E+01	4.11103E+01
48					
ALEATOIRE	9.41176E+01	9.41176E+01	17	7.79853E+01	1.00235E+02
49					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.45869E+02	2.61163E+02
50					
ALEATOIRE	8.88889E+01	8.88889E+01	18	4.99752E+02	2.72575E+02