Techniques d'implantation de programmes parallèles sur architectures réparties

Nibal Idlebi

Université de Nancy 1 U.E.R Sciences Mathématiques

Centre de Recherche en Informatique de Nancy
INRIA Lorraine

THESE

pour l'obtention du

Doctorat de l'Université de Nancy 1
(Spécialité Informatique)

par

Nibal IDLEBI

Techniques d'implantation de programmes parallèles sur architectures réparties

Présentée et soutenue le 16 Janvier 1991

Devant la commision composée de :

Président:	Jean-Claude Derniame
Rapporteurs:	Françoise André
	Claude Kirchner
Examinateurs:	Jean-Pierre Finance
	Guy-René Perrin

A travers ces lignes, je tiens à remercier tous ceux qui m'ont aidée pour mener ce travail:

Monsieur Jean-Pierre FINANCE, Professeur à l'Université de Nancy 1 et Directeur du CRIN, qui m'a accueillie dans son équipe Prograis et qui a accepté d'être mon directeur de recherche. Il m'a guidée pour apprendre la méthodologie du travail et de la résolution de problèmes informatiques. Je le remercie sincèrement pour ses multiples interventions et sa bienveillante exigence, et je profite de l'occasion pour lui faire part de toute ma gratitude.

Monsieur Guy-René PERRIN, Professeur à l'Université de Besançon, qui a suivi ce travail. Je lui dois énormément pour ses compétences et son expérience. Ses suggestions constructives, ses encouragements et sa gentillesse m'ont été très précieux tout au long de ce travail. Je tiens à lui exprimer toute ma reconnaissance.

Monsieur Claude KIRCHNER, Directeur de Recherche à l'INRIA Lorraine, pour avoir accepter d'être rapporteur de ce mémoire. Il a été un lecteur critique sachant mettre l'accent sur les points clés de ce travail.

Madame Françoise ANDRE, Professeur à l'Université de Rennes, qui a pris de son temps pour être rapporteur de ce travail. Ses remarques et ses critiques m'ont permis de l'améliorer considérablement.

Monsieur Jean-Claude DERNIAME, Professeur à l'Université de Nancy 1, le président de ce jury.

J'adresse mes sincères remerciments à Francis ALEXANDRE. Sa lecture attentive a contribué à l'amélioration de ce texte.

Je remercie Dominique MERY, François CARREZ, Noureddine LAZRAK et tous les membres de l'équipe COMETE qui ont participé à l'avancement de ce travail avec leurs remarques et leurs questions.

Merci à tous mes amis du CRIN pour leur appui amical et pour leur présence rassurante, en particulier Hani TOLBA.

A mes racines.

A l'eau de ma vie.

A toutes mes fleurs.

Implementation techniques for the distribution of parallel programs

Abstract

The aim of this thesis is to define techniques for the implementation of parallel programs on distributed architectures.
The distribution of parallel program requires the assignement of its component parts to nodes in distributed architecture and the representation of its different opertaions on such an architecture. In this study a solution for these two problems is presented.
In order to generalise this study a formalisation of the implementation techniques which are required for the distribution of parallel programs is proposed. This formalisation allows the introduction of technical constraints related to a particular distributed architecture or a given programming language to be differentiated.
On the experimental side, a software tool has been developed to allow implementation of parallel programs on T-NODE : distributed memory parallel computers.

Résumé

L'objectif de notre travail est l'étude et la définition de techniques d'implantation de programmes parallèles sur des architectures réparties.
La répartition d'un programme parallèle nécessite le placement de ses composants sur les sites de l'architecture répartie et la représentation de ses différentes opérations sur une telle architecture. Dans cette thèse nous présentons une résolution de ces deux problèmes.
En vue de la généralisation de l'étude nous proposons une formalisation des techniques d'implantation. Cette formalisation permet de différer l'introduction des détails techniques liés à une architecture répartie donnée ou à un langage de programmation particulier.
D'un point de vue experimental nous développons un outis logiciel permettant l'implantation de programmes parallèles sur la machine parallèle à mémoire distribuée : T-NODE.

Mots-clés

Application parallèle, concurrence, communication, asynchronisme, répartition, système réparti, architecture répartie, formalisation, abstraction, types abstraits, spécification, implantation, placement, règles heuristiques, dérivation, représentation, compilation.

Table des matières

Introduction

0.1 Contexte du travail

Avec le développement des architectures réparties, des réseaux et des machines parallèles à mémoire distribuée, nous pouvons nous attendre à l'évolution de la recherche dans le domaine de la répartition. Cette recherche concerne les architectures, les systèmes d'exploitation et les applications répartis.

Les applications réparties seront vraisemblablement mises en œuvre sur des systèmes répartis, offrant des mécanismes de base analogues à ceux des systèmes d'exploitation mais traitant de façon plus complète les aspects liés à la répartition : processus, synchronisation et communication.

Il est intéressant de noter que la répartition des éléments d'un système n'est pas nécessairement la solution la plus économique et la plus simple à réaliser. Néanmoins des considérations de disponibilité, de commodité d'utilisation ou d'adaptation à la structure de l'application peuvent conduire à décentraliser les fonctions d'un système entre un grand nombre de processeurs spécialisés.

Par ailleurs la recherche de la puissance de calcul et de la minimisation du temps d'exécution conduisent à répartir les tâches d'un programme parallèle sur les processeurs d'un réseau ou d'une machine parallèle.

Devant l'intérêt et le progrès de la répartition, la notion d'algorithmique distribuée est apparue. Cette notion reste encore difficile aussi bien au niveau de la conception qu'à celui de la programmation. La difficulté est due aux interactions (communication, coopération, concurrence) entre les éléments distribués sur les différentes machines de l'architecture répartie. Ces interactions ne nous permettent pas de spécifier complètement ni facilement le comportement global d'un programme exécuté sur une telle architecture.

Les algorithmes répartis sont des algorithmes parallèles écrits pour une machine virtuelle, composée de plusieurs processeurs qui ne partagent pas de mémoire. En conséquence leur conception est proche de celle de certains types d'algorithmes parallèles.
Le domaine de la programmation parallèle est en effet un domaine vaste et très varié qui va de la programmation des machines parallèles, vectorielles, piplinées, jusqu'à la programmation de multiprocesseurs à mémoire distribuée. Dans ce domaine on s'intéresse à la conception des

langages d'expression du parallélisme, à la sémantique de ces langages, aux systèmes de preuve associés et au développement des applications spécifiques à certaines architectures parallèles. Parmi ces différents axes du domaine de programmation parallèle **nous nous intéressons dans le cadre de cette thèse à l'étude et à la définition de techniques d'implantation des applications parallèles sur des architectures réparties.**

Ce travail a été réalisé dans le cadre de projet COMETE (COMmunication Et TEmps), développé au sein des laboratoires CRIN à Nancy et LIB à Besançon et soutenu par le projet CNRS C^3.

Le projet COMETE se préoccupe de méthodologie de construction de programmes parallèles. Les principaux thèmes de recherche développés dans ce projet sont les suivants :

- la définition d'un cadre formel pour la spécification des problèmes et l'expression de leurs solutions,
- l'étude de la représentation algorithmique des solutions obtenues en termes de systèmes de processus communicants, de leur programmation et de leur validité.

La notion de communication proposée dans [Per85] constitue l'outil de base de la démarche proposée. Cette démarche consiste à concevoir un système parallèle comme un ensemble de processus indépendants qui communiquent entre eux par l'intermédiaire d'objets de communication. Chaque objet est caractérisé par un type de communication exprimant la relation qui existe entre les suites de valeurs échangées entre les processus.

Le langage du projet COMETE est basé sur le concept de processus communicants qui est un des concepts de base de l'algorithmique distribuée. Dans notre travail, ce langage nous inspire pour définir les applications parallèles et leurs images sur une architecture répartie.

Du point de vue de l'architecture, les machines qui nous intéressent font partie de la classe MIMD (Multiple Instruction stream Multiple Data stream). Selon la classification de Hockney [Hoc85], cette classe représente toutes les architectures constituées de plusieurs entités de base : processeurs, mémoires, nœuds de commutation reliés par un réseau d'interconnexions. Dans cette classe nous considérons ce qu'on appelle maintenant les machines DMPC (Distributed Memory Parallel Computers).

0.2 Objectif du travail

L'objectif de notre travail est l'étude et la définition de techniques d'implantation de programmes parallèles sur des architectures réparties.
La répartition d'un programme parallèle nécessite le placement de ses composants sur les sites de l'architecture répartie et la représentation de ses différentes opérations sur une telle architecture. Dans cette thèse nous présentons une résolution de ces deux problèmes.

En vue de la généralisation de l'étude nous proposons une formalisation des techniques d'implantation. Cette formalisation permet de différer l'introduction des détails techniques liés à une architecture répartie donnée ou à un langage de programmation particulier.

D'un point de vue expérimental nous développons un outil logiciel (système assistant) permettant l'implantation de programmes parallèles sur la machine parallèle à mémoire distribuée : T-Node.

0.2.1 Formalisation

Le domaine des langages de programmation parallèle et celui des réseaux et des architectures réparties sont des domaines vastes, évolutifs et mouvants. Malgré leur évolution et l'emploi des architectures réparties pour l'exécution des programmes parallèles, il n'existe pas actuellement de méthodes générales permettant l'implantation de programmes parallèles sur une architecture répartie. Notre objectif est de contribuer à l'étude et à la définition formelle de techniques de répartition de programmes parallèles.

La formalisation de ces techniques nécessite une **description abstraite** des différents concepts intervenant lors de la répartition des programmes parallèles. Ces concepts sont :

- Le programme parallèle : le concept logique de base de l'implantation.
- L'architecture répartie : le concept physique de l'implantation.

L'abstraction de ces concepts permet de mettre en évidence leur aspects fondamentaux.
Vu que l'abstraction est développée afin de formaliser l'implantation de programmes parallèles sur les architectures réparties, seuls les aspects nécessaires pour cette implantation seront prises en compte. Nous définissons ainsi :

- L'**application parallèle** : l'abstraction du programme parallèle. Il est composé d'un ensemble de processus communicant par l'intermédiaire d'objets de communication [Per85].
- Le **système réparti** : l'abstraction de l'architecture répartie. Il est constitué d'un ensemble de sites reliés par un réseau de communication.

L'étape de la formalisation est la première étape de notre travail. Elle nous permettera de formaliser la répartition de programmes parallèles.

0.2.2 Implantation

La répartition des programmes parallèles nécessite :

- la définition de l'implantation de l'application parallèle sur une architecture répartie. Cette implantation est nommée : **application répartie**,
- la définition d'un procédé de construction d'une application répartie correspondant à une application parallèle. Ce procédé est nommé : la **fonction d'implantation.**

Lors de la répartition nous considérons que les programmes parallèles et les architectures réparties sont décrits par leur formalisation : application parallèle et système réparti.

L'implantation d'une application parallèle sur une architecture répartie doit résoudre trois types de problème :

- le placement des processus sur les processeurs et le placement des communications sur les liaisons entre les processeurs.
- la communication entre les processus dans l'environnement réparti,
- l'exécution des processus.

Nous supposons dans notre travail que chaque processus est exécuté sur un seul processeur (pas de migration de processus). En conséquence le dernier point peut être considéré comme convenablement résolu (cas de machines centralisées).

Le premier et le deuxième point ne sont que partiellement résolus : il existe des solutions pour certains langages et pour des architectures particulières. L'objectif de notre travail est de contribuer à la formalisation de la résolution de ces points.

La construction d'une application répartie nécessite la création de ses objets et la représentation des opérations de l'application parallèle dans l'environnement réparti. La fonction d'implantation définit un procédé de dérivation d'une application répartie.

Deux approches ont été suivies lors du développement de la fonction d'implantation : une approche heuristique pour déterminer le placement de composants de l'application parallèle sur les sites de l'architecture répartie et une approche méthodologique pour la dérivation de l'application répartie et la réalisation de la communication entre les processus dans l'environnement réparti. Cette étape constitue le noyau de notre étude.

0.2.3 Réalisation

Afin de concrétiser notre étude nous développons un outil logiciel permettant l'implantation d'une application parallèle sur une machine physique réelle.

La machine physique choisie est le T-NODE, qui est une machine parallèle à base de transputers. La mémoire de cette machine est répartie et son résau est reconfigurable.
L'outil logiciel qui réalise la fonction d'implantation est le système COMPIL. Ce système tient compte de contraintes liées :

- au langage de programmation C-Parallel, utilisé pour l'implantation des applications réparties;
- à l'environnement logiciel disponible sur la machine T-NODE;
- à l'architecture physique de la machine T-NODE.

Le système COMPIL est un système interactif d'aide à l'implantation. Il est composé, comme la fonction d'implantation, d'une partie heuristique de placement et d'une partie méthodologique de dérivation de l'image exécutable de l'application parallèle sur la machine T-NODE.
L'utilisateur du système peut intervenir pendant l'exécution de la partie heuristique pour : modifier un placement proposé par le système, proposer un placement ou choisir un placement parmi plusieurs possibles.

0.3 Plan du travail

Après ce bref exposé informel de notre travail, nous allons donner le plan de cette thèse :

- Dans une première partie nous définissons le cadre formel du travail.
  - La formalisation d'une application parallèle est développée dans le premier chapitre. Cette formalisation est déduite des langages de programmation permettant l'expression du parallélisme et plus précisement du langage du projet COMETE.
  - Dans le deuxième chapitre nous donnons la formalisation d'un système réparti, considéré comme l'abstraction d'une architecture répartie. Cette spécification met en évidence les caractéristiques principales de l'architecture sans tenir compte de l'aspect physique et technique lié à une architecture particulière.
- La deuxième partie concerne l'implantation d'applications parallèles sur des systèmes répartis. Nous y définissons l'application répartie et nous y présentons formellement la fonction d'implantation.
  La fonction d'implantation est composée de deux fonctions : une fonction heuristique de placement et une fonction de dérivation de l'application répartie.
  - La définition de l'application répartie est donnée au troisième chapitre.
  - Dans le chapitre 4 nous présentons la fonction de placement de composants de l'application parallèle sur un système réparti. Cette fonction est fondée essentiellement sur des règles heuristiques liées aux caractéristiques de l'application parallèle. Ces règles sont complétées par des règles liées à la charge des sites et du réseau de l'architecture répartie.
  - L'implantation de l'application parallèle nécessite la création de l'application répartie correspondant à un placement donné et la représentation des opérations de communication dans l'environnement réparti. La résolution de ces deux problèmes est donnée par la fonction de dérivation développée au chapitre 5.
- Dans la troisième partie nous décrivons la réalisation pratique de notre travail sur la machine T-NODE.
  - Une description de l'architecture physique de la machine T-NODE, de son environnement logiciel de travail et de ses langages de programmation est donnée au chapitre 6.
  - Dans le chapitre 7 nous présentons le système COMPIL.
    La réalisation du système a nécessité la définition de l'image réelle de l'application répartie sur la machine T-NODE, cette image est le résultat du système COMPIL.
    Pour l'exécution de l'application répartie il a fallu introduire une couche système nommée la couche de communication. Cette couche est transparante vis-à-vis l'utilisateur et elle constitue une partie de l'image réelle de l'application répartie sur la machine T-NODE. La couche de communication est générée par le système COMPIL.
    Lors de la description de système COMPIL nous séparons sa partie heuristique de sa partie automatique et nous présentons son interface avec l'utilisateur.
    Des exemples d'exécution sont donnés à la fin du chapitre.

Partie I

Concepts intervenant dans la répartition d'une application parallèle

L'implantation d'une application parallèle sur une architecture répartie (ou la répartition d'une application parallèle) nécessite une définition précise des différents concepts intervenant lors de cette implantation (voir schéma).

L'**application parallèle** est la description logique (abstraite) de l'application que nous voulons implanter. Elle est constituée d'un ensemble de processus qui s'exécutent en parallèle et qui communiquent par échange de messages. L'application parallèle peut être décrite à l'aide d'un langage de programmation parallèle tel que : ADA, CSP, etc.

L'**architecture répartie** est la description de l'architecture physique sur laquelle va se faire l'implantation. Elle est constituée d'un ensemble de machines indépendantes reliées par un réseau. Il existe actuellement plusieurs sortes d'architectures réparties, ces architectures se différencient par leurs machines et par la topologie de leurs réseaux.

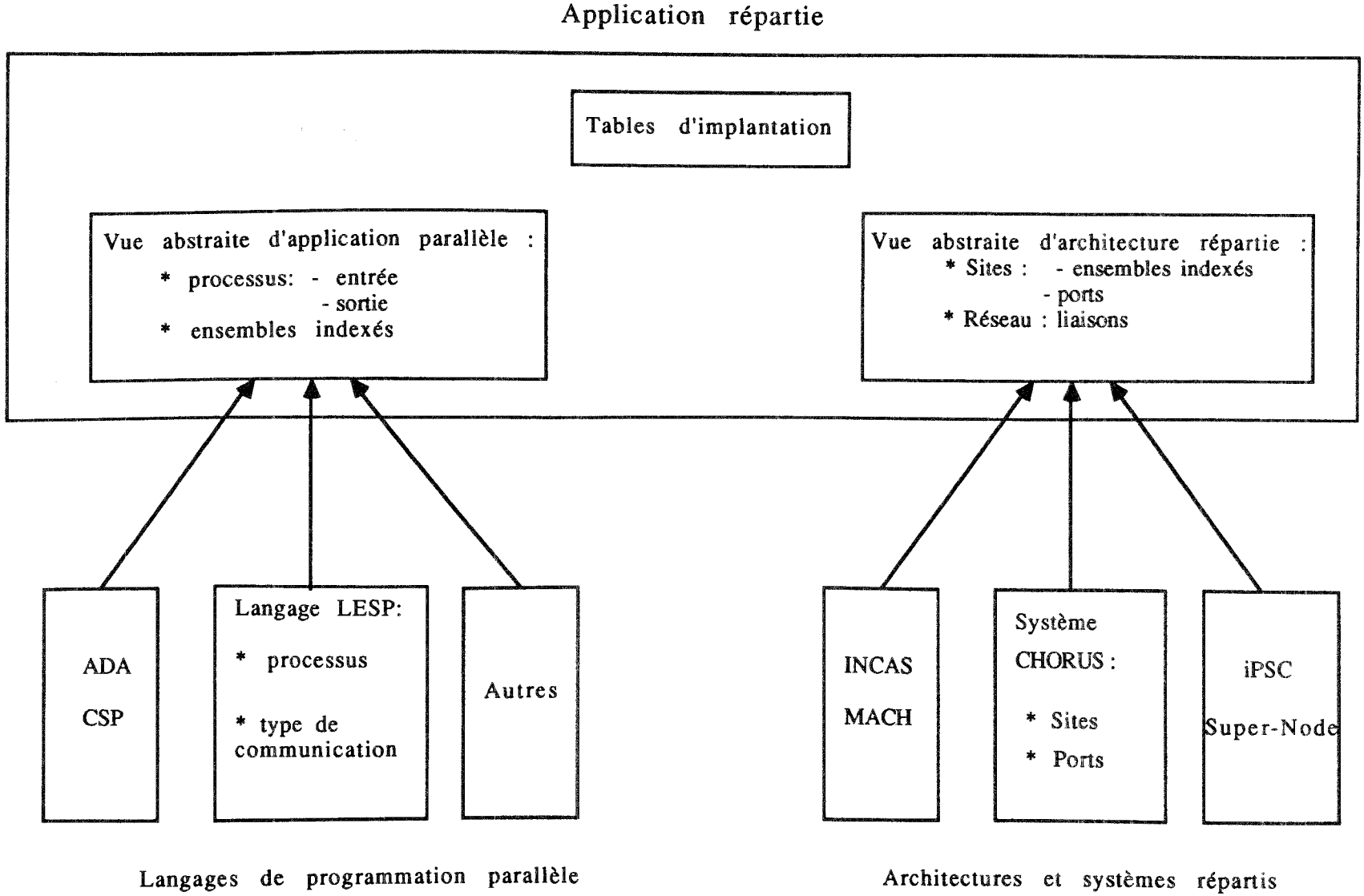
L'application répartie est le résultat de l'implantation d'une application parallèle sur une architecture répartie. C'est une image exécutable de l'application parallèle qui reflète sa distribution.

La formalisation de la répartition nécessite une description abstraite de ces concepts. Cette abstraction sert à la généralisation de notre étude, nous ne nous limitons pas à un seul langage de programmation ni à une architecture particulière, et elle permet de mettre en évidence les aspects fondamentaux de chaque concept et de différer l'introduction des détails techniques.

Pour définir le mécanisme de la répartition nous allons développer dans cette partie :

- La formalisation d'une application parallèle : à partir de différents langages de programmation parallèle (CSP, ADA, etc), nous proposons une vue abstraite d'application parallèle (cf : chapitre 1).
- La formalisation d'un système réparti : à partir de différentes architectures réparties existant (iPSC, Super-Node, etc) et en nous inspirant de quelques systèmes répartis tel que : CHORUS, INCAS, MACH, nous définissons une vue abstraite d'architecture répartie (cf : chapitre 2).

Ces formalisations nous permettrons de **définir l'application répartie** (cf : partie 2) ainsi que le mécanisme de sa construction.



Les concepts intervenant dans la répartition d'une application parallèle

Chapitre 1

La formalisation d'une application parallèle

Introduction

Le domaine des langages de programmation parallèle est un domaine vaste, évolutif et mouvant. Il existe actuellement plusieurs langages de programmation parallèle. Ces langages se différencient par : les domaines d'applications traités, les types de parallélisme (vectorisation, communication, concurrence) et les modèles de communication : synchrones ou asynchrones.

Le but de ce chapitre est de définir **une vue abstraite des langages de programmation parallèle** permettant de préciser leurs points communs et de dégager leur concepts fondamentaux.

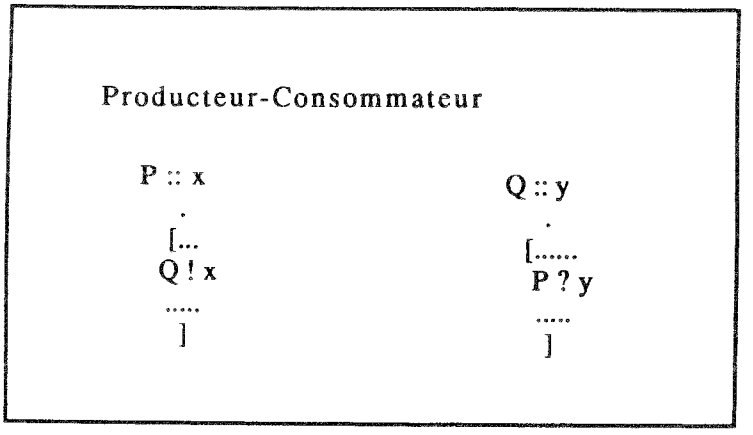
Commençons par donner quelques exemples de langages de programmation parallèle.
CSP (Communicating Sequantial Processes) de Hoare [Hoa78] est un langage de programmation parallèle fondé sur un modèle de communication synchrone. Un programme de ce langage est constitué d'un ensemble de processus séquentiels, non déterministes. Les processus communiquent des valeurs de données locales via un canal, sans mémorisation des messages.

Un canal est défini implicitement et dynamiquement par le nom des processus interlocuteurs et de la donnée communiquée. Un symbole réservé indique la direction d'utilisation dudit canal : " ! " pour l'émission, et " ? " pour la réception. Ces deux notations définissent des opérations complémentaires sur un canal, et sont activées simultanément dans une situation dite de " rendez-vous ". Ainsi est réalisé le système producteur-consommateur synchrone de la figure 1.

ADA est un autre langage de programmation fondé aussi sur le modèle de communication synchrone [Ada80]. Il permet la décomposition d'un programme parallèle en processus parallèles appelés tâches, qui peuvent éventuellement partager des variables globales. Le nombre de tâches peut évoluer dynamiquement et les communications se font par appels de points d'entrée.

Les appels sont exécutés lors de rendez-vous entre la tâche appelant et la tâche qui active la procédure appelée. Toute acceptation d'un appel de procédure dans une tâche (où celle ci est

définie), est précisée dans le corps de tâche (éventuellement, dans une sélection non déterministe) et elle constitue un point d'entrée dans cette tâche.



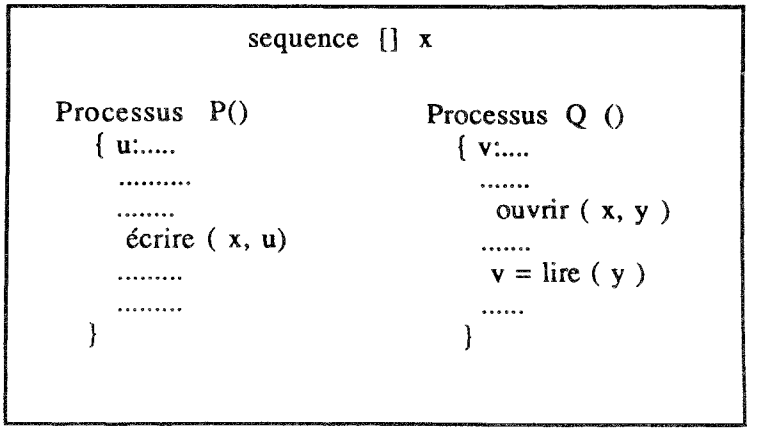
Communication par rendez-vous (CSP)

Figure 1.1

Un autre modèle de langage de programmation parallèle est le modèle de communication asynchrone. Le principe de ce modèle est que le médium de communication permet la mémorisation des messages dans une file d'attente bornée.

LC3 [Cer86] est un langage parallèle fondé sur le modèle asynchrone. C'est une extension du langage C où les média sont constitués de séquences typées. Les processus peuvent accèder , en lecture, à des séquences partagées par l'intermédiaire de canaux locaux à chaque processus.

A titre de comparaison avec CSP, un système de producteur-consommateur asynchrone, écrit en LC3 est présenté par la figure 2. Les deux processus communiquent via une séquence non bornée " x " accessible en lecture par un canal " y " ouvert sur " x " dans le processus Q.



Communication asynchrone en LC3

Figure 1.2

Cette présentation brève de quelques langages montre quelques points communs des langages de programmation parallèle :

- un programme parallèle est constitué d'un ensemble de **processus communicants**,
- l'existence d'un **médium de communication** permettant l'échange d'information entre les processus.

Dans le cadre de cette étude nous nous intéressons aux langages de programmation parallèles fondés sur le modèle de communication **asynchrone.** Parmi ces langages nous choisissons le langage d'expression de systèmes parallèles, développé dans le cadre de projet COMETE, comme support de notre étude. Tout au long de cette thèse nous appelerons ce langage : **LESP** (Langage d'Expression de Systèmes Parallèles). Ce langage est défini dans [Per85] [Mar89], il a été conçu pour être un outil de construction méthodologique et de validation de programmes parallèles. Dans ce langage nous pouvons distinguer deux types d'entité : les processus et les objets de communication qui constituent le médium de communication.

Dans cette thèse l'abstraction de programmes décrits par les langages parallèles considérés concerne :

- les **entrées-sorties de processus** du programme parallèle,
- le **média de communication** du programme parallèle.

Ce chapitre est divisé en deux parties. La première est un rappel des concepts de base de langage de projet COMETE. La seconde est la présentation d'une vue abstraite de ce langage, appelée formalisation d'applications parallèles, qui nous permettra dans les chapitres suivants de définir l'implantation sur un système réparti.

1.1 Les concepts de base de LESP

Dans le cadre du projet COMETE un formalisme a été défini pour la spécification et la conception de systèmes parallèles [Per85]. **Ce formalisme est fondé sur le concept de type de communication qui décrit un protocole de communication entre les processus d'un programme parallèle.**

Contrairement aux travaux qui s'intéressent à la synchronisation entre les processus d'un système parallèle [RK83] [BR83], le projet COMETE propose comme formalisme d'expression un modèle de communication **asynchrone** fondé sur les notions de suite temporelle et de relation de communication, mises en évidence dans [FJ80] [Per85] et [Mar89].

Un algorithme est décrit, dans ce formalisme, sous forme de processus communicants. Les communications entre les processus sont réalisées par l'intermédiaire d'objets de communication typés par des types de communication. Le type de communication est la traduction algorithmique d'une relation de communication.

La notion de relation de communication, et par conséquent le type de communication, permet de séparer les interactions des processus de leurs calculs. Cette séparation fait du système de processus communicants un outil approprié pour spécifier les applications parallèles de notre

étude, car la communication est l'aspect critique de l'implantation des applications parallèles sur des architectures réparties.

1.1.1 Type de communication

Comme nous l'avons signalé dans l'introduction, le formalisme proposé permet d'exprimer le parallélisme en termes de processus communicants. Les communications entre les processus sont définies par une relation entre les valeurs de données échangées.
En effet dans un univers parallèle, deux types de relation sont distingués :

- Une relation statique, appelée relation de calcul, entre les suites de données et les suites de résultats
- Une relation de communication [JP85], qui définit les suites de résultats à partir des suites de données, sans modifier leurs valeurs, et en prenant en compte l'état courant du système pour définir chaque terme du résultat.

Intuitivement une relation de communication exprime le choix par le processus consommateur de la valeur à consommer parmi les valeurs émises par le processus producteur. Elle permet aussi d'indiquer comment doit être modifié cet ensemble après cette consommation. Par exemple une relation de communication "égalité" spécifie un système composé d'un producteur et un consommateur, tel que à tout instant de communication la valeur utilisée par le processus consommateur soit la plus ancienne valeur émise par le producteur et non encore consommée. Cette valeur doit disparaitre de l'ensemble des valeurs émises après la consommation. Ainsi dans une telle stratégie de communication le processus producteur peut avoir plusieurs productions d'avance par rapport au processus consommateur.

Un autre exemple d'une relation de communication, totalement différent de l'égalité, est celle nommée "aléatoire". Un processus consommateur d'une telle relation reçoit à tout instant la valeur émise le plus récemment. Le processus producteur peut être en retard vis-à-vis du consommateur et dans ce cas une valeur pourra être consommée plusieurs fois consécutives.
Les relations de communication sont exprimées dans un premier temps en termes de relation entre des suites temporelles [Laz86] et [Mar89]. En fait, il s'agit d'établir les relations entre les suites de données échangées. La nature même de ces relations sous-entend l'expression du temps à travers des expressions comme "à tout instant" , "plus ancienne" , "plus récente" , etc.

En ce qui concerne les relations de communication, nous nous limitons à cette brève présentation, par contre nous allons détailler le formalisme d'expression des types de communication qui n'est que la représentation algebrique et constructive de ces relations. La preuve de représentation d'une relation de communication par un type de communication est etudiée dans [Laz90] et [Mar89].

Définition d'un type de communication

Un type de communication est la description algorithmique d'une relation de communication. Il est exprimé en termes de types abstraits de donnée qui répond à l'objectif de modularité visé par l'expression algorithmique des systèmes parallèles et permet de valider sa spécification.

Chaque type de communication est défini par l'intermédiaire des composants des objets de ce type; ces composants sont :

- Un ensemble produit (EP) contenant l'ensemble de valeurs émises par le producteur. Cet ensemble représente l'historique de la production.
- Un ensemble consommable (A) contenant l'ensemble de valeurs émises par le processus producteur et dont dispose le processus consommateur.
- Un ensemble consommé (EC) contenant l'ensemble de valeurs consommées par le processus consommateur. Cet ensemble représente l'historique de la consommation.

Un objet "tc" du type TYPCOM est défini formellement par un triplet d'objets, noté tc = < EP, A, EC >, chacun d'eux étant du type ensemble indexé[1].
Ce triplet < EP, A, EC > est manipulé par trois opérations :

- init-com : Initialiser la communication entre les processus,
- produire : Emettre une donnée. Tout processus qui active cette opération est appelé producteur de l'objet.
- consommer : Recevoir une donnée communiquée. Tout processus qui active cette opération est appelé consommateur de l'objet.

Le type abstrait TYPECOM est défini par une spécification en termes de **pré** et **post** condition [Mar89]. Il est paramétré par le type des données communiquées et les trois opérations caractéristiques du type de communication.
Ces opérations sont :

- pré-cons : l'opération qui définit la pré condition de l'opération de consommation,
- val-cons : l'opération qui delivre la valeur consommée,
- post-cons : l'opération qui définit l'état des composants de l'objet du type suite à l'opération de consommation.

En supposant que le type TYPCOM est paramétré par le type des données ELT le profil des opérations de ce type est le suivant :

- init-com : → TYPECOM.
- produire : TYPECOM * ELT → TYPECOM.
- consommer : TYPECOM → TYPECOM.
- pré-cons : TYPECOM → bool.
- val-cons : TYPECOM → ELT.
- post-cons : TYPECOM → TYPECOM.

[1] La définition de type "ensemble indexé" est donnée dans [Per85]

Remarque : La définition de l'opération de production est la même quelque soit le type de communication, elle consiste à mettre la valeur produite dans l'ensemble produit et l'ensemble consommable de l'objet du type. En revanche, l'opération de la consommation est définie en termes des opérations caractéristiques du type (pré-cons, val-cons, post-cons) et caractérise ainsi le type de communication.

Pour clarifier les idées, nous donnons comme exemple les trois types de communications "égalité", "aléatoire" et "pile". Une présentation détaillée des différents types est donnée dans [Per85].

Exemples

1. **Egalité :** le type de communication "égalité" est caractérisé par le fait que toute valeur émise est reçue une et une seule fois, et l'ordre de réception est identique à l'ordre d'émission.
   Les opérations caractéristiques de ce type sont :

   - pré-cons (< EP, A, EC >) = non vide ? (A)
   - val-cons (< EP, A, EC >) = première (A)
   - post-cons (< EP, A, EC >) = décapite (A)

   L'opération décapite est définie sur le type d'ensemble indexé, elle consiste à supprimer le message le plus ancien de l'ensemble indexé.

2. **Aléatoire :** le type de communication "aléatoire" est caractérisé par le fait que chaque valeur reçue est la dernière émise à cet instant, certaines valeurs ne sont pas reçues et d'autres sont reçues plusieurs fois.
   Les opérations caractéristiques de ce type sont :

   - pré-cons (< EP, A, EC >) = non vide ? (A)
   - val-cons (< EP, A, EC >) = dernière (A)
   - post-cons (< EP, A, EC >) = finir (A)

   Où finir, est une opération définie sur le type d'ensemble indexé et elle consiste à supprimer toutes les valeurs de A sauf la dernière.

3. **Pile :** le type de communication "pile" est caractérisé par le fait que chaque valeur reçue est la dernière émise à cet instant et elle est reçue une et une seule fois. Les opérations caractéristiques de ce type sont :

   - pré-cons (< EP, A, EC >) = non vide ? (A)
   - val-cons (< EP, A, EC >) = dernière (A)
   - post-cons (< EP, A, EC >) = disparaitre (A)

   L'opération disparaitre consiste à supprimer la dernière valeur de l'ensemble indexé.

1.1.2 Le langage d'expression du parallélisme

Le type de communication est le concept de base de LESP, pour l'expression des systèmes parallèles.

L'expression du parallélisme est fondée sur la coopération de processus non déterministes qui communiquent de manière asynchrone par l'intermédiaire des objets de communication. L'utilisation des types de communication permet de séparer, dès le début de la conception, les relations de calcul et les relations de communication. Elle permet aussi de construire les différents processus du système parallèle d'une manière indépendante et sans souci de la synchronisation. Les seules synchronisations sont dûes à la contrainte d'accès en exclusion mutuelle aux objets de communication et aux attentes qui se produisent lorsqu'un processus est prêt à réaliser une opération de de consommation alors que la pré condition de la consommation n'est pas satisfaite.

Plusieurs études ont éte réalisées autour du langage. Le passage aux autres langages d'expression du parallélisme a été étudié dans [KFJP88] pour le langage ADA ou dans [EJ89] pour le langage OCCAM. Sa mise en oeuvre a donné lieu à la réalisation d'un environnement de programmation appelé COMEDIE.

Présentation du langage

Le langage est construit autour de quatre entités qui constituent le noyau parallèle : le processus, l'objet de communication, la donnée communiquée et le concept du type de communication.
Un programme, ou système parallèle, est constitué d'un ensemble hiérarchique de processus communicants. Dans cette hiérarchie nous pouvons distinguer deux types de processus :

- Les processus élémentaires : ce sont des processus séquentiels non déterministes qui utilisent des constructions proches des processus CSP, ou des tâches ADA.
- Les processus composés obtenus par une composition parallèle d'autre processus communicants par des objets de communication.

Un **processus élémentaire** a la structure suivante :

1. **Entrée, Sortie :** définition des données communiquées : nom et type de la donnée, objet de communication utilisé pour transmettre les valeurs de cette donnée. Cette partie constitue l'interface du processus et elle utilise les objets déclarés au niveau du processus englobant.
2. Déclaration d'**objets locaux** au processus (variables, fonctions).
3. **Corps** : définition des actions du processus. L'activation des opérations : produire et consommer, réalisant la communication, est faite dans cette partie.

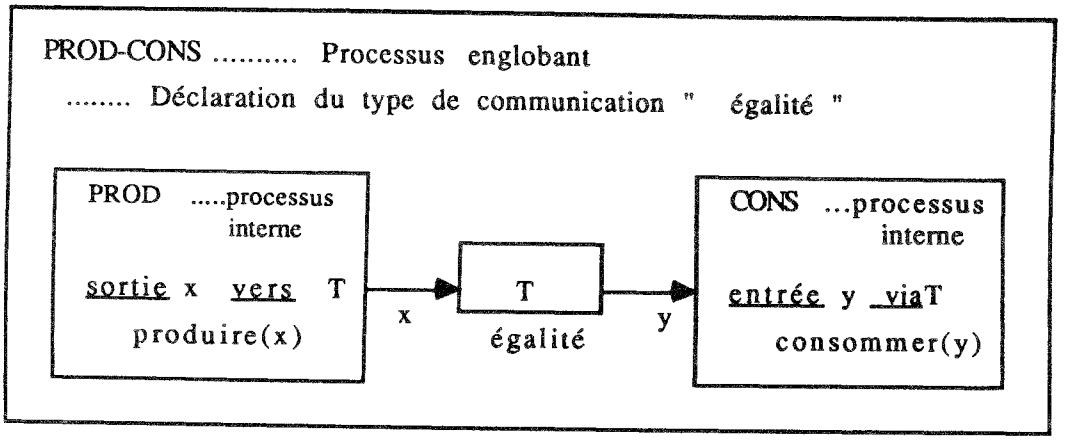
La structure d'**un processus composé** est de la forme :

1. Déclaration des **processus internes** et description de leurs relations de communication en précisant :

- les objets de communication utilisés : le nom et le type de communication de l'objet.
- les types de communication associés et non prédéfinis.

2. **Composition** : mise en composition parallèle des processus internes déclarés dans la partie 1. Cette partie réalise l'activation parallèle des processus définis de manière statique, elle est alors réduite à l'opérateur de composition //.

Le schéma de la figure 3 représente la structure d'un système parallèle simple composé d'un producteur (PROD), un consommateur (CONS) et un objet de communication (T).



Un système parallèle composé de deux processus

Figure 1.3

Exemple d'utilisation de LESP pour le calcul de PGCD

Le langage LESP a été utilisé pour exprimer des problèmes en calcul numérique [Per85]. Comme exemple nous citons un programme écrit en ce langage pour le calcul de plus grand commun diviseur *pgcd* de "n" nombres entiers strictement positifs "$a_1, a_2, ..., a_n$", par décomposition de ces nombres en facteurs premiers.
La formule qui donne le pgcd est :

$$pgcd = \prod_{j=1}^{k} p_j^{s_j}$$

où

p : suite des nombres premiers ($p_1 = 2$)
s : suite des exposants de la décomposition de pgcd en facteurs premiers, définie par :

s_j = max { s entier >= 0 tel que $\forall i \in [1..n]$, p_j^s divise a_i }

k : entier tel que $\forall$ i $\in [1..n]$ $k > k \Rightarrow$ non (p^k divise a_i)

La réalisation de cet énoncé peut être effectuée par un système parallèle constitué des processus suivants :

- un processus P calcule la suite "p" des nombres premiers,
- un processus S calcule la suite "s" des exposants,
- un processus Q calcule le résultat "pgcd".

Par propriété de l'opérateur "max" et de la conjonction, nous pouvons substituer à la définition du terme général de la suite "s", les définitions suivantes :

$s_j = min\{\underline{r_{ij}}, i \in [1,n]\}$
$\forall$ i $\in$ [1.n], $\underline{r_i}$: suite des exposants de la décomposition de a_i en facteurs premiers, définie par :

$\underline{r_{ij}}$ = max { r entier >= 0 tel que p_j^r divise a_i }
% r_i = ($r_{i1}, r_{i2}, \ldots, r_{ik}$) %

Le système parallèle obtenu comporte alors "n" processus "P_i" qui calculent les suites "r_i" des exposants.

La définition des suites dans l'énoncé précédent suppose l'introduction des communications suivantes :

- Chaque terme "p_j" calculé par le processus "P" est utilisé une fois au plus par les processus "P_i" et "Q", dans l'ordre de sa production : cette suite détermine donc (n+1) données communiquées, utilisant chacune un objet associé au type de communication "égalité",
- De même, la suite "r_i" détermine, pour tout i de 1 à n une donnée communiquée utilisant un objet associé au type de communication "égalité",
- Enfin, nous déterminons la valeur de "k" par la communication d'une suite de valeurs booléennes "c_i" des processus "P_i" vers le processus "Q". La terminaison du processus "P" est obtenue par "consultation" d'une valeur booléenne "c" émise par "Q" : ceci est réalisé par une donnée communiquée utilisant un objet associé au type de communication "aléatoire".

Nous obtenons finalement le système suivant :

processus **PGCD** ::

typecom égalité(elt) ... définition algébrique du type de communication " égalité ".

fin typecom.

typecom aléatoire(elt) ... définition algébrique du type de communication " aléatoire "

fin typecom.

communication sp, (sr_i, sc_i; i=1..n) : égalité(entier); sc : aléatoire(booléen).

processus **P** ::

entrée c : booléen via sc;

sortie p : entier via sp;

corps

p := 2;
produire(p);
consommer(c);
répéter non c
.... calcul nombre premier suivant ...
produire(p);
consommer(c);
fin répéter

fin P;

processus P_i ::

entrée p : booléen via sp;

sortie r : entier vers sr_i; c : booléen vers sc_i,

entier a_i,

corps

c := faux;
consommer(p);
répéter p < a_i,
r := 0;
répéter p divise a_i
$a_i = a_i/p$;
r := r + 1;
produire(r);
produire(c);
consommer(p);
fin répéter;
c := vrai;
produire(c);

fin P_i;

processus **Q** ::

entrée p : entier via sp;
(r_i : entier via sr_i; i=1..n)
(c_i : booléen via sc_i; i=1..n)

sortie c : booléen vers sc;

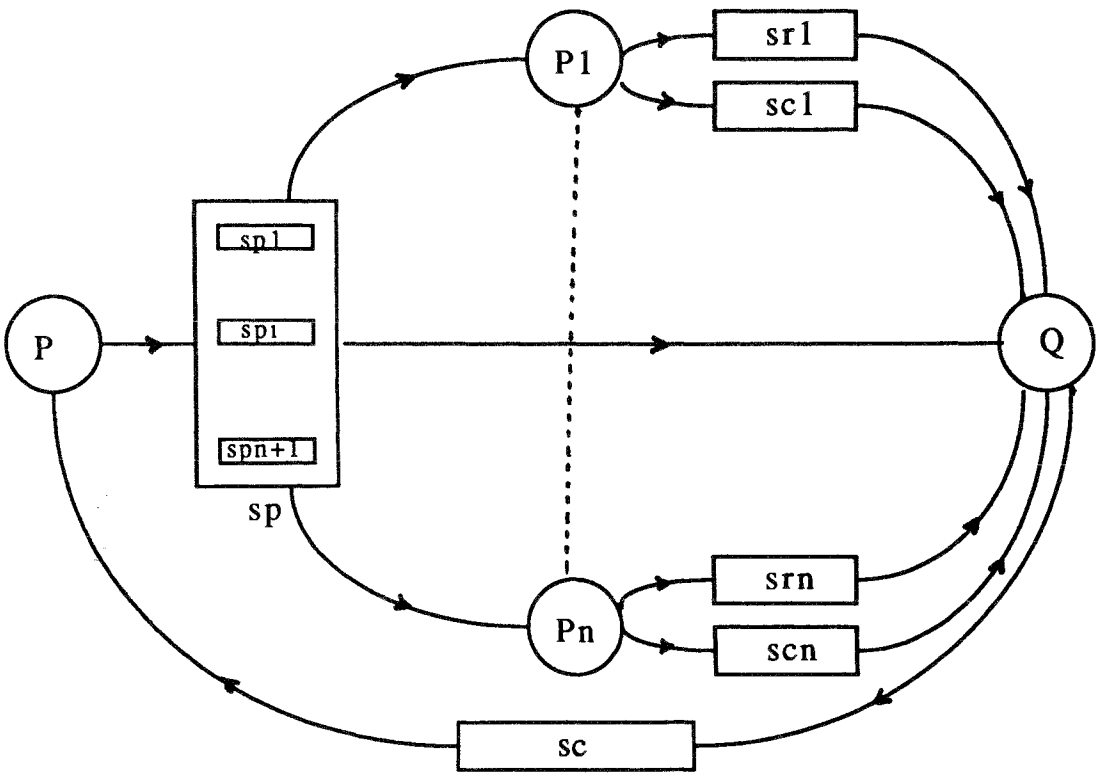
entier pgcd, s;

corps

pgcd := 1;
c := faux;
consommer(p);
(consommer(c_i); i=1..n);
répéter $\bigwedge_{i=1}^{n}$ non c_i

```
            (consommer(r_i); i=1..n );
            s := min( r_i: i=1..n );
            pgcd := pgcd * (p^s);
            consommer(p);
            (consommer(c_i); i=1..n );
        fin répéter;
        c := vrai;
        produire(c);
    fin Q
    composition P ( // P_i; i=1..n ) // Q
fin S.
```

La figure 4 illustre le schéma correspondant au système précédent.



Le système parallèle de calcul de pgcd

Figure 1.4

1.1.3 La topologie des objets de communication

Nous avons vu qu'un système parallèle peut être décrit par une composition de processus communicants et que les objets de communication constituent le moyen de communication des processus.

Selon le schéma de composition de processus, nous pouvons distinguer différentes topologies de l'objet de communication.

La topologie bipoint

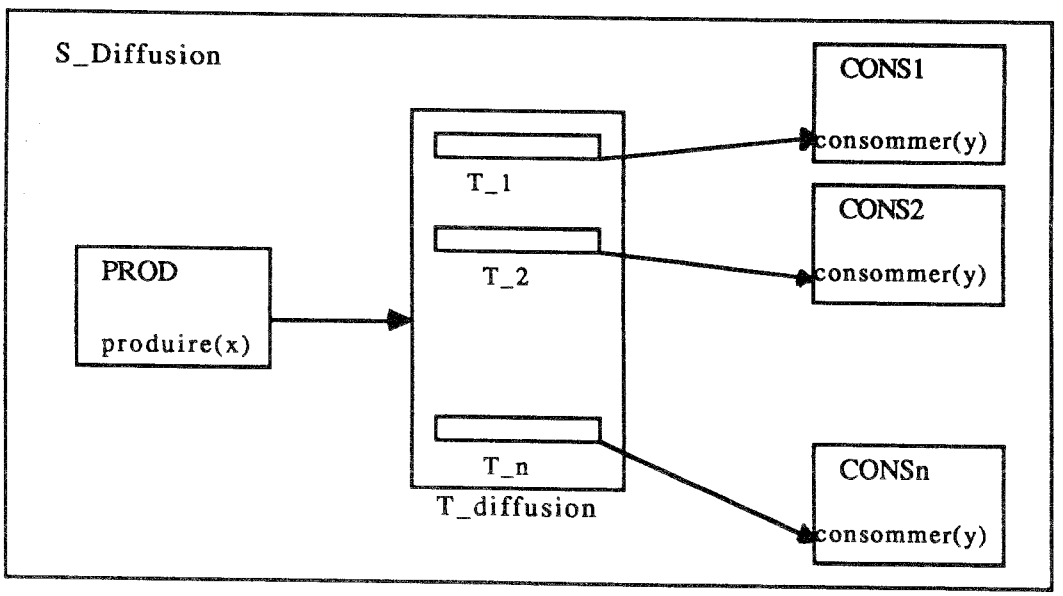
Le schéma de composition le plus simple est le schéma illustré par le système " producteur-consommateur " donné précédemment, figure 3. Nous dirons que la topologie de l'objet (T) est "bipoint".

La diffusion

C'est la généralisation de la communication bipoint à un système "un producteur-n consommateurs" où les valeurs de la donnée communiquée sont diffusées vers "n" objets, où n est défini statiquement.

La situation de diffusion se présente alors comme un ensemble de n communications bipoint ayant le même producteur et utilisant, des objets de même type de communication, ou de types de communication distincts.

La figure 5 schématise un objet de communication de topologie diffusion, et son utilisation par des processus.



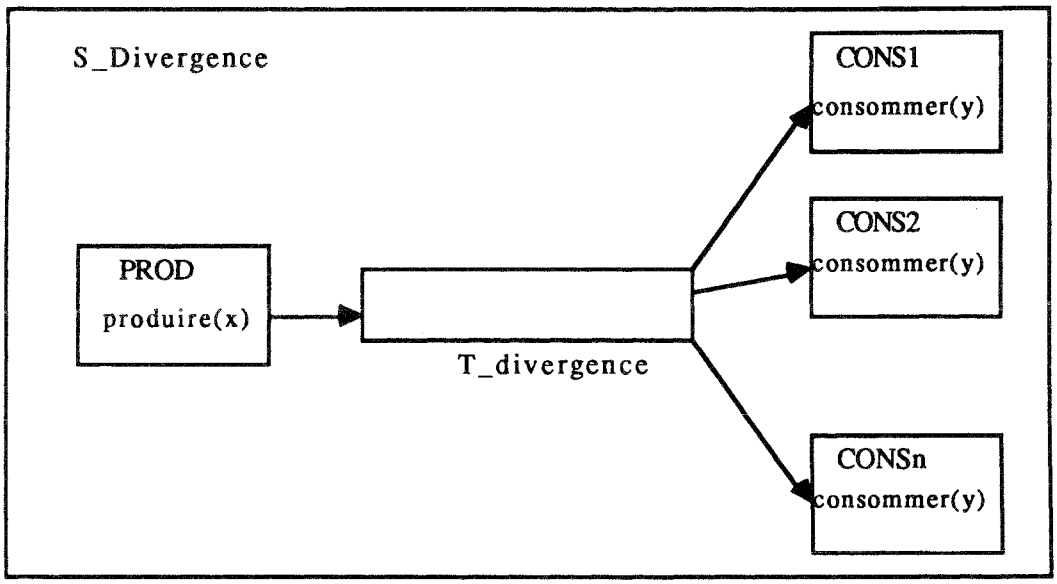
Un système parallèle dont l'objet est de topologie "diffusion"

Figure 1.5

La divergence

C'est le cas d'un système " un producteur-n consommateurs " où les consommateurs coopérent pour la consommation des valeurs produites par le producteur.
Les différents processus d'un tel système partagent un objet de communication de topologie divergence : le producteur agit sur l'objet par une opération de production et les différents consommateurs agissent par une opération de consommation, au hasard de leurs activations, en exclusion mutuelle.

La figure 6 illustre un système composé d'un processus producteur et "n" processus consommateurs communicant par un objet de topologie "divergence".



Un système parallèle dont l'objet est de topologie "divergence"

Figure 1.6

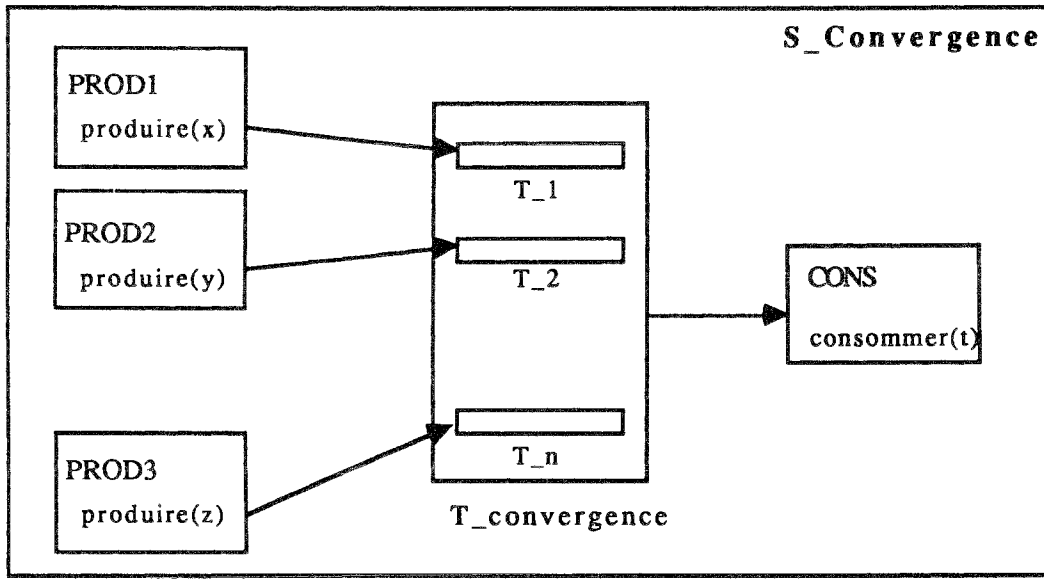
La convergence

Cette situation est topologiquement la symétrique de la diffusion. Le système est composé de "n" producteurs et un consommateur, et la communication entre un producteur et le consommateur est une communication bipoint.
Chaque producteur produit dans un seul objet, par contre le consommateur a accès à tous les objets. Il choisit l'objet dans lequel il veut consommer avant de choisir la valeur à consommer. Par conséquent il détermine le producteur du message avant de déterminer le message. Un objet de communication d'un tel système est un objet de topologie "convergence", il est composé de "n" objets bipoint. La figure 7 illustre un tel objet.

La concentration

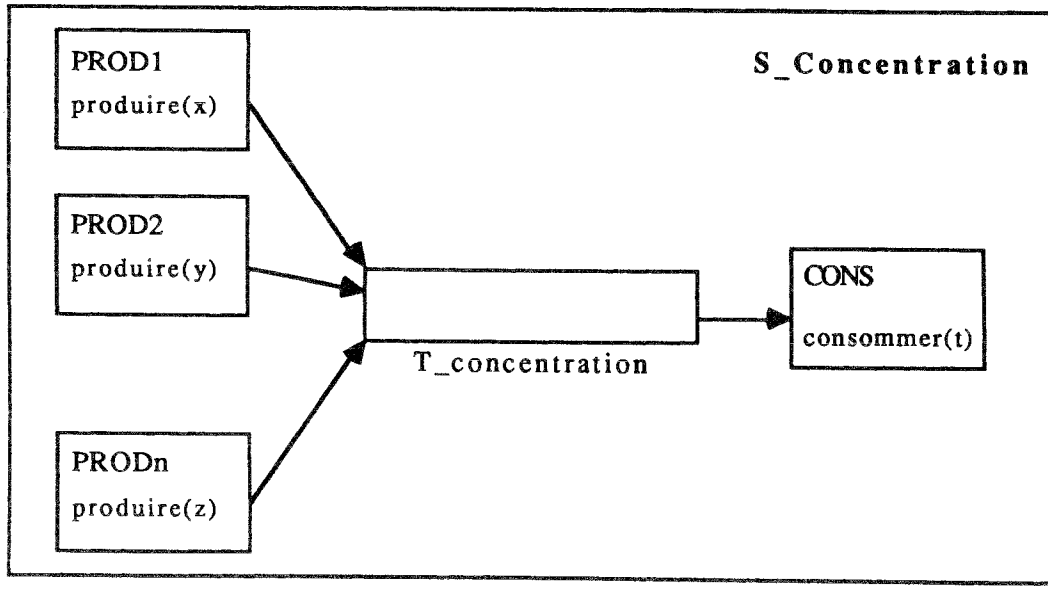
Comme dans la situation précédente, le système parallèle est composé de "n" producteurs et un consommateur. Mais dans cette situation et lors de la communication, le consommateur choisit une valeur parmi toutes les valeurs produites par les différents producteurs. Comme dans le cas de divergence, une synchronisation est nécessaire pour l'accès à cet objet en production pour réaliser l'exclusion mutuelle entre les producteurs.

L'objet de communication qui lie les processus d'un tel système est un objet de topologie "concentration". La figure 8 illustre un tel objet.



Un système parallèle dont l'objet est de topologie "convergence"

Figure 1.7



Un système parallèle dont l'objet est de topologie "concentration"

Figure 1.8

1.2 La formalisation d'applications parallèles

Pour formaliser l'implantation d'applications parallèles sur des architectures réparties, il est utile d'en définir une vue abstraite.

Dans la littérature nous trouvons plusieurs langages de spécification des systèmes parallèles; chacun définit une abstraction particulière. Ces langages sont caractérisés par :

- l'objectif de l'abstraction,
- les concepts de base,
- les entités manipulées et les expressions de communication et de concurrence.

Avant de détailler notre formalisation nous allons présenter brièvement quelques langages de spécification des systèmes parallèles.

Afin de définir un modèle mathématique qui représente les systèmces parallèles Milner a introduit un algebre de processus (CCS : Communicating Concurrent Systems) [Mil88] [Mil89].
D'aprés Milner, un système parallèle est composé d'agents (processus) et de connexions. Les agents sont indépendants. Ils s'exécutent en parallèle et ils se communiquent pour assurer l'unité du système parallèle.
Dans un tel système Milner distingue plusieurs protocoles de communication entre les agents. Il introduit dans son algebre plusieurs primitives permettant : l'expression des protocoles de communication et la construction des systèmes parallèles complexes.

Proposées initialement par Campbell et Heberman [CH74], les expression de chemins est une méthode permettant la spécification de la synchronisation dans un système parallèle. Cette spécification repose sur la séparation de la partie synchronisation de la partie active du système. Un système parallèle est composé de processus, d'objets partagés par les processus et d'un chemin de synchronisation. Chaque processus exécute une séquence d'opération sur un ensemble d'objets.
Pour résoudre le problème de conflit d'accès aux objets partagés, une synchronisation entre les processus est nécessaire. Les règles de synchronisation sont données par les expression de chemins et elle sont intégrées dans la définition des types.
Les types sont introduits pour définir l'abstraction des ressources partagées.

Un autre langage de spécification de système parallèle fondé sur le concept de module abstrait est proposé dans [Kro87]. Il inclut deux concepts importants dans la spécification des systèmes informatiques : les types abstraits de données et la logique temporelle.
Les spécifications fondées sur ces concepts sont cependant assez différentes : les types abstraits de données permettent de spécifier les propriétés statiques tandis que la logique temporelle permet surtout de rendre compte de phenomènes temporels ou à caractère dynamique.
Les modules abstraits présentés dans [Kro87] unifient ces deux aspects :

- Les types abstraits de données spécifiés algebriquement décrivent la partie fonctionnelle du module.

- La partie opérationnelle spécifie les règles que doivent respecter les procédures du module dans un environnement concurrentiel. Ces règles sont spécifiées à l'aide de la logique temporelle.

Les réseaux de Petri [Pet81] constituent un des modèles adaptés à la spécification du comportement de systèmes paralléles. Ils permettent d'exprimer les contraintes de temps et sont adaptés à la fois à la validation et à la simulation dans de nombreaux domaines d'application, plus particulièrement dans le domaine des automatismes industriels et des protocoles de communication.
Les réseaux de Petri sont fondés sur deux concepts : Le premier concerne la synchronisation et le second concerne la transition.
Plusieurs études méthodologique fondées sur les réseaux de Petri ont été développées. Parmi ces études nous citons le travail de Tankoano [Tan88] qui définit une approche méthodique pour la conception des systèmes de commande des automatismes industriels répartis.

Précisons que les différences des objectifs d'abstraction de ces langages ont pour conséquences une différence de leur concept, de leurs entités et de leur expression de concurrence et de communication.

Aprés cette brève description de certains langages de spécification de systèmes parallèles nous allons présenter notre formalisation. Rappellons que l'objectif de cette formalisation est la définition d'un mécanisme d'implantation des programmes paralléles décrits dans [Per85] sur des architectures réparties. Une raison pour laquelle nous n'avons pas adopté une des formalisations présentées ci-dessus.

Nous avons cité dans l'introduction et dans la première partie différents langages de programmation parallèle. Notre but ici est de dégager les concepts fondamentaux communs, et qui seront utiles pour la définition d'une implantation répartie.

Le choix du niveau d'abstraction qui est fait ici est donc guidé par cet objectif de définition de l'implantation répartie et s'inspire des concepts généraux du langage présenté dans la première partie.

Cette implantation répartie est définie dans le contexte suivant :

- chaque processus est implanté sur un seul site de l'architecture répartie.

- la migration d'un processus d'un site à un autre n'est pas autorisée lors de son exécution.

En prenant en compte ces contraintes, les problèmes qui interviennent lors de l'implantation sont, des problèmes liés à la **communication**. Nous ne nous intéressons pas aux actions locales à un seul site.

La formalisation de l'application parallèle doit donc préciser les éléments qui interviennent lors des communications et les états de ces éléments suite aux opérations de communication.

Pour pouvoir exprimer ces deux aspects, nous avons choisi une formalisation en termes de **pré** et **post** condition.

1.2.1 Formalisation d'une application parallèle

Vu que l'aspect de communication d'applications parallèles est l'aspect fondamental dans notre travail, la formalisation de ces applications consiste à donner une vue abstraite de la sémantique de leurs communications.
Les seules entités considérées par cette formalisation sont les entités intervenant lors de communications, et les opérations prises en compte sont les opérations de communication.
En prenant le langage LESP comme langage de description de programmes parallèles, une application parallèle : l'abstraction de programme parallèle, est composée de :

- un ensemble de processus qui constituent les entités communicants,
- un ensemble d'ensembles indexés qui constituent le médium de communication.

Chaque processus de l'application parallèle est en interaction avec le médium de communication. Nous distinguons deux sortes d'interaction :

- interaction en production : où le processus est un producteur d'un ensemble indexé,
- interaction en consommation : où le processus est un consommateur d'un ensemble indexé.

Un processus est décrit dans notre formalisation par ses interactions avec le médium de communication :

- ses interactions en production, nommées SORTIE,
- ses interactions en consommation, nommées ENTREE.

Les ensembles indexés du médium de communication sont les ensembles consommables des objets de communication de langage LESP [2]. Chaque ensemble est caractérisé par le type de ses données et les opérations caractéristiques de son type de communication. Ces opérations permettent la manipulation des ensembles indexés par les processus de l'application parallèle.

Nous allons associer chaque composant de l'application (processus et ensemble indexé) à un nom. Cette association nous permettra de :

- séparer la partie immuable d'un composant de sa partie dynamique,
- désigner un composant par son nom sans qu'il soit impliqé par cette désignation.

Et elle nous conduira à définir de nouveaux objets, chacun est composé d'un nom d'un composant de l'application parallèle et du composant lui même.

Afin de formaliser l'application parallèle nous allons introduire des types. Chaque type permet de préciser la structure et les opérations définies sur ses objets [3].

[2] Les ensembles produits et consommés, retracent l'histoire des productions, et de consommations sont des expressions locales aux processus, qui ne nous intéressent pas dans notre abstraction.

[3] Dans ce qui suit nous supposons que les types : ensemble, type de communication et le constructeur de produit cartésien sont prédéfinis [Lev84]

Un processus de l'application parallèle est du type **accès-processus**. Ce type est le produit cartésien :

- d'un **nom** de processus,
- d'une **entrée**, qui indique les ensembles indexés manipulés par le processus lors de ses interaction en consommation,
- d'une **sortie**, qui indique les ensembles indexés manipulés par le processus lors de ses interaction en production.

Un ensemble indexé de l'application parallèle est de type **cons-ensemble-indexé**[4]. Ce type est paramétré par :

- le type de données de l'ensemble indexé,
- le triplet d'opérations caractéristiques de son type de communication.

Les opérations de ce type sont définies en terms du triplet d'opérations donné en paramètre.

Pour une raison de désignation, nous introduisons le type **accès-ensemble-indexé**. Ce type est le produit cartésien :

- d'un **nom** d'un ensemble indexé,
- d'un **cons-ensemble-indexé**.

L'accès-ensemble-indexé est paramétré par les mêmes paramètres que son cons-ensemble-indexé.

Le dernier type introduit dans cette formalisation est le type **application-parallèle**. Chaque objet de ce type est composé de :

- un **ensemble d'accès processus**,
- **plusieurs accès-ensemble-indexé**.

Trois opérations sont définies sur le type **application-parallèle** :

1. l'opération de **production** : qui permet la production d'un message à un accès-ensemble-indexé de l'application parallèle,
2. l'opération de **consommation** : qui permet la consommation d'un message d'un accès-ensemble-indexé de l'application parallèle.
3. l'opération **message-à-consommer** : qui détermine le message à consommer dans un accès-ensemble-indexé à un instant donné.

Le type application parallèle est paramétrée par :

[4]Ce type définit le type de l'ensemble consommable d'un objet de communication.

- les types de données de ses accès-ensembles-indexés,
- les triplets d'opérations caractéristiques de ses accès-ensembles-indexés,

La formalisation d'une telle application parallèle est donnée ci-dessous, sous la forme de types abstraits.

Notations : En vue d'alléger l'écriture des différents types introduits pour la formalisation d'une application parallèle nous allons donner quelques notations :

- Nous notons TC le triplet d'opérations caractéristiques d'un type de communication réduites à l'ensemble consommable[5].
  TC = (pré-cons, val-cons, post-cons)
- $\overline{V_j}$ est un vecteur de taille j, ses éléments sont de même type mais ils ne sont pas nécessairement paramétrés par les mêmes paramètres.
  $\overline{V}$ est un vecteur de taille indeterminée.
  $\overline{V_j} = (V_1, \ldots, V_j)$
  $\overline{V} = (V_1, \ldots, V_n)$ où "n" est un entier quelconque.
- Le symbole "@" signifie : l'expression du type de l'objet.
  Si "OBJ" est un objet de type "type-obj", "@OBJ" signifie l'expression de type "type-obj" de l'objet.
- type est un type indéterminé. Dans notre contexte type est un entier, réel ou booléen.
- <> signifie : le constructeur de produit cartésien.
- Toutes les phrases écrites entre "% %" sont des commentaires ajoutées pour simplifier la lecture et la compréhension de la formalisation.
- Les noms de différents objets sont des chaines de caractères.
- L'opération de projection sur un champ d'un produit cartésien est noté :
  $champ_i$(OBJ)
  où OBJ est de type "ex-type" et l'expression de "ex-type" est : $< champ_1, \ldots, champ_n >$.

Type **n-indexé**

    Expression chaine de caractères;

    % Un objet de ce type est un nom d'un ensemble indexé %

End

Type **n-processus**

    Expression chaine de caractères;

[5] Les ensembles produits et consommés, retracent l'histoire des productions, et de consommations sont des expressions locales aux processus, qui ne nous intéressent pas dans notre abstraction.

% Un objet de ce type est un nom d'un processus %

End

Type **cons-ensemble-indexé (typelem : type, (pré-cons, val-cons, post-cons)[6])**

Expression :

ensemble-indexé(typelem)

Opérations :

prod-EI (E : cons-ensemble-indexé, M : typelem)
$\acute{E}$: cons-ensemble-indexé;

pré vrai
post $\acute{E}$ = ajouter (@E, M)[7]

pré-cons-EI (E : cons-ensemble-indexé) b : booléen;

pré vrai
post b = pré-cons (@E). [8]

val-cons-EI (E : cons-ensemble-indexé) M : typelem;

pré pré-cons-EI (E)
post M = val-cons (@E).

post-cons-EI (E : cons-ensemble-indexé) $\acute{E}$: cons-ensemble-indexé;

pré pré-cons-EI (E)
post $\acute{E}$ = post-cons (@E).

End

Type **accès-ensemble-indexé (typelem : type, TC)**

Expression

< NOM : n-indexé;
IND : cons-ensemble-indéxé(typelem, TC) >

End

Type **accès-processus**

Expression

[6] Ce type définit le type de l'ensemble consommable d'un objet de communication.

[7] @ : signifie l'expression du type de l'objet E. E : est de type cons-ensemble-indexé, @E : est de type ensemble-indexé

[8] Cette définition dépend du type de communication en paramètre.

< NOM : n-processus;
ENTREE : ensemble (n-indexé);
SORTIE : ensemble (n-indexé) >

End

Type **application-parallèle** $_j$ ($\overline{\textbf{\textit{typelem}}_j}$: $\overline{\underline{\textbf{\textit{type}}_j}}$,$\overline{\textbf{\textit{TC}}_j}$)

Expression

< PROC : ensemble (accès-processus);
$\overline{ENS-IND_j}$: accès-ensemble-indexé($\overline{typelem_j}$, $\overline{TC_j}$) >

% Où :
$\overline{ENS-IND_j}$ =

$ENS-IND_{j1}$: accès-ensemble-indexé ($typelem_1$, TC_1)
$ENS-IND_{j2}$: accès-ensemble-indexé ($typelem_2$, TC_2)
⋮
$ENS-IND_{jj}$: accès-ensemble-indexé ($typelem_j$, TC_j)

$\overline{typelem_j} = typelem_1, \ldots, typelem_j$.
$\overline{TC_j} = TC_1, \ldots, TC_j$
$\overline{\underline{type_j}} = \underline{type_1}, \ldots, \underline{type_j}$.%

Opérations

produire (AP : application-parallèle; E : accès-ensemble-indexé ($typelem_k, TC_k$);
M : $typelem_k$) $\acute{A}P$: application-parallèle;

pré
∃ i, E = $ENS-IND_{ji}$(@AP) ∧
(∃ P : accès-processus, P ∈ PROC(@AP) ∧
NOM(@E) ∈ SORTIE(@P))

% L'accès processus P est un producteur de l'accès ensemble-indexé E %

post
$\acute{A}P$ = chang [9] (@AP, $ENS-IND_{ji}$, $\acute{E}$)
$\acute{E}$ = chang(@E, IND(@E), $\acute{E}I$)
$\acute{E}I$ = prod-EI (IND(@E), M)

consommer (AP : application-parallèle; E : accès-ensemble-indexé ($typelem_k, TC_k$))
$\acute{A}P$: application-parallèle;

pré

[9] L'opération : chang, est une opération définie sur le constructure de produit cartésien. Elle consiste à modifier la valeur d'un de ses composants. Par exemple : chang (@AP, $ENS-IND_{ji}$, $\acute{E}$) est une opération qui modifie la valeur de $ENS-IND_{ji}$. La nouvelle valeur de $ENS-IND_{ji}$ est $\acute{E}$.

$\exists$ i, E = $ENS - IND_{ji}$ (@AP) $\wedge$
($\exists$ P : accès-processus, P $\in$ PROC(@AP) $\wedge$
NOM(@E) $\in$ ENTREE(@P)) $\wedge$
pré-cons-EI (IND (@E))

%L'accès processus P est un consommateur de E, et pré-cons-EI %

post
$\acute{AP}$ = chang (@AP, $ENS - IND_{ji}$, $\acute{E}$)
$\acute{E}$ = chang(@E, IND(@E), $\acute{EI}$)
$\acute{EI}$ = post-cons-EI (IND(@E))

message-à-consommer (AP : application-parallèle;
E : accès-ensemble-indexé ($typelem_k, TC_k$)) M : $typelem_k$;

pré
$\exists$ i, E = $ENS - IND_{ji}$ (@AP) $\wedge$
($\exists$ P : accès-processus, P $\in$ PROC(@AP) $\wedge$
NOM(@E) $\in$ ENTREE(@P)) $\wedge$
pré-cons-EI (IND (@E))

%L'accès processus P est un consommateur de E, et pré-cons-EI %

post M = val-cons-EI(IND(@E))

End

Avec cette formalisation de l'application parallèle nous satisfaisons une première partie de la définition d'une application répartie (voir schéma).

Seul l'aspect de communication d'une application parallèle a été pris en compte dans cette formalisation. Ni les variables locales, ni les instructions du calcul d'un processus ne sont considérés. Les opérations définies sur l'application parallèle sont les opérations de communication exécutées par les processus et agissant sur les objets de communication représentés ici par : accès-ensemble-indexé.

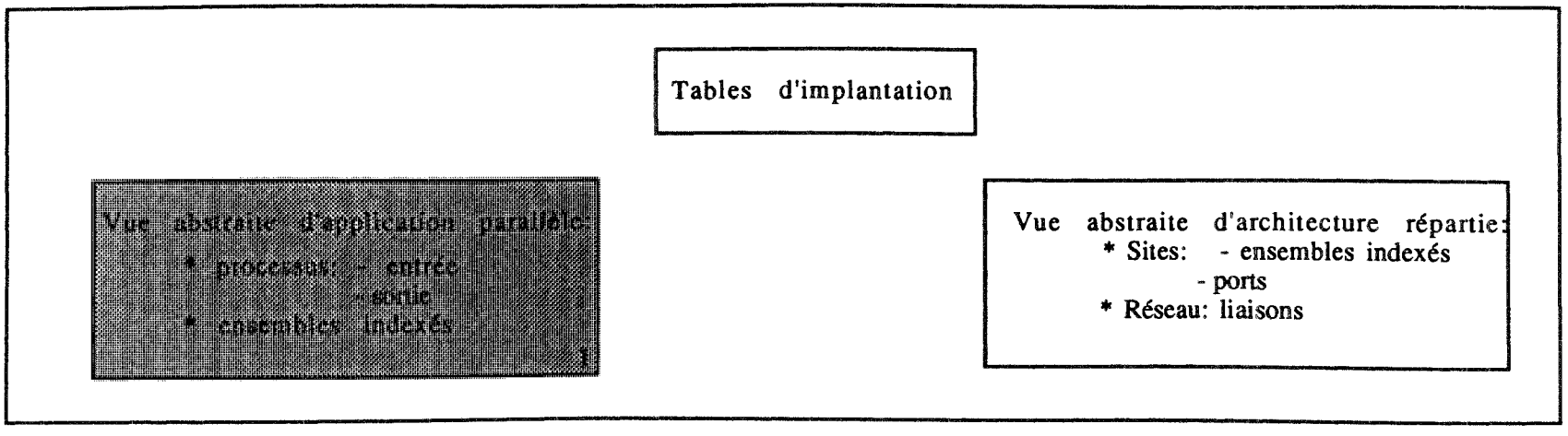
La formalisation d'une application parallèle se déduit facilement à partir de son programme écrit en LESP, en effet il suffit de considérer :

- la partie communication du programme pour déterminer les accès-ensembles-indexés de l'application parallèle,
- les interfaces de processus pour déterminer les accès-processus de l'application parallèle.

Nous allons constater, sur des exemples, la facilité de la déduction de la formalisation d'une application parallèle à partir de son programme écrit en LESP. Cette formalisation est la seule partie de l'application qui sera considerée lors de son implantation sur des architectures réparties.

le choix du langage LESP, comme langage de base de notre étude, a entrainé quelques implications sur notre formalisation. Citons ces implications :

Application répartie



- La définition du type accès-ensemble-indexé est fondé sur la définition du type ensemble-indexé et la notion du type de communication du langage LESP.
- Les entrées-sorties des processus sont explicites dans le langage LESP. En conséquence le passage d'un processus du programme à l'accès-processus correspondant de l'application parallèle peut être automatique.
- Les opérations de communication définies sur le type application-parallèle sont inspirées des opérations de communication de langage LESP.

Ces implications ne présentent pas des restrictions sur notre formalisation car :

- Le type accès-ensemble-indexé peut être considéré comme l'abstraction du moyen de communication défini dans les langages de programmation parallèle. Ce moyen est nommé en général : port, canal, etc. En fait la diversité des types de communication nous permettra de choisir celui qui convient au moyen de communication du langage de programmation utilisé.
- Si les entrées-sorties des processus ne sont pas explicites dans le langage de programmation, ils peuvent être déduites du contexte processus.
- Les opérations de communication de l'application parallèle peuvent être considérées comme des abstractions des primitives de communication des autres langages de programmation parallèle (envoyer, recevoire, etc).

1.2.2 Exemples

Exemple 1

La spécification du système parallèle donnée en (1.2.2), et qui calcule le *pgcd* de "n" nombres entiers est :

accès-ensemble-indexé :

SC = (%NOM% : sc, %IND% : $EI2$(booléen, TC(aléatoire))),

$\forall$ i=1..n+1, SP_i = (%NOM% : sp_i, %IND% : EP_i(entier, TC(égalité)));

$\forall$ i=1..n, SR_i = (%NOM% : sr_i, %IND% : ER_i(entier, TC(égalité)));

$\forall$ i=1..n, SC_i = (%NOM% : sc_i, %IND% : EC_i(entier, TC(égalité)));

accès-processus :

P = (%NOM% : np, %ENTREE% : {sc}, %SORTIE% : {sp_i, i=1..n+1 }),

$\forall$ i=1..n, P_i = (%NOM% : np_i, %ENTREE% : {sp}, %SORTIE% : {sr_i, sc_i });

Q = (%NOM% : nq, %ENTREE% : { sp,(sr_i; i=1..n),(sc_i; i=1..n)}, %SORTIE% : {sc});

application-parallèle :

APPL1 =

(%PROC% : { P_i(P_i; i=1..n), Q },
%ENS-IND% : { SC, (SP_i; i=1..n+1), (SR_i; i=1..n), (SC_i; i=1..n) })

Exemple 2

Prenons comme exemple, le système parallèle représenté par le graphe de la figure 9. Ce système est composé de trois processus (P1, P2, Q), et de quatre objets de communication (N1, N2, F1, F2). P1 produit des messages dans (N1, F1), les messages de N1 sont consommés par P2 et par Q, tandis que les messages de F1 sont consommés par Q. P2 produit des message dans (N2, F2), les messages de N2 sont consommés aussi bien par P1 que par Q, qui consomme les messages de F2. La topologie des objets (N1, N2) est la diffusion et leur type de communication est aléatoire, par conséquent les consommateurs ne partagent pas le même ensemble d'objets. Comme le type de communications de (N1, N2) est aléatoire, les processus consommateurs ne consomment pas nécessairement la même suite de messages.
Les objets (F1, F2) sont des bipoints et leur type de communication est égalité.

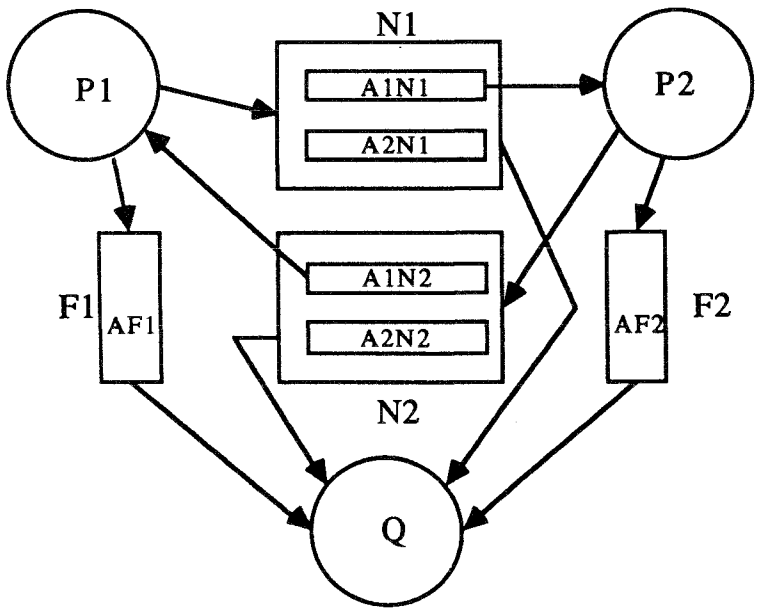


Figure 1.9

La description du système par LESP en négligeant la partie calcul des processus, est :

Processus **S** ::

Communication N1, N2 : aléatoire; F1, F2 : égalité.

Processus **P1** ::

entrée n2 : entier via N2;

sortie n1 : entier vers N1; f1 :entier vers F1;

Corps

opérations de calcul,

opérations de communication :
produire(n1), consommer(n2), produire(f1)

End P1

Processus **P2** ::

entrée n1 : entier via N1;

sortie n2 : entier vers N2; f2 : entier vers F2;

Corps

opérations de calcul,

opérations de communication :
produire(n2), consommer(n1), produire(f1)

End P2

Processus **Q** ::

entrée n1 :

entier via N1,
n2 : entier via N2,
f1 : entier via F1,
f2 : entier via F2;

Corps

opérations de calcul,

opérations de communication :
consommer(n1), consommer(n2), consommer(f1), consommer(f2)

End Q

composition P1 // P2 // Q

End S

Avant de donner la formalisation du système notons que N1 et N2 sont des objets de topologie diffusion. Chacun est composé de deux composants :

N1 = (A1N1, A2N1)
N2 = (A1N2, A2N2)

La spécification du système en utlisant le langage de formalisation est :

accès-ensemble-indexé :

A1N1 = (%NOM% : a1n1, %IND% : EI1(entier, TC(aléatoire))),

A2N1 = (%NOM% : a2n1, %IND% : EI2(entier, TC(aléatoire)))

A1N2 = (%NOM% : a1n2, %IND% : EI3(entier, TC(aléatoire))),

A2N2 = (%NOM% : a2n2, %IND% : EI4(entier, TC(aléatoire))),

AF1 = (%NOM% : af1, %IND% : EI5(entier, TC(égalité))),

AF2 = (%NOM% : af2, %IND% EI6(entier, TC(égalité)));

accès-processus :

P1 = (%NOM% : np1, %ENTREE% : {a1n2}, %SORTIE% : {a1n1, a2n1, af1}),

P2 = (%NOM% : np2, %ENTREE% : {a1n1}, %SORTIE% : {a1n2, a2n2, af2}),

Q = (%NOM% : nq, %ENTREE% : { a2n1, a2n2, af1, af2 }, %SORTIE% : { });

application-parallèle :

APPL2 =

(%PROC% : { P1, P2, Q },
%ENS-IND% : { A1N1, A2N1, A1N2, A2N2, AF1, AF2})

Conclusion

L'objectif de ce chapitre était de donner une formalisation d'applications parallèles afin d'en formaliser les implantations sur des architectures réparties. Pour satisfaire cet objectif nous avons choisi comme applications de base, les applications décrites par LESP. Cette formalisation peut généralement être considérée comme une abstraction de tous les langages asynchrones à base de processus communicants.

La modularité et la qualité du langage ainsi que le concept du type de communication nous ont permis de formaliser rigoureusement ces applications.

Chapitre 2

La formalisation d'un système réparti

Introduction

Le domaine de réseaux et systèmes répartis a connu ces dernières années une évolution rapide : évolution technique, avec la généralisation des réseaux à grande distance, le développement des stations de travail et celui des réseaux; évolution conceptuelle tant sur le plan d'analyse et de modélisation que sur celui des méthodes et de conception. Par ailleurs, l'évolution des microprocesseurs a permis l'apparition de machines à mémoire distribuée telles que : iPSC (hypercube), Super-Node (réseau reconfigurable), etc.

Malgré cette évolution, il n'existe pas actuellement de méthodes générale permettant la répartition de programmes parallèles sur les différents sites d'une architecture répartie.

Comme nous avons mentioné dans l'introduction de cette partie, la formalisation de notre travail : l'implantation de pogrammes parallèle sur des architectures réparties, nécessite l'abstraction des différents concepts intervenant lors de cette implantation. Dans le chapitre précedent nous avons présenté une formalisation du concept logique de l'implantation : l'application parallèle. **Nous nous intéressons dans ce chapitre à la formalisation du concept physique de l'implantation : l'architecture répartie.** Cette architecture est la cible de l'implantation.

La formalisation des architectures réparties est une vue abstraite de ces architectures qui met en évidence leurs caractéristiques principales sans tenir compte de l'aspect physique et technique lié à une architecture particulière. Dans la formalisation de l'architecture répartie nous tenons compte du problème posé qui est celui de l'implantation et nous la définissons adaptée à ce problème.

Ayant la formalisation de l'application parallèle d'une part (chapitre 1) et la formalisation de système réparti d'autre part, une fonction d'implantation réalisera l'implantation de l'application parallèle sur l'architecture répartie en fournissant une application répartie (cf : chapitre suivant).

Dans ce chapitre nous présentons dans une première partie le concept de système réparti au travers de quelques exemples de systèmes existants. Ensuite, dans une deuxième partie, nous

donnons la formalisation de ces systèmes répartis en termes de "pré" et "post" conditions.

2.1 Les systèmes répartis

Un système réparti est un ensemble de machines informatiques (sites dans un réseau, et processeurs dans une machine répartie) reliées par un système de communication qui leur permet d'échanger des informations. Il présente à ses utilisateurs une machine virtuelle, facilitant l'écriture et l'exécution des applications à distance. Il fournit un ensemble de services et vise à assurer l'utilisation optimale et le partage équitable du matériel, la protection et la conservation sûre de l'information.

Sur chaque machine du système nous trouvons un noyau, des processus système, des processus d'application, et des ressources. Le noyau permet de contrôler l'exécution de processus et de garder une certaine indépendance vis-à-vis les autres machines. Chaque processus se trouve sur une seule machine et il communique avec d'autres processus du système par échange de messages.

Le support de communication est un réseau local de topologie quelconque (bus, étoile, anneau, grille ou reconfigurable). Il doit assurer la fiabilité de la transmission ce qui veut dire que les messages sont transmis sans perte, sans duplication, sans altérnance ni erreur. Le mode de transmission sur le réseau est synchrone ou asynchrone.

La répartition d'informations et de services dans un système réparti introduit des problèmes nouveaux que nous pouvons notamment attribuer aux causes suivantes :

- Distance accrue entre la machine virtuelle à réaliser (qui reste proche de celle fournie par les systèmes centralisés) et le réseau de machines physiques, souvant hétérogènes, communiquant par des voies sujettes aux retards et aux défaillances.
- Impossibilité, pour l'allocateur de ressources, d'avoir une perception exacte de l'état de l'ensemble du système; difficulté pour concilier une gestion globale des ressources avec l'autonomie souhaitée pour chaque site.

Nous allons présenter les caractéristiques et l'intérêt d'un système réparti.

2.1.1 Pourquoi un système réparti ?

Malgré certains problèmes qui apparaissent lors de la répartition, l'étude et le développement de systèmes répartis sont en progrès. Ce progrès est dû aux facteurs suivants :

1. Les besoins de **communication** qui conduisent notamment à interconnecter des applications existantes, développées indépendamment, pour constituer un système unique intégré, tout en conservant à chaque site une certaine autonomie.
2. Le souhait d'**augmenter la disponibilité** d'un système informatique en permettant le remplacement immédiat d'un composant défaillant par un composant équivalent : ce mode de fonctionnement nécessite une communication permanente entre les composants du système, ainsi qu'un logiciel prévu pour gérer les reconfigurations et reprises en cas d'incident.

3. Le souhait de **partager les ressources** en mettant en commun les ressources les plus couteuses, et en permettant éventuellement une répartition de la charge par migration des programmes ou des données.

4. Le souhait de faciliter l'**évolution et l'adaptation** du système, par remplacement progressif ou addition de composants.

5. Le souhait d'**adapter la structure d'un système informatique à celle de l'application qu'il traite** : c'est notamment le cas des systèmes de conduite de procédés industriels (ou système en "temps réel") où il est maintenant courant d'associer un ou plusieurs processeurs à chaque élément du système commandé; la gestion globale de l'ensemble, ainsi que les considérations de disponibilité déjà mentionnées, imposent une communication entre ces processeurs et l'utilisation d'un logiciel adapté.

6. La recherche de la puissance de calcul et de la minimisation de temps d'exécution implique la répartition des programmes parallèles.

2.1.2 Qu'est ce qu'un système réparti ?

La répartition de données, de programmes, de logiciels, et de fonctions système implique certaines caractéristiques de système réparti. Les principales caractéristiques sont :

1. Les systèmes répartis présentent un degré élevé de **parallélisme réel**.

2. L'absence de mémoire commune entre les processus situés sur des sites différents; en conséquence la **communication par messages** et le seul moyen d'échange d'information ou de synchronisation entre les processus.

3. L'**absence de référence temporelle** commune et la variabilité des délais de transmission a pour conséquence l'impossibilité de définir un état global du système.

4. La répartition augmente les **risques de défaillance**, cela est dû au nombre élevé de composants et le fait que la fiabilité des systèmes de communication est généralement inférieure à celles des systèmes informatiques qu'ils relient.

Le progrès dans le domaine des systèmes répartis nécessite la résolution des problèmes dus à la répartition, et l'introduction de nouveaux mécanismes permettant d'éliminer ou de réduire l'effet des défaillances (reprise après panne, redondance, reconfiguration, etc). Des études ont été développées pour résoudre ces problèmes. Dans [Mat88] [Lam78] nous trouvons une définition d'une référence temporelle commune aux différents sites d'un système réparti, ce qui est équivalent à la détermination de l'état global de système [HPR88a]. Pour pouvoir exécuter les différents programmes dans un environnement réparti, il a fallu étudier la synchronisation entre les processus [Boc79], et le partage de la mémoire dans un tel environnement [UW87] [FR86].

D'autre part des algorithmes ont été introduits pour la détection de la terminaison de calcul effectué sur des systèmes répartis [Mat87], et pour la détection de l'interblocage des processus [HMR85]. L'objectif visé par ces algorithmes est de réduire l'effet de défaillance. Pour réaliser

le même objectif le système SATURNE, qui est un système réparti tolérant les fautes et les intrusions volontaires ou accidentelles, a été développé au LAAS-CNRS-INRIA [DFLP85]. Ce système apporte des solutions aux besoins des systèmes informatiques répartis, en matière de la fiabilité par la tolérance aux fautes, et en matière de confidentialité par la tolérance aux intrusions.

2.2 Quelques systèmes répartis existant

Les premiers systèmes informatiques répartis sont apparus au début des années 80, TRIX [War 80] MEDUSA [OSS80]. La plupart d'entre eux sont des systèmes expérimentaux, qui permettent d'approfondir les problèmes liés à la répartition, la fiabilité, la synchronisation, le contrôle de l'exécution, l'expression du parallélisme, et l'allocation de ressources. Il existe actuellement plusieurs systèmes répartis INCAS [NHM+87], SOS [SAG+87], ACCENT [RR81]. Certains sont en cours de développement GUIDE [Kra88], et d'autres sont disponibles commercialement CHORUS [ZGMB84]. Malgré la commercialisation, de nombreaux problèmes de conception restent ouverts.

Les systèmes existant différent par :

- leur conception : les entités manipulées, le mode et les primitives de communication et de synchronisation,
- leur structure : les différentes couches système,
- leur fonctionnalité : les fonctions système assurées localement et celles assurées à distance.

Dans [Gui84], nous trouvons une synthèse et une étude comparative de quelques systèmes répartis existant.

Pour clarifier les idées, nous décrivons les principes de quelques systèmes répartis existant afin de dégager leur points communs et de donner une vue abstraite convenable à ces système, et adaptée à notre objectif.

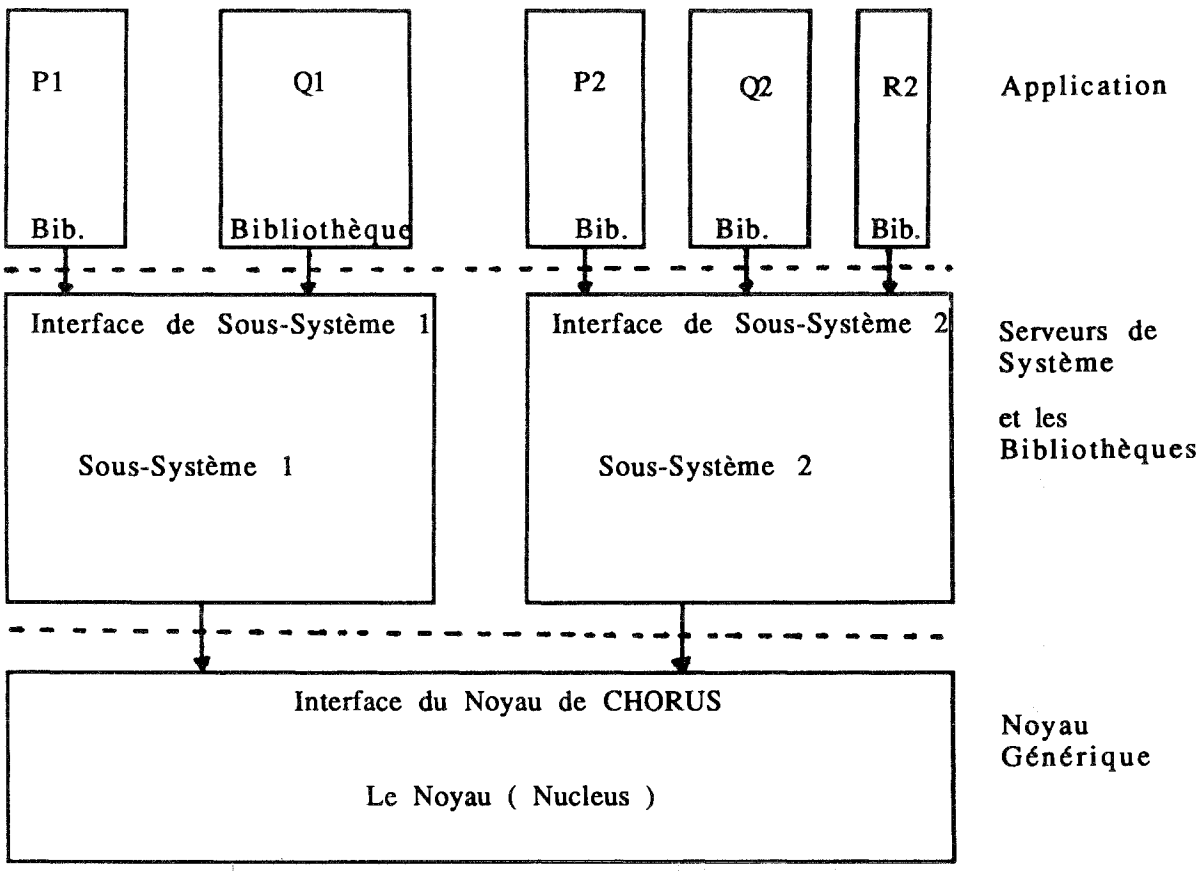
2.2.1 CHORUS

C'est un système réparti, ouvert et portable construit autour d'un ensemble réduit de concepts simples et généraux. Il propose une méthode de conception d'applications réparties et supporte l'exécution de ces applications indépendamment de la nature des supports matériels d'exécution et de communication.

Le **système d'exploitation** de CHORUS est structuré selon deux niveaux logiques : le **noyau** (Nucleus) et les **serveurs** (System Servers), figure 1. Le noyau assure deux sortes de services de base :

- service local tel que : calcul local, allocation locale des processeurs, gestion de mémoire virtuelle etc.
- service global : la communication tranparente entre les processus **IPC** (Inter-Processes Communication).

Les serveurs coopèrent, dans le contexte de "Subsystem", pour fournir aux utilisateurs du système un ensemble de services cohérant et une interface simple à manipuler. Ils se communiquent par le protocole IPC, assuré par le noyau.



L'Architecture de CHORUS

Figure 2.1

Dans CHORUS nous pouvons distinguer deux sortes d'entité : les entitées actives et les entités de communication, figure 2.

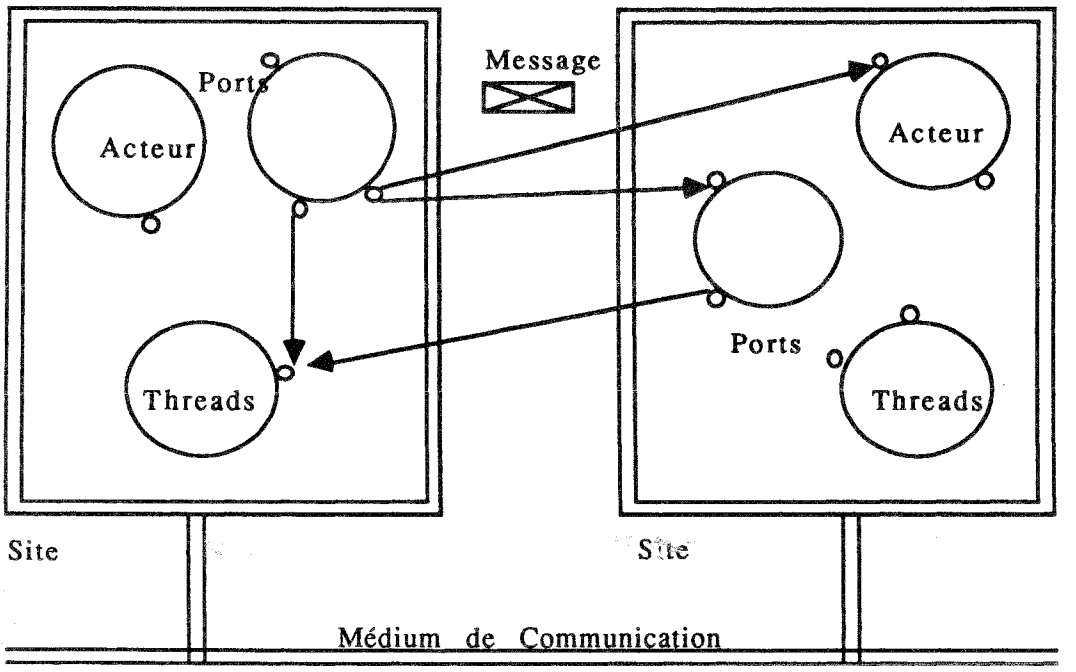
Les entités actives

Il existe trois sortes d'entités actives dans CHORUS, ces entités sont :

- **Sites** : Le **support physique** de CHORUS est composé d'un ensemble de **sites** connectés par un **réseau**. Chaque site est une machine locale qui comporte un ensemble de ressources physiques, un ou plusieurs processeurs, une mémoire centrale, et des interfaces Homme-Machine. Sur chaque site il existe un noyau du système.
- **Acteurs** : L'acteur est l'unité logique de distribution dans CHORUS. Il définit l'espace

mémoire partagé, et protégé, par un ensemble de threads (processus). Chaque acteur est attaché à un site, et les threads supportés par un acteur sont exécutés toujours sur le site d'attachement de leur acteur. Plusieurs acteurs peuvent être attachés simultanément au même site.

- **Threads** : Le thread est l'unité d'exécution de CHORUS. C'est une séquence d'instructions de contrôle et de traitement caractérisée par le contexte de thread. Chaque thread est attaché à un et un seul acteur qui définit l'espace mémoire utilisé par le thread. Les threads d'un acteur partagent les ressources (mémoire et ports) et ils peuvent s'exécuter en parallèle.
  La communication et la synchronisation entre les threads sont réalisées par échange de messages même s'ils sont sur le même site (protocole IPC). Mais la synchronisation des threads d'un acteur peut se réaliser par un mécanisme basé sur la mémoire partagée.



Les entités principales manipulées par CHORUS

Figure 2.2

Les entités de communication

Les threads se communiquent par échange de **messages** via des **ports**. Chaque thread peut communiquer et synchroniser avec n'importe quel thread exécutable sur n'importe quel site. La communication dans CHORUS est trasparente vis-à-vis la localisations des threads.

- **Messages** : Le message est une suite de caractères transmis d'un émetteur à un récepteur. La transmission est réalisée par une copie logique du message de l'espace mémoire de l'émetteur à l'espace mémoire du récepteur.

- **Ports :** Les messages sont transmis entre les threads par l'intermédiaire des ports. Un port représente : une ressource pour la mémorisation de message et une addresse à laquelle les messages sont envoyés.
  Au niveau du système, la notion de port permet une séparation entre l'interface d'une service et son implantation ce qui facilite en conséquence la reconfiguration dynamique.
  L'établissement de connexion entre deux ports n'étant pas nécessaire, les communications locales et distantes sont banalisées.
  Chaque port est attaché à un seul acteur, et il peut être utilisé par tous ses threads pour l'envoi ou la réception de messages. Il peut migrer d'un acteur à un autre durant sa période de vie.

Le protocole IPC de CHORUS permet deux modes d'échange de message : le mode **asynchrone** et le mode de communication par **appel de procédure.**

Selon CHORUS une application répartie est donc un ensemble d'acteurs et un ensemble de threads.

Le système CHORUS est conçu pour une large classe d'applications désirant tirer parti de la répartition en ce qui concerne la reconfiguration dynamique, le partage des ressources et la tolérance aux fautes. Il a été développé en plusieurs étapes. Dans la première version V0, CHORUS a été implanté sur Intel 8086 connectés par le un réseau "Danube" a 50 Kb/s. Le prototype est écrit en Pascal. La version V1 a été réalisé sur des sites SM90 (Motorola 68000) connectés par Ethernet 10 Mb/s. Et dans la dernière version V2, il a fallu adapter CHORUS selon UNIX.

2.2.2 INCAS (Incremental Architecture for Distributed System)

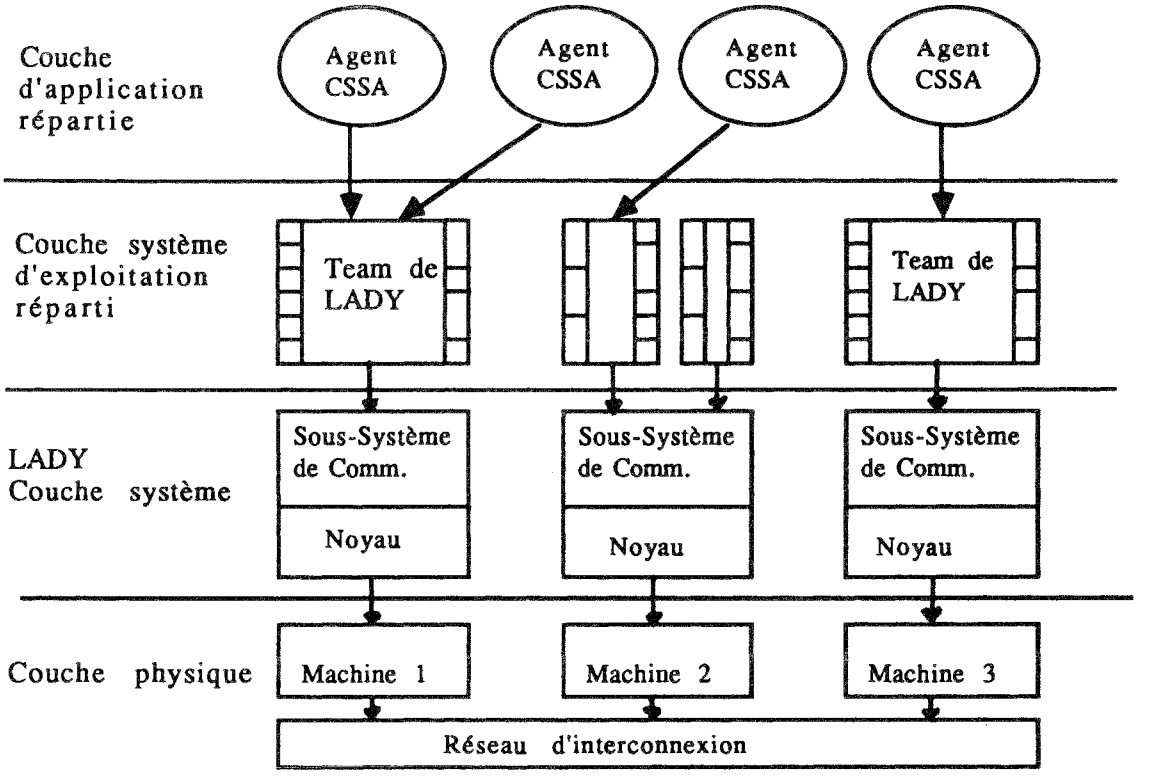
Il a été conçu pour le développement d'une méthodologie permettant la conception et l'implantation des systèmes et des applications répartis.

INCAS est structuré selon trois niveaux : le niveau physique (un réseau local constitué de plusieurs machine identiques), le niveau système (assuré par LADY qui est un système d'exploitation réparti) et le niveau application (le langage de programmation de INCAS est CSSA (Computing System for Societies of Agents)). La figure 3 illustre la structure de INCAS.

Le niveau système de INCAS est divisé en deux couches : une couche assurant les fonctions du noyau (création et destruction des processus, synchronisation et communication locale, etc) et une couche de système réparti composée des unités communicantes réparties sur l'ensemble de machines. Ce système est écrit par un langage typé et structuré, nommé LADY.
Le concept de base de LADY est le "team", chaque "team" est composé d'un ensemble de processus coopérent pour satisfaire une fonction précise de système, et ils communiquent par une mémoire partagée. Le "system" est un autre concept de LADY, il est défini recursivement, chaque "system" est composé d'un ensemble de "system" et d'un ensemble de "team". La figure 4 montre les différentes couches de LADY.

Les "teams" sont les unités de répartition, chaque "team" est placé sur une machine, et les différents "teams" peuvent être placés sur la même machine ou sur des machines différentes.

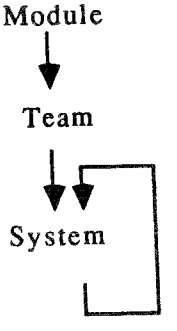
La partie visible d'un "team" est son interface qui est composé de ports d'entrée et de ports de sortie. L'interaction entre "teams" est réalisé par des échanges de messages via leur interfaces.



La structure de INCAS

Figure 2.3

L'établissement d'une connexion entre un port d'entrée et un port de sortie est nécessaire pour la transmission de message. Deux types de connexion sont définis dans LADY, une con-



La structure de LADY

Figure 2.4

nexion simple qui lie un port de sortie à un port d'entrée, et une connexion multiple "bus" qui lie plusieurs ports de sortie à plusieurs ports d'entrée. Dans LADY nous distinguons deux sortes de transmission : la transmission bipoint et la diffusion. La communication sous LADY est une communication unidirectionnelle, fiable si elle est bipoint. Les ports d'interface, et par conséquent les connexions, sont typés. Chaque port peut être associé à un buffer; si les ports d'une connexion sont associés à des buffers, la communication sur cette connexion est asynchrone, et elle synchrone dans le cas contraire.

CSSA est un langage de programmation de haut-niveau permettant l'expression des algorithmes répartis. Il est plus abstrait que LADY, en conséquence, les détails de synchronisation et de communication ne sont pas considérés dans CSSA. Les composants de base de CSSA sont les "agents". Les "agents sont des objets actifs communicant par échange de messages, le seul moyen de communication dans CSSA.

2.2.3 MACH

MACH [Fis89] [JR86] est un système d'exploitation conçu pour assurer les calculs parallèles et répartis sur un réseau de processeurs ou de multi-processeurs. Il est développé au (Carnegie-Mellon-University (CMU)). Sa première version a été réalisé en 1986 sur les mini-ordinateurs DEC-VAX. Actuellement MACH existe sur plusieurs machines : IBM PC/RT, les Sun3, Encore Multix, etc.
Malgré la compatibilité avec le système Berkley 4.3 Unix, la conception et l'organisation du système MACH sont différentes de celles d'Unix. En fait les concepteurs de MACH ont choisi Unix comme système de référence accuse de sa réputation sur le marché americain.
MACH est le système réparti concurrent de système CHORUS, et il est possible d'imaginer que MACH remplacera Unix dans un futur proche.

MACH est fondé sur quelques concepts de base de système d'exploitation. Les concepts de communication sont:

- **Task** : c'est l'environnement d'exécution d'un programme. Il constitue l'unité de base d'allocation de ressources et comporte un espace virtuel d'adressage et des droits d'accès aux ressources.

- **Thread** : c'est l'unité de base de l'exécution de MACH. Plusieurs threads peuvent partager les ressources d'un task et chaque thread est attaché à un seul task.

- **Port** : c'est le canal de communication qui assure la transmission de messages. Il est implanté comme une queue de messages contrôlée par le noyau.
  Les messages sont transmis entre les threads et entre les tasks via les ports.

- **Message** : c'est une collection de données d'une taille déterminée ou quelconque. Il constitue l'unité de communication et ses données peuvent être typées ou non typées.

L'échange par message est le moyen de communication entre les threads et entre les tasks et le noyau du système.

Les seules fonctions-système réalisées par des interruptions sont celles concernantes la communication par messages : msg_send, msg_receive. Toutes les autres fonctions sont assurées par échange de messages.

Les présentations brèves de MACH et CHORUS montrent que les concepts : thread, port et message sont des concepts communs et que l'acteur de CHORUS joue le rôle de task de MACH.

2.2.4 Autres systèmes

ARGUS [Lis88] [LS83] est un langage de programmation et un système développés pour l'implantation et l'exécution des programmes répartis. Il fournit des mécanismes permettant la résolution de certains problèmes de programmation répartie tels que les défaillances des machines à distance et des réseaux de communication.
ARGUS est conçu pour des logiciels qui conservent les informations échangées en ligne pour une longue periode de temps. Parmi ces logiciels nous citons le système de courrier éléctronique et le système de transfert de fichiers.
ARGUS est fondé sur deux concepts : le guardian et l'action. Le guardian est l'unité de la distribution d'Argus. C'est un objet abstrait qui contrôle localement l'accès à une ou plusieurs ressources physiques ou logiques. Le concept d'action dans ARGUS assure l'atomicité des transactions locales ou réparties.
En resumé ARGUS assure : la concurrence, la répartition, la consistance malgré le risque de défaillance et l'extensibilité du réseau de processeurs.

GOTHIC est un autre système d'exploitation développé à l'INRIA-IRISA-Rennes. L'objectif de ce système est la réalisation d'une mémoire virtuelle globale. Cet objectif est aussi l'un des objectifs de ACCENT (A communication oriented network operating system kernel), les autres objectifs d'ACCENT sont : un degré élevé du parallélisme, la fiabilité, la modularité, la transparence, et la détection rapide des erreurs.
Dans une autre approche, l'approche langage à objets de système réparti, nous trouvons CSA [BMH87] et SOS [SAG+87].

Comme dernier système nous citons HELIOS [Bal90]. Il a été conçu comme un système d'exploitation distribué. Sa première implantation a été réalisée pour le transputer, bien que sa conception lui permettre d'accomoder d'autres processeurs ayant des mécanisme de communication et de gestion de mémoire différents. Les concepts de base d'HELIOS sont décrits dans le chapitre 6.

2.3 La formalisation de systèmes répartis

Quelques études ont été développées pour spécifier des problèmes concernant les systèmes répartis. Une spécification des protocoles de communication se trouve dans [BR83]. Dans une autre approche, l'objectif de l'étude présentée dans [CY83], est la spécification du comportement et de la structure de système réparti. Deux relations fondamentales entre les événements sont utilisées pour exprimer les propriétés concernant le contrôle réparti et la transmission

d'informations. Ces relations sont la relation de précédence et la relation de causalité, elles permettent, entre autre, d'ordonner les événements exécutés sur les différents sites.

SPANNER [ABM88] est un environement logiciel pour l'analyse, la spécification, et la vérification de problèmes répartis, précisement les protocoles de communication. Il est basé sur le modèle de machine à état fini, appelé modèle sélection/ résolution.

Avant de donner les formalisations, il faut noter l'existence des différences entre notre travail et les travaux développés pour spécifier, les programmes parallèles [Lam83], les systèmes concurrents [BR83], ou les systèmes répartis [CY83]; pour concevoir un système d'exploitation réparti comme CHORUS [ZGMB84], GOTHIC [BB86], INCAS [NHM+87]; ou pour introduire un langage de programmation réparti CSSA [MB85].

Ces différences sont dues au fait que nous nous intéressons à l'implantation répartie des programmes parallèles et la formalisation de système réparti est développée pour réaliser cet objectif. La charge du système, la disponibilité de ressources , etc, ne sont pas considérés lors de la formalisation. Notre travail peut être vu comme étant un **outil abstrait pour la répartition des applications parallèles.**

Seuls les éléments et les opérations nécessaires pour réaliser cette implantation sont considérés lors de la formalisation. Cette formalisation doit préciser la **structure** du système, son **comportement** et le **changement d'état** suite à certaines opérations.

Comme l'aspect de communication est le seul aspect pris en compte dans la formalisation d'application parallèle il sera aussi le seul aspect considéré dans la formalisation de système réparti. Nous nous intéressons donc aux opérations d'interaction entre les composants du système réparti.

2.3.1 Formalisation d'un système réparti

Un système réparti est une vue abstraite d'une architecture répartie. Il est composé de sites et d'un ensemble de liaisons, figure 5. Intuitivement chaque site est l'abstraction d'un site de l'architecture, et l'ensemble de liaisons représente un réseau logique de cette architecture.
Lors de l'abstraction nous considérons que le réseau de communication de l'architecture répartie est un réseau complétement maillé et il assure la fiabilité de la transmission, ce qui veut dire que les messages sont transmis sans perte, sans duplication, sans altérnance ni erreur.

Chaque site du système répartie est composé :

- d'une interface composée de ports de réception et de ports d'émission,
- d'une mémoire vue comme des ensembles indexés,

On voit ainsi que cette formalisation ne retient de l'idée mémoire que des structures de données abstraites d'ensembles indexés. Sans perte de généralité un tel modèle permettra de se concentrer sur l'aspect de communication de l'architecture répartie et sur notre problématique

d'implantation répartie d'applications parallèles.
Un port de réception d'un site reçoit les messages d'autres sites et il les dirige vers un ensemble indexé. Un port d'émission reçoit les messages d'ensembles indexés et les dirige vers d'autres sites du système.

Une liaison dans un système réparti est l'association de deux ports du système, l'un est un port de sortie d'un site et l'autre est un port d'entrée d'un autre site.

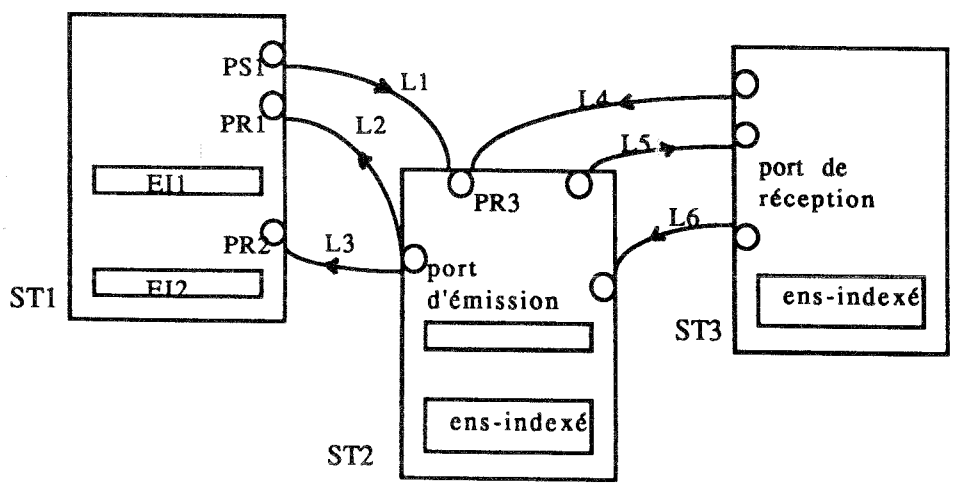
Exemple La figure 5 illustre un système réparti composé de :

- trois sites : { ST1, ST2, ST3 },
- six liaisons : { L1, L2, L3, L4, L5, L6 }

Le site ST1 comporte :

- deux ensembles indexés : { EI1, EI2 },
- une interface composée d'un port d'émission : { PS1 }, et deux ports de réception : { PR1, PR2 }.

La liaison L1 relie le port d'émission PS1 de ST1, au port de réception PR3 de site ST2.



Un système réparti

Figure 2.5

Afin de formaliser la notion de système réparti nous allons définir son type ainsi que les types de ses différents composants. Nous présentons ces types d'une manière ascendente.

Le type **port** est introduit pour formaliser les ports d'interface de sites de système réparti. Chaque objet de ce type est une **suite** paramétrée par le type de ses messages et sa taille. Trois opérations sont définies sur le type port : mettre, enlever, et prendre; les opérations mettre et enlever modifient l'état des objets de ce type.

- **mettre** : c'est une opération permettant l'insertion d'un message dans le port,
- **enlever** : c'est une opération permettant la suppression d'un message du port,
- **prendre** : c'est une opération qui définit le message à transmettre à un instant donné.

Comme pour les composants d'une application parallèle, nous allons associer chaque objet du système réparti à un nom de l'objet, et nous définissons le type **accès-port** comme étant le produit cartésien d'un **nom** d'un port et d'un **port**. Ce type est paramétré par le type de données et la taille de son port.

Le nom d'un objet de type accès-port est la partie immuable (statique) de l'objet, elle n'est pas modifiable par les opérations appliquées sur l'objet. Le port d'un tel objet est la partie variable (dynamique) permettant d'exprimer le changement d'état lors de l'exécution des opérations sur l'objet.

De même nous définissons le type **accès-site** qui est le produit cartésien du **nom** du site (partie immuable) et d'un **site** (partie variable).

Le type **site** est le produit cartésien de :

- **plusieurs accès-ensemble-indexé**
- **plusieurs port d'émission**
- **plusieurs port de réception**

Le type **accès-site** ainsi que le type site sont paramétrés par :

- les types de données des ensembles indexés,
- les triplets d'opérations caractéristiques des ensembles indexés,
- les types de données de ports d'émission,
- les tailles de ports d'émission,
- les types de données de ports de réception,
- les tailles de ports de réception.

Nous précisons que l'introduction des noms dans les différents types : accès-port, accès-site, etc, est nécessaire pour :

- pouvoir préciser à la fois la structure et le comportement d'un objet d'un type donné,
- séparer la partie immuable de la partie variable d'un objet,
- pouvoir désigner un objet sans qu'il soit impliqué par cette désignation.

Une liaison est une association de deux ports de système réparti, elle fait référence aux ports à travers leur nom. Elle est donc un objet immuable du système. Le type **liaison** est défini comme étant le produit cartésien de **nom du port d'émission** et de **nom du port de réception** de la liaison. Les ports désignés par une liaison doivent être paramétrés par le même type de message.

Nous définissons le type **système-réparti** comme étant le niveau abstrait de l'architecture répartie. Ce type est le produit cartésien d'un **ensemble de liaison** et de **plusieurs sites**. Et il est paramétré par les différentes paramètres de ses sites. La transmission de messages dans le système réparti est décrite par une opération "**transmettre**" sur une liaison. Cette opération permet de préciser le changement d'état du système résultant de la transmission. Elle est conditionnée par une pré-condition sur l'état du système.

La formalisation de système réparti est donnée en termes de pré, post condition [1]. Elle constitue la seconde partie de la définition d'une application répartie (voir schéma).

Dans la définition de certains types nous avons introduits :

- des conditions dont le rôle est de restreindre l'expression du type en précisant la relation existant entre les paramètres du type et son expression.
- des prédicats qui précisent l'existance, ou non existance, d'une relation déterminée entre les composants du type. Nous les avons adopté afin de simplifier l'écriture des pré-condition des opérations du type.

Type **n-port**

Expression chaine de caractères;

% Un objet de ce type est un nom de port %

End

Type **n-site**

Expression chaine de caractères;

% Un objet de ce type est un nom de site %

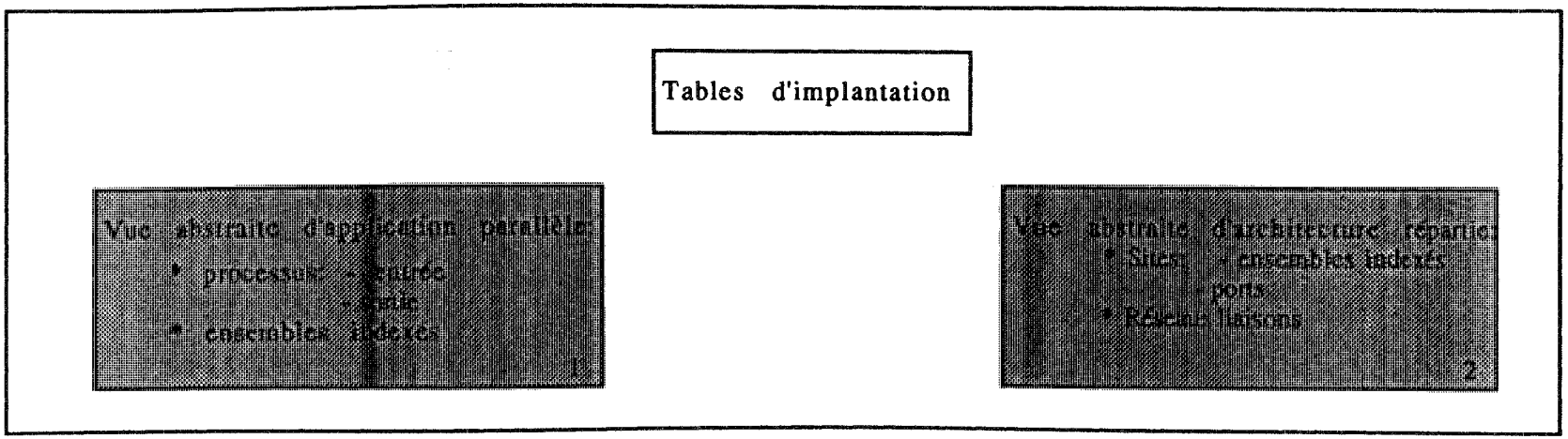
End

Type **port (typelem : type, L : entier)** [2]

[1] Voir "Notations" dans le chapitre 1 pour la lecture de la formalisation d'un système réparti

[2] Les opérations : plein, vide, taille, premier, retirer, et cons sont des opérations définies sur le type suite.

Application répartie



Expression : suite (typelem)

Condition : $\forall$ P : port, taille (@P) $\leq$ L

Opérations :

mettre (P : port, M : typelem) $\acute{P}$: port;

pré non plein (@P)

post $\acute{P}$ = cons(@P, M)

enlever (P : port) $\acute{P}$: port;

pré non vide (@P)

post $\acute{P}$ = retirer(@P)

prendre (P : port) M : typelem;

pré non vide (@P)

post M = premier(@P)

End

Type **accès-port (typelem : type, L : entier)**

Expression < NOM : n-port, PORT(typelem, L) : port >;

End

Type $\boldsymbol{site_{ijk}}$ ($\overline{\boldsymbol{typelem_i}}$: $\overline{\boldsymbol{type_i}}$, $\overline{\boldsymbol{TC_i}}$, $\overline{\boldsymbol{typelem_j}}$: $\overline{\boldsymbol{type_j}}$, $\overline{\boldsymbol{L_j}}$: $\overline{\boldsymbol{entier_j}}$, $\overline{\boldsymbol{typelem_k}}$: $\overline{\boldsymbol{type_k}}$, $\overline{\boldsymbol{L_k}}$: $\overline{\boldsymbol{entier_k}}$)

Expression

< $\overline{EIS_i}$: accès-ensemble-indexé($\overline{typelem_i}$, $\overline{TC_i}$);
$\overline{EMISSION_j}$: accès-port($\overline{typelem_j}$,$\overline{L_j}$);
$\overline{RECEPTION_k}$: accès-port($\overline{typelem_k}$,$\overline{L_k}$) >

% Où

$\overline{EIS_i}$ =

EIS_{i1} : accès-ensemble-indexé ($typelem_1$, TC_1)
EIS_{i2} : accès-ensemble-indexé ($typelem_2$, TC_2)
⋮
EIS_{ii} : accès-ensemble-indexé ($typelem_i$, TC_i)

$\overline{EMISSION_j}$ =

$EMISSION_{j1}$: accès-port ($typelem_1$, L_1)
$EMISSION_{j2}$: accès-port ($typelem_2$, L_2)
⋮

$EMISSION_{jj}$: accès-port ($typelem_j$, L_j)

$\overline{RECEPTION_k}$ =

$RECEPTION_{k1}$: accès-port ($typelem_1$, L_1)

$RECEPTION_{k2}$: accès-port ($typelem_2$, L_2)

⋮

$RECEPTION_{kk}$: accès-port ($typelem_k$, L_k)

$\overline{typelem_n} = typelem_1, \ldots, typelem_n.$

$\overline{TC_n} = TC_1, \ldots, TC_n$

$\overline{L_n} = L_1, \ldots, L_n$

$\overline{type_n} = type_1, \ldots, type_n.$

$\overline{entier_n} = entier_1, \ldots, entier_n.$ %

End

Type $\boldsymbol{acces - site_{ijk}}$ ($\overline{\boldsymbol{typelem_i}}$: $\overline{\boldsymbol{type_i}}$, $\overline{\boldsymbol{TC_i}}$, $\overline{\boldsymbol{typelem_j}}$: $\overline{\boldsymbol{type_j}}$, $\overline{\boldsymbol{L_j}}$: $\overline{\boldsymbol{entier_j}}$, $\overline{\boldsymbol{typelem_k}}$: $\overline{\boldsymbol{type_k}}$, $\overline{\boldsymbol{L_k}}$: $\overline{\boldsymbol{entier_k}}$)

Expression

< NOM : n-site,
OBJ : $site_{ijk}$ ($\overline{typelem_i}$, $\overline{TC_i}$, $\overline{typelem_j}$, $\overline{L_j}$, $\overline{typelem_k}$, $\overline{L_k}$) >

End

Type **liaison**

Expression < SEND : n-port, RECEIVE : n-port >

End

Remarques :

Pour simplifier la notation nous ne précisons pas la taille des paramètres d'un système réparti. Ces tailles seront précisées lors de développement des exemples dans ce chapitre et dans le chapitre suivant.

De même les paramètres ne seront pas précisées dans la formalisation des opérations que s'il y a des conditions portant sur ces paramètres.

Type $\boldsymbol{systeme - reparti_n}$ ($\overline{\boldsymbol{typelem - ind}}$: $\overline{\boldsymbol{type}}$, $\overline{\boldsymbol{TC - ind}}$, $\overline{\boldsymbol{typelem - pe}}$: $\overline{\boldsymbol{type}}$, $\overline{\boldsymbol{L - pe}}$: $\overline{\boldsymbol{entier}}$, $\overline{\boldsymbol{typelem - ps}}$: $\overline{\boldsymbol{type}}$, $\overline{\boldsymbol{L - ps}}$: $\overline{\boldsymbol{entier}}$)

Expression

< RESEAU : ensemble(liaison),

$\overline{SITES_n}$: accès-site ($\overline{typelem_{ni}}$, $\overline{TC_{ni}}$, $\overline{typelem_{nj}}$, $\overline{L_{nj}}$, $\overline{typelem_{nk}}$, $\overline{L_{nk}}$) >

% Où

$\overline{SITES}_n =$

$\overline{SITES}_{n1}$: accès-site ($\overline{typelem_{1i}}$, $\overline{TC_{1i}}$, $\overline{typelem_{1j}}$, $\overline{L_{1j}}$, $\overline{typelem_{1k}}$, $\overline{L_{1k}}$)

$\overline{SITES}_{n2}$: accès-site ($\overline{typelem_{2i}}$, $\overline{TC_{2i}}$, $\overline{typelem_{2j}}$, $\overline{L_{2j}}$, $\overline{typelem_{2k}}$, $\overline{L_{2k}}$)

$\vdots$

$\overline{SITES}_{nn}$: accès-site ($\overline{typelem_{ni}}$, $\overline{TC_{ni}}$, $\overline{typelem_{nj}}$, $\overline{L_{nj}}$, $\overline{typelem_{nk}}$, $\overline{L_{nk}}$)%

Prédicats

1. **send-sp** : accès-site * accès-port * liaison → booléen
   send-sp (S, P, L)⇔ P ∈ $\overline{EMISSION}$ (@OBJ (@S)) ∧ NOM (@P) = SEND(@L)

   % send-sp (S,P,L) est vrai, si le port P est un port d'émission de site S, et s'il est le port "SEND" de liaison L %
2. **receive-sp** : accès-site * accès-port * liaison → booléen
   receive-sp (S, P, L)⇔ P ∈ $\overline{RECEPTION}$ (@OBJ (@S)) ∧
   NOM (@P) = RECEIVE (@L)

   % receive-sp (S,P,L) est vrai, si le port P est un port de réception de site S, et s'il est le port "RECEIVE" de liaison L %

Condition :

∀ L : liaison, L ∈ RESEAU (@SR) ⇒
(∃ S1, S2 : accès-site, S1 ∈ $\overline{SITES}$ (@SR), S2 ∈ $\overline{SITES}$ (@SR), S1 ≠ S2) ∧
(∃ P1 (typelem, K), P2 (typelem, J) : accès-port),
send-sp(S1,P1, L) ∧ receive-sp(S2,P2, L)

% Chaque liaison L de système réparti lie un port d'émission (P1) d'un site (S1) à un port de réception (P2) d'un autre site (S2) du système réparti (SR). Les deux ports sont paramétrés par le même type de message : typelem %

Opération :

transmettre (SR : système-réparti, L : liaison) $\acute{SR}$: système-réparti;

pré

L ∈ RESEAU (@SR) ∧
(∃ (P1, P2 : accès-port), ∃ (S1, S2 : accès-site),
S1 ∈ $\overline{SITES}$ (@SR) ∧ S2 ∈ $\overline{SITES}$ (@SR) ∧
send-sp (S1, P1, L) ∧ receive-sp (S2, P2, L) ∧
non vide (@PORT(P1)) ∧ non plein (@PORT (P2)))

%Le port PORT(@P1) désigné par SEND(@L) n'est pas vide et le port PORT(@P2) désigné par RECEIVE(@L) n'est pas plein.%

post

SR^1 = chang (@SR , S1, $\acute{S}1$)
$\acute{SR}$ = chang (@SR^1 , S2, $\acute{S}2$)

$\acute{S}1$ = chang (@S1, P1, $\acute{P}1$)
$\acute{S}2$ = chang (@S2, P2, $\acute{P}2$)
PORT(@$\acute{P}1$) = enlever (PORT(@P1))
PORT(@$\acute{P}2$) = mettre (PORT(@P2), prendre (P1))

End

2.3.2 Exemples

Prenons comme exemple le système réparti illustré par la figure 6. La spécification de ce système est :

accès-ensemble-indexé :

EI_1 = (%NOM% : ens_1, %IND% : ET_1(entier, TC(égalité))),

EI_2 = (%NOM% : ens_2, %IND% : ET_2(réel, TC(aléatoire))),

EI_3 = (%NOM% : ens_3, %IND% : ET_3(entier, TC(très-aléatoire))),

EI_4 = (%NOM% : ens_4, %IND% : ET_4(réel, TC(élastique))),

EI_5 = (%NOM% : ens_5, %IND% : ET_5(entier, TC(égalité))),

accès-port :

PS_1 = (%NOM% : ps_1, %PORT% : PT_1(entier,4)),

PS_2 = (%NOM% : ps_2, %PORT% : PT_2(réel,2)),

PS_3 = (%NOM% : ps_3, %PORT% : PT_3(entier,1)),

PS_4 = (%NOM% : ps_4, %PORT% : PT_4(entier,2)),

PS_5 = (%NOM% : ps_5, %PORT% : PT_5(réel,2)),

PR_1 = (%NOM% : pr_1, %PORT% : PT_6(réel,3)),

PR_2 = (%NOM% : pr_2, %PORT% : PT_7(réel,2)),

PR_3 = (%NOM% : pr_3, %PORT% : PT_8(entier,4)),

PR_4 = (%NOM% : pr_4, %PORT% : PT_9(réel,5)),

PR_5 = (%NOM% : pr_5, %PORT% : PT_{10}(entier,2)),

site

ST_1 =

(%EIS% : { EI_1, EI_2 };
%EMISSION% : {PS_1};
%RECEPTION% : {PR_1, PR_2})

ST_2 =

(%EIS% : { EI_3, EI_4 };
%EMISSION% : { PS_2, PS_3};
%RECEPTION% : { PR_3, PR_4 })

ST_3 =

(%EIS% : { EI_5 };
%EMISSION% : {PS_4, PS_5 };
%RECEPTION% : PR_5 });

accès-site

S_1 = (%NOM% : s_1, %OBJ% : ST_1)

S_2 = (%NOM% : s_2, %OBJ% : ST_2)

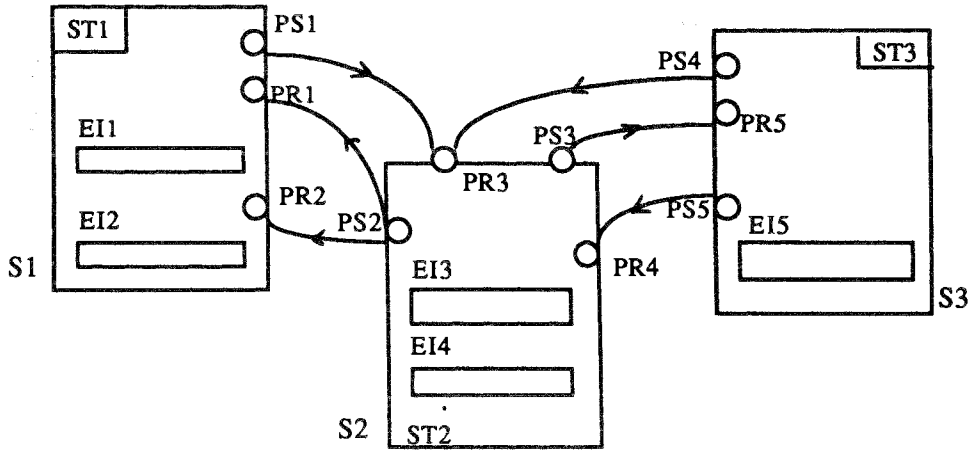
S_3 = (%NOM% : s_3, %OBJ% : ST_3)

système-réparti

SR =

(%RESEAU% : { $(ps_1, pr_3), (ps_4, pr_3), (ps_2, pr_1), (ps_2, pr_2), (ps_3, pr_5), (ps_5, pr_4)$ };

%SITES% : { S_1, S_2, S_3 })



Exemple d'un système réparti (SR)

Figure 2.6

Conclusion

Afin de formaliser l'implantation d'une application parallèle sur une architecture répartie, nous avons donné dans ce chapitre une abstraction de cette architecture, nommée **système réparti**.

Ce formalisme s'inspire des concepts fondamentaux de systèmes répartis existant tels que : CHORUS, INCAS. Et il nous servira pour la définition de l'application répartie.

Partie II

La répartition d'une application parallèle

Dans cette partie nous allons définir formellement l'application répartie : l'implantation d'une application parallèle sur un système réparti, et nous présentons un procédé pour sa construction à partir d'une application parallèle. Ce procédé est nommé : la fonction d'implantation.

La fonction d'implantation définit un mécanisme permettant la répartition d'une application parallèle. Elle est donc le moyen de passage d'une application parallèle à une application répartie correspondante.

Deux problèmes interviennent lors de la répartition d'une application parallèle :

- le placement des constituants de l'application parallèle sur les sites du système réparti,
- la création de l'application répartie correspondante à un placement donné.

Deux approches ont été suivies lors de développement de la fonction d'implantation : une approche heuristique pour la détermination du placement et une approche méthodologique pour la création de l' application répartie.

La **fonction d'implantation** est composée de deux fonctions :

- La **fonction heuristique de placement** dont le rôle est de choisir un " bon " placement des constituants de l'application parallèle sur le système réparti.
- La **fonction de dérivation de l'application répartie** dont le rôle est de créer cette application et de donner la représentation des opérations de communication dans l'environnement réparti.

Pour définir le mécanisme de la répartition d'une application parallèle nous allons :

- Définir la structure de l'application répartie : à partir de la formalisation d'une application parallèle et de la formalisation d'un système réparti, nous déduisons la définition d'une application répartie (cf : chapitre 3).
- Présenter les règles heuristiques et la fonction de placement (cf : chapitre 4).
- développer la fonction de dérivation de l'application répartie (cf : chapitre 5).

Chapitre 3

La formalisation d'une application répartie

3.1 Définition d'une application répartie

Une application répartie est l'implantation d'une application parallèle sur un système réparti. C'est une extension d'un système réparti permettant l'exécution d'une application parallèle donnée.

Plusieurs applications réparties peuvent être les implantations d'une même application parallèle. Le choix et la création d'une implantation parmi plusieurs possibles est réalisé par la fonction d'implantation. Une application répartie est donc le résultat d'exécution d'une fonction d'implantation sur une application parallèle.
La fonction d'implantation est composée de deux fonctions :

- Une fonction de placement : son rôle est de déterminer le placement de composants de l'application parallèle sur les sites du système réparti.
- Une fonction de dérivation : son rôle est la création d'une application répartie correspondant à un placement donné.

La fonction de placement est une fonction non-déterministe fondée sur des règles heuristiques de placement et elle tient compte de contraintes et de choix d'utilisateur (cf : chapitre 4). Par ailleurs la fonction de dérivation est une fonction déterministe définie d'une manière rigoureuse (cf : chapitre 5).

Précisons que l'introduction de l'application répartie nous a permis de distinguer entre deux problèmes liés à l'implantation des programmes parallèles sur les architectures réparties. Ces deux problèmes sont distincts et par conséquent leurs résolutions sont distinctes. Le premier est lié à la notion de répartition dans le sens général du terme (caractéristiques d'un système réparti), et le deuxième est lié à l'architecture physique choisie comme cible d'implantation et au langage de programmation.
La résolution du premier est donnée dans cette partie de la thèse et la résolution du deuxième

est développée dans la troisième partie.

Dans ce chapitre nous nous limitons à la présentation de la structure de l'application répartie. Le procédé de sa génération et les différentes fonctions de création sont données dans les chapitres 4 et 5. Ce chapitre est divisé en deux parties : dans la première partie nous précisons la structure et la définition formelle d'une application répartie, et dans la deuxième partie nous montrons quelques exemples d'applications réparties.

3.2 Formalisation d'une application répartie

Une application répartie est l'implantation d'une application parallèle sur un système réparti. Dans cette implantation chaque processus de l'application parallèle est exécuté sur un site du système réparti et chaque ensemble indexé est placé sur un de ces sites. Le placement de composants de l'application parallèle sur les sites du système réparti est indiqué par des tables de placement.

En d'autre terms une application répartie est une extension d'un système réparti permettant l'exécution d'une application parallèle. Elle est constituée donc d'un ensemble de sites et d'un ensemble de liaisons.

Afin de formaliser la notion d'application répartie nous allons introduire le type **application répartie** : qui est une vue abstraite de l'implantation d'une application parallèle sur un système réparti. Ce type est paramétré par :

- une famille de tables de placement,
- un ensemble de processus,
- les paramètres de son système réparti :
  - les types de données des ensembles indexés,
  - les triplets d'opérations caractéristiques des ensembles indexés,
  - les types de données de ports d'émission,
  - les tailles de ports d'émission,
  - les types de données de ports de réception,
  - les tailles de ports de réception,

  De même que pour le système réparti et pour simplifier la notation nous ne précisons pas la taille des paramètres de l'application réparti. Ces tailles seront précisées dans les exemples.

La famille de tables de placement est constituée de plusieurs tables qui indiquent : le placement de composants de l'application parallèle sur les sites du système réparti et les associations entre certains composants du système réparti. Ces associations permettent l'exécution des opérations de communication de l'application parallèle dans l'environnement réparti.

Ces tables sont :

- une table (TAB-PROC) indiquant le placement des processus de l'application parallèle sur les sites du système réparti. Les entrées de cette table sont les noms de processus, et chaque élément indique le site de placement du processus désigné par l'entrée,
- une table (TAB-COM) indiquant le placement des ensembles indexés, de l'application parallèle sur les sites du système réparti. Les entrées de cette table sont les noms des ensembles indexées, et chaque élément indique le site de placement de l'ensemble indexé désigné par l'entrée,
- une table (TAB-EMISSION) indiquant l'association des ensembles-indexés aux ports d'émission du système. Les entrées de cette table sont les noms de certains ensembles indexés de l'application parallèle. Chaque élément de la table indique le port d'émission utilisé pour envoyer les messages, retirés d'un ensemble indexé, aux autres sites du système. L'ensemble indexé, désigné par une entrée, et le port d'émission associé doivent être des composants d'un même site,
- une table (TAB-RECEPTION) indiquant l'association des ensembles-indexés aux ports de réception du système. Les entrées de cette table sont les noms de certains ensembles indexés de l'application parallèle. Chaque élément de la table indique le port de réception utilisé pour recevoir les messages destinés à l'ensemble indexé de son entrée. L'ensemble indexé, désigné par une entrée, et le port de réception associé doivent être des composants du même site.

Nous définissons le type **famille-tab-place** comme étant le produit cartésien de quatre tables : TAB-PROC, TAB-COM, TAB-EMISSION, TAB-RECEPTION .

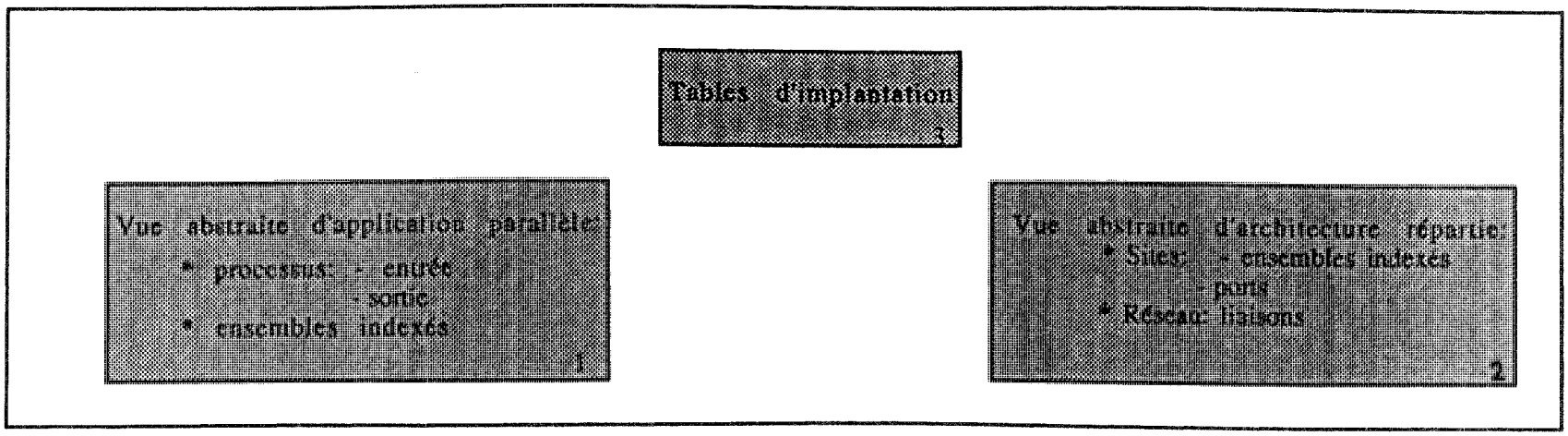
Les processus de l'application répartie sont les processus de l'application parallèle. De même les ensembles indexés des différents sites de l'application répartie sont les ensembles indexés de l'application parallèle.

Les opérations définies sur l'application répartie sont les images des opérations de communication sur un système réparti. Ces images : φ(produire), φ(consommer), φ(message-à-consommer); sont définies par la fonction de dérivation de l'application répartie, et elles sont décrites dans le chapitre 5.

Les associations des composants de l'application répartie décrites par les différentes tables, sont exprimées par l'association des noms de ces composants, en conséquence ces tables sont des éléments immuables (non modifiable à l'éxecution) de l'application répartie.
La définition de l'application répartie se termine par la définition des tables de placement (voir schéma).

La costruction d'une application répartie nécessite la détermination de paramètres de son type et sa création. Cette construction est réalisée par une fonction nommée la fonction d'implantation.
Vu que l'implantation nécessite des informations concernant les sites du système réparti : la

Application répartie



cible d'implantation, la fonction d'implantation est donné en fonction du système réparti. La fonction d'implanatation est définie sur une application parallèle est elle rend une application répartie.

Comme nous l'avons précisé dans l'introduction, la fonction d'implantation est composée de deux fonctions chacune réalise une étape de la construction :

- la fonction de placement : qui détermine les paramètres de l'application répartie et plus précisement la famille de tables de placement (les autres paramètres sont déduites à partir de l'application parallèle),
- la fonction de dérivation : qui crée l'application répartie correspondant à une application parallèle pour un placement donné. La création d'une application répartie consiste à créer : les ensembles indexés, les ports d'émission, les ports de réception et le réseau de l'application répartie.

Notons :

- ρ : une famille de fonction d'implantation,
- θ : une fonction d'implantation correspondant à un système réparti donné,
- δ : une famille de fonction de placement,
- ζ : une fonction de placement correspondant à un système réparti donné,
- η : une famille de fonction de dérivation,
- φ : une fonction de dérivation correspondant à un système réparti donné,
- τ : une famille de tables de placement,
  τ : **famille-tab-place**
- $\mathcal{R}$: des règles heuristiques de placement.
- soit α : une application parallèle,
  α : **application-parallèle(t1, t2,..,tn)**
- soit r : un système réparti,
  r : **système-réparti(t1, t2, .., tm)**
- soit ar : une application répartie,
  ar : **application-répartie(τ, β, t1, t2, ..,tm)**

Avec ces notations nous pouvons écrire :

- $\rho = \eta \circ \delta$
- $\theta = \varphi \circ \zeta$
- $\theta = \rho\ (\ r\ )$

- θ : **application-parallèle** $\xrightarrow{\mathcal{R}}$ **application-répartie**
  Ce qui est équivalent à :
  $ar = \theta\ (\ \alpha\ ) = \rho\ (\ r\ )\ (\ \alpha\ ) = r_\alpha$
  avec
  $\tau = \zeta\ (\ \alpha\ ), \beta =$ PROC (@α)

- $\zeta = \delta\ (\ r\ )$

- ζ : **application-parallèle** $\xrightarrow{\mathcal{R}}$ **famille-tab-place**
  Ce qui est équivalent à :
  $\tau = \zeta\ (\ \alpha\ ) = \delta\ (\ r\ )\ (\ \alpha\ )$

- $\varphi = \eta\ (\ r\ )$

- φ : **famille-tab-place * application-parallèle** $\longrightarrow$ **application-répartie**
  Ce qui est équivalent à :
  $ar = \varphi\ (\ \tau, \alpha\ ) = \eta\ (\ r\ )\ (\ \tau, \alpha\ )$

La formalisation d'une application répartie est donnée ci-dessus en terms de **pré** et **post** condition.

Type **famille-tab-place**

Expression

< TAB-PROC : table [n-processus] n-site,
TAB-COM : table [n-indexé] n-site,
TAB-EMISSION : table [n-indexé] n-port,
TAB-RECEPTION : table [n-indexé] n-port) >

End

Type **application-répartie (PLACEMENT : famille-tab-place, PROCESSES : ensemble (accès-processus),** $\overline{typelem-ind}$: $\underline{type}$, $\overline{TC-ind}$, $\overline{typelem-pe}$: $\underline{type}$, $\overline{L-pe}$: $\overline{entier}$, $\overline{typelem-ps}$: $\underline{type}$, $\overline{L-ps}$: $\overline{entier}$)

Expression

système-réparti($\overline{typelem-ind}, \overline{TC-ind}, \overline{typelem-pe}, \overline{L-pe}, \overline{typelem-ps}, \overline{L-ps}$),

Conditions :

1. $\forall$ X : accès-ensemble-indexé(typelem, TC), DA : application-répartie,
   NOM(@X) $\in$ domaine (TAB-EMISSION (@PLACEMENT))
   $\Rightarrow$
   ($\exists$ S : accès-site, S $\in \overline{SITES}$ (@DA),
   X $\in \overline{EIS}$(@OBJ (@S)))
   ($\exists$ P : accès-port(typelem.l), P $\in \overline{EMISSION}$(@OBJ (@S)) ,

TAB-COM(@PLACEMENT) [NOM (@X)] = NOM (@S)
TAB-EMISSION(@PLACEMENT) [NOM (@X)] = NOM (@P))

% Si X est un ensemble indexé du site S, et si P est le port d'émission associé à X alors X et P doivent être sur le site S et ils doivent être paramétrés par le même type de message : typelem %

2. ∀ X : accès-ensemble-indexé(typelem, TC), DA : application-répartie,
NOM(@X) ∈ domaine (TAB-RECEPTION (@PLACEMENT))
⇒
(∃ S : accès-site, S ∈ $\overline{SITES}$ (@DA),
X ∈ $\overline{EIS}$(@OBJ (@S)))
(∃ P : accès-port(typelem,I), P ∈ $\overline{RECEPTION}$(@OBJ (@S)) ,
TAB-COM(@PLACEMENT) [NOM (@X)] = NOM (@S)
TAB-RECEPTION(@PLACEMENT) [NOM (@X)] = NOM (@P))

% Si X est un ensemble indexé du site S, et si P est le port de réception associé à X alors X et P doivent être sur le site S et ils doivent être paramétrés par le même type de message : typelem %

End

Rappellons que les conditions du type application-répartie sont introduites afin de préciser les relations entre les paramètres du type et son expression.

3.3 Exemples

Nous avons présenté dans le chapitre 1, la formalisation de deux applications parallèles. Pour chacune de ces applications, nous allons donner une application répartie correspondante. La seule contrainte physique considérée lors du choix de l'application répartie est le nombre de sites de l'architecture répartie dont nous disposons. C'est à dire que l'on fait les hypothèses suivantes :

- les vitesses de transmission sont identiques entre les différents sites,
- le réseau de connexion est complètement maillé.

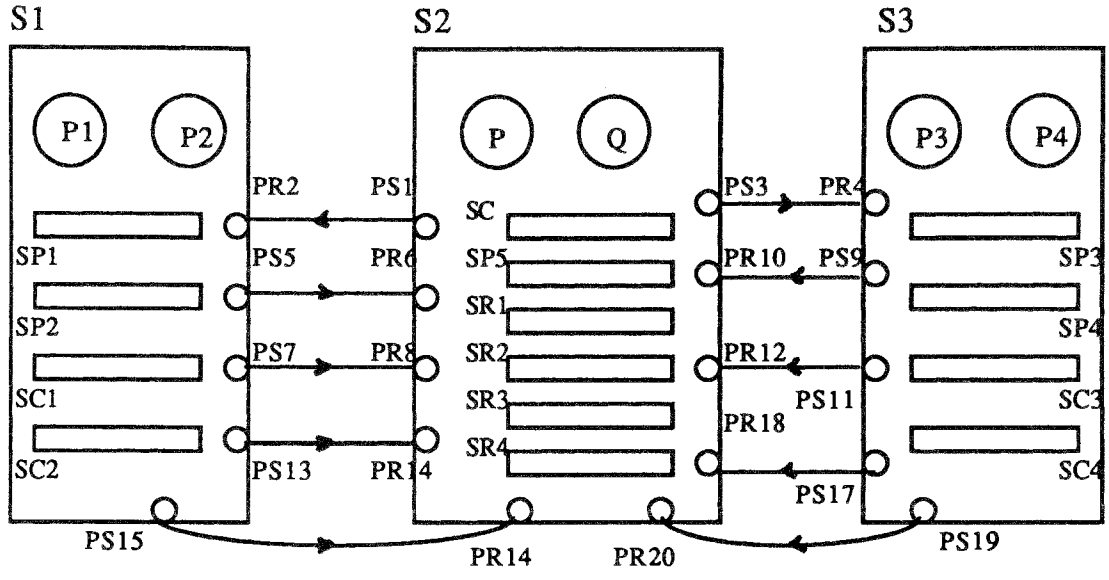
Comme nous l'avons déjà dit, le choix d'une application répartie parmi plusieurs possibles est fait par la fonction d'implantation. Une des règles de placement de processus sur les sites est d'avoir l'équité de chargement des sites en nombre de processus, ceci pour avoir le maximum de parallélisme.

3.3.1 Exemple 1

Pour donner la formalisation d'une application répartie correspondante à l'application parallèle de calcul de *pgcd*, prenons l'application calculant le *pgcd* de 4 nombres entiers (n = 4). Le

nombre de processus de cette application est 6 (cf : chapitre 1) qui sont (P, P_1, P_2, P_3, P_4, Q), et nous supposons que le nombre de sites de l'architecture dont nous disposons est 3.

L'application répartie représentée par à la figure 1, et dont la formalisation est donnée ci-dessous n'est pas nécessairement la meilleure, nous la citons ici juste comme exemple. Le meilleur choix sera discuté dans le chapitre 4.



Application répartie de calcul de pgcd

Figure 3.1

Les paramètres : PLACEMENT et PROCESSES de l'application répartie

PLACE1=

(%TAB-PROC% : [$np_1 \rightarrow s_1, np_2 \rightarrow s_1, np_3 \rightarrow s_3, np_4 \rightarrow s_3, np \rightarrow s_2,$ $nq \rightarrow s_2$],
%TAB-COM% : [$sp_1 \rightarrow s_1, sp_2 \rightarrow s_1, sc_1 \rightarrow s_1, sc_2 \rightarrow s_1, sc \rightarrow s_2,$ $sp_5 \rightarrow s_2, sr_1 \rightarrow s_2, sr_2 \rightarrow s_2, sr_3 \rightarrow s_2, sr_4 \rightarrow s_2, sp_3 \rightarrow s_3, sp_4 \rightarrow s_3,$ $sc_3 \rightarrow s_3, sc_4 \rightarrow s_3$],
%TAB-EMISSION% : [$sc_1 \rightarrow ps_{13}, sc_2 \rightarrow ps_{15}, sc_3 \rightarrow ps_{17}, sc_4 \rightarrow ps_{19}$],
%TAB-RECEPTION% : [$sp_1 \rightarrow pr_2, sp_2 \rightarrow pr_2, sp_3 \rightarrow pr_4, sp_4 \rightarrow pr_4,$ $sr_1 \rightarrow pr_6, sr_2 \rightarrow pr_8, sr_3 \rightarrow pr_{10}, sr_4 \rightarrow pr_{12}$])

PROC1= { P, P1, P2, P3, P4, Q }, (voir chapitre 1, exemple 1)

accès-port

$\forall$ i=1..10, PS_{2i-1} = (%NOM% : ps_{2i-1}, %PORT% : PT_{2i-1}(entier,1));

$\forall$ i=1..10, PR_{2i} = (%NOM% : pr_{2i}, %PORT% : PT_{2i}(entier,l));

site

ST_1 =

(%EIS% : { SP_1, SP_2, SC_1, SC_2 };
%EMISSION% : { PS_5, PS_7, PS_{13}, PS_{15} };
%RECEPTION% : { PR_2 })

ST_2 =

(%EIS% : { SC, SP_5, SR_1, SR_2, SR_3, SR_4 };
%EMISSION% : { PS_1, PS_3 };
%RECEPTION% : { PR_6, PR_8, PR_{10}, PR_{12}, PR_{14}, PR_{16}, $PR18$, PR_{20} }),

ST_3 =

(%EIS% : { SP_3, SP_4, SC_3, SC_4 };
%EMISSION% : { PS_9, PS_{11}, PS_{17}, PS_{19} };
%RECEPTION% : { PR_4});

accès-site

S_1 = (%NOM% : s_1, %OBJ% : ST_1)

S_2 = (%NOM% : s_2, %OBJ% : ST_2)

S_3 = (%NOM% : s_3, %OBJ% : ST_3)

système-réparti

SR1 =

(%RESEAU% : { ((ps_{2i-1}, pr_{2i}); i=1..10)};
%SITES% : { S_1, S_2, S_3 })

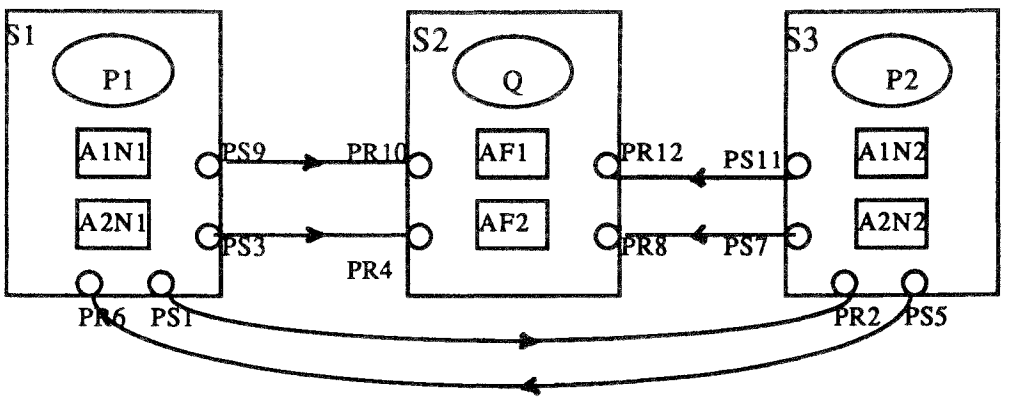
application-répartie

DA1 = DA1 (PLACE1, PROC1,...) = SR1

3.3.2 Exemple 2

Comme deuxième exemple nous considerons l'application parallèle APPL2 schématisé par la figure 9 de chapitre 1.
Supposons que l'architecture répartie soit composée de n sites, où n $\geq$ 3.
L'application parallèle est composée de trois processus, chaque processus est placé sur un site de l'architecture.

Une application répartie correspondante est donnée par la figure 2.



L'application répartie de l'exemple 2

Figure 3.2

La formalisation de cette application est :

Les paramètres : PLACEMENT et PROCESSES de l'application répartie

PLACE2 =

(% TAB-PROC% : [$np_1 \rightarrow s_1, np_2 \rightarrow s_3, nq \rightarrow s_2$];
%TAB-COM% : [a1n1 $\rightarrow s_1$, a2n1 $\rightarrow s_1$, a1n2 $\rightarrow s_3$, a2n2 $\rightarrow s_3$, af1 $\rightarrow s_2$, af2 $\rightarrow s_2$c];
%TAB-EMISSION% : [a1n1 $\rightarrow ps_1$, a2n1 $\rightarrow ps_3$, a1n2 $\rightarrow ps_5$, a2n2 $\rightarrow ps_7$];
%TAB-RECEPTION% : [af1 $\rightarrow pr_{10}$, af2 $\rightarrow pr_{12}$])

PROC2 : { P1, P2, Q }, (voir chapitre 1, Exemple 2)

accès-port :

$\forall$ i= 1..6, PS_{2i-1} = (%NOM% : ps_{2i-1}, %PORT% : PT_{2i-1}(entier,l)),

$\forall$ i= 1..6, PR_{2i} = (%NOM% : pr_{2i-1}, %PORT% : PT_{2i}(entier,l))

site

ST_1 =

(%EIS% : { A1N1, A2N1 };
%EMISSION% : { PS_1, PS_3, PS_9 };
%RECEPTION% : { PR_6 })

$\%ST_2$ =

(%EIS% : { AF1, AF2 };
%EMISSION% : { };
%RECEPTION% : { PR_4, PR_8, PR_{10}, PR_{12} })

ST_3 =

(%EIS% : { A1N2, A2N2 };
%EMISSION% : { PS_5, PS_7, PS_{11} };
%RECEPTION% : { PR_2 })

accès-site

$\forall$ i= 1..3, S_i = (%NOM% : s_i, %OBJ% : ST_i)

système-réparti

SR2 =

(%RESEAU% : { ((ps_{2i-1}, pr_{2i}); i=1..10) };
%SITES% : { S_1, S_2, S_3 })

application-répartie

DA2 = DA2 (PLACE2, PROC2, ...) = SR2

Conclusion

Afin de formaliser le mécanisme d'implantation d'une application parallèle sur une architecture répartie, nous avons donné dans ce chapitre la formalisation d'une application répartie, l'implantation de l'application parallèle sur une architecture répartie.

La déduction d'une application répartie correspondant à une application parallèle donnée, est réalisée par une **fonction d'implantation** développée dans les chapitres suivants. Cette fonction est un outil logiciel qui permettra le choix d'une application répartie appropriée correspondant à une application parallèle. Ceci en considérant certaines contraintes d'implantation imposées, par l'architecture dont nous disposons, ou par l'utilisateur.

La prise en compte de ces contraintes pour définir un placement des constituants de l'application parallèle est l'objet du chapitre 4.

A partir de ces règles de placement la définition de l'application répartie, et la représentation des opérations de communication dans un environnement réparti sont données au chapitre 5.

Chapitre 4

Le problème du placement

Introduction

La première étape de l'implantation d'une application parallèle sur un système réparti est le placement des constituants de l'application sur les sites du système.

Cette étape est réalisée par une fonction, nommée fonction de placement, dont le rôle est de choisir un placement convenable des composants de l'application parallèle sur les sites de système réparti.

La **fonction de placement est un outil interactif d'aide au placement** (système d'assistance). **Cet outil propose à l'utilisateur des solutions avec des évaluations de leurs qualités en fonction de certains critères.** L'utilisateur peut intervenir pour :

- modifier certaines solutions,
- imposer certaines contraintes,
- prendre une décision.

La fonction de placement peut être exécutée automatiquement, le choix d'une solution parmi plusieurs équivalentes est aléatoire dans ce cas.

Le contexte d'utilisation de la fonction de placement est le suivant :

- L'application parallèle est décrite par ses spécification (cf : chapitre 1), elle est donc constituée d'un ensemble de processus qui communiquent par l'intermédiaire d'objets de communication.
- Les architectures réparties sont constituées de plusieurs processeurs identiques reliés par un réseau permettant la transmission supposée parfaite des messages.

Le placement des processus et des objets de l'application parallèle sur les sites du système est un placement statique.
Lors du placement nous considérons que les processus s'exécutent en parallèle et qu'ils seront répartis sur tous les sites du système.

Chaque objet de communication a des processus producteurs et des processus consommateurs, chaque ensemble indexé de cet objet est placé autour d'un de ces processus selon la topologie et le type de communication de l'objet.

Notons que le modèle d'application parallèle et celui de l'architecture répartie considérés ici sont des modèles très simplifiés. Pour que cet étude soit plus réaliste nous pouvons la compléter par le résultat d'autres études de la litérature qui concernnent le problème de placement et qui sont mentionnées dans la suite de ce chapitre.
La complexité des processus n'est pas considérée, l'évaluation de la charge d'un site dépend, donc, du nombre de processus placés sur ce site. D'autre part l'évaluation de charge de réseau nécessite l'évaluation de nombre des messages échangés entre les sites. Ce nombre dépend des données et des contextes des processus placés sur les sites. Comme le placement est statique et les contextes des processus ne sont pas pris en compte, la charge du réseau est évaluée en fonction du nombre de liaisons correspondant au placement.

Les critères d'évaluation d'un placement sont :

- La valeur d'une fonction qui exprime la synchronisation entre les processus placés sur les différents sites.
- Le nombre de liaisons entre les sites.
- Le nombre de processus par site.
- Le nombre d'ensembles indexés par site, ce nombre mesure la place mémoire occupée du site.

4.1 Présentation générale du problème

Le problème du placement des processus ou des tâches d'une application parallèle sur les sites d'un système réparti est un problème connu dans le domaine du système réparti, figure 1. Il a pris de l'importance ces dernières années suite au développement, d'une part des langages parallèles (ADA, CSP,...etc), et d'autre part des architectures réparties (iPSC, Super-node,..etc). Le même problème se pose dans le domaine de base de données répartis pour le placement, d'un fichier, ou de différentes copies d'un fichier, sur l'ensemble de sites du système [BCS89].

Ce problème est un problème NP-complet. Nous trouvons dans la littérature des méthodes de résolution exacte dont le but est de trouver le placement optimal, et des méthodes de résolution approchée dont le but est de trouver un "bon" placement.
La résolution exacte d'un problème NP-complet nécessite un temps non polynominal d'exécution. Pour minimiser ce temps il est possible d'imposer certains compromis de résolution : par exemple le temps de communication est le seul facteur critique de placement, ou bien d'avoir une méthode de résolution pour une classe particulière d'architecture comme dans [BS87] où le graphe de l'architecture est un tableau linéaire de processeurs.

Les méthodes de résolution approchée sont plus intéressantes que celles de résolution exacte pour plusieurs raisons, d'une part leurs temps d'exécution est polynominal, d'autre part ces méthodes sont adaptées à un grand nombre d'applications et d'architectures.

Plusieurs algorithmes heuristiques existent pour résoudre le problème du placement. Certains ont été développés pour une classe particulière d'applications, les applications temps réel [DV87] [SS84] ; d'autres pour une classe particulière d'architectures, [Sei85] [LA87] [BS87] pour une architecture linéaire; ou pour des tâches d'une structure précise comme dans [Bok81a] où l'auteur considère la structure arborescente de tâches.

Malgré le but commun de ces algorithmes, qui est d'avoir le maximum de parallélisme et le minimum de communication à distance entre les tâches de l'application parallèle, les hypothèses concernant la classe de l'application ou la classe de l'architecture sont variées et par conséquent leurs critères d'évaluation sont distincts. Dans [BCS89] nous trouvons un classement des études concernant le problème de placement selon plusieurs modèles, les modèles sont basés sur : le nombre de processeurs de l'architecture, la topologie du réseau, le placement statique ou dynamique [Bok79] des tâches, la considération de contraintes de placement tel que la capacité de la mémoire [RSH79], les contraintes d'affectation [MLT82], et les contraintes de redondance.

Le coût de communication entre les tâches de l'application parallèle est un facteur pris en compte dans [DV87] tandis que l'existence d'une communication entre deux tâches est le seul facteur important pour le placement dans [Sei85].

En ce qui concerne l'architecture, certaines études supposent que les différents processeurs sont identiquement reliés [Lo84]. d'autres supposent que les relations inter-processeurs sont différentes et elles prennent en compte le coût de routage [DV87] [AP88] [PW87]. Il existe aussi des études qui considérent que les processeurs ainsi que leurs liens sont diversifiés [Ma84].

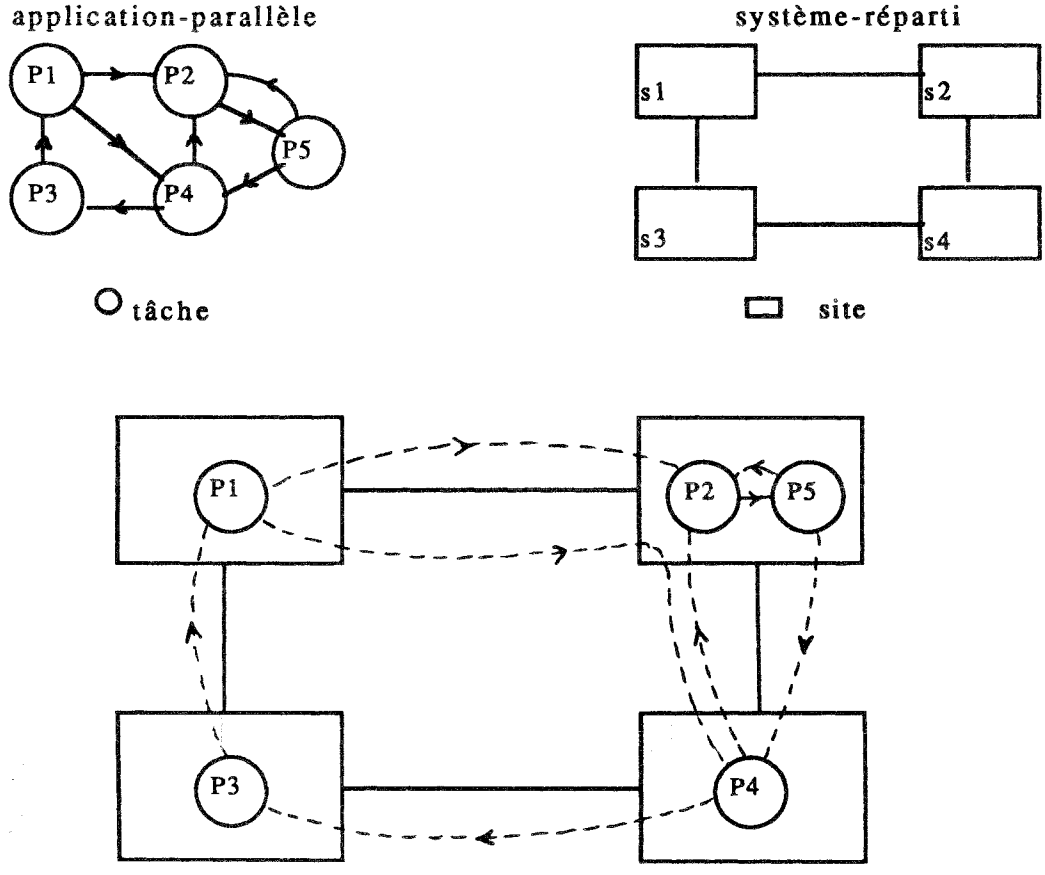
Dans [LA87] nous trouvons un algorithme de placement qui tient compte de contexte des tâches. Selon ces contextes l'algorithme distingue deux sortes de parallélisme entre les tâches : le parallélisme virtuel et le parallélisme réel. Ensuite il transforme le graphe de l'application en un autre graphe qui montre les deux sortes du parallélisme. Lors de placement de tâches, l'algorithme ne considére que le parallélisme réel entre les tâches.

L'article [AP88] donne une intéressante étude comparative entre les différentes classes d'algorithmes heuristiques existants, et les évaluations de certains de ces algorithmes appliqués sur le calculateur CHEOPS.

Dans les différentes études, les algorithmes des méthodes de résolution approchées sont fondées sur des règles heuristiques dépendant de l'application et de l'architecture.
Le schéma général de ces algorithmes est :

- Affectation initiale intelligente (fondée sur des règles heuristiques)
- Cycle
  - Évaluation du placement selon certains critères
  - Permutation
- Fin cycle
- Décision

La suite de ce chapitre décrit la fonction de placement que nous avons bâti. Il est organisé de la manière suivante : au paragraphe 4.2 nous citons les règles heuristiques qui guident le placement ainsi que la formulation des critères d'évaluation; ces règles seront utilisés par l'algorithme développé en 4.3. Nous terminons le chapitre par l'application de l'algorithme de placement sur un exemple 4.



Le placement d'une application parallèle sur un système réparti

Figure 4.1

4.2 Les règles heuristiques retenues

Nous distinguons deux sortes de règles heuristiques de placement :

- les règles de placement des objets de communication : ces règles determinent l'association des ensembles indexés des objets aux processus de l'application parallèle;
- les règles de placement des processus : ces règles determinent les processus les plus aptes à un regroupement éventuel.

Les règles de placement des objets de communication seront appliquées quel que soit le nombre de processus de l'application et le nombre de sites du système. Tandis que les règles de placement des processus seront appliquées dans le cas où le nombre de processeurs est inférieur au nombre de processus.

En effet dans le cas où le nombre de processus de l'application est inférieur ou égal au nombre des sites du système, et pour atteindre le maximum de parallélisme; un seul processus est placé sur chaque site de système réparti. Comme nous avons supposé que le réseau de l'architecture complétement maillé et que ses sites identiques, le problème du placement se ramène au placement des objets de communication (ou leurs composants) autour des processus. Autrement dit le problème est d'associer chaque ensemble indexé à un processus de l'application parallèle (et donc au site où sera implanté ce donnée). Cette association indique le placement de l'ensemble indexé sur le site de placement de processus.

Dans le cas où le nombre de processus est supérieur au nombre de sites, un regroupement des processus est nécessaire. Un regroupement parmi tous les regroupements possibles sera choisi en fonction de plusieurs critères d'évaluation.

4.2.1 Association des objets de communication aux processus

Plusieurs processus ont accès à un objet de communication. Parmi ces procesus nous distinguons des producteurs et des consommateurs. L'association d'un objet de communication, ou de ses composants, à un de ces processus dépend de la topologie de l'objet et de son type de communication. Certaines topologies, comme bipoint ou diffusion, n'affectent pas cette association; le seul facteur important pour déterminer l'association est le type de communication. D'autres topologies, comme divergence ou concentration, imposent des règles d'association.

Règles heuristiques liées aux types de communication

Les types de communication sont très variés et ils peuvent être classés selon différents critères. Dans certains types l'ordre de consommation des messages est le même que l'ordre de leurs productions (égalité, égalité presque partout, égalité avec trous); dans d'autres cet ordre n'est pas conservé (aléatoire, rafraichie, pile, etc). Pour certains on consomme tous les messages produits (égalité, égalité prèsque partout, pile,..etc); dans d'autres il y a perte éventuelle de messages. La perte dépend, de la fréquence de la consommation par rapport à la fréquence de la production (aléatoire, échantillon,.. etc). ou bien des valeurs des messages (croissante, non répétitive).
D'aprés leur définition, certains types de communication sont tels que la consommation consomme toujours le message le plus récent, en effaçant les plus anciens, qui sont donc perdus

pour toujours. Pour d'autres types de communication, cette perte de messages est " moins systématique " soit parce qu'on ne considère pas uniquement le message le plus récent, soit parce qu'on n'efface pas tous les autres.
Le tableau 1 montre les différents types de communication que nous avons considéré avec leurs caractéristiques; tableau 2 montre les principales caractéristiques de ces types. La spécification de ces types de communication se trouve dans [Per85].

Type de communication	Consom. du plus ancien mess.	Consom. du plus recent mess.	Perte des messages	Dépendance de parametre	Dépenance des valeurs
Egalité	#				
Aléatoire		#	#		
Trés aléatoire		#	#		
Rafraichie		#	#		
Egalité près que partout	#				
Rafraichie avec trous		#	#		
Egalité avec trous	#		##		
A retard borné		#	#	#	
Echantillon		#	#	#	
Elastique	#		##	#	
Circulaire	#		##	#	
Pile		#			
Croissant	#		##		#
Non répétitive	#		##	#	#

Type de communication	Consom. de ω au début	Consom. de ω si pas de mess.	Consom. d'un mess. n fois
Egalité			
Aléatoire			#
Trés aléatoire	#	#	#
Rafraichie	#		
Egalité presque partout	#	#	
Rafraichie avec trous	#	#	
Egalité avec trous	#	#	
A retard borné			#
Echantillon			#
Elastique			
Circulaire			
Pile			
Croissant			
Non répétitive			

Note: ## indique une perte " moins systématique ".

Tableau (1)

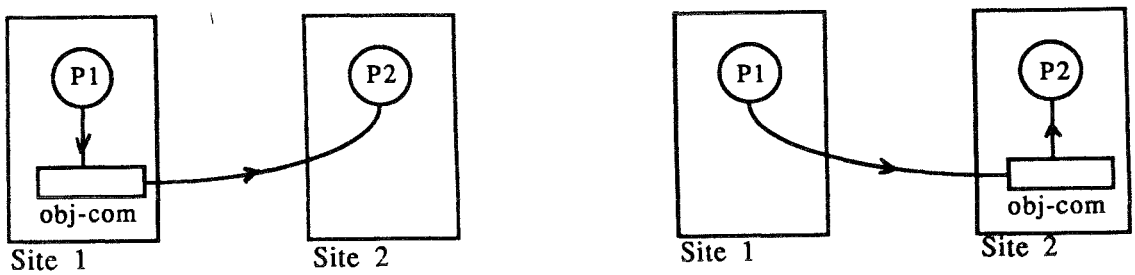
Nous allons détailler quelques types de communication avec les conséquences de chaque choix d'association (l'association de l'objet au producteur ou au consommateur); nous déduirons par la suite les critères et les règles heuristiques d'association.

Type de communication	Conservation de l'ordre de messages	Perte de messages
Egalité	#	
Aléatoire		#
Trés aléatoire		#
Rafraichie		#
Egalité prèsque partout	#	
Rafraichie avec trous		#
Egalité avec trous	#	##
A retard borné		#
Echantillon		#
Elastique	#	##
Circulaire	#	##
Pile		
Croissant	#	##
Non répétitive	#	##

Note: ## indique une perte " moins systématique ".

Tableau (2)

Le placement du producteur et du consommateur d'un objet de communication sur deux sites distincts et l'**association de l'objet au producteur** nécessite l'envoi d'une requête de consommation du consommateur au site de production pour chaque consommation, figure 2-a.



Le placement d'une application parallèle composée de deux processus et d'un objet de communication sur un système réparti

Figure 4.2

Le site de production choisit le message à consommer et il l'envoie au consommateur. Dans cette association, **seuls les messages consommés sont transmis entre le site de production et le site de consommation** et pour **chaque consommation** le consommateur **se bloque** entre l'envoi de la requête de consommation et la réception du message à consommer. **Chaque consommation** nécessite la transmission de **deux messages** à distance, un pour la requête et un pour le message à consommer, figure 2-b.
L'**association de l'objet au consommateur** oblige le producteur à envoyer, à chaque production, le message produit au site de consommation. **Tous les messages produits sont transmis entre le site de production et le site de consommation** même s'ils ne sont pas tous consommés. **Chaque production** nécessite alors la transmission **d'un message à distance.** La consommation est une opération locale par conséquent **la durée de blocage** du consommateur, à la consommation, **est négligable** (sauf s'il n'y a rien à consommer, mais ceci n'est pas un problème de placement).

Type de communication égalité Dans ce type de communication tout message produit est consommé une et une seule fois, et l'ordre des consommations est identique à l'ordre des productions.
L'association de l'objet de communication à son consommateur semble plus adéquate pour ce type de communication car, d'une part tous les messages produits sont consommés par conséquent il n'y a pas de transmission des messages non consommés; et d'autre part cette association, comparée à l'association de l'objet au producteur, diminue le temps de blocage du consommateur et le nombre de messages transmis à distance.

Type de communication aléatoire Ce type exprime une relation telle que chaque message consommé est le dernier produit à cet instant : ainsi, certains messages ne sont pas consommés et d'autres peuvent l'être plusieurs fois.
Comme dans ce type de communication on ne consomme pas tous les messages produits l'association de l'objet de communication à son producteur semble plus intéressant. Ceci permet de diminuer le nombre de messages échangés entre le site de production et le site de consommation. En suivant le même raisonnement, un objet de communication du type : très aléatoire, rafraichie ou aléatoire avec trous est associé au producteur.

Type de communication pile La politique de consommation est celle de la pile; le consommateur consomme le message le plus récent, chaque message est consommé une seule fois et tous les messages sont consommés.
Malgré la consommation du dernier message et la nécessité d'avoir l'image la plus exacte de l'objet de communication par le consommateur à l'instant de la consommation pour consommer le dernier message, il semble plus intéressant d'associer l'objet de communication au consommateur car d'une part tous les messages produits sont consommés, le consommateur ne se bloque pas systématqiuement à chaque consommation, et d'autre part l'association de l'objet de communication au producteur n'assure pas nécessairement la consommation du dernier message, car entre l'envoi du message par le site de production et sa réception par le site de consommation, la production d'un ou de plusieurs messages est possible.

Type de communication élastique La politique décrite par ce type de communication est telle que tout message émis est reçu une fois et une seule, dans l'ordre des émissions, sauf lorsque le processus consommateur prend un retard supérieur à "p" : dans ce cas tous les messages émis sont "sautés" jusqu'au dernier.
La perte des messages dans ce type dépend du retard de consommateur par rapport au producteur et de la valeur de "p". Ce retard est difficile à évaluer statiquement, il est lié aux contextes du producteur et du consommateur, du degré de synchronisation entre eux et de leurs exécutions.
Comme cette perte n'est pas systématique, l'association de l'objet de communication au consommateur est plus appropriée pour les raisons citées précédemment concernant le nombre de message et la durée de blocage. De même l'objet de communication dont le type est circulaire, égalité presque partout ou égalité avec trous, est associé au consommateur pour les mêmes raisons.

Type de communication croissante Ce type de communication est tel que les messages émis sont reçus une seule fois, dans l'ordre de leur production à la condition de former une suite croissante (par rapport à leurs valeurs).
L'objet de communication ayant ce type de communication sera associé au consommateur. Pour que la transmission se réalise seulement pour les messages consommés, un message produit est transmis au site de consommation si sa valeur est supérieur à la valeur du dernier message transmis.
Le même procédé est appliqué pour le type de communication "non répétitive", où le site de production mémorise la valeur du dernier message transmis ainsi que le nombre consécutif de fois de transmission des messages de même valeur. Un message est transmis si sa valeur est différente du dernier transmis ou si il a la même valeur mais le nombre consécutif de fois de sa transmission est inférieur à "p".

Type de communication	Association au producteur	Association au consommateur
Egalité		#
Aléatoire	#	
Trés aléatoire	#	
Rafraichie	#	
Egalité presque partout		#
Rafraichie avec trous	#	
Egalité avec trous		#
A retard borné	#	
Echantillon	#	
Elastique		#
Circulaire		#
Pile		#
Croissant		#
Non répétitive		#

Tableau (3)

Conclusion

Rappelons les arguments pris en compte lors de l'association des objets de communication au processus : miser le temps d'attente des consommateurs et éviter des transferts inutiles, ce qui conduit au choix suivant :
Si les messages d'un objet de communication sont tous consommables ou dans le cas d'une perte " moins systématique " des messages (voir tableau 2), l'objet sera associé au consommateur. Dans le cas contraire il sera associé au producteur.
Le tableau 3 précise les règles d'association des différents types de communication.

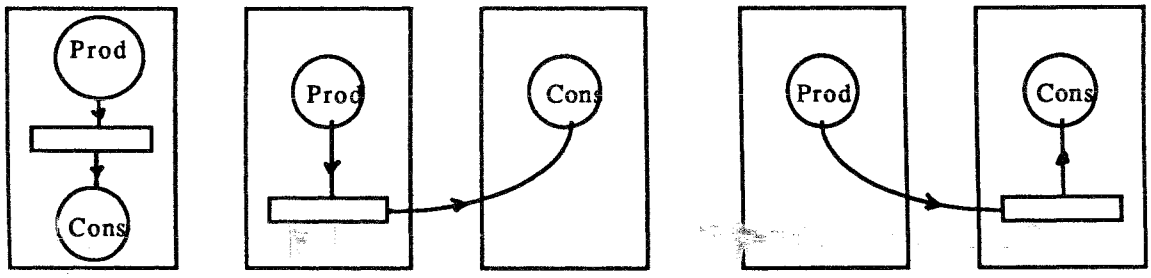
Règles heuristiques liées à la topologie de l'objet de communication

Nous avons décrit les différentes topologies possibles de l'objet de communication au chapitre 1. Précisons dans ce paragraphe les règles heuristiques liées à chaque topologie.

1) Bipoint : La figure 3 montre les différentes possibilités de placement d'une application parallèle composée d'un producteur et d'un consommateur communicant par un objet bipoint : placement des différents composants de l'application parallèle sur le même site 3-a, placement de l'objet sur le même site que son producteur 3-b, placement de l'objet sur le même site que son consommateur 3-c.

Les conséquences de l'association de l'objet de communication à son producteur ou à son consommateur sont liés au type de communication de l'objet. Cette topologie n'impose donc aucune contrainte sur l'association de l'objet à un parmi ces deux processus (producteur, consommateur). Les règles liées aux types de communication sont appliquées.

Application parallèle: toplogie BIPONT

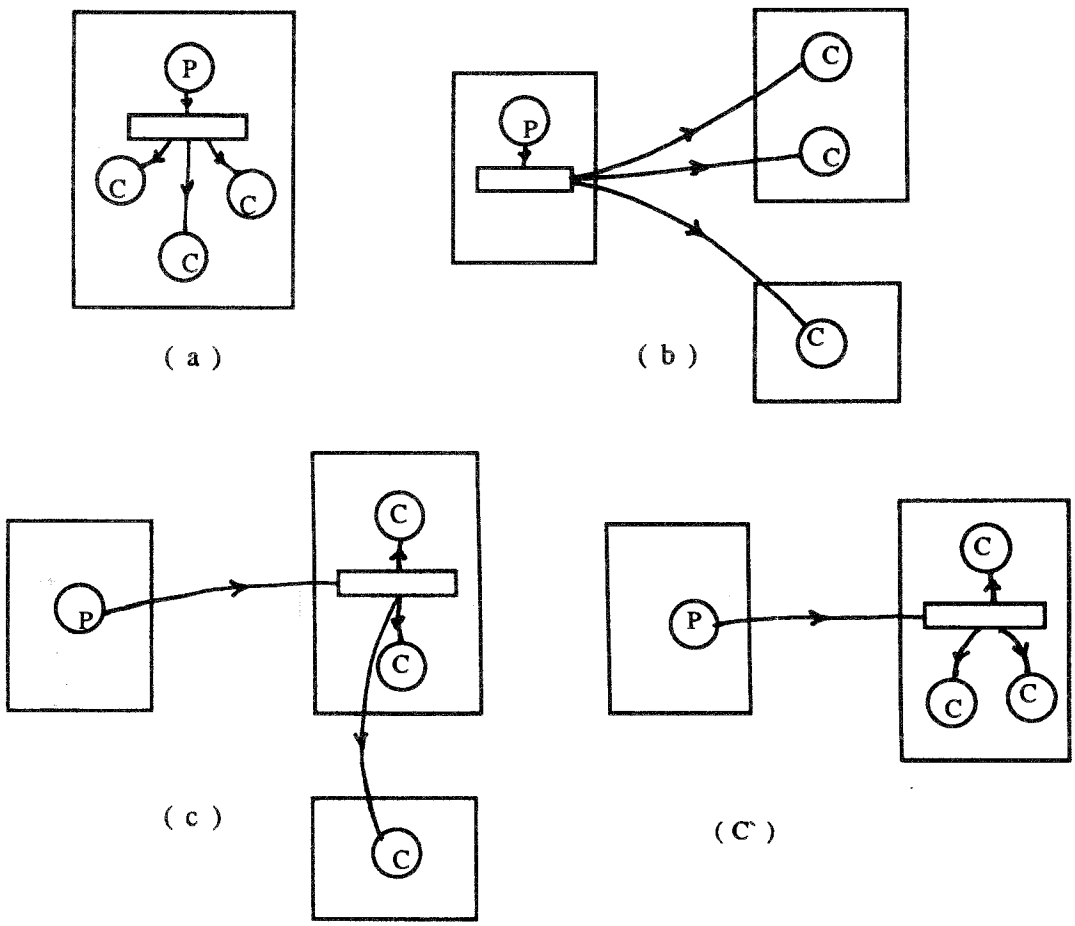


Les différentes possibilités de placement d'une application parallèle dont la topologie est BIPOINT

Figure 4.3

2) Divergence : La figure 4 montre les différentes possibilités de placement d'une application parallèle dont les processus communiquent par un objet de topologie divergence : placement des différents composants de l'application parallèle sur le même site 4-a, placement de l'objet de communication sur le même site que son producteur 4-b, placement de l'objet de communication sur le même site qu'un de ses consommateurs 4-c, placement de l'objet de communication sur le site de placement de ses consommateurs s'ils sont tous placés sur le même site 4-c.

Les avantages de l'association d'un tel objet de communication à un de ses consommateurs par rapport à son association au producteur ne sont significatifs que pour le consommateur en question. Pour les autres consommateurs l'objet de communication est placé à distance. Cela nous conduit à la règle suivante :
Dans le cas général et pour avoir la symétrie des différents consommateurs vis-à-vis l'objet de communication, cet objet est associé au producteur quelque soit le type de communication. Dans un cas particulier où les différents consommateurs sont placés sur le même site, l'objet de communication sera placé sur le site de production ou sur le site de consommation selon la règle liée à son type de communication.

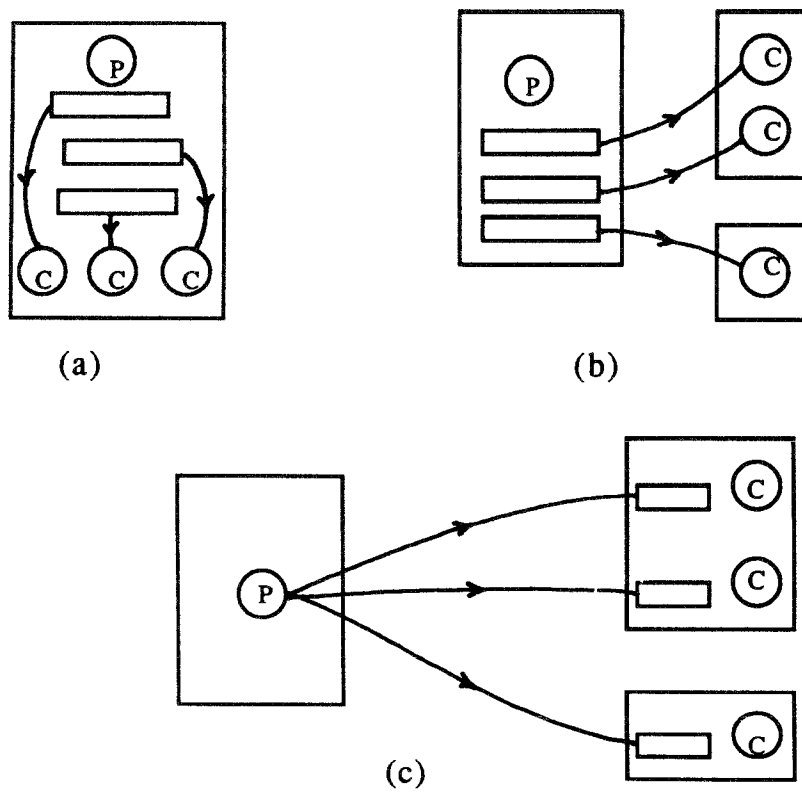


Les différentes possibilités de placement d'une application parallèle dont la toplogie est DIVERGENCE

Figure 4.4

3) Diffusion : Les différentes possibilités de placement d'une application parallèle dont les processus communiquent par un objet de communication de topologie diffusion sont illustrées par la figure 5 : placement des différents composants de l'application parallèle sur le même site 5-a, placement des composants de l'objet de communication sur le même site que le producteur 5-b, placement de chaque composant de l'objet de communication sur le même site que son consommateur 5-c.

Un objet de cette topologie peut se ramener à des objets bipoints ayant le même producteur. En conséquence l'association d'un composant de cet objet à son producteur ou à son consommateur est determinée selon le type de communication de l'objet (cas de bipoint).

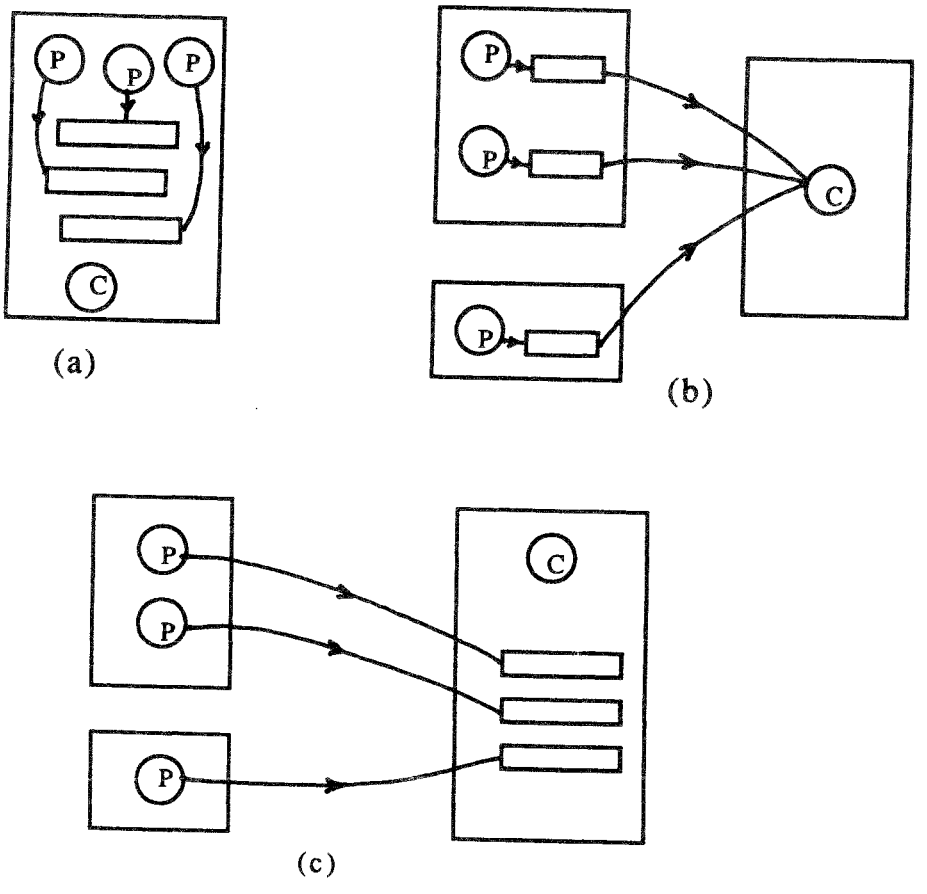


Les différentes possibilités de placement d'une application parallèle de topologie DIFFUSION

Figure 4.5

4) Convergence : Figure 6 montre le placement d'une application parallèle ayant un objet de communication de topologie convergence : placement des différents composants de l'application parallèle sur le même site 6-a, placement de chaque composant de l'objet de communication sur le même site que son producteur 6-b, placement des composants de l'objet de communication sur le même site que le consommateur 6-c.

De même qu'un objet de topologie diffusion, un objet de topologie convergence peut se ramener à des objets bipoints ayant le même consommateur. L'association d'un ensemble indexé est determinée en fonction du type de communication de l'objet (cas de bipoint).

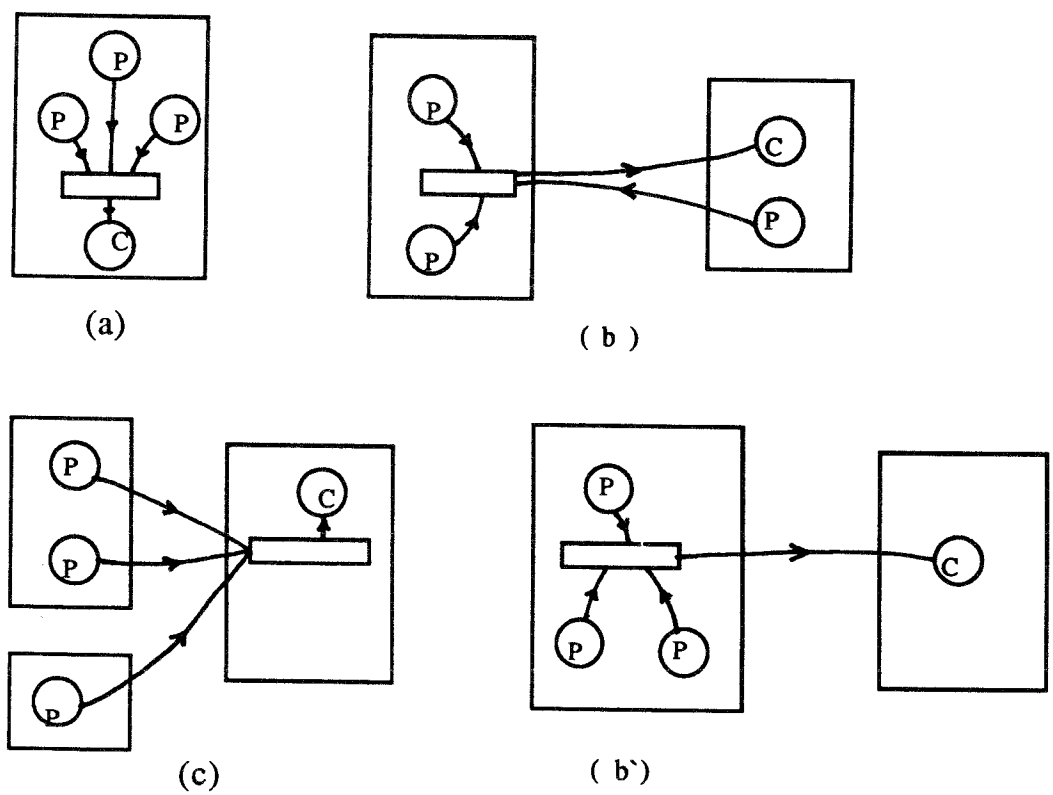


Les différentes possibilités de placement d'une application parallèle de topologie CONVERGENCE

Figure 4.6

5) Concentration : Figure 7 illustre les différentes possibilités de placement d'une application parallèle dont les processus communiquent par un objet de topologie concentration : placement des différents composants de l'application parallèle sur le même site 7-a, placement de l'objet de communication sur le même site qu'un de ses producteurs 7-b, placement de l'objet de communication sur le même site que son consommateur 7-c, placement de l'objet de communication sur le site de placement de ses producteurs s'ils sont tous placés sur le même site 7-b.

Cette topologie est le cas inverse de divergence, donc par analogie, et pour avoir la symétrie des différents producteurs vis-à-vis l'objet de communication, cet objet sera associé au consommateur quelque soit son type de communication. Dans le cas particulier où les différents producteurs sont placés sur le même site, l'objet de communication sera placé sur le site de production ou sur le site de consommation selon son type de communication.



Les différentes possibilités de placement d'une application parallèle dont la topologie est CONCENTRATION

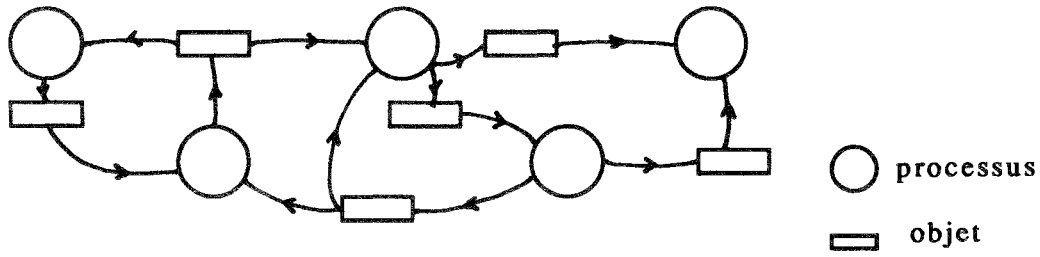
Figure 4.7

Conclusion

Seuls les topologies "divergence" et "concentration" imposent des contraintes sur l'association de l'objet de communication. En résumé :

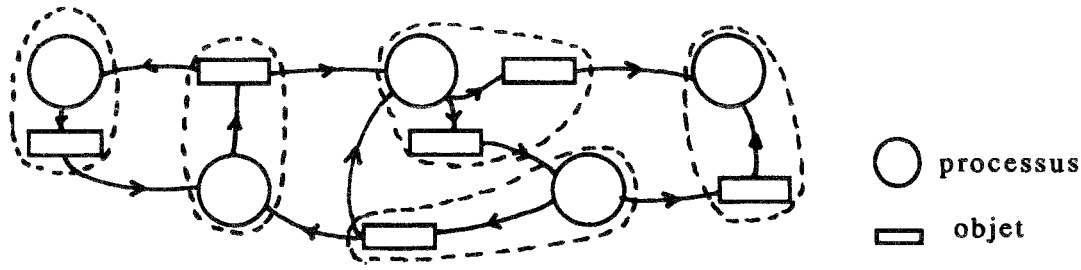
- Cas topologie
  - divergence : l'objet est associé au producteur.
  - concentration : l'objet est associé au consommateur.
  - autre cas : selon le type de communication.

Suite à l'application des règles d'association des objets aux processus, le graphe de l'application parallèle, figure 8, se transforme en un nouveau graphe ayant autant de nœuds qu'il y a de processus. Chaque nœud est constitué d'un processus et de l'ensemble d'ensembles indexés associés à ce processus, figure 9.



Un exemple d'une application parallèle

Figure 4.8



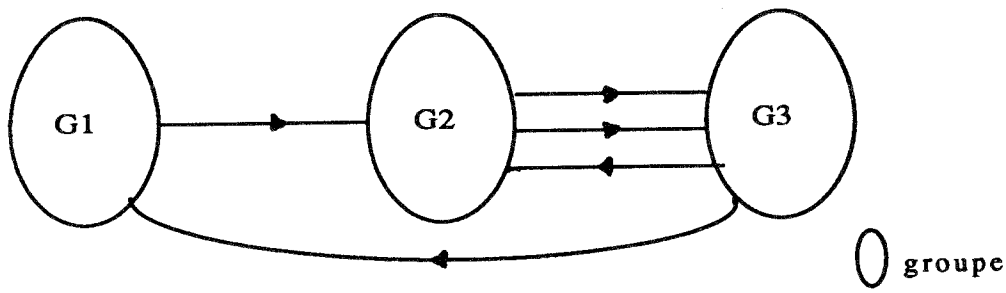
L'association des objets de communication aux processus

Figure 4.9

4.2.2 Placement des processus

Le regroupement de certains processus est nécessaire si le nombre de processus de l'application est supérieur au nombre de sites du système dont nous disposons. (Dans le figure 10 nous supposons que nous disposons de trois sites). Plusieurs solutions sont possibles dans le cas général. Pour choisir une "bonne" solution qui minimise la communication entre les différents sites et maximise le parallélisme réel, certains critères permettent d'évaluer les différentes solutions.

Les critères les plus importants dans notre étude sont les critères liés à la synchronisation et à la dépendance entre les processus de l'application parallèle. Nous commancons ce paragraphe par une présentation de ces critères ensuite nous donnons la définition d'une fonction qui évalue ces critères et nous terminons le paragraphe par d'autres critères d'évaluation.



Regroupement de processus

Figure 4.10

Critères de minimisation de la synchronisation

Les différents processus d'une application paral èle décrite, par le langage LESP ou par sa formalisation (cf : chapitre 1), *ne sont pas indépendants, ils se communiquent par échange de messages.* Dans une telle application parallèle nous pouvons distinguer deux sortes de **synchronisation** entre ses processus : une liée au type de communication de l'objet, et l'autre liée à la topologie de l'objet.

La synchronisation des processus a plusieurs conséquences sur le comportement et l'exécution de l'application parallèle. Une de ces conséquences est le blocage de certains processus en attendant leurs interlocateurs. Ce blocage diminue le degré de parallélisme et augmente le temps d'exécution de l'application paralléle.

Sachant que la communication par messages est le seul moyen de synchronisation dans l'environnement réparti et que le delai de transmission de messages, dans un tel environnement, est non négligable nous constatons qu'il sera souhaitable de **minimiser la synchronisation entre les processus d'une application parallèle lors de son implantation sur un système réparti.** Cet objectif se réalise en choisissant une **implantation qui minimise la synchronisation à distance** entre les processus de l'application parallèle. Comme l'objectif de ce paragraphe est de regrouper certains processus de l'application parallèle afin d'obtenir "n" groupes de processus, où n est le nombre de sites de l'architecture répartie dont nous disposons, nous

allons introduire une fonction qui évalue la synchronisation entre les processus d'une application parallèle. La minimisation de la valeur de cette fonction est l'un des critères de regroupement.

Dans ce paragraphe nous allons présenter les deux sortes de synchronisation entre les processus d'une application parallèle.

- **Synchronisation liée au type de communication**
  Mis à part le fait qu'un type de communication exprime la politique de communication des processus qui partagent un objet, il reflète aussi le degré de synchronisation entre ces processus. Le degré de synchronisation entre un producteur et un consommateur d'un objet de type égalité est élevé; car ce type ne permet pas la perte des messages et les messges sont consommés dans l'ordre de leurs production. Malgré l'asynchronisme entre les processus des applications considérées, ce type impose un synchronisme entre le producteur et le consommateur (blocage du consommateur si l'objet est vide, et chaque message est consommé une seule fois).

  A l'autre extrême, le degré de synchronisation entre un producteur et un consommateur d'un objet de type très aléatoire est trés peu élevé car le consommateur peut consommer même si le producteur n'a pas produit (consommation de ω), et après une production le consommateur n'est jamais bloqué (consommation du même message plusieurs fois), et d'autre part aucune contrainte n'est imposée sur les valeurs des messages ni sur le retard relatif du consommateur par rapport au producteur.
  Le degré de synchronisation entre deux processus communicant par un objet de communication est lié au type de communication de l'objet. Un automate a été associé à chaque type de communication [Per85] pour pouvoir comparer les types de communication du point de vue de l'asynchronisme. Deux relations d'ordre partiel permettant cette comparaison ont été introduites dans [Per85]. La formalisation de ces relations se trouve dans [Laz90].

Dans ce paragraphe nous allons citer les idées de base permettant la compréhension de ces relations et le diagramme qui met en évidence les relations entre les différents types de communication.

Automate associé à un type de communication
Pour modéliser un système parallèle réduit à ses opérations de production et de consommation sur un objet donné, nous associons à chaque type de communication un automate figuré par un graphe orienté représentant les transitions possibles. Les noeuds de ce graphe représentent les états du système. Les arcs sont étiquetés par l'opération sur la donnée communiquée :

- p : correspond à une production.
- c : correspond à une consommation.

Ce graphe doit respecter les conditions suivantes :

- Une opération de consommation c est activable si et seulement si la pré-condition de consommation pré-cons est vérifiée.
- A chaque étiquette p ou c d'un arc est associé un indice tel que :

  * Pour une opération de production p cet indice est égal à l'indice dans l'ensemble produit de l'élément résultant de l'occurrence de production associée.
  * Pour une opération de consommation c, cet indice s'il existe est égal à :
    · l'indice dans l'ensemble consommable de l'élément résultant de l'occurrence de consommation associée,
    ou
    · ω, si ω est la valeur indéfinie résultant de cette opération.
- Les opérations sont étiquetées dans l'ordre croissant des indices.

Les automates associés aux différents types de communication sont donnés dans [Per85]. Comme exemple, la figure 11 illustre les automates correspondants aux types : égalité, aléatoire, et égalité-presque-partout.

Ordre de communication
La définition de ces graphes permet d'introduire deux relations d'ordre partiel sur l'ensemble des types de communication. Ces relations sont telles que si pour une donnée communiquée nous substitutions un type de communication TC_2 à un type de communication TC_1 tel que TC_1 soit en relation avec TC_2, le système devient plus asynchrone dans le sens que nous précisons ci-dessous.

Relation d'asynchronisme avec perte des valeurs :
Soient TC_1 et TC_2 deux types de communication. TC_2 est dit plus asynchrone que TC_1 avec possibilité de perte de valeurs, ce qui est noté :

$$TC_1 \text{ " } as < \text{ " } TC_2$$

si :

- tout chemin dans le graphe associé à TC_1 est un chemin du graphe associé à TC_2,
- sur les arcs correspondants, les indices du graphe de TC_1 sont inférieurs ou égaux aux indices du graphe de TC_2.

Cette relation "$as <$" est une relation d'ordre partiel sur l'ensemble des types de communication.
Exemples :

- égalité "$as <$" aléatoire,
- rafraichie "$as <$" échantillon "$as <$" aléatoire.

Relation d'asynchronisme avec respect des valeurs :
Soient TC_1 et TC_2 deux types de communication. TC_2 est dit plus asynchrone que TC_1 en respectant les valeurs, ce qui est noté :

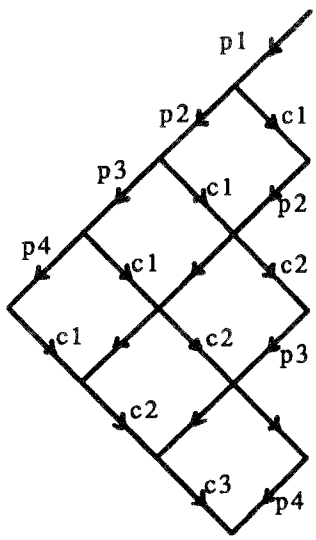
$$TC_1 \text{ "}op < \text{ " } TC_2$$

si :

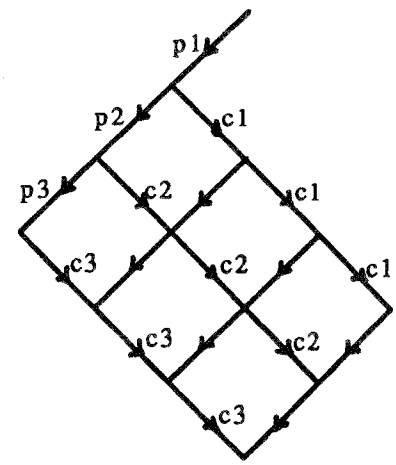
- tout chemin dans le graphe associé à TC_1 est un chemin du graphe associé à TC_2.
- sur les arcs correspondants, les indices du graphe de TC_1 sont égaux aux indices du graphe de TC_2.

Cette relation "$op <$" est une relation d'ordre partiel sur l'ensemble des types de communication, telle que :

$$TC_1 \text{ "}op < \text{" } TC_2 \Rightarrow TC_1 \text{ "}as < \text{" } TC_2$$

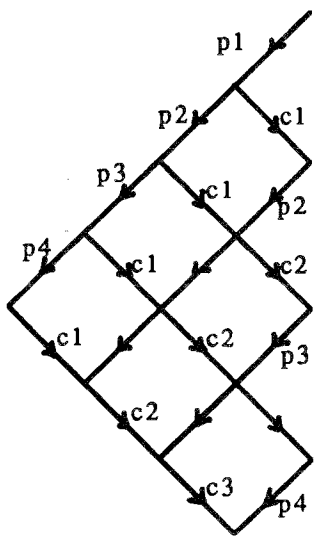


Automate associé au type de communication égalité

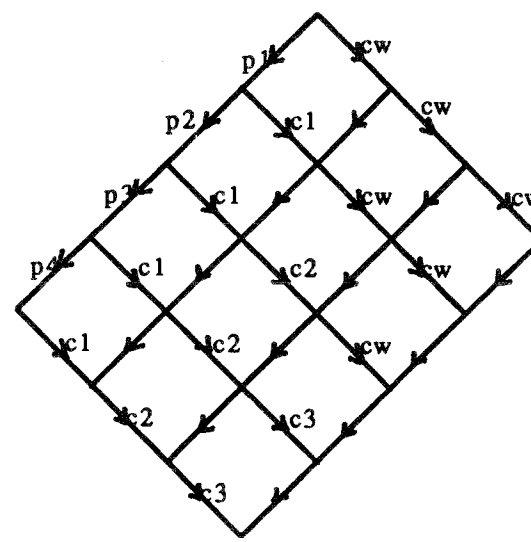


Automate associé au type de communication aléatoire

égalité **as<** **aléatoire**



Automate associé au type de communication égalité



Automate associé au type de communication égalité presque partout

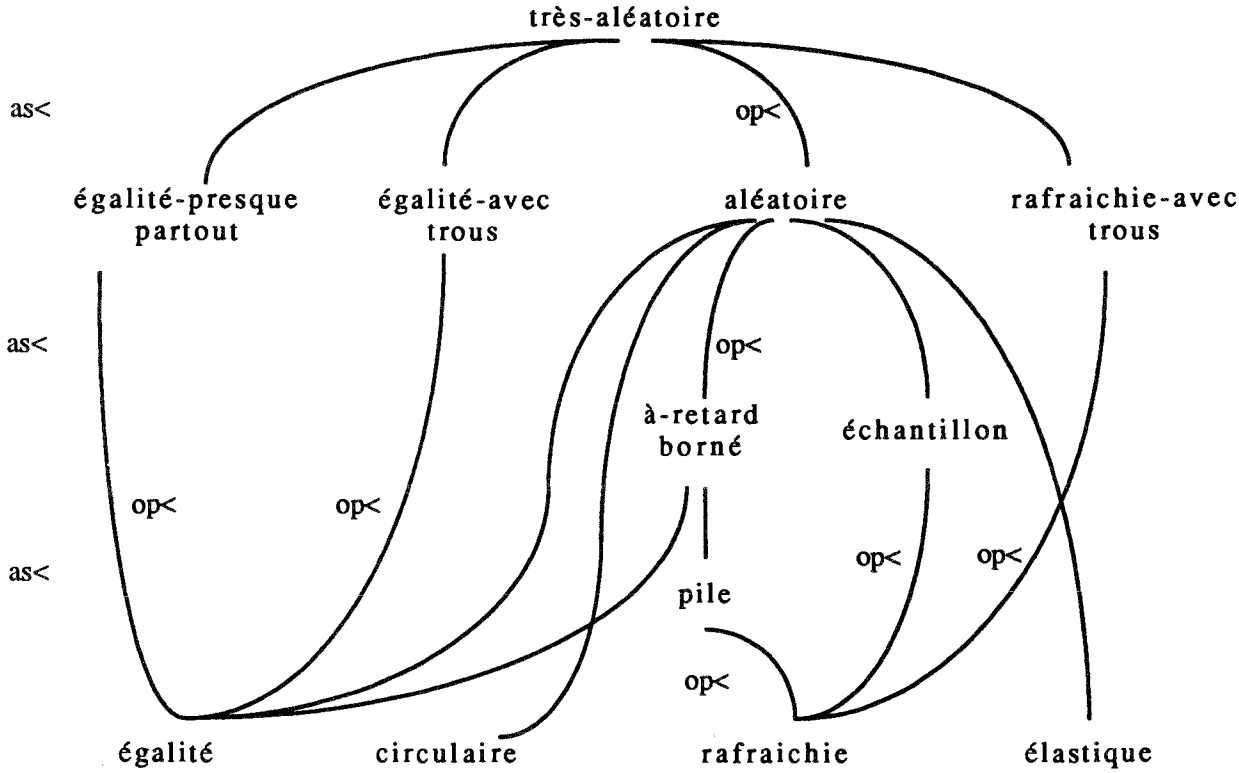
égalité **op<** **égalité-presque partout**

Figure 4.11

Exemples :

- égalité "*op* <" égalité-presque-partout,
- rafraichie "*op* <" rafraichie-avec-trous.

Pour les types de communication décrits dans le paragraphe précedant, nous obtenons le diagramme de figure 12.



Le diagramme décrivant la relation d'asynchronisme entre les types de communication

Figure 4.12

Les relations d'ordre "*as* <" et "*op* <" nous permettent de donner un critère de regroupement de processus.

Les processus communicants par un objet dont le type de communication est TC_1 sont plus susceptible au regroupement que les processus communicants par un objet dont le type de communication est TC_2 si : TC_1 " as < " TC_2

En effet TC_2 est plus asynchrone que TC_1, et par conséquent le degré de synchronisation entre les processus de TC_1 est plus élevé qu'entre les processus de TC_2.

- **Dépendance liée à la topologie de l'objet de communication**
  Nous pouvons distinguer plusieurs relations de dépendance entre les processus d'un objet de communication (les producteurs et les consommateurs de l'objet de communication). Ces relations sont liées à la topologie de l'objet et au rôle que joue chaque processus (producteur ou consommateur). Ces relations sont :

  - **prod-cons** : Deux processus sont liés par une relation (prod-cons), si l'un est producteur et l'autre est un consommateur du même objet de communication. Quelque soit la toplogie de l'objet il y a au moins un couple de processus liés par cette relation.
  - **prod-conc** : Deux processus producteurs d'un objet dont la topologie est la concentration sont liés par une relation "prod-conc".
  - **cons-div** : Deux processus consommateurs d'un objet dont la topologie est la divergence sont liés par une relation "cons-div".
  - **prod-conv** : Deux processus producteurs d'un objet dont la topologie est la convergence sont liés par une relation "prod-conv".
  - **cons-diff** : Deux processus consommateurs d'un objet dont la topologie est la diffusion sont liés par une relation "cons-diff".

Dans un environement centralisé les relations : prod-conv et cons-diff n'ont pas d'effet sur l'implantation car les deux processus, producteur de prod-conv et consommateurs de cons-diff, n'aggissent pas sur le même ensemble indexé de l'objet. L'effet des relations : cons-div et prod-conc sur l'implantation dans l'environnement centralisé se traduit par la synchronisation des deux processus liés par une telle relation pour l'accès en exclusion mutuelle à l'objet de communication.

Dans un environement réparti et précisement dans le cas où le nombre de processus est supérieur au nombre de sites, ces relations de dépendance interviennent lors de regroupement de processus. Chaque relation de dépendance favorise le regroupement de ces processus pour atteindre un objectif déterminé. Précisons l'objectif de regroupement des processus liés par une des relations précédentes :

- La relation **prod-cons** :
  L'objectif de regroupement des processus liés par cette relation est de **diminuer le nombre de liaisons** entre les sites de système réparti et de **minimiser le nombre de messages échangés à distance** (production locale, consommation à distance)
- La relation **prod-conc** :
  Les messages produits sont rangés dans l'objet de communication selon leurs dates de production. Le placement des producteurs sur des sites différents implique l'absence d'une référence temporelle commune et par conséquent cette implantation nécessite l'exécution d'un algorithme simulant l'horloge globale.
  Le regroupement des producteurs **simplifie l'exécution de l'algorithme de simulation de l'horloge globale.**

Un autre objectif de regroupement, atteignable si l'objet est associé au consommateur, est de **diminuer le nombre de liaisons** du système réparti. Les producteurs regroupés partagent le même port d'envoi et ils utilisent la même liaison pour la transmission des messages produits, figure 7-*ć*.

- La relation **cons-div** :
  L'objectif de regroupement des processus liés par cette relation est multiple :
  * **Diminuer le nombre de liaisons** de système réparti figure 4-*b́* : deux consommateurs placés sur le même site partagent le même port de réception, et par conséquent une seule liaison est utilisée pour la transmission de leurs messages.
  * **Avoir la même réference temporelle** pour les requêtes de consommation et par conséquent simplifier l'utilisation de l'algorithme de simulation de l'horloge globale (cf : prod-conc).
- La relation **prod-conv** :
  L'objectif de regroupement des processus liés par cette relation est de **diminuer le nombre de liaisons** du système réparti. Cet objectif est atteignable si les ensembles indexés sont associés aux producteurs car le consommateur demande la consommation d'un seul ensemble indexé, par conséquent la même liaison peut être associée aux différents ensembles indexés placés sur le même site, figure 6-*b́*.
- La relation **cons-diff** :
  De même que le cas précédent l'objectif de regroupement des processus liés par cette relation est de **diminuer le nombre de liaisons** de système réparti. Cet objectif est atteignable si les ensembles indexés sont associés aux consommateurs. La valeur produite sera diffusée localement sur le site de réception, figure 5-*ć*. Ce regroupement **diminue** aussi **le nombre de messages échangés à distance.**

Nous remarquons que les objectifs de regroupement sont les combinaisons des critères de base suivants :

- Diminuer le nombre de liaisons entre les sites de système réparti.
- Diminuer le nombre de messages échangés à distance.
- Avoir la même référence temporelle.

Tableau 4, montre les liens entre les critères de base et les objectifs de regroupement des processus liés par les relations précédentes.

Définition de la fonction de synchronisation

Cette fonction évalue la synchronisation entre les différents processus de l'application. La valeur maximale de cette fonction peut être calculée en considérant tous les liens existant entre les processus de l'application.
La valeur de cette fonction est calculée pour chaque regroupement en ne considérant que les liens entre les processus qui n'appartiennent pas au même groupe. Le meilleur regroupement vis-à-vis le critère de minimisation de la synchronisation est celui qui minimise la valeur de cette fonction.

La formulation de la fonction de synchronisation est difficile à définir et nous lui préférons une forme linéaire qui tient compte de facteurs liés aux types de communication et de facteurs exprimant la relation de dépendance entre les processus. Cette formulation est définie par :

$$\mathcal{F} = \sum_{k \in K} C_k \sum_{i \in I} L_{ik} \sum_{j \in I \wedge j > i} L_{jk} * f_{ijk}$$

- C_k : facteur de synchronisation lié au type de communication de l'objet k.
- f_{ijk} : facteur de dépendance exprimant la relation entre les processus i et j vis-à-vis l'objet k.
- L_{ik}= 1 si le processus i est un producteur ou consommateur de l'objet k.
- I : l'ensemble des processus.
- K : l'ensemble des objets de communication.

Pour pouvoir évaluer cette fonction, il faut donner des valeurs numériques aux différents facteurs.

Les valeurs de C_k :
Etant donné que les types de communication sont ordonnés **partiellement** par une des relations d'ordre "$as <$" ou"$op <$", nous ne pouvons pas donner des valeurs numériques, aux facteurs C_k, justifiable pour toutes les applications parallèles.
Un sous ensemble de l'ensemble des types de communication intervient dans chaque application parallèle. Pour ce sous-ensemble, nommé TC-APP (types de communication d'une application parallèle), nous pouvons distinguer trois cas possibles :

1. Les types de communication de TC-APP sont ordonnés totalement : dans ce cas nous pouvons donner des valeurs numériques aux facteurs de types de communication de TC-APP. Naturellement ces valeurs doivent respecter l'ordre total entre les types.
2. Les types de communication de TC-APP ne sont pas ordonnés ni totalement ni partiellemen : dans ce cas nous ne pouvons pas donner des valeurs aux facteurs C_k. Ces facteurs ne doivent pas affecter la valeur de la fonction de synchronisation. Pour éliminer leurs effet nous leurs donnons la même valeur.

3. Les types de communication de TC-APP sont ordonnés partiellement : dans ce cas nous éliminons l'effet des types de communication non comparables en donnant leur facteur la même valeur numérique et nous associons les facteurs des types comparables a des valeurs qui respecte l'ordre partiel.

(Voir exemples en 4-4).
Précisons quelques facteurs qui favorisent la prise en compte d'un critère de regroupement par rapport aux autres.

- La complexité de la procédure de céation d'une liaison entre deux sites distant du système réparti favorise la prise en compte du critère de diminution de nombre de liaison.
- Le nombre limité de connexions physiques entre les sites de l'architecture répartie favorise aussi la prise en compte du critère de diminution de nombre de liaison.
- La minimisation de charge du réseau de communication de l'architecture répartie favorise la la prise en compte du critère de diminution de nombre de message.

Les valeurs f_{ijk} :
Les facteurs f_{ijk} sont des facteurs de dépendance exprimant les relations entre les processus. Comme nous l'avons vu au paragraphe précédent, plusieurs critères interviennent pour l'évaluation de la dépendance des processus liés par ces relations.
Il est difficile de donner les priorités des critères car cela dépend de plusieurs facteurs liés à l'application parallèle, à la machine physique utilisée, et au choix de l'utilisateur.
Précisons quelques facteurs qui favorisent la prise en compte d'un critère de regroupement par rapport aux autres.

- La complexité de la procédure de céation d'une liaison entre deux sites distant du système réparti favorise la prise en compte du critère de diminution de nombre de liaison.
- Le nombre limité de connexions physiques entre les sites de l'architecture répartie favorise aussi la prise en compte du critère de diminution de nombre de liaison.
- La minimisation de charge du réseau de communication de l'architecture répartie favorise la la prise en compte du critère de diminution de nombre de message.

Nous allons montrer sur un exemple (4) le choix de valeurs de ces facteurs, et leurs effet sur le résultat du regroupement.

Autres critères de placement

- **Une répartition équitable des processus sur les sites**
  Pour avoir une répartition de la charge entre tous les sites les différents processus sont répartis sur l'ensemble des sites. Chaque site est le site de placement d'au moins K processus et au maximum (K+1) processus (où K = N/M; K, N, M sont des entiers ; N = nombre de processus; M = nombre de sites).
  Comme chaque processus s'exécute sur un seul site, il y aura P sites de (K+1) processus et

(M-P) sites de K processus, avec (P=N-K*M). La fonction qui évalue l'équité de la charge est G :

$$\mathcal{G} = \sum_{i \in S \wedge j \in S \wedge j > i} (NPS(i) - NPS(j))$$

où S représente l'ensemble des sites, NPS(i) est le nombre de processus de site numéro i.

- **Minimisation du nombre de liaisons entre les sites**
  Une liaison entre deux sites indique l'appartenance d'un processus à un site, et d'un ensemble indexé à l'autre site et que le processus est un producteur ou un consommateur de cet ensemble indexé.
  La fonction qui évalue le nombre de liaisons est définie par :

$$\mathcal{D} = \sum_{i \in I \wedge q \in Q} x_{iq}$$

où Q représente l'ensemble d'ensembles indexés et $x_{iq} = 1$ si le processus i est un producteur ou un consommateur de l'ensemble indexé q et que le processus i et l'ensemble indexé q n'appartiennent pas au même groupe.

- **Une répartition équitable des ensembles indexés sur les sites**
  Pour avoir l'équité de charge des différents sites, il faut minimiser la fonction L, définie de la manière suivante :

$$\mathcal{L} = \sum_{i \in S \wedge j \in S \wedge j > i} (NES(i) - NES(j))$$

où NES(i) est le nombre d'ensembles indexés sur le site i.

Les différentes fonctions $\mathcal{F}, \mathcal{G}, \mathcal{D}, \mathcal{L}$ permettent l'évaluation d'un regroupement. Un algorithme de regroupement, minimisant ces quatre fonctions, est développée au 3.2. Cet algorithme cherche un bon regoupement, et dans certain cas le meilleur, par rapport aux critères considérés.

Le résultat de regroupement est un ensemble de groupe de processus. Chaque groupe est constitué d'un ensemble de processus et de l'ensemble d'ensembles indexés associés à ces processus.

Le graphe de l'application parallèle se transforme en un nouveau graphe, figure 14, dont les nœuds représentent les groupes.

4.3 L'algorithme de placement proposé

C'est l'algorithme d'utilisation des règles heuristiques afin d'obtenir un " bon " placement des constituants de l'application parallèle sur les sites d'un système réparti.

Les entrées de cet algorithme sont :

1. Les processus de l'application parallèle avec leurs entrées/sorties.

2. Les objets de communication en précisant leurs topologies et leurs types de communication.
3. Le nombre de sites du système dont nous disposons.

Cet algorithme rend les groupes des processus avec leurs ensembles indexés, et il est composé de deux procédures :

- La procédure d'association des ensembles indexés aux processus.
- La procédure du regroupement des processus.

La deuxième procédure est appliquée seulement si le nombre de processus est supérieur au nombre de sites.

Dans la formalisation d'une application parallèle le medium de communication est constitué d'un ensemble d'ensembles indexés. Certains de ces ensembles sont des composants d'un seul objet de communication.

Pour pouvoir appliquer les règles heuristiques liées à la topologie de l'objet de communication, nous allons introduire un nouveau type nommé " objet-composé ". Ce type regroupe les ensembles indexés appartenant au même objet de communication et il précise la topologie de l'objet.

La formalisation d'une application parallèle peut donc être exprimée par son ensemble de processus et son ensemble d'objets de communication.

Type **n-topologie**

Expression : chaine de caractère; { bipoint, diffusion, ...}

End
Type **objet-composé**

Expression :

< COMPOS : ensemble (accès-ensemble-indexé), TOPO : n-topologie >

Invariant :

1. $\forall$ OBJ : objet-composé, $\forall$ E1, E2 : accès-ensemble-indexé /
   E1 $\in$ COMPOS(OBJ) $\wedge$ E2 $\in$ COMPOS(OBJ) $\Rightarrow$
   TYP-COM(E1) = TYP-COM(E2) $\wedge$
   ($\exists$ A1(TYP), A2(TYP) : cons-ensemble-indexé /
   IND(E1)= A1 $\wedge$ IND(E2)= A2)
   { Les ensembles indexés d'un objet de communication ont le même type de communication et le même type de donnée }
2. $\forall$ OBJ : objet-composé,
   (TOPO(OBJ) = bipoint $\vee$ TOPO(OBJ) = divergence
   $\vee$ TOPO(OBJ) = concentration $\Rightarrow$ taille(COMPOS(OBJ)) = 1)
   (TOPO(OBJ) = diffusion $\vee$ TOPO(OBJ) = convergence $\Rightarrow$

taille(COMPOS(OBJ)) > 1)
{ Un objet de communication de topologie : bipoint, concentration ou divergence est composé d'un seul ensemble indexé }

End

4.3.1 La procédure d'association des ensembles indexés aux processus

L'objectif de cette procédure est d'associer chaque ensemble indexé au processus le plus approprié. L'application des règles d'associations déjà citées en 2.1 permet de déterminer l'association de chaque ensemble indexé à un et un seul processus; ceci quel que soit la topologie et le type de communication de l'objet de communication.

Procédure d'association

- Entrées
  - Ensemble de processus avec leurs entrées, sorties.
  - Objets de communication.
- Sorties
  - Table d'association des ensembles indexés aux processus.
- Corps
  - Pour chaque objet de communication faire
    - Case topologie
      - Bipoint : associer l'objet (ensemble indexé) à son producteur ou consommateur en fonction de son type de communication (voir tableau 3).
      - Divergence : associer l'objet (ensemble indexé) à son producteur.
      - Diffusion, convergence : associer chaque ensemble indexé de l'objet à son producteur ou consommateur en fonction du type de communication de l'objet (voir tableau 3).
      - Concentration : associer l'objet (ensemble indexé) à son consommateur.
  - Fin pour

Fin

4.3.2 La procédure du regroupement

L'objectif de cette procédure est de déduire à partir du graphe obtenu lors de l'application de la procédure d'association, un nouveau graphe composé de M nœuds où M est le nombre de sites du système réparti.

La première étape de cette procédure consiste à **calculer le nombre moyen de processus par site** pour satisfaire l'équité de charge entre les sites. Ensuite elle **déduit les**

différents liens existants entre les processus de l'application parallèle à partir des entrées, des sorties des processus et des objets de communication. Une **détection optionnelle de la symétrie** entre les processus de l'application permet de diminuer le nombre de permutations dans l'algorithme. Cette étape est la troisième étape de la procédure. Avant d'entrer dans le **cycle d'évaluation et de permutation** la procédure choisit un **regroupement initial** qui minimise la fonction de synchronisation prenant en compte le nombre moyen de processus par site.

La déduction des liens consiste à déterminer si l'ordre entre les types de communication des objets de l'application parallèle considèrée est un ordre partiel ou total et à accumuler les différents liens de communication entre ses processus. Selon le résultat de cette déduction, l'utilisateur doit donner des valeurs numériques permettant le calcul de la valeur de la fonction de synchronisation $\mathcal{F}$.

La détection de la symétrie entre les processus d'une application parallèle :
Deux processus sont symétriques par rapport à un autre processus s'ils ont les mêmes liens de communication avec ce troisième. Même lien veut dire, dans ce contexte, la même relation de communication (prod-cons, cons-diff, ...) et le même type de communication.
Deux processus sont symétriques par rapport à un ensemble de processus s'ils sont symétriques par rapport à chaque processus de l'ensemble.
Exemples :

1. Les processus P1 et P2 de (chapitre1, figure 9) sont symétriques par rapport au processus Q car :

   - P1 et Q sont liés par une relation de communication (prod-cons) vis-à-vis l'objet F1. Le type de communication de cet objet est égalité.
   - De même P2 et Q sont liés par une relation de communication (prod-cons) vis-à-vis l'objet F2 et le type de communication de cet objet est égalité.
   - P1 et Q sont liés par une relation de communication (prod-cons) vis-à-vis l'objet N1. Le type de communication de cet objet est aléatoire.
   - De même P2 et Q sont liés par une relation de communication (prod-cons) vis-à-vis l'objet N2 et le type de communication de cet objet est aléatoire.

   En résumé :

   - (P1, Q) : **prod-cons** avec type **égalité**, et **prod-cons** avec le type **alétoire**.
   - (P2, Q) : **prod-cons** avec type **égalité**, et **prod-cons** avec le type **alétoire**.

2. Les processus $P_1, P_2, ..., P_n$ de (chapitre 1, exemple 1) sont symétriques par rapport aux processus (P, Q).
   Chacun parmi les processus P_i a une relation (prod-cons) avec le processus P, le type de communication de cette relation est égalité.
   Et chacun de ces processus a deux relations (prod-cons) avec le processus Q, le type de communication de ces relations est égalité.
   D'autre part les processus $P_1, .., P_{i-1}, P_{i+1}, ..., P_n$ sont symétriques par rapport au processus P_i. Chacun d'eux a une relation (cons-diff), vis-à-vis l'objet SP de type égalité, avec P_i.

La complexité de détection de la symétrie dans une application parallèle dépend de : nombre de processus, nombre d'objets de communication et de la complexité de la topologie de l'application.
L'objectif de cette détection est de diminuer le nombre de permutations lors de l'exécution de la procédure de regroupement. Cette détection peut être plus coûteuse que la permutation si l'application parallèle est assez complexe. Pour cette raison la détection de la symétrie est éventuelle et elle applicable sur des applications parallèles simples de point de vue de topologie.
Si la détection de la symétrie a été effectuée, la procédure de regroupement ne tient pas compte de permutations des processus symétriques par rapport à un ensemble de processus (A) et qui n'ont de liens que avec cet ensemble de processus (A). L'ensemble (A) peut se reduire à un seul processus. Cette règle est appliquée dans l'exemple (4).
Malgré la simplicité de cette règle, elle est applicable sur toutes les exemples d'applications parallèles citées dans [Per85], et elle diminue le nombre de leurs permutations (Annexe D).

Procédure de regroupement

- Entrées
  - Ensemble de processus avec leurs entrées, sorties.
  - Objets de communication avec leurs types de communications.
  - Le nombre des sites de l'architecture répartie.
- Sorties
  - Les groupes des processus
- Corps
  - A partir du nombre de processus et du nombre des sites, déduire le nombre moyen de processus par site.
  - Déduire à partir des entrées les différents liens existants entre les processus :
    - déduire l'ordre entre les types de communication,
    - déduire les relations existant entre les processus,
    - donner des valeurs numériques aux facteurs C_k, f_{ijk}.
  - Détection éventuelle de la symétrie entre les processus.
  - Regroupement initial qui minimise la valeur de la fonction de synchronisation.
  - Tant que il y a des permutations possibles et qu'on n'est pas satisfait faire
    - Evaluation des différentes fonctions : $\mathcal{F}, \mathcal{G}, \mathcal{D}, \mathcal{L}$.
    - Permutation.
  - Fin tant que
  - Regroupement final.

Fin

Le résultat de la fonction heuristique de regroupement est un ensemble de groupe, chaque groupe est constitué d'un ensemble de processus et un ensemble d'ensembles indexés.

Regroupement :
apllication parallèle * nombre de sites du système réparti → groupes
groupes : ensemble (groupe) / taille(groupes) ≤ nombre de sites de système réparti
groupe = < PRCS : ensemble (accès-processus), INS : ensemble (accès-ensemble indexé) >
Invariant : ∀ G : group ⇒ PRCS(G) ≠ { }

4.4 Application de l'algorithme de placement sur un exemple

Nous allons appliquer la fonction de regroupement sur l'application parallèle APP2 donnée par sa formalisation dans le chapitre 1, figure 7.
Cette application est composée de trois processus : { P1, P2, Q }, et de quatre objets de communication : { N1, N2, F1, F2 }. La formalisation de ses objets est :
objet-composé :

N1 = (COMPOS : { A1N1,A2N1 }, TOPO : diffusion)

N2 = (COMPOS : { A1N2,A2N2 }, TOPO : diffusion)

F1 = (COMPOS : { AF1 }, TOPO : bipoint)

F2 = (COMPOS : { AF2 }, TOPO : bipoint)

L'application du procédure d'association :
TOPO(N1) = diffusion ∧ TYPE(N1) = aléatoire ⇒
A1N1 et A2N1 sont associés au processus P1.
TOPO(N2) = diffusion ∧ TYPE(N2) = aléatoire ⇒
A1N2 et A2N2 sont associés au processus P2.
TOPO(F1) = bipoint ∧ TYPE(N1) = égalité ⇒
AF1 est associé au processus Q.
TOPO(F2) = bipoint ∧ TYPE(F2) = égalité ⇒
AF1 est associé au processus Q.
La table d'association des ensembles indexés aux processus est :
[a1n1 → np1, a2n1 → np1, a1n2 → np2, a2n2 → np2, af1 → nq, af2 → nq]

L'application de procédure du regroupement :
Si le nombre de sites du système réparti dont nous disposons est supérieur à trois sites, le résultat de cette procédure est, figure 13 :
Groupes = { G1, G2, G3 }
G1 = (PRCS : { P1 }, INS : { A1N1, A2N1 })
G2 = (PRCS : { P2 }, INS : { A1N2, A2N2 })
G3 = (PRCS : { Q }, INS : { AF1, AF2 })

Dans le cas contraire nous aurons autant de groupes qu'il y a de sites.
Si le nombre de sites est **deux** alors :

Le nombre de processus par groupe :
Un groupe d'un processus et un groupe de deux processus.

Les liens existant entre les processus :

- (P1, P2) :
  1. Une relation (**prod-cons**) vis-à-vis N1 dont le type est **aléatoire**.
  2. Une relation (**prod-cons**) vis-à-vis N2 dont le type est **aléatoire**.
- (P1, Q) :
  1. Une relation (**prod-cons**) vis-à-vis F1 dont le type est **égalité**.
  2. Une relation (**cons-diff**) vis-à-vis N2 dont le type est **aléatoire**.
- (P2, Q) :
  1. Une relation (**prod-cons**) vis-à-vis F2 dont le type est **égalité**.
  2. Une relation (**cons-diff**) vis-à-vis N1 dont le type est **aléatoire**.

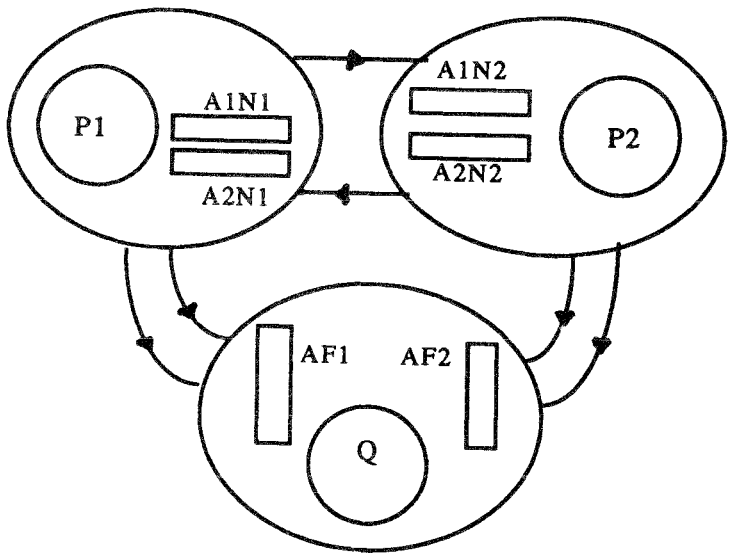


Figure 4.13

Détection de la symétrie entre les processus :
Nous remarquons que P1 et P2 ont les mêmes relations de communication avec le processus Q par conséquent ils sont symétriques vis-à-vis ce processus et les deux regroupements ((P1,Q), (P2)), ((P2,Q), (P1)) sont équivalents.
Deux regroupement seulement nécessitent l'évaluation :

1. ((P1,P2), (Q)), figure 14.
2. ((P1), (P2, Q)), figure 15.

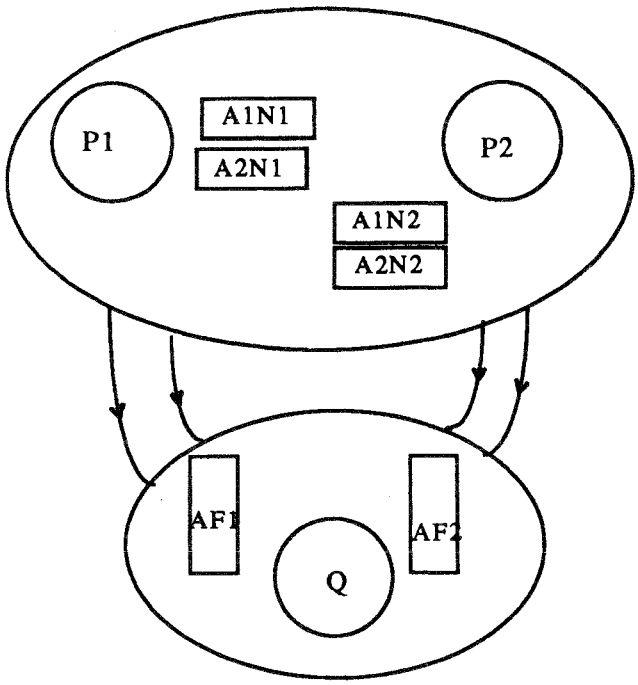


Figure 4.14

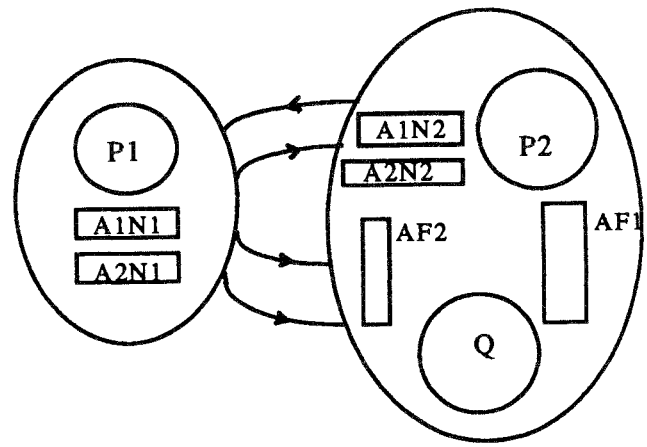


Figure 4.15

Evaluation des regroupements :

1. L'évaluation des regroupement nécessite l'association des facteurs C_k et f_{ijk} de la fonction de synchronisation à des valeurs numériques.

   Les facteurs C_k : Les types de communication utilisés dans cette application parallèle sont : égalité et aléatoire. Ces deux types sont ordonnés totalement par la relation "$as <$" (le type aléatoire est plus asynchrone que le type égalité avec perte des valeurs). En supposant que cette relation est une relation suffisante pour comparer les types de communication, nous pouvons prendre comme valeurs d'essai :
   $C_1 = 1$, $C_2 = 0.5$, (1= égalité, 2= aléatoire).

   Les facteurs f_{ijk} : Les relations de communication existant entre processus de l'application

parallèle sont : prod-cons et cons-diff. Nous donnons la priorité à la relation prod-cons et nous prenons comme valeurs :
$f_1 = 1$, $f_2 = 0.4$, (1= prod-cons, 2= cons-diff).

L'évaluation des regroupements est donnée par le tableau 5. Les valeurs des fonctions $\mathcal{G}, \mathcal{D}, \mathcal{L}$ sont les mêmes dans les deux regroupements. Le deuxième regroupement ((P1), (P2,Q)) minimise la fonction $\mathcal{F}$. Donc ce regroupemnt sera retenu, figure 15.
Groupes = { G1, G2 }
G1 = (PRCS : { P1 }, INS : { A1N1, A2N1 })
G2 = (PRCS : { P2, Q }, INS : { A1N2, A2N2, AF1, AF2 })

Regroupement	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{D}$	$\mathcal{L}$
((P1,P2), (Q))	2.4	1	4	2
((P1), (P2,Q))	2.2	1	4	2

Tableau (5)

2. En supposant que la relation d'ordre entre les types de communication qui nous intéressent est la relation "*op* <", les deux types de communication "égalité" et "aléatoire" ne sont pas comparables. Pour éliminer leur effet sur la fonction de synchronisation nous associons les facteurs C_k à la même valeur numérique :
$C_1 = 1$, $C_2 = 1$.
Et nous gardons les mêmes valeurs pour les facteurs f_{ijk}.

Dans ce cas, l'évaluation des regroupements est donnée par le tableau 6.

Regroupement	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{D}$	$\mathcal{L}$
((P1,P2), (Q))	2.8	1	4	2
((P1), (P2,Q))	3.4	1	4	2

Tableau (6)

Nous remarquons que le premier regroupement minimise la fonction de synchronisation $\mathcal{F}$, et ce regroupement doit être retenu, figure 14.
Groupes = { G1, G2 }
G1 = (PRCS : { P1, P2 }, INS : { A1N1, A2N1, A1N2, A2N2 })
G2 = (PRCS : { Q }, INS : { AF1, AF2 })

3. En prenant comme valeurs des facteurs C_k les valeurs prises dans le premier cas, ($C_1 = 1$, $C_2 = 0.5$), et supposant que les deux relations de communication prod-cons et cons-diff ont la même priorité ($f_1 = 1$, $f_2 = 1$), nous trouvons les évaluations de tableau 7. Dans ce cas le deuxième regroupement minimise la fonction de dépendance $\mathcal{F}$, figure 15.
Groupes = { G1, G2 }
G1 = (PRCS : { P1 }, INS : { A1N1, A2N1 })
G2 = (PRCS : { P2, Q }, INS : { A1N2, A2N2, AF1, AF2 })

Cet exemple montre l'effet de choix de valeurs numériques des facteurs C_k et f_{ijk} sur le placement.

Regroupement	$\mathcal{F}$	$\mathcal{G}$	$\mathcal{D}$	$\mathcal{L}$
((P1,P2), (Q))	2.5	1	4	2
((P1), (P2,Q))	2	1	4	2

Tableau (7)

Conclusion

Dans notre étude les règles heuristiques de la fonction de placement sont essentiellement des règles basées sur les caractèristiques des applications parallèles de langage LESP, et surtout sur le concept de type de communication. Cela implique la différence entre notre fonction de placement et les différents travaux développés pour réaliser le même objectif.

Nos règles heuristiques peuvent être complétées, dans l'avenir par des règles plus générales qui prennent en compte : le contexte de processus, la topologie de l'architecture, etc. Il est possible aussi de compléter ces règles par des règles utilisées dans d'autres études de la littérature et qui sont mentionnées dans l'introduction de ce chapitre.

Chapitre 5

Dérivation d'une application répartie

Introduction

La deuxième étape de l'implantation d'une application parallèle sur un système réparti est la dérivation de l'application répartie correspondante à l'application parallèle pour un placement déterminé.

Cette **dérivation**, réalisée par la fonction φ (voir chapitre 3), est divisée en deux parties :

- la **création** de l'application répartie. Cette création s'agit de :
  - l'allocation de composants de l'application parallèle aux sites de l'architecture répartie,
  - la création de médium de communication de l'application répartie. Ce médium est composé de ports d'émission, de ports de réception et d'un réseau de communication.
- la **représentation** des opérations de communication dans l'environnement réparti.

Naturellement le placement considéré lors de cette dérivation est le résultat de la fonction de placement : les groupes (cf : chapitre 4).

5.1 La création de l'application répartie

La création de l'application répartie s'agit ici d'allouer les processus et les ensembles indexés de l'application parallèle aux sites de l'architecture répartie, de créer : les ports nécessaires pour la communication entre les sites et les tables de placement de l'application répartie.
Pour cette création nous allons définir un **procédé incremental** (pas à pas) d'allocation de ressources à l'application répartie. Et nous proposons de commencer par une application répartie "vide". Chaque étape de la création enrichit l'application répartie initiale par l'allocation d'un nouveau objet.
La fonction de création est composée de quatre procédures :

- La procédure d'**initialisation** : son rôle est la création d'une application répartie vide.

- La procédure d'**allocation** de composants de l'application parallèle aux sites de l'architecture répartie.
- La procédure **"émission-réception"** : son rôle est la création de ports d'émission et de ports réception associés aux ensembles indexés de l'application répartie. Elle définit aussi les tables : TAB-EMISSION, TAB-RECEPTION de l'application répartie.
- La procédure **réseau** : son rôle est la création des ports et et du réseau.

Les entrées de la fonction de création de l'application répartie sont : l'application parallèle, les groupes retenus par la fonction de placement et les sites du système réparti.

Dans les différentes procédures qui suivent nous utilisons des opérations de création de composants d'application répartie. Ces opérations sont :

- creer-tab : une opération de création d'une table.
  creer-tab : → table.
- creer-index : une opération de création d'un accès-ensemble-indexé.
  creer-index : → accès-ensemble-indexé.
- creer-port : une opération de création d'un accès-port.
  creer-port : → accès-port.

5.1.1 La procédure d'initialisation

Cette procédure crée une application répartie vide : DA. Les processus de cette application (PROCESSES est une paramètre du type application-répartie) sont les processus de l'application parallèle. Ses tables de placement sont des tables vides. Et ses sites sont les sites du système réparti que nous disposons.

Procédure d'initialisation

- Entrées
  - APPL : application-parallèle.
  - { S_1,S_2,...,S_n} : ensemble (accès-site)
    % Les sites du système réparti.%
- Sorties
  - DA : application-répartie.
    DA = DA (PLAC, PROCES. . . .)
    % DA est paramétré par la famille de tables de placement PLAC et par l'ensemble de processus PROCES %
- Corps
  - PROCES = PROC(@APPL)
  - TPROC = creer-tab()

- TCOM = creer-tab()
- TEMISS = creer-tab()
- TRECEP = creer-tab()
- PLAC = (%TAB-PROC% : TPROC, %TAB-COM% : TCOM, %TAB-EMISSION% : TEMISS, %TAB-RECEPTION% : TRECEP)
- RES = { }
- Pour chaque S_i, i=1 ...n faire
  * OBJ(@$\acute{S}_i$) = (%EIS% : { }; %EMISSION% : { }; %RECEPTION% : { })
- DA = (%RESEAU% : RES; %SITES% : { $\acute{S}_1$,$\acute{S}_2$,...,$\acute{S}_n$})

5.1.2 La procédure d'allocation

Nous supposons dans notre étude que les sites de l'architecture répartie sont identiques et que son réseau de communication est complétement maillé. Par conséquent le choix du site de placement d'un groupe donné est arbitraire à condition d'avoir **un groupe par site.** L'association (groupe → site) se déduit, donc, facilement.
Cette association permet :

- la création des ensembles indexés des sites,
- la définition des tables : TAB-PROC, TAB-COM. de l'application répartie.

Procédure d'allocation

- Entrées
  - APPL : application-parallèle.
  - DA : application-répartie.
    DA = DA (PLAC, PROCES,...)
    % DA est paramétré par la famille de table de placement PLAC et par l'ensemble de processus PROCES. %
    DA = (%RESEAU% : RES, %SITES% : { S_1, ..., S_n })
    % DA est composé de réseau RES et des sites { S_1, ..., S_n } %
  - Groupes = { G_1,..., G_n }, un regroupement (voir chapitre 4).
- Sorties
  - $\acute{DA}$: application-répartie.
    $\acute{DA}$ = $\acute{DA}$ ($\acute{PLAC}$, PROCES, ...)
- Corps
  - TPROC = TAB-PROC(@PLAC)
  - TCOM = TAB-COM(@PLAC)
  - Pour chaque G_i i = 1,..., n faire

* Associer G_i à S_i
  % Associer chaque groupe à un site du système réparti %

- Fin pour
- Pour chaque G_i, i = 1,..., n faire
  * S_i : accès-site telque G_i est associé à S_i
  * Pour chaque P : accès-processus telque $P \in PRCS(G_i)$ faire
    · $TP\acute{R}OC$ = ajout-tab(TPROC, NOM(@P), NOM(@S_i))
    % Allouer chaque processus de groupe G_i au site S_i %
  * Fin pour
  * Pour chaque E : accès-ensemble-indexé(typelem, TC) telque $E \in INS(G_i)$ faire
    · EI = creer-index()
    · EIS(@OBJ(@$\acute{S}_i$)) = EIS(@OBJ(@S_i)) ∪ { EI }
    · $TC\acute{O}M$ = ajout-tab(TCOM, NOM(@EI), NOM(@S_i))
    % Création et allocation d'un ensemble indexé au site S_i %
  * Fin pour
- Fin pour
- TAB-PROC(@$PL\acute{A}C$) = $TP\acute{R}OC$
- TAB-COM(@$PL\acute{A}C$) = $TC\acute{O}M$
- $D\acute{A}$ = (%RESEAU% : RES, %SITES% : { $\acute{S}_1, \ldots, \acute{S}_n$ })

Fin

5.1.3 La procédure "émission-réception"

La création d'un couple de ports est nécessaire pour satisfaire les opérations de communication entre un processus et un ensemble indexé, s'ils ne sont pas placés sur le même site. La procédure "émission-réception" crée les ports d'émission ou de réception qui sont associés aux ensembles indexés, ainsi que les deux tables TAB-EMISSION, TAB-RECEPTION de l'application répartie.

Remarques : Dans les schémas des figures (3,..., 7) du chapitre 4, nous remarquons que :

- L'association entre un ensemble indexé et un port d'émission (resp : de réception) est telle que : chaque ensemble indexé est associé à, au maximum, un port d'émission (resp : de réception) et chaque port d'émission (resp : de réception) est associé à, au maximum, un ensemble indexé.
- Un port d'émission (resp : de réception) peut être le port d'émission "SEND" (resp : de réception "RECEIVE") de plusieurs liaisons.
- D'aprés les schémas (4-b, 7-c), nous remarquons que le regroupement des producteurs ou des consommateurs ne diminue pas le nombre de liaisons, car chaque processus produit (figure 7-c) dans un port d'émission ou consomme d'un port de réception (figure 4-b). La minimisation du nombre de liaisons est possible par la création d'un port d'émission (resp :

de réception) par site et pour un ensemble indexé donné.

Ces remarques ont été integrées lors de la conception des procédures qui suivent.

Procédure émission-réception

- Entrées
  - APPL : application-parallèle.
  - DA : application-répartie.
    DA = DA (PLAC, PROCES,...)
    % PLAC est la famille de table de placement de DA, PROCES est l'ensemble de processus de l'application DA. %
    DA = (%RESEAU% : RES, %SITES% : { $S_1, \ldots, S_n$ })
- Sorties
  - DA : application-répartie.
    $DA = DA$ ($PLAC$, PROCES, ...)
- Corps
  - TEMISS = TAB-EMISSION (@PLAC)
  - TRECEP = TAB-RECEPTION (@PLAC)
  - TCOM = TAB-COM (@PLAC)
  - TPROC = TAB-PROC (@PLAC)
  - Pour chaque EI : accès-ensemble-indexé(typelem, TC) telque
    NOM(@EI) ∈ domaine(TCOM) faire
    % Pour chaque accès-ensemble-indexé de l'application répartie %
    * S : accès-site telque TCOM[NOM(@EI)] = NOM(@S)
      % S est le site de placement de EI %
    * Si (∃ P : accès-processus telque
      P ∈ PROCES ∧ NOM(@EI) ∈ ENTREE(@P) ∧
      TPROC[NOM(@P)] ≠ NOM(@S) ∧
      ¬ (NOM(@EI) ∈ domaine(TEMISS))) alors
      % S'il existe un consommateur P, de EI, qui n'est pas placé sur le même site que EI %
      · PE = creer-port()
      · EMISSION(@OBJ(@S)) = EMISSION(@OBJ(@S)) ∪ { PE }
        % Création de port d'émission PE sur le site de placement de EI %
      · $TEMISS$ = ajout-tab(TEMISS, NOM(@EI), NOM(@PE))
        % Association du port d'émission PE à l'accès ensemble indexé EI %
    * Fin si
    * Si (∃ P : accès-processus telque
      P ∈ PROCES ∧ NOM(@EI) ∈ SORTIE(@P) ∧
      TPROC[NOM(@P)] ≠ NOM(@S) ∧

¬ (NOM(@EI) ∈ domaine(TRECEP))) alors
% S'il existe un producteur P, de EI, qui n'est pas placé sur le même site que EI %
· PR = creer-port()
· RECEPTION(@OBJ(@$\acute{S}$)) = RECEPTION(@OBJ(@S)) ∪ { PR }
% Création de port de réception PR sur le site de placement de EI %
· $TRE\acute{C}EP$ = ajout-tab(TRECEP, NOM(@EI), NOM(@PR))
% Association du port de réception PR à l'accès ensemble indexé EI %
* Fin si
- Fin pour
- TAB-EMISSION(@$PL\acute{A}C$) = $TEMISS$
- TAB-RECEPTION(@$PL\acute{A}C$) = $TRE\acute{C}EP$
- $D\acute{A}$ = (%RESEAU% : RES, %SITES% : { $\acute{S}_1, \ldots, \acute{S}_n$ })

Fin

5.1.4 La procédure réseau

L'objectif de cette procédure est de créer les ports du système de l'application répartie qui ne sont pas associés aux ensembles indexés et de définir le réseau du système.

Procédure réseau

- Entrées
  - APPL : application-parallèle,
  - DA : application-répartie.
    DA = DA (PLAC, PROCES,...)
    % PLAC est la famille de table de placement de DA, PROCES est l'ensemble de processus de l'application DA. %
    DA = (%RESEAU% : RES, %SITES% : { $S_1, \ldots, S_n$ })
- Sorties
  - $D\acute{A}$: application-répartie.
- Corps
  - TEMISS = TAB-EMISSION (@PLAC)
  - TRECEP = TAB-RECEPTION (@PLAC)
  - TCOM = TAB-COM (@PLAC)
  - TPROC = TAB-PROC (@PLAC)
  - Pour chaque P : accès-processus telque P ∈ PROCES faire
    * S :accès-site telque TPROC[NOM(@P)] = NOM(@S)

* Pour chaque EI : accès-ensemble-indexé (typelem, TC) telque
NOM(@EI) ∈ ENTREE(@P) faire
  - Si TCOM[NOM(@EI)] ≠ NOM(@S) alors
    % Si P est un consommateur de EI et que P et EI ne sont pas placés sur le même site %
  - PE : accès-port telque NOM(@PE) = TEMISS[NOM(@EI)]
  - PR = creer-port()
  - RECEPTION(@OBJ(@$\acute{S}$)) = RECEPTION(@OBJ(@S)) ∪ { PR }
    % Création de port de réception PR sur le site de placement de processus P %
  - $\acute{RES}$ = RES ∪ { (NOM(@PE), NOM(@PR)) }
    % Création d'une liaison entre le site de placement de processus P et le site de placement de l'ensemble indexé EI.
    La liaison est crée entre les ports PE et PR %
  - Fin si
* Fin pour
* Pour chaque EI : accès-ensemble-indexé(typelem, TC) telque
NOM(@EI) ∈ SORTIE(@P) faire
  - Si TCOM[NOM(@EI)] ≠ NOM(@S) alors
    % Si P est un producteur de EI et que P et EI ne sont pas placés sur le même site %
  - PR : accès-port telque NOM(@PR) = TRECEP[NOM(@EI)]
  - PE= creer-port()
  - EMISSION(@OBJ(@$\acute{S}$)) = EMISSION(@OBJ(@S)) ∪ { PE }
    % Création de port d'émission PE sur le site de placement de P %
  - $\acute{RES}$ = RES ∪ { (NOM(@PE), NOM(@PR)) }
    % Création d'une liaison entre le site de placement de processus P et le site de placement de l'ensemble indexé EI.
    Cette liaison est crée entre les ports PE et PR %
* Fin si

– Fin pour

• Fin pour

• $\acute{DA}$ = (%RESEAU% : $\acute{RES}$, %SITES% : { $\acute{S}_1, \ldots, \acute{S}_n$ })

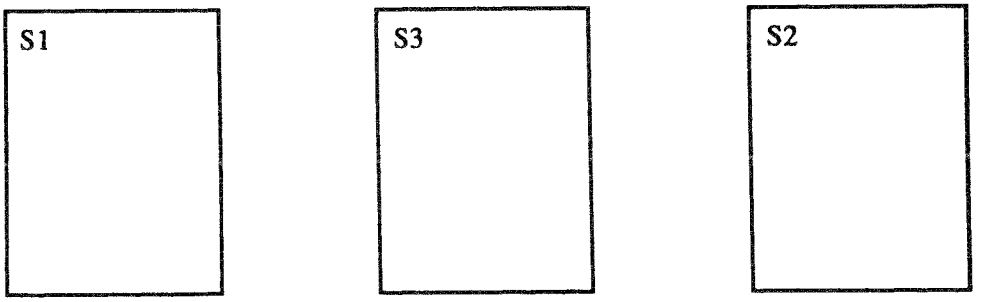
Fin

L'exécution de ces différentes procédures permet la création de l'application répartie, image de l'application parallèle sur un système réparti.

5.1.5 Application à un exemple

Nous allons illustrer la mise en œuvre de la fonction **Création** sur l'exemple développé au chapitre 4, (figure 13).
L'application parallèle considérée est l'application APPL2 dont la spécification est donnée dans le (chapitre 1). Supposons que le regroupement retenu soit le regroupement suivant :
Groupes = { G1, G2, G3 }
G1 = (%PRCS% : { P1 }, %INS% : { A1N1, A2N1 })
G2 = (%PRCS% : { P2 }, %INS% : { A1N2, A2N2 })
G3 = (%PRCS% : { Q }, %INS% : { AF1, AF2 })

Application de la procédure d'initialisation : (figure 1)
Nous supposons que le système réparti est composé de trois sites { S_1, S_2, S_3 } , et que l'application répartie que nous créons s'appelle DA.
TPR = creer-tab()
TCO = creer-tab()
TEM = creer-tab()
TRE = creer-tab()
PLAC = (%TAB-PROC% : TPR, %TAB-COM% : TCO, %TAB-EMISSION% : TEM,
%TAB-RECEPTION% : TRE)
PROCES = { P1, P2, Q }
DA = DA (PLAC, PROCES, ...)
RES = { }
$\forall i$, $1 \leq i \leq 3$; OBJ(@S_i) = (EIS : { }; EMISSION : { }; RECEPTION : { })
DA = (%RESEAU% : RES; %SITES% : { S_1,S_2, S_3})



L'initialisation de l'application répartie
Figure 5.1

Application de la procédure d'allocation : (figure 2)

Associer G1 à S1 ⇒

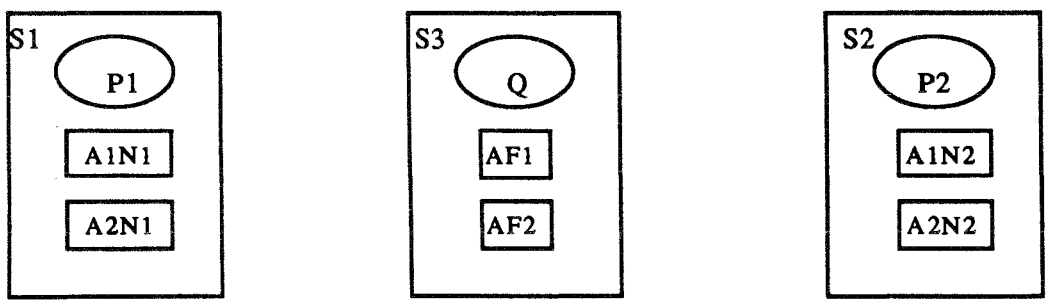
TPR^1 = ajout-tab(TPR, np1, s1)
TCO^1 = ajout-tab(TCO, a1n1, s1)
TCO^2 = ajout-tab(TCO^1, a2n1, s1)
A1N1 = creer-index ()
A2N1 = creer-index ()
EIS(@OBJ($@S_1^1$)) = EIS(@OBJ($@S_1$)) ∪ { A1N1, A2N1 }

Associer G2 à S2 ⇒

TPR^2 = ajout-tab(TPR^1, np2, s2)
TCO^3 = ajout-tab(TCO^2, a1n2, s2)
TCO^4 = ajout-tab(TCO^3, a2n2, s2)
A1N2 = creer-index ()
A2N2 = creer-index ()
EIS(@OBJ($@S_2^1$)) = EIS(@OBJ($@S_2$)) ∪ { A1N2, A2N2 }

Associer G3 à S3 ⇒

TPR^3 = ajout-tab(TPR^2, nq, s3)
TCO^5 = ajout-tab(TCO^4, af1, s3)
TCO^6 = ajout-tab(TCO^5, af1, s3)
AF1 = creer-index () entier, TC.égalité)
AF2 = creer-index (entier, TC.égalité)
EIS(@OBJ($@S_3^1$)) = EIS(@OBJ($@S_3$)) ∪ { AF1, AF2 }



L'allocation des processus et des objets de communication aux sites du système réparti

Figure 5.2

Application de la procédure "émission-réception" : (figure 3)

TCO = TCO^6
TPR = TPR^3
TCO[a1n1] = s1 ∧ a1n1 ∈ ENTREE(@P2) ∧ TPR[np2] ≠ s1 ⇒
PS1 = creer-port(entier, 1)
TEM^1 = ajout-tab(TEM, a1n1, ps1)
EMISSION(@OBJ($@S_1^2$)) = { PS1 }

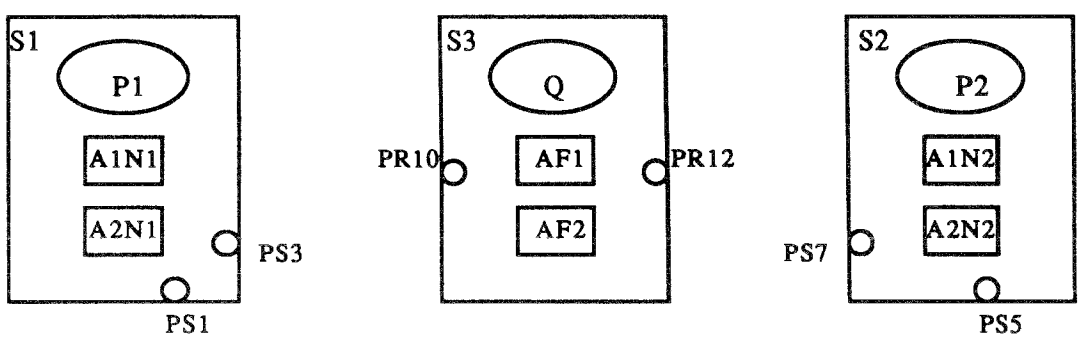
TCO[a2n1] = s1 ∧ a2n1 ∈ ENTREE(@Q) ∧ TPR[nq] ≠ s1 ⇒
PS3 = creer-port(entier, 1)
TEM^2 = ajout-tab(TEM^1, a2n1, ps3)
EMISSION(@OBJ($@S_1^3$)) = { PS1 } ∪ { PS3 }

TCO[a1n2] = s2 ∧ a1n2 ∈ ENTREE(@P1) ∧ TPR[np1] ≠ s2 ⇒
PS5 = creer-port(entier, 1)
TEM^3 = ajout-tab(TEM^2, a1n2, ps5)
EMISSION(@OBJ($@S_2^2$)) = { PS5 }

TCO[a2n2] = s2 ∧ a2n2 ∈ ENTREE(@Q) ∧ TPR[nq] ≠ s2 ⇒
PS7 = creer-port(entier, 1)
TEM^4 = ajout-tab(TEM^3, a2n2, ps7)
EMISSION(@OBJ($@S_2^3$)) = { PS5 } ∪ { PS7 }

TCO[af1] = s3 ∧ af1 ∈ SORTIE(@P1) ∧ TPR[np1] ≠ s3 ⇒
PR10 = creer-port(entier, 1)
TEM^5 = ajout-tab(TEM^4, af1, pr10)
RECEPTION(@OBJ($@S_3^2$)) = { PR10 }

TCO[af2] = s3 ∧ af2 ∈ SORTIE(@P2) ∧ TPR[np2] ≠ s3 ⇒
PR12 = creer-port(entier, 1)
TEM^6 = ajout-tab(TEM^5, af2, pr12)
RECEPTION(@OBJ($@S_3^3$)) = { PR10 } ∪ { PR12 }



La création des ports associés aux ensembles indexés.

Figure 5.3

L'application de la procédure réseau : (figure 4)

TPR[np1] = s1 ∧ a1n2 ∈ ENTREE(@P1) ∧ TCO[a1n2] ≠ s1 ⇒
PR6 = creer-port(entier, 1)
RECEPTION(@OBJ(@S_1^4)) = { PR6 }
RES^1 = RES ∪ { (ps5,pr6) }

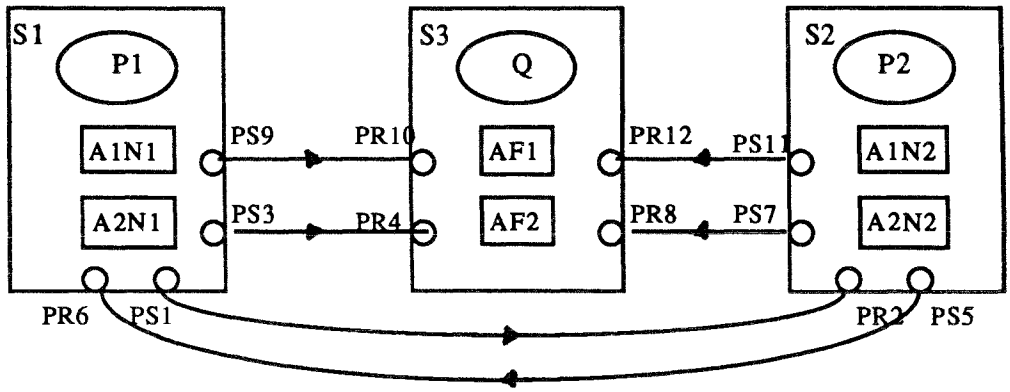
TPR[np2] = s2 ∧ a1n1 ∈ ENTREE(@P2) ∧ TCO[a1n1] ≠ s2 ⇒
PR2 = creer-port(entier, 1)
RECEPTION(@OBJ(@S_2^4)) = { PR2 }
RES^2 = RES^1 ∪ { (ps1, pr2) }

TPR[nq] = s3 ∧ a2n1 ∈ ENTREE(@Q) ∧ TCOM[a2n1] ≠ s3 ⇒
PR4 = creer-port(entier, 1)
RECEPTION(@OBJ(@S_3^4)) = { PR10, PR12 } ∪ { PR4 }
RES^3 = RES^2 ∪ { (ps3, pr4) }

TPR[nq] = s3 ∧ a2n2 ∈ ENTREE(@Q) ∧ TCO[a2n2] ≠ s3 ⇒
PR8 = creer-port(entier, 1)
RECEPTION(@OBJ(@S_3^5)) = { PR4, PR10, PR12 } ∪ { PR8 }
RES^4 = RES^3 ∪ { (ps7, pr8) }

TPR[np1] = s1 ∧ af1 ∈ SORTIE(@P1) ∧ TCO[af1] ≠ s1 ⇒
PS9 = creer-port(entier, 1)
EMISSION(@OBJ(@S_1^5)) = { PS1, PS3 } ∪ { PS9 }
RES^5 = RES^4 ∪ { (ps9, pr10) }

TPROC[np2] = s2 ∧ af2 ∈ SORTIE(@P2) ∧ TCO[af2] ≠ s2 ⇒
PS11 = creer-port(entier, 1)
EMISSION(@OBJ(@S_2^5)) = { PS5, PS7 } ∪ { PS11 }
RES^6 = RES^5 ∪ { (ps11, pr12) }



Création du réseau de l'application répartie

Figure 5.4

L'application répartie DA :
PLAC = (%TAB-PROC% : TPR^3, %TAB-COM% : TCO^6, %TAB-EMISSION% : TEM^6,
%TAB-RECEPTION% : TRE^6)
PROCES = { P1, P2, Q }
DA = DA (PLAC, PROCES, ...)
DA = (%RESEAU% : RES^6, %SITES% : { S_1^5, S_2^5, S_3^5 })

5.2 La représentation des opérations de communication

L'implantation d'un programme parallèle sur un système réparti entraine l'exécution à distance (répartie) de certaines opérations. Une opération à distance est exécutée sur deux sites ou plus, et elle nécessite des opérations de transmission de messages d'un site à un autre. D'une manière générale cette opération est déclenchée (provoquée) sur un site et se termine sur un autre site.

En considérant les applications parallèles citées au chapitre 1, et en prenant en compte le fait que chaque processus s'exécute sur un seul site et que les opérations de communication sont les seules opérations non locales aux processus, nous déduisons que l'exécution de ces applications parallèles rend certaines opérations de communication (production, consommation, message-à-consommer), des opérations à distance et ce sont les seules opérations qui peuvent l'être. Une opération de communication sur un objet de communication est une opération locale si le processus qui la provoque et les différents composants de l'objet concernés par cette opération sont placés sur le même site du système réparti, et elle est à distance dans le cas contraire. Pour simplifier la distinction entre une opération locale et une opération à distance, nous introduisons le prédicat suivant :

local : application-répartie * accès-port * accès-ensemble-indexé → booléen

Soit DA : application-répartie telque DA = DA(PLAC, PROC,...)
local(DA, P, E) ⇔
TAB-PROC(@PLAC)[NOM(@P)] = TAB(@PLAC)[NOM(@E)]
% local(DA,P, E) est valide si le processus P et l'ensemble indexé E sont placés sur le même site %

L'exécution des opérations de communication dépend du placement des processus et des objets de communication sur les différents sites du système réparti. Nous allons présenter dans ce paragraphe la représentation des opérations de communication dans l'environnement réparti pour un objet de topologie "bipoint". Pour les autres topologies, nous allons détailler certains cas qui ne se ramènent pas au cas de " bipoint ".

Dans cette **représentation** nous distinguons **deux aspects** :

- L'**aspect formel** décrit par la spécification de la représentation des opérations de communication dans l'environnement réparti. Cette spécification est donnée en terms de pré et post condition.
- L'**aspect dynamique** décrit par des procédures qui satisfont les opérations de communication.

5.2.1 La spécification de la représentation des opérations de communication dans l'environnement réparti

Chaque opération de communication de l'application parallèle est définie par des opérandes, des résultats, une pré-condition déterminant la condition nécessaire pour sa réalisation et une post-condition qui détermine l'état résultant de l'opération.

L'exécution de certaines opérations de communication de l'application parallèle sur un système réparti nécessite la transmission des données entre les sites du système, par conséquent de nouvelles conditions concernant les liaisons ou d'autres éléments de l'application répartie doivent être verifiées pour l'exécution de ces opérations à distance. D'autre part ces opérations modifient les états de plusieurs sites.
En supposant que chaque opération est définie par :

Opération : (opérandes, résultats, pré-condition, post-condition)

La représentation de cette opération dans l'environnement répartie φ **(opération)**, est définie par :

φ **(opération)** =

(φ_1 **(opérandes)**, φ_2 **(résultat)**, φ_3 **(pré-condition)**, φ_4 **(post-condition)**)

Une opération de communication est une opération du type application parallèle. Plus précisement, cette opération modifie l'état de l'application parallèle. Comme l'image d'une application parallèle est une application répartie, la représentation d'une opération de communication dans un environnement réparti est définie sur le type : application répartie, elle modifie l'état de cette application et sa pré-condition (resp : post-condition) se transforme en une nouvelle pré-condition (resp : post-condition).

Dans cette représentation nous distinguons deux cas : le cas local et le cas réparti. Le cas local n'implique pas de modifications dans la pré-condition ni dans la post-condition de l'opération, tandis que le cas réparti implique des modifications que nous préciserons par la suite, pour chaque opération de communication.

L'opération de production

L'opération de production est une opération définie sur le type : application parallèle. Les opérandes de cette opération sont : une application parallèle, un ensemble-indexé et un message. Le résultat de cette opération est l'application parallèle dans un nouvel état.
Pour satisfaire cette opération (pré-condition) il faut qu'il y ait un processus producteur de l'ensemble-indexé appartenant à l'application parallèle et que l'ensemble indexé appartienne aussi à cette application.
L'effet de cette opération (post-condition) est d'ajouter le message produit dans l'ensemble-indexé.

Produire :
opérandes : PA : application-parallèle, E : accès-ensemble-indexé, M : message.

résultat : $\acute{PA}$: application-parallèle.
pré : E $\in \overline{ENS-IND}$(@PA) $\wedge$
($\exists$ P : accès-processus , P $\in$ PROC(@PA) $\wedge$ NOM(@E) $\in$ SORTIE(@P))
post : $\acute{PA}$ = chang(@PA, E, $\acute{E}$), $\acute{E}$ = chang(@E, IND(@E), $\acute{EI}$), $\acute{EI}$ = prod-EI (IND(@E), M)

La représentation de l'opération de production notée φ **(produire)** est définie par :

φ **(produire)** =

(φ_1 **(opérandes(produire))**, φ_2 **(résultat(produire))**,
φ_3 **(pré-condition(produire))**, φ_4 **(post-condition(produire))**)

Les opérandes de φ (produire) sont : une application répartie qui est l'image de l'application parallèle sur un système réparti, un accès-ensemble-indexé et le message produit.

φ_1 **(opérandes(produire))** = φ **(PA)**, φ **(E)**, φ **(M)**.
φ (PA) = DA = DA(PLAC, PROC,...) où DA : application-répartie,
φ (E) = E
φ (M) = M.

Le résultat de φ (produire) est l'application répartie dans un nouvel état.
φ_2 **(résultat(produire))**= φ ($\acute{PA}$)
φ ($\acute{PA}$) = $\acute{DA}$ = $\acute{DA}$($\acute{PLAC}$, PROC....) où $\acute{DA}$: application-répartie.

La pré-condition de φ (produire) est **une des deux** pré-conditions suivantes : l'une représente la pré-condition dans le cas local et l'autre représente la pré-condition dans le cas réparti :

- Cas local : c'est la pré-condition de produire appliquée à l'application répartie : nous considerons les processus de l'application parallèle donnés en paramètres et les ensembles indexés des sites de l'application répartie.
- Cas réparti : il faut que les conditions suivantes soient verifiées :
  - la pré-condition de produire appliquée à l'application répartie.
  - l'existance d'une liaison entre le site de placement du processus producteur de l'ensemble-indexé et le site de placement de l'ensemble indexé.
  - la pré-condition de l'opération de transmission d'un message sur la liaison L reliant les deux sites.

φ_3 **(pré(produire))** =

local (DA, P, E) $\Rightarrow$ φ_3 **local (pré(produire))**

non local (DA, P, E) $\Rightarrow$ φ_3 **réparti (pré(produire))**

φ_3 **local (pré(produire))** :
$\exists$ P : accès-processus, S : accès-site telque
S $\in \overline{SITES}$(@DA) $\wedge$ E $\in \overline{EIS}$(@OBJ(@S)) $\wedge$ P $\in$ PROC $\wedge$ NOM(@E) $\in$ SORTIE(@P)

φ_3 réparti (pré(produire)) :
($\exists$ L : liaison telque L $\in$ RESEAU(@DA)),
($\exists$ S1,S2 :accès-site telque S1 $\in \overline{SITES}$(@DA), S2 $\in \overline{SITES}$(@DA),)
($\exists$ P1, P2 : accès-port telque
send-sp(S1,P1,L), receive-sp(S2,P2,L))
NOM(@S1) = TAB-PROC(@PLAC)[NOM(@P)] $\wedge$
NOM(@S2) = TAB-COM(@PLAC)[NOM(@E)] $\wedge$
RECEIVE(@L) = TAB-RECEPTION(@PLAC)[NOM(@E)] $\wedge$
non plein(PORT(@P1)) $\wedge$ non plein(PORT(@P2)) $\wedge$
($\exists$ P : accès-processus, S : accès-site telque
S $\in \overline{SITES}$(@DA) $\wedge$ E $\in \overline{EIS}$(@OBJ(@S)) $\wedge$ P $\in$ PROC $\wedge$ NOM(@E) $\in$ SORTIE(@P))

La post-condition de φ (produire) est **une des deux** post-conditions suivantes :

- La post-condition dans le cas d'une opération locale : elle est la post-condition de "produire", appliquée à l'application répartie.
- La post-condition dans le cas d'une opération répartie : la réalisation de la post-condition de "produire" implique de modifications d'état de plusieurs composant de l'application répartie. Précisement : l'état de site de placement de processus et l'état de site de placement d'accès-ensemble-indexé.

φ_4 (post(produire)) =

local (DA, P, E) $\Rightarrow$ φ_4 local (post(produire))

non local (DA, P, E) $\Rightarrow$ φ_4 réparti (post(produire))

φ_4 local (post(produire)) :
$\acute{DA}$ = chang(@DA, $S1$, $\acute{S1}$)
$\acute{S1}$ = chang(@$S1$, E, $\acute{E}$)
$\acute{E}$ = chang(@E, IND(@E), $\acute{EI}$)
$\acute{EI}$ = prod-EI(IND(@E), M)

φ_4 réparti (post(produire)) :
DA^1 = chang(@DA, S1, $\acute{S1}$)
$\acute{S1}$ = chang(@S1, P1, $\acute{P1}$)
PORT(@$\acute{P1}$) = mettre(PORT(@P1),M)
DA^2 = transmettre (DA^1, L)
DA^3 = chang(@DA^2, S2, $S2^1$)
$S2^1$ = chang(@S2, P2, $\acute{P2}$)
PORT(@$\acute{P2}$) = enlever(PORT(@P2))
$\acute{DA}$ = chang(@DA^3, $S2^1$, $\acute{S2}$) $\acute{S2}$ = chang(@$S2^1$, E, $\acute{E}$)
$\acute{E}$ = chang(@E, IND(@E), $\acute{EI}$)
$\acute{EI}$ = prod-EI(IND(@E), prendre(PORT(@$\acute{P2}$)))

L'opération de consommation

L'opération de consommation est une opération définie sur le type : application parallèle.
Les opérandes de cette opération sont : une application parallèle et un ensemble-indexé.
Le résultat de cette opération est l'application parallèle dans un nouvel état.
Pour satisfaire cette opération (pré-condition) il faut qu'il y ait un processus consommateur de l'ensemble-indexé appartenant à l'application parallèle et que l'ensemble-indexé appartienne aussi à cette application.
L'effet de cette opération (post-condition) est d'enlever un message de l'ensemble-indexé.

Consommer :
opérandes : PA : application-parallèle, E : accès-ensemble-indexé.
résultat : $\acute{PA}$: application-parallèle.
pré : E $\in \overline{ENS-IND}$(@PA) $\wedge$
($\exists$ P : accès-processus , P $\in$ PROC(@PA) $\wedge$ NOM(@E) $\in$ ENTREE(@P)) $\wedge$
pre-cons-EI (IND(@E))
post : $\acute{PA}$ = chang(@PA, E, $\acute{E}$), $\acute{E}$ = chang(@E, IND(@E), $\acute{EI}$),
$\acute{EI}$ = post-cons-EI(IND(@E))

La représentation de l'opération de consommation φ **(consommer)** est définie par :
φ **(consommer)** =

(φ_1 **(opérandes(consommer))**, φ_2 **(résultat(consommer))**,
φ_3 **(pré-condition(consommer))**, φ_4 **(post-condition(consommer))**)

Les opérandes de φ (consommer) sont : une application répartie, l'image de l'application parallèle sur un système réparti, et un ensemble-indexé.

φ_1 **(opérandes(consommer))**= φ **(PA)**, φ **(E)**
φ (PA) = DA = DA(PLAC, PROC,...) où DA : application-répartie,
φ (E) = E

Le résultat de φ (consommer) est l'application répartie dans un nouvel état.
φ_2 **(résultat(consommer))**= φ ($\acute{PA}$)
φ ($\acute{PA}$) = $\acute{DA}$ = $\acute{DA}$ ($\acute{PLAC}$, PROC,...) où $\acute{DA}$: application-répartie.

La pré-condition de φ (consommer) est **une des deux** pré-conditions : l'une représente la pré-condition dans le cas local et l'autre représente la pré-condition dans le cas réparti :

- Cas local : c'est la pré-condition de "consommer" appliquée à l'application répartie.
- Cas répartie : il faut que les conditions suivantes soient verifiées :
  - la pré-condition de consommer appliquée à l'application répartie,
  - l'existance d'une liaison entre le site de placement du processus producteur de l'ensemble-indexé et le site de placement de l'ensemble indexé.
  - la pré-condition de l'opération de transmission d'un message sur la liaison reliant les deux sites.

φ_3 (pré(consommer)) =

local (DA, P, E) $\Rightarrow$ φ_3 local (pré(consommer))

non local (DA, P, E) $\Rightarrow$ φ_3 réparti (pré(consommer))

φ_3 local (pré(consommer)) :
$\exists$ P : accès-processus, S : accès-site telque
S $\in \overline{SITES}$(@DA) $\wedge$ E $\in \overline{EIS}$(@OBJ(@S)) $\wedge$ P $\in$ PROC $\wedge$ NOM(@E) $\in$ ENTREE(@P) $\wedge$ pre-cons-EI(IND(@E))

φ_3 réparti (pré(consommer)) :
($\exists$ L : liaison telque L $\in$ RESEAU(@DA)),
($\exists$ S1, S2 :accès-site telque S1 $\in \overline{SITES}$(@DA), S2 $\in \overline{SITES}$(@DA)),
($\exists$ P1, P2 : accès-port telque
(send-sp(S1,P1,L) , receive-sp(S2,P2,L)) $\wedge$
NOM(@S2) = TAB-PROC(@PLAC)[NOM(@P)] $\wedge$
NOM(@S1) = TAB-COM(@PLAC)[NOM(@E)] $\wedge$
SEND(@L) = TAB-EMISSION(@PLAC)[NOM(@E)] $\wedge$
non plein(PORT(@P1)) $\wedge$ non plein(PORT(@P2)) $\wedge$
($\exists$ P : accès-processus, S : accès-site telque
S $\in \overline{SITES}$(@DA) $\wedge$ E $\in \overline{EIS}$(@OBJ(@S)) $\wedge$ P $\in$ PROC $\wedge$ NOM(@E) $\in$ ENTREE(@P) $\wedge$ pre-cons-EI(IND(@E)))

La post condition de φ (consommer) est la post-condition de "consommer" appliquée à l'application répartie.

φ_4 (post(consommer)) :
DA' = chang(@DA, $S1$, $S1'$)
$S1'$ = chang(@$S1$, E, E')
E' = chang(@E, IND(@E), EI')
EI' = post-cons-EI(IND(@E), M)

L'opération : message-à-consommer

Les opérandes et la pré-condition de cette opération sont les mêmes que celles de l'opération de consommation. Son résultat est un message défini par la post-condition.

Message-à-consommer :
opérandes : PA : application-parallèle, E : accès-ensemble-indexé.
résultat : M : message.
pré : E $\in \overline{ENS-IND}$(@PA) $\wedge$
($\exists$ P : accès-processus , P $\in$ PROC (@PA) $\wedge$
NOM(@E) $\in$ ENTREE(@P)) $\wedge$ pre-cons-EI(IND(@E))
post : M = val-cons-EI (IND(@E))

La représentation de l'opération "message-à-consommer" **φ (message-à-consommer)** est définie

par :

φ (message-à-consommer) =

(**φ_1 (opérandes(message-à-consommer))**,
φ_2 (résultat(message-à-consommer)),
φ_3 (pré-condition(message-à-consommer)),
φ_4 (post-condition(message-à-consommer)))

Nous définissons la représentation de l'opération message-à-consommer en s'inspirant de la représentation de l'opération consommer.

φ_1 (opérandes(message-à-consommer))= φ (PA), φ (E)
φ (PA) = DA = DA(PLAC, PROC,...) où DA : application-répartie,
φ (E) = E

φ_2 (résultat(message-à-consommer))= φ (M)
φ (M) = M

La représentation de la pré-condition est définie par:

φ_3 (pré(message-à-consommer)) =

local (DA, P, E) $\Rightarrow$ φ_3 local (pré(message-à-consommer))

non local (DA, P, E) $\Rightarrow$ φ_3 réparti (pré(message-à-consommer))

φ_3 local (pré(message-à-consommer)) :
$\exists$ P : accès-processus, S : accès-site telque
S $\in \overline{SITES}$(@DA) $\wedge$ E $\in \overline{EIS}$(@OBJ(@S)) $\wedge$ P $\in$ PROC $\wedge$ NOM(@E) $\in$ ENTREE(@P) $\wedge$
pre-cons-EI(IND(@E))

φ_3 réparti (pré(message-à-consommer)) :
($\exists$ L : liaison telque L $\in$ RESEAU(@DA)),
($\exists$ S1, S2 :accès-site telque S1 $\in \overline{SITES}$ (@DA), S2 $\in \overline{SITES}$(@DA)),
($\exists$ P1, P2 : accès-port telque
(send-sp(S1,P1,L), receive-sp(S2,P2,L) $\wedge$
NOM(@S2) = TAB-PROC(@PLAC)[NOM(@P)] $\wedge$
NOM(@S1) = TAB-COM(@PLAC)[NOM(@E)] $\wedge$
SEND(@L) = TAB-EMISSION(@PLAC)[NOM(@E)] $\wedge$
non plein(PORT(@P1)) $\wedge$ non plein(PORT(@P2)) $\wedge$
($\exists$ P : accès-processus, S : accès-site telque
S $\in \overline{SITES}$(@DA) $\wedge$ E $\in \overline{EIS}$(@OBJ(@S)) $\wedge$ P $\in$ PROC $\wedge$ NOM(@E) $\in$ ENTREE(@P) $\wedge$
pre-cons-EI(IND(@E)))

La représentation de la post-condition est définie par :

φ_4 (post(message-à-consommer)) =

local (DA, P, E) $\Rightarrow$ φ_4 local (post(message-à-consommer))

non local (DA, P, E) $\Rightarrow$ φ_4 réparti (post(message-à-consommer))

φ_4 local (post(message-à-consommer)) :
M = val-cons-EI(IND(@E))

φ_4 réparti (post(message-à-consommer)) :
DA^1 = chang(@DA, S1, $\acute{S}1$)
$\acute{S}1$ = chang(@S1, P1, $\acute{S}1$)
PORT(@$\acute{P}1$) = mettre(PORT(@P1),val-cons-EI(IND(@E)))
DA^2 = transmettre(DA^1, L)
$\acute{D}A$ = chang(@DA^2, S2, $\acute{S}2$)
$\acute{S}2$ = chang(@S2, P2, $\acute{P}2$)
PORT(@$\acute{P}2$) = enlever (PORT(@P2))
M = prendre(PORT(@P2)).

5.2.2 L'aspect dynamique de la représentation des opérations de communication dans un environnement réparti

L'exécution des opérations de communication sur un système réparti est une succession d'opérations, de recherche locale ou à distance, de transfert entre deux sites, et d'opérations locales définies sur les ensembles indexés. Les opérations de recherche sont réalisées par l'interrogation des tables de l'application répartie correspondant à l'application parallèle.

Dans ce paragraphe nous allons présenter les procédures d'exécution des différentes opérations de communication dans l'environnement réparti.
Chacune de ces procédures satisfait la spécification de la représentation d'une opération de communication : φ(opération), (cf : 5.2.1). Elle détecte la validité de la pré condition de l'opération : φ_3(pré(opération)), et elle exécute les opérations nécessaires pour satisfaire la post condition de l'opération : φ_4(post(opération)). Chaque procédure distingue deux cas :

- une exécution locale de l'opération : dans ce cas la procédure satisfait φ_4 local (post(opération));
- une exécution répartie de l'opération : dans ce cas la procédure satisfait φ_4 réparti (post(opération)).

La procédure d'exécution de l'opération de production

En supposant que l'application répartie est DA, l'exécution d'une opération de production, déclenchée par le producteur P, sur l'accès-ensemble-indexé E est réalisée de la manière suivante :

1. Soit DA(PLAC, PROC,...) : application-répartie, E : accès-ensemble-indexé,
   P : accès-processus telque
   P $\in$ PROC $\wedge$ NOM(@E) $\in$ SORTIE(@P).
   % P est un producteur de E %

2. TPR = TAB-PROC(@PLAC)
   TCO = TAB-COM(@PLAC)
   TEM = TAB-EMISSION(@PLAC)
   TRE = TAB-RECEPTION(@PLAC)

3. Soit S : accès-site telque TPR[NOM(@P)] = NOM(@S)
   % S est le site de placement du producteur P %

4. **Production locale :**
   Si local (DA, P, E) alors
   % Le producteur et l'ensemble indexé sont placés sur le même site %
   Exécuter la post-condition de l'opération de la production : φ_4 local (post(production)) sur le site S ⇒
   prod-EI (IND(@E), M)

5. **Production à distance :**
   Si non local (DA, P, E) alors
   % Le producteur et l'ensemble indexé ne sont pas placés sur le même site %

   (a) La recherche du site de placement de l'ensemble indexé en consultant la table TCO.
   ∃ S_k : accès-site telque s_k = NOM(@S_k) ∧ TCO[NOM(@E)] = s_k

   (b) La recherche du port d'accès de cet ensemble indexé en consultant la table TRE.
   ∃ PR : accès-port telque pr = NOM(@PR) ∧ TRE[NOM(@EI)] = pr.

   (c) La recherche du port d'émission du site S qui est connecté à PR, en consultant l'ensemble RESEAU de l'application DA et les ports d'émission de site S : EMISSION (@OBJ(@S)).
   ∃ PE : accès-port telque pe = NOM(@PE) ∧ PE ∈ EMISSION (@OBJ(@S)) ∧
   (pe,pr) ∈ RESEAU (@DA)

   (d) L'exécution de la post condition : φ_4 réparti (post(produire)) si la pré condition de l'opération de transmission est satisfaite. (cas de production à distance)
   Si pré-transmettre (DA, (pe,pr)) alors
   { Les autres pré-conditions sont déjà satisfaites %

   mettre (PORT(@PE), M)
   transmettre (DA, (pe,pr))
   enlever (PORT(@PR))
   prod-EI (IND(@E), prendre(PORT(@PR)))

   Fin si

   Fin si

Les procédures d'exécution des opérations : consommation et message-à-consommer

En suivant les mêmes étapes, l'exécution de l'opération de consommation (resp : message-à-consommer) de l'ensemble indexé E déclenchée par le consommateur P est réalisée de la manière suivante :

1. Soit DA(PLAC, PROC,...) : application-répartie, P : accès-processus,
   E : accès-ensemble-indexé telque
   P ∈ PROC ∧ NOM(@E) ∈ ENTREE(@P).
   % P est un producteur de E %

2. TPR = TAB-PROC(@PLAC)
   TCO = TAB-COM(@PLAC)
   TEM = TAB-EMISSION(@PLAC)
   TRE = TAB-RECEPTION(@PLAC)

3. Soit S : accès-site telque TPR[NOM(@P)] = NOM(@S)
   % S est le site de placement de processus P %

4. **Consommation locale :**
   Si local (DA, P, E) alors
   % Le consommateur et l'ensemble indexé sont placés sur le même site %
   Exécuter la post-condition de l'opération de consommation φ_4 local (post(consommation)) (resp : de message-à-consommer φ_4 local (post(message-à-consommer))) sur le site S si la pré-condition de cette opération est vrai ⇒
   Si pré-cons-EI (IND(@E)) alors

   consommer post-cons-EI (IND(@E))

   message-à-consommer val-cons-EI(IND(@E))

   Fin si

5. **Consommation à distance :**
   Si non local (DA, P, E) alors
   % Le consommateur et l'ensemble indexé ne sont pas placés sur le même site %

   (a) La recherche du site de placement de l'ensemble indexé en consultant la table TCO.
   $\exists S_k$: accès-site telque s_k = NOM(@S_k) ∧ TCO[NOM(@E)] = s_k.

   (b) La recherche du port d'accès à cet ensemble indexé en consultant la table TEM,
   ∃ PE : accès-port telque pe = NOM(@PE) ∧ TEM[NOM(@E)] = pe.

   (c) La recherche du port de réception de site S qui est connecté à PE, en consultant l'ensemble RESEAU de l'application répartie DA et les ports de réception de site S : RECEPTION (@OBJ(@S)).
   ∃ PR : accès-port telque pr = NOM(@PR) ∧ PR ∈ RECEPTION (@OBJ(@S))
   (pe,pr) ∈ RESEAU (@DA)

   (d) L'exécution de la post condition de l'opération de communication :
   φ_4 réparti (post(consommer)) (resp : φ_4 réparti (post(message-à-consommer))) si la pré condition de l'opération de transmission est satisfaite, (cas d'une opération répartie)
   Si pré-transmettre (DA, (pe,pr)) ∧
   pré-cons-EI (IND(@E)) alors
   % Les autres pré-conditions sont déjà verifiées %

   consommer

post-cons-EI (IND(@E))
message-à-consommer
M = val-cons-EI (IND(@E))
mettre (PORT(@PE), M)
transmettre (DA, (pe, pr))
enlever (PORT(@PR))

Fin si

Fin si

Généralisation

La représentation des opérations de production, de consommation et "message-à-consommer" dans le cas d'une topologie **divergence** ou **concentration** *n'est pas différente de la représentation dans le cas d'une topologie bipoint.*

Cas de diffusion

La représentation de l'opération de consommation (resp : message-à-consommer) d'un objet de topologie diffusion est la même que celle de la topologie bipoint car chaque consommateur consomme d'un ensemble indexé de l'objet de communication. L'opération de production est la composition de plusieurs opérations de production de même message dans les différents ensembles indexés de l'objet. Les différents ensembles indexés ne sont pas tous placés sur le même site, par conséquent l'exécution d'une opération de production sur un objet de topologie diffusion est une exécution répétée de la procédure d'exécution de l'opération de production sur un objet de topologie bipoint, citée précédement. Cette procédure sera répétée autant de fois qu'il y a d'ensembles indexés dans l'objet.

Procédure prod-diff

- Si OBJ : objet-composé telque TOPO(OBJ) = diffusion alors
  - Pour chaque EI $\in$ COMPOS(OBJ) faire
    - Appliquer la procédure de la production dans le cas de topologie bipoint en considérant le même message produit.
  - Fin pour
- Fin si

Fin

Cas de convergence

La représentation de l'opération de production n'est pas différente de sa représentation dans

le cas de bipoint car chaque producteur produit dans un ensemble indexé, de l'objet de communication, qui lui est propre. Le consommateur choisit l'ensemble indexé duquel il veut consommer avant d'exécuter l'opération de consommation, par conséquent la représentation de cette opération n'est pas différente de sa représentation dans le cas de bipoint.

5.2.3 Exemples

Nous allons montrer la démarche de l'exécution des proécedures citées en 2.2 sur quelques opérations de communication de l'application répartie (DA2(PLAC, PROC,...)) donnée au (chapitre 3, figure 2).

Production locale

L'exécution de l'opération de production d'un message (M) dans l'objet de communication N1.

1. TPR = TAB-PROC(@PLAC)
   TCO = TAB-COM(@PLAC)
   TEM = TAB-EMISSION(@PLAC)
   TRE = TAB-RECEPTION(@PLAC)

2. N1 : objet-composé telque TOPO(N1) = diffusion ⇒
   appliquer la procédure de la production dans le cas de diffusion.
   % Le choix de la procédure à appliquer %

3. (A1N1 : accès-ensemble-indexé telque A1N1 ∈ COMPOS(N1)) ∧
   (P1 : accès-processus telque P1 ∈ PROC ∧ NOM(@A1N1) ∈ SORTIE(@P1)) ⇒
   P1 est un producteur de A1N1.
   % La déduction de processus concerné par l'opération %

4. NOM(@P1) = np_1 ∧ TPR[np_1] = s_1 ∧ (∃ S_1 : accès-site telque NOM(@S_1) = s_1) ⇒
   S_1 est le site du placement de P1.
   % La déduction de site de placement de processus producteur %

5. TPR[np_1] = s_1, TCO[a1n1] = s_1 ∧ TPR[np_1] = TCO[a1n1] ⇒ Production locale.
   % La précision de la nature de l'opération "locale ou à-distance" %

6. Pour réaliser une opération de production, il suffit d'ajouter le message produit (m) dans l'ensemble (A1N1) localement car la pré-condition est déjà satisfaite.
   prod-EI (IND (@A1N1), M)

7. Les mêmes étapes doivet être appliquées pour le composant (A2N1).

Production à distance

L'exécution de la production d'un message dans l'ensemble-indexé AF1 :

1. P1 : accès-processus, AF1 : accès-ensemble-indexé telque
   P ∈ PROC ∧ NOM(@AF1) ∈ SORTIE(@P1) ⇒
   P1 est un producteur de AF1.

2. NOM(@P1) = np_1 ∧ TPR[np_1] = s_1 ∧ (∃ S_1 : accès-site telque NOM(@S_1) = s_1) ⇒
   S_1 est le site du placement de P1.

3. TPR[np_1] = s_1, TCO[af_1] = s_2 ∧ TPR[np_1] ≠ TCO[af_1] ⇒ Production réparti.

4. ∃ S_2 : accès-site telque NOM(@S_2) = s_2 ∧ TCO[af_1] = s_2 ⇒
   S_2 est le site du placement de AF1.
   % Le recherche de site de placement de l'accès-ensemble-indexé %

5. ∃ PR_{10} : accès-port telque NOM(@PR_{10}) = pr_{10}∧ TRE[af_1] = pr_{10} ⇒
   PR_{10} est le port de réception de AF1.
   % La recherche de port de réception associé à l'accès-ensemble-indexé %

6. ∃ PS_9 : accès-port telque NOM(@PS_9) = ps_9 ∧ PS_9 ∈ EMISSION (@OBJ(@S_1)) ∧ (ps_9, pr_{10}) ∈ RESEAU (@DA1) ⇒
   (ps_9, pr_{10}) est la liaison utilisé pour la transmission du message produit.
   % La recherche de la liaison reliant le port de réception à un port d'émission de site de placement du processus %

7. Réalisation de l'opération de production :
   Pour réaliser l'opération de la production, il suffit que la pré condition de l'opération de transmission soit vrai, car les autres conditions sont déjà verifées.
   Si pré-transmettre alors

   mettre (PS_9, M)

   transmettre (DA1, (ps_9, pr_{10}))

   enlever (PORT(@PR_{10}))

   prod-EI (IND(@AF1), prendre(PORT(@PR_{10})))

Consommation à distance

La consommation d'un message de A2N1 :

1. Q : accès-processus, A2N1 : accès-ensemble-indexé telque
   Q ∈ PROC ∧ NOM(@A2N1) ∈ ENTREE(@Q) ⇒
   Q est un consommateur de A2N1.

2. NOM(@Q) = nq ∧ TPR[nq] = s_2 ∧ (∃ S_2 : accès-site telque NOM(@S_2) = s_2) ⇒
   S_2 est le site de placement de Q.

3. TPR[nq] = s_2, TCO[a2n1] = s_1 ∧ TPR[nq] ≠ TCO[a2n1] ⇒ Consommation à distance.

4. ∃ S_1 : accès-site telque NOM(@S_1) = s_2 ∧ TCO[a2n1] = s_1 ⇒
   S_1 est le site de placement de A2N1.

5. ∃ PS_3 : accès-port telque NOM(@PS_3) = ps_3∧ TEM[a2n1] = ps_3 ⇒
   PS_3 est le port d'envoi de A2N1.

6. ∃ PR_4 : accès-port telque NOM(@PR_4) = pr_4∧ PR_4 ∈ RECEPTION (@OBJ(@S_2)) ∧ (ps_3, pr_4) ∈ RESEAU (@DA1) ⇒
   (ps_3, pr_4) est la liaison utilisée pour la transmission du message à consommer.

7. Réalisation de l'opération de consommation :
   Pour réaliser l'opération de la consommation, il faut que les pré-conditions :
   pré-transmettre et pré-cons-EI (IND(@A2N1)) soient vrais, car les autres conditions sont déjà verifées.
   Si pré-transmettre (DA1), (ps_3, pr_4)) $\wedge$
   pré-cons-EI (IND(@A2N1)) alors
   % Les autres pré-conditions sont déjà verifiées %

   finir (IND(@A2N1))

Conclusion

A l'opposé de la fonction de placement qui est une fonction heuristique, la dérivation de l'application répartie corréspondant à un placement donné est une dérivation méthodologique et automatique.

Par la création de l'application répartie et la représentation des opérations de communication dans l'environnement réparti nous satisfaisons l'objectif de notre étude : l'implantation d'applications parallèles sur une architecture répartie.

La réalisation pratique de la fonction heuristique de placement et de la fonction de création de l'application répartie sont faites sur la machine T-NODE par un système assistant nommé **COMPIL**. Ce système génére, à partir de la spécification de l'application parallèle, une application répartie exécutable sur la machine T-NODE. Le système COMPIL est décrit dans la partie 3.

Partie III

L'application de l'étude sur une architecture réelle

Nous avons défini dans la partie précedente la fonction d'implantation d'une application parallèle sur une architecture répartie. Lors de cette définition, l'application parallèle et l'architecture répartie sont décrites par leur spécification. En conséquence les contraintes imposées par le langage de programmation et celles imposées par l'architecture physique ne sont pas considérées.

L'objectif de cette partie est de **décrire la réalisation pratique de la fonction d'implantation sur la machine T-NODE.**
L'utilisation de la machine T-NODE impose des contraintes liées : au langage de programmation, à l'environnement logiciel du travail et à l'architecture physique de la machine. Ces différentes contraintes seront considérées dans cette partie de la thèse.

La fonction d'implantation est réalisée sur la machine T-NODE par un système nommé COMPIL. Ce système génére, à partir de la spécification d'une application parallèle, une application répartie correspondant à la première et exécutable sur la machine T-NODE.

Pour décrire le système COMPIL nous allons développer dans cette partie :

- La description de la machine T-NODE de point de vue matériel et logiciel (chapitre 6).
- Une description de l'**image exécutable de l'application répartie sur la machine T-NODE** : cette image est le résultat de système COMPIL (chapitre 7).
- Une description de système COMPIL : la réalisation de la fonction d'implantation sur la machine T-NODE (chapitre 7).

Chapitre 6

L'environnement de la réalisation

Introduction

La réalisation pratique de ce travail a été faite sur la machine T-NODE, qui est une machine parallèle à base de transputers.
Nous allons présenter dans ce chapitre, les caractéristiques principales de cette machine et de son environnement logiciel de travail.

6.1 Description de la machine T-NODE

La machine T-NODE est développée dans le cadre de projet Esprit P1085, par la société Telmat Informatique. L'objectif de projet était : la création d'une machine parallèle à configuration modulaire et d'une grande puissance de calcul scientifique, et le développement des outils logiciels permettant une utilisation simple et éfficace de la machine.

Le T-NODE est une machine parallèle composée d'un réseau de transputers et d'un composant électronique nommé " switch ". La caractéristique principale de cette machine est la possibilité de la reconfiguration de la topologie du réseau de transputers par l'intermédiaire du switch.

6.1.1 Les caractéristiques de la machine utilisée

L'expérimentation a été développée sur la configuration suivante :

- un ordinateur frontal STE30 fonctionnant sous Unix(MPIX Telmat),
- une carte d'interface STE30/T-NODE (**ITFTP32**), contenant un transputer T414 et 4 M octets, mono utilisateur;
- le T-NODE en lui-même, soit pour ce modèle (TN352) :
  - 32 transputers T800-20 Mhz, possédant chacun une mémoire locale de 1 M octets (4 cartes de 8 transputers),
  - un switch reliant les 32 transputers entre eux, géré par un T800 (le controleur) et une Eprom (T-Kernel),

- un transputer T800 gérant une mémoire partagée de 16 Moctets,
- un transputer T800 gérant un disque global et une mémoire de 16 Moctets.

6.1.2 La description des principaux composants de la machine T-NODE

Le transputer Le transputer est l'élément fondamental du T-Node. Il a été conçu en vue d'assurer la concurrence d'exécution des opérations. Cette concurrence est complétement exploitée lors de l'utilisation du langage de programmation OCCAM. Le transputer est un microprocesseur qui posséde une mémoire locale et quatre liens bidirectionnels permettant sa connexion avec d'autres transputers. C'est un circuit VLSI, RISC, fabriqué par **Inmos.**

Il existe plusieurs types de transputers, ces types se différencient par l'horloge interne, la taille mémoire, le processeur qui les contrôle,...etc. Les quatre types de transputers utilisées dans la machine T-NODE sont : T212, T414, T800, M212.

D'une manière générale, le transputer est un " chip " composé d'un processeur, d'une mémoire, de quatre connexions permettant la communication : bipoint, bidirectionnelle, en série et synchrone entre les transputers. De plus chaque transputer posséde une interface ou un circuit spéciale adapté à son utilisation. Figure 1, montre l'architecture du transputer.

Pour assurer la concurrence, le transputer posséde au niveau de son micro-code un ordonnanceur qui partage le temps d'accès au processeur entre les différents processus exécutés sur le transputer.

Dans la machine T-NODE nous distinguons deux sortes de transputers : les transputers de travail, utilisés pour l'exécution des programmes d'utilisateurs; et les transputers de contrôle utilisés par le système.

Le transputer peut être utilisé comme un seul processeur, ou dans un réseau de transputers constituant ainsi un système parallèle, figure 2.

Le switch L'intérêt de T-NODE par rapport aux autres machines paralléles tient dans le fait que sa topologie est reconfigurable. Ceci est possible grâce au switch qui permet, en utilisant les propriétés des cycles Eulériens, de connecter les transputers selon un réseau prédéfini (hypercube, arbre, ...etc). Tous les liens des transputers de travail sont connectés au switch qui est contrôlé par un transputer de contrôle. Ces deux organes, le switch et le transputer de contrôle permettent la configuration de réseau de transputers selon une topologie quelconque à condition de connecter chaque transputer avec quatre autres au maximum.

Le switch a été conçu pour que la reconfiguration du réseau soit statique (modification de réseau avant une exécution), pseudo dynamique (modification de réseau avant et durant l'exécution), ou dynamique (modification de réseau durant l'exécution); néanmoins seule la reconfiguration statique est possible actuellement.

6.1.3 Le système d'exploitation

Le système d'exploitation de la machine T-NODE est HELIOS. L'objectif de sa conception était de fournir un système qui, tant au niveau des commandes qu'au niveau de la programmation soit familier aux utilisateurs de Unix.

L'approche multi-tâches de HELIOS est complètement similaire aux concepts qui ont prévalu à la conception du transputer. HELIOS reconnait deux types d'objets. Le premier est appelé process. Un process partage la mémoire avec les autres process et la commutation d'un process à un autre est très rapide. Le second type d'objet est appelé tâche. Une tâche a un nom et est créée par un appel système à HELIOS. Des tâches différentes ne peuvent pas partager de la mémoire commune. Chaque tâche sous HELIOS peut être soit l'exécution d'un programme pour un utilisateur ou bien une tâche système.
La réalisation des communications sur le transputer est telle qu'un message peut ne pas être transmis mais s'il est transmis : il est correct. HELIOS réalise un contrôle de transmission par une technique de chien de garde pour chaque message mais ne réalise aucun contrôle sur son contenu.

HELIOS s'appuie sur le modèle client serveur qui est largement utilisé dans de nombreux systèmes d'exploitation. Bien que caché par de nombreuses couches de logiciels, la communication entre les processeurs est du type transfert de messages.
La conception interne d'HELIOS utilise ce modèle client-serveur mais avec la différence que les process qui représentent le client et le serveur peuvent résider sur des processeurs différents. Les messages seront transmis d'une tâche à l'autre par la mémoire ou bien transmis au processeur voisin à travers un lien.
Dans chaque cas, le client réalise toujours le même appel au serveur, c'est le processus interne de transmission qui passera le message par la mémoire au process dans la même machine ou bien qui enverra ce message à travers un nombre quelconque de transputers pour atteindre le destinataire final. La localisation du destinataire est inconnue de l'émetteur de même que le chemin d'accès à celui-ci. Le routage du message est donc totalement transparent pour les clients, ce qui implique que la topologie du réseau leur est aussi transparente.

HELIOS est implanté comme un vrai système distribué. Chaque processeur, nœuds dans la topologie, contient le noyau d'HELIOS qui gère la mémoire et le mécanisme de communication. Chaque nœud contient aussi deux serveurs : le gestionnaire du processeur et le chargeur. Le gestionnaire du processeur est chargé de la création des process localement à ce processeur et des tâches de maintenance de son état tandis que le chargeur effectue les chargements et déchargements des programmes et des librairies résidentes lorsqu'elles sont requises.
Chaque calculateur sous HELIOS exécute un gestionnaire de réseau, qui réalise l'initialisation du réseau. Ce serveur utilise un fichier qui décrit les ressources disponibles sur le réseau.

Il est possible de disposer dans un réseau HELIOS de processeurs qui n'utilisent pas HELIOS. En effet la plupart des réseaux de transputers sont généralement connectés à un autre système qui le calculateur hôte. Plus généralement, n'importe quel réseau HELIOS peut contenir un nombre quelconque de nœuds transputer ainsi que de nœuds système hôte.

L'algorithme de Bokhari est l'algorithme utilisé actuellement dans HELIOS pour la répartition de process et de tâches d'un programme parallèle sur les transputers. Cette information nous l'avons eu suite à une disscusion avec les ingénieurs système de la société TELMAT.

Précisons que le système HELIOS n'est pas disponible sur la machine T-NODE du CRIN. Nous n'avons donc pas pu bénéficier des avantages d'HELIOS lors de la réalisation de notre système.

6.2 Les environnements logiciels de la machine T-NODE

Trois langages de programmation parallèle sont disponibles sur la machine T-NODE. Ces langages sont : Occam, C-parallèle et Fortran-parallèle. Le langage Occam " le langage de transputer " a été créé spécialement pour exploiter le maximum de parallélisme du transputer. Les deux autres langages sont les C et Fortran standard, sur lesquels sont diférées des couches supplémentaires pour assurer la communication entre les tâches et gérér le parallélisme. Il existe une très grande analogie entre les coucnes supplémentaires de ces deux langages, nous nous limitons dans le cadre de cette thèse à la présentation de la couche de C-parallèle, qui est le langage utilisé dans notre système, et aux aspects de base de langage Occam pour justifier notre choix.

L'utlisateur de la machine T-NODE dispose d'un environnement de travail TNT (T-NODE TOOLS) qui facilite le développement et l'exécution de ses programmes. L'environnement TDS est utilisé pour les programmes écrits en OCCAM, et l'environnement 3l est utilisé pour les programmes écrit en C-Parallèle ou en Fortran-Parallèle.

6.2.1 La programmation en OCCAM

Le langage de programmation Occam

Occam est un langage de programmation créé par Inmos pour son transputer. Il permet l'expression de la concurrence et des liens entre les tâches d'un programme parallèle.

C'est un langage très simple qui repose sur un minimum d'instruction et sur le modèle de langage impératif à structure de blocs. Son originalité réside dans les primitives de communication et les constructeurs de parallélisme.

Les primitives d'Occam :

- L'affectation simple ou multiple.
- Les primitives de communication : Les messages sont transmis entre deux processus par l'intermédiaire d'un canal. Chaque canal assure une communication unidirectionnelle, bipoint et sans mémorisation de message (par rendez-vous). Un protocole est associé à chaque canal permettant de spécifier le type et le format de ses messages.
  La communication est assurée par des primitives d'envoi et des primitives de réception de message. Comme la communication se fait par rendez-vous, ces primitives sont bloquantes.
  { L'envoi de valeur de la variable var1 sur le canal can1 s'écrit en Occam : can1 ! var1.
  La réception d'une valeur de canal can1 et l'affectation de cette valeur à la variable var1 s'écrit en Occam : can1 ? var1}

Les constructeurs d'Occam : Un programme d'Occam est une composition de processus, chaque processus est la combinaison de processus plus élémentaires par un des constructeurs suivants :

- Le constructeur de séquencement **SEQ** : les processus combinés par ce constructeur sont exécutés séquentiellement.

- Le constructeur conditionnel **IF**, le constructeur de sélection **CASE**, et le constructeur d'itération **WHILE**.

- Le constructeur de parallélisme **PAR** : c'est le constructeur du langage Occam permettant la combinaison parallèle de processus concurrents.
  Pour assurer l'exclusion mutuelle, le compilateur vérifie qu'il n'y a pas de conflit d'accès aux variables partagées.

- Le constructeur d'altération **ALT** : il combine un ensemble de processus guardés par des conditions ou par des primitives de communication. Et il exécute le processus associé au guard satisfaisable. Si plusieurs guardes sont possibles, le choix de processus à exécuter est aléatoire.
  Ce constructeur permet le multiplexage et la création d'un mécanisme de lecture non bloquant.

- Le constructeur de priorité PRI : c'est un constructeur permettant d'associer des priorités aux différents processus qui sont combinés par le constructeur ALT ou par le constructeur PAR.

Les éléments manipulés par Occam Les éléments manipulés par Occam sont les variables, les canaux et les timers. Les canaux transmettent des valeurs, et les timers produisent des valeurs qui représentent le temps.

Les types des variables sont : booléen (**BOOL**), octet (**BYTE**), les entiers (**INT, INT16, INT32, INT64**), des réels (**REAL32, REAL64**) et les chaînes de caractères.
Il existe un seul **type complexe** en Occam : le **tableau.**
Les canaux sont des éléments spécifiques au Occam, chaque canal respecte un certain protocole, ce protocole peut être :

- un **protocole simple** (INT, ANY, BYTE,...etc) : le canal reçoit toujours des valeurs de même type,

- un **protocole séquentiel** : le canal reçoit toujours une suite de valeur chacune d'un type donné,

- un **protocole variant** : le format de message reçu sur le canal est variable.

Le timer produit des valeurs représentant le temps, c'est un élément partagé accessible par des processus concurrents.

Les fonctions et les procédures : Les procédures en Occam sont identiques à celles de Pascal.
Les fonctions ne prennent que des paramétres passées par valeurs et peuvent rendre un ensemble de valeurs.

La structure d'un programme Occam : Un programme Occam est la combinaison d'un ensemble de processus par les constructeurs d'Occam. Les processus ont une structure hiérachique,

un processus peut être une seule instruction, un ensemble d'instructions exécutées séquentiellement, ou une combinaison de processus plus élémentaires.

Un **programme Occam** peut être exécuté sur un réseau de transputers. **Pour pouvoir le placer sur les différents transputers, il faut le diviser en plusieurs procédures, chacune regroupant les processus s'exécutant sur le même transputer. De plus chaque procédure est paramétrée par les canaux de communication avec l'extérieur.**

La création de telles procédures oblige l'utilisateur à les modifier lorsqu'il veut modifier le placement de son programme sur le réseau de transputer. Ceci alourdit le placement et rend l'**affectation des tâches aux transputers explicite.**

Il faut noter que la notion de procédure en Occam n'a pas été introduite pour ce but uniquement, et que plusieurs procédures peuvent être exécutées sur le même transputer mais dans ce cas il faut créer une autre procédure qui les regroupe.

Nous remarquons que **le programmeur doit créer en plus de son programme logique, une couche supplémentaire pour pouvoir placer les tâches du programme sur le réseau de transputers.**

La gestion du parallélisme en Occam : Sur un réseau de transputers nous distinguons deux sortes de parallélisme :

- Le **parallélisme virtuel** : c'est un parallélisme entre les processus s'exécutant sur le même transputer. Dans ce faux parallélisme, l'ordonanceur de transputers partage le temps d'accès au processeur pour exécuter les différents processus.
- Le **parallélisme réel** : c'est un parallélisme entre les processus s'exécutant sur différents transputers.

La gestion du parallélisme virtuel est transparente : Elle est assurée par le constructeur PAR qui créé un processus pour chaque instruction contenue dans son bloc.
Le parallélisme réel est explicite : Il est exprimé dans un fichier de configuration qui indique pour chaque transputer utilisé : la procédure associée, et les canaux associés à chacune des liens physiques.

La configuration d'un programme Occam :

La configuration consiste à associer les composants d'un programme logique à l'ensemble des ressources physiques.
Les processus constituant le programme sont distribués sur les unités de traitement disponibles dans l'environnement dans lequel le programme s'exécute (les transputers pour le T-NODE). Les canaux reliant les différents processus distribués sont affectés aux liens physiques reliant les différents transputers.
La configuration n'affecte pas la logique du programme à condition que celui ci soit constitué d'une liste de procédures communiquant par canaux. Ainsi les processus associés à ces procédures combinées par le constructeur PAR peuvent s'exécuter chacun sur un processeur dédié.

Deux instructions en Occam permettent le placement des unités logiques sur les unités physiques :

- l'instruction **PLACED PAR** qui affecte une procédure à un processeur spécifique,
- l'instruction **PLACE** canal **AT** canal-physique : qui associe un canal logique reliant deux processus à un lien physique reliant deux transputers.

Le fichier de configuration d'un programme parallèle doit préciser, en plus de l'association, certains caractèristiques des ressources physiques : le type de transputer et le numéro de chaque lien physique utilisé par le programme parallèle.

L'environnement logiciel TDS (Inmos Transputer Developpement System)

TDS est un outil logiciel permettant le développement, la compilation d'un programme parallèle écrit en OCCAM ainsi que la configuration, le chargement et l'exécution du programme sur un réseau de transputers.

L'environnement logiciel de TDS est composé d'un ensemble d'utilitaires, chacun est spécifique à une fonction particulière. Parmi ces utilitaires nous citons les plus importants :

- **T-Configurer** : Son rôle est de réaliser la configuration d'un programme exécutable sur T-NODE et de fournir des informations, nécessaires, sur la configuration de la machine T-NODE pour l'exécution du programme.
- **T-Switcher** : Il est utilisé pour la configuration de la machine T-NODE. L'utilisateur fournit le T-Switch par le programme configuré et les informations de configuration déduites par T-Confiurer. T-Switcher génére et envoie les commandes de connexion des transputers au T-Kernel.
- **T-Loader** : Il est utilisé pour charger et exécuter un programme sur un réseau de transputers déjà configuré.
- **T-Kernel** : Il contrôle le réseau de transputers lors de l'exécution d'un programme, et il administre : l'arrêt des processus, le switching et la communication en série.

6.2.2 La programmation en C-Parallèle

Le langage de programmation C-parallèle :

Ce langage est constitué de C-standard auquel a été diféré : une couche permettant d'utiliser le parallélisme et d'exprimer la communication entre les tâches, et un outil permettant l'implantation des exécutables C-parallèle sur un réseau.

Ce langage (comme Fortran parallèle) est basé sur le modèle CSP (Communicating Séquential Processes) dans lequel un programme parallèle est composé d'un ensemble de tâches séquentielles communicants. Les tâches se communiquent par l'intermédiaire des canaux et leur communication est par rendez-vous, donc : synchronisée et bloquante. Les canaux sont unidirectionnels, bipoints.

Chaque tâche comporte un espace code, un espace données, un vecteur de ports d'entrées et un vecteur de ports de sortie. Ce modèle de programme est adapté au réseau de transputers où deux tâches communicantes peuvent être allouées sur le même transputer ou sur des transputers

distincts. Et selon leur allocation les canaux qui les relient sont des canaux logiques, si elles sont allouées au même transputer (nombre illimité), ou des liens physiques, si elles sont sur deux transputers distincts (nombre limité).

La couche supplémentaire Cette couche permet la gestion de communication entre tâches, de réaliser l'exclusion mutuelle pour l'accès aux variables partagées et l'attente multiple ou l'équivalence de constructeur ALT.

- **La gestion de communication :** Cela est satisfait par l'introduction des canaux (channel) et des primitives d'envoi et de réception des message. Ces primitives sont dans la biblothèque (chan.h).

  Dans ce langage : un canal de communication est de type **CHAN**, une des primitives d'envoi de message est **chan_out_word**, et une des primitives de réception de message est **chan_in_word**.

- **L'exclusion mutuelle :** La résolution du problème de conflit d'accès aux variables partagées a été réalisé par l'introduction de sémaphore et des primitives d'attente, d'entrée et de libération de section critique, (bibliothèque sema.h).

  Dans ce langage un sémaphore est de type **SEMA**, et les principaux primitives d'utilisation de sémaphore sont :

  - **sema_init** : l'initialisation d'un sémaphore,
  - **sema_signal** : exécuter l'opération " signal " sur un semaphore,
  - **sema_wait** : exécuter l'opération " attendre " sur un semaphore,

- **L'attente multiple :** Réalisation du constructeur ALT d'Occam : attente sur plusieurs conditions et plusieurs signaux, le premier disponible se l'approprie avec indéterminisme. Cette réalisation est faite par la création d'une ou de plusieurs " **thread** " dans une tâche, (biblothèque thread.h).

- **La gestion du temps :** Il existe quelques primitives permettant de connaître la valeur de l'horloge des transputers, de comparer les horloges des transputers, et d'attendre un certain délai.

La structure d'un programme en C-parallèle Un programme écrit en C-parallèle est un ensemble de tâches, chaque tâche s'exécute séquentiellement et elle communique avec les autres tâches par l'envoi et la réception de messages. Les différentes tâches sont écrites indépendamment, un fichier par tâche, et elles peuvent être exécutées sur le même transputer ou sur des transputers distincts, cela est précisé dans le fichier de configuration.

La gestion du parallélisme : Malgré l'absence en C-parallèle de l'équivalent de constructeur PAR d'Occam, la notion du parallélisme virtuel existe, et ce parallélisme est géré par le système lors de l'exécution de deux tâches concurrentes sur le même transputer.

Le fichier de configuration d'un programme C-parallèle

Le rôle de ce fichier est multiple :

1. Déterminer les différentes tâches (modules) du programme parallèle et préciser le nombre de canaux d'entrée et le nombre de canaux de sortie de chaque tâche (aspect logique).
2. Décrire la topologie du programme parallèle en précisant les liens entre les différentes tâches (aspect logique).
3. Préciser le nombre de transputers (ou processeurs) utilisés pour l'exécution du programme (aspect physique).
4. Le placement des tâches sur les transputers.
5. L'association de certains canaux logiques du programme aux liens physiques des transputers utilisés. Cette association peut être déduite par le configurateur.

Il faut noter que plusieurs liens logiques entre les tâches de deux transputers peuvent corréspondre au même lien physique reliant les deux transputers.

L'environnement de programmation 3l

L'environnement 3l est un logiciel pour l'utilisation des langages parallèles : C-Parallèle, Fortran-Parallèle et Pascal-Parallèle sur la machine T-NODE.

La création d'un programme séquentiel exécutable nécessite :

1. l'édition du texte de programme,
2. la compilation,
3. l'édition des liens,
4. l'éxécution.

La création d'un programme parallèle composé de plusieurs tâches et exécutable sur un transputer ou sur un réseau de transputers non-reconfigurable nécessite en plus des trois premières étapes précedentes :

- La création d'un fichier de configuration indiquant :
  - les liens entre les différents tâches du programme,
  - le placement des tâches sur les transputers, et des canaux logiques sur les liens physiques.
- La création d'un programme exécutable sur un réseau de transputers.

La machine T-NODE est une machine dont le réseau est reconfigurable. Pour pouvoir exécuter un programme parallèle sur une telle machine, il faut configurer le réseau physique selon la configuration du programme logique. Dans l'environnement 3l, la configuration de réseau de transputers est réalisée par l'intermédiaire de deux utilitaires : T_Config et T_Switch.

- **T_Config** : Il lit le fichier de configuration (fich.cfg) et il crée deux fichiers (fich.wdg, fich.nwk) contenant des ordres de connexion physiques des transputers et l'utilisation de leur liens.
- **T_Switch** : Il lit les fichiers résultant de T_Config, transforme les ordres de connexion en instructions, et il envoie ces instructions à la carte de contrôle qui programme les "switches".

L'exécution des programmes parallèles dans l'environnement 3l nécessite la création d'un fichier make décrivant les commandes : de compilation, d'édition de lien, de configuration et la génération de module chargable et exécutable sur cette machine.

Conclusion

Parmi les environnements du travail disponibles nous avons choisi l'environnement 3l pour réaliser notre système. Ce choix est justifié par les avantages de cet environnement par rapport à l'environnement TDS, et les avantages de langage C-Parallèle par rapport à OCCAM.

Chapitre 7

La réalisation pratique sur le Super-Node

Introduction

Nous allons décrire dans ce chapitre la réalisation pratique de notre travail sur la machine T-NONE. Cette description tient compte de l'environnement matériel et logiciel de la machine. Et elle est divisée en deux parties :

- La représentation d'une application répartie sur la machine T-NODE : sachant que l'image d'une application parallèle sur une architecture répartie est une application répartie, nous allons donner la représentation de l'application répartie dans l'environnement 3l : environnement du travail.
- La réalisation de la fonction d'implantation sur la machine T-NODE : le système COMPIL. Ce système génére l'application répartie exécutable sur la machine T-NODE et qui correspond à une application parallèle donnée.

7.1 L'implantation d'applications réparties

Nous avons défini au chapitre 3 l'application répartie d'un point de vue abstrait. Notre objectif ici est de définir l'implantation réelle de l'application répartie sur la machine T-NODE. Cette définition consiste à exprimer l'image de chaque composant de l'application répartie par un composant matériel ou logiciel de la machine T-NODE et de son environnement de travail. Pour la définition de cette implantation nous avons considéré d'une part :

- la spécification de l'application répartie,

et d'autre part :

- les caractéristiques de la machine T-NODE et les contraintes matérielles imposées par cette machine,
- le langage de programmation et l'environnement logiciel disponible sur la machine T-NODE.

Nous avons choisi comme environnement logiciel : l'environnement 3l; et comme langage de programmation : le langage C-Parallèle. Plusieurs raisons ont guidé notre choix :

- Le langage C est un langage portable et plus général que le langage OCCAM.
- La couche supplémentaire de C-Parallèle permet de gérér les primitives nécessaires pour l'implantation des applications réparties : la communication entre les tâches, l'exclusion mutuelle, l'attente multiple,...etc.[1]
- Le développement des programmes est plus souple dans l'environnement 3l que dans l'environnement TDS.
- La configuration des programmes parallèles est implicite dans l'environnement 3l tandis qu'elle est explicite dans l'environnement TDS. En effet la correspondance entre les liaisons logiques du programme parallèle et les liaisons physiques de la machine T-NODE est générée automatiquement dans l'environnement 3l

7.1.1 L'image exécutable d'une application répartie sur la machine T-NODE

La définition de l'image exécutable d'une application répartie sur la machineT-NODE nécessite la définition des images de ses composants et la représentation des opérations de communication dans l'environnement de travail choisi.

Comme nous allons le constater par la suite, les images des composants de l'application répartie seront déduites facilement à partir des composants de la machine T-NODE.
L'exécution des opérations de communication nécessite l'introduction d'une couche de contrôle qui gére ces opérations d'une manière transparente vis-à-vis l'utilisateur. Cette couche sera nommée **la couche de communication** . Sa génération n'est pas contrôlée par l'utilisateur qui ne s'aperçoit pas de son existence. Elle est générée par la fonction d'implantation : système COMPIL.

Les images des composants de l'application répartie

L'application répartie est une extension d'un système réparti. Elle est paramétrée par des tables de placement, des processus et les types des différents composants du système réparti. Nous allons préciser l'image de chacun de ces composants : les composants du système réparti et les paramètres de l'application répartie, sur la machine T-NODE.

L'image du système réparti : L'architecture répartie dont nous disposons est un réseau reconfigurable de 32 transputers. Chaque transputer peut être lié avec l'extérieur par quatre liens physiques bidirectionnels.

Le système réparti est une vue abstraite de l'architecture répartie, en conséquence ses composants sont représentés par des composants de cette architecture.

[1] L'absence d'une instruction équivalente à l'instruction PAR d'OCCAM n'est pas génant car le parallélisme dans nos applications est un parallélisme au niveau de processus et pas au niveau d'instruction.

Chaque **site** est représenté par un **transputer** et chaque **liaison** est représenté par un **lien physique** entre deux transputers. Le **système réparti** est représenté donc par les **transputers** et **les liens physiques** de la machine T-NODE qui **sont utilisés pour l'exécution de l'application parallèle**.

Les contraintes physiques de la machine T-NODE imposent que :

- le nombre de sites soit inférieur ou égale à 32 sites.
- chaque site posséde, au maximum, quatre liaisons bidirectionnelles, lui permettant de communiquer avec l'extérieur.

Chaque **ensemble indexé** d'un site est représenté par une **espace mémoire (tableau)** partagée par les processus ayant accès à cet ensemble. Une **bibliothèque de gestion d'ensembles indexés** est développée, sous forme de fonctions, pour réaliser les différentes opérations définies sur ce type (ajouter, finir, dernière ...etc). L'accès à un ensemble indexé est réalisé en exclusion mutuelle par l'intermédiaire de sémaphore.
Remarque : Nous allons voir dans la suite de ce chapitre que chaque ensemble indexé de l'application parallèle appartiendra, dans notre modèle d'implantation, à un processus de gestion qui gére l'accès en exclusion mutuelle et toutes les opérations sur cet ensemble. Nous pouvons dire donc que l'image de chaque ensemble indexé est un processus de gestion.

Chaque **port d'émission** d'un site est représenté par une **sortie d'un transputer**, et chaque **port de réception** par une **entrée d'un transputer**.

L'image des processus de l'application répartie : En utilisant le langage de programmation C-Parallèle, chaque **processus** de l'application répartie est représenté par un **module**. Les **entrées/sorties** du processus sont les **canaux : in_ports, out_ports** du module; et le corps de processus est une séquence d'instructions C.

Les opérations de communication (produire, consommer, message-à-consommer) sont des opérations complexes; leurs représentations seront détaillées dans d'autres paragraphes de ce chapitre.

L'image des tables de placement : Les **différentes tables de placement d'une application répartie** indiquent : le placement de composants de l'application parallèle sur les sites du système réparti et les associations entre les différents composants. En conséquence leurs images dans l'environnement 3l sont **inclues** dans le **fichier de configuration**.

- Chaque élément de la table **TAB-PROC** est représenté dans le fichier de configuration par une déclaration :
  place nom_processus nom_site.
- Chaque élément de la table **TAB-COM** est représenté dans le fichier de configuration par une déclaration :
  place nom_processus nom_site.
  L'image de chaque ensemble indexé est un processus de gestion

- Les éléments de **TAB-EMISSION, TAB-RECEPTION** ainsi que les éléments de **RESEAU** du système réparti sont représentés par les déclarations :
  connect ? nom_processus1[i] nom_processus2[j]

7.1.2 Répartition des opérations de communication : la couche de communication

La répartition des opérations de communication d'un programme parallèle sur un réseau du transputers nous a amené à introduire une **couche de communication**. Le rôle de cette couche est de **gérer les opérations de communication entre les processus de l'application parallèle. Elle satisfait ainsi les procédures d'exécution des opérations de communication**, définies dans le chapitre 5.
Cette couche a deux rôles :

- l'exécution des opérations de communication,
- l'accès aux ensembles indexés en exclusion mutuelle.

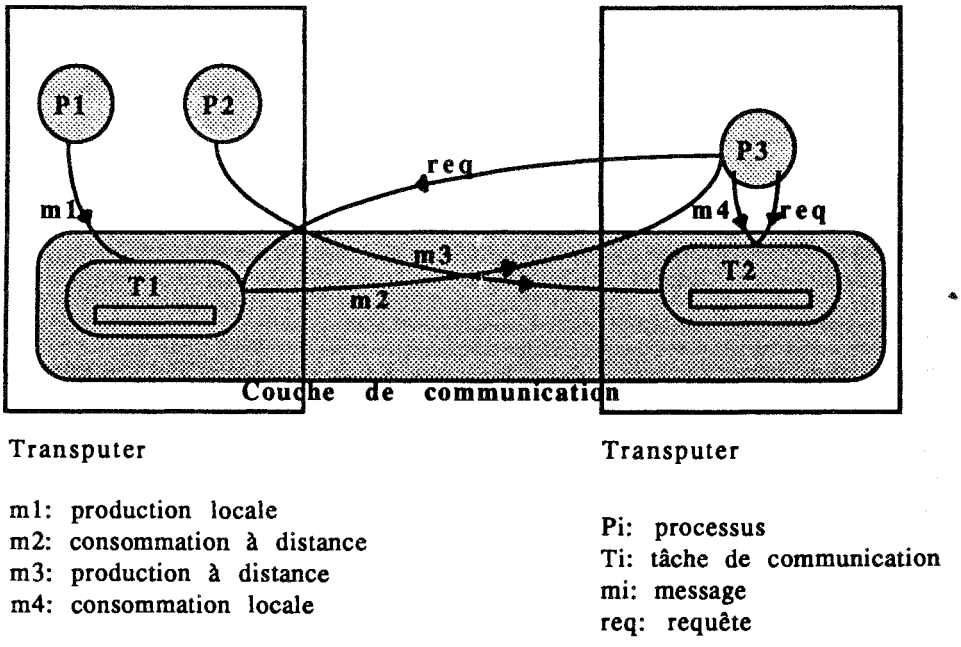
Cette couche est constituée d'un ensemble de tâches de communication distribuées sur l'ensemble des sites de l'application répartie. Elle est générée par la fonction d'implantation.
Chaque tâche gére les opérations de communication d'un ensemble indexé. Elle assure une communication asynchrone entre les processus producteurs et les processus consommateurs de l'ensemble indexé. Elle communique avec les processus ayant accès à un ensemble indexé et elle contrôle l'accès en exclusion mutuelle à cet ensemble. Chaque ensemble indexé appartient à la tâche de communication qui gére ses opérations de communication.

Les processus communicant avec une tâche de communication peuvent être sur le site de placement de la tâche ou sur un autre site. En effet la localisation des tâches de communication est transparente vis-à-vis les processus de l'application répartie.
Les différentes tâches de communication d'un site s'exécutent en parallèle avec les processus de l'application parallèle (pseudo-parallélisme). La figure 1 montre un exemple de ce schéma d'implantation.

La tâche de communication reçoit deux sortes de message : les messages produits par les producteurs et les requêtes de consommation de la part des consommateurs. Lors de la réception d'un message produit, elle mémorise ce message dans l'ensemble indexé, en satisfaisant ainsi l'opération de la production. A la réception d'une requête de consommation, elle choisit le message à consommer selon le type de communication associé à l'ensemble indexé et elle envoie ce message au consommateur demandeur, en satisfaisant ainsi les opérations : message-à-consommer et l'opération de consommation.
Malgré la communication synchrone entre les modules de C-Parallèle, il est possible d'introduire l'attente multiple dans les modules de ce langage. L'attente multiple permet d'attendre indifféremment un appel parmi plusieurs possibles. La tâche de communication utilise cette possibilité pour répondre aux différents appels de production et ceux de consommation.

Le rôle de la tâche de communication est de gérér les opérations de communication d'un ensemble indexé, en conséquence aucun calcul n'est réalisé par cette tâche.

Il faut préciser que la couche de communication générée par la fonction d'implantation dépend de l'application parallèle sur laquelle est appliquée cette fonction. D'une manière générale, la fonction d'implantation ajoute une tâche de communication par ensemble indexé. Cette tâche est alloué au site de placement de l'ensemble indexé.



Une tâche de communication par ensemble indexé

Figure 7.1

Les différents modules correspondant aux tâches de communication sont développés, à part, en langage C-Parallèle. Lors de la génération de la couche de communication, le role de la fonction d'implantation est de choisir les modules à intégrer dans la couche, et de créer le nombre d'exemplaires nécessaires de chaque module.
Nous avons développé quatre modules de communication, ces modules constituent les modules types de base de la couche de communication. Chaque tâche de communication est une image exécutable d'un de ces modules.
Un module de communication est paramétré par le nombre de canaux en entrée et le nombre de canaux en sortie, la valeur de ces paramétres est fixé par la fonction d'implantation.

Les quatres modules types de base de la couche de communication sont :

1. **Le module "bipoint"** : C'est un module qui gére les opérations de communication d'un ensemble indexé ayant un producteur et un consommateur. Ce module a deux canaux en entrée : un canal de réception de messages produits et un canal de réception de requêtes de consommation, et un canal en sortie pour l'envoi de messages au consommateur.
   Ce module est un module de base de la couche de communication car plusieurs topologies de communication l'utilisent. La figure 2-a montre le schéma de connexion d'une tâche de ce type au producteur et au consommateur de l'ensemble indexé de la tâche.

2. **Le module "multipoint"** : C'est un module qui gére les opérations de communication d'un ensemble indexé ayant "n" producteurs et "m" consommateurs. C'est une généralisation de module "bipoint". Il est paramétré par le nombre de canaux en entrée et le nombre de canaux en sortie.
   Le nombre de canaux en sortie est égale au nombre de consommateurs. Le nombre de canaux en entrée est égale au nombre de producteurs plus le nombre de consommateurs (production et requête de consommation).
   Ce module est utilisé dans le cas d'un objet de communication de topologie divergence, figure 2-b, ou de topologie concentration, figure 2-c.

3. **Le module "diffuseur"** : C'est un module de diffusion de messages. Il est utilisé dans le cas d'un objet de topologie diffusion. Il reçoit un message de son canal d'entrée et il l'envoie via les "n" canaux de sortie. La figure 2-c montre la couche de communication d'une application parallè composée d'un producteur et de n consommateurs communicant par un objet de topologie diffusion.

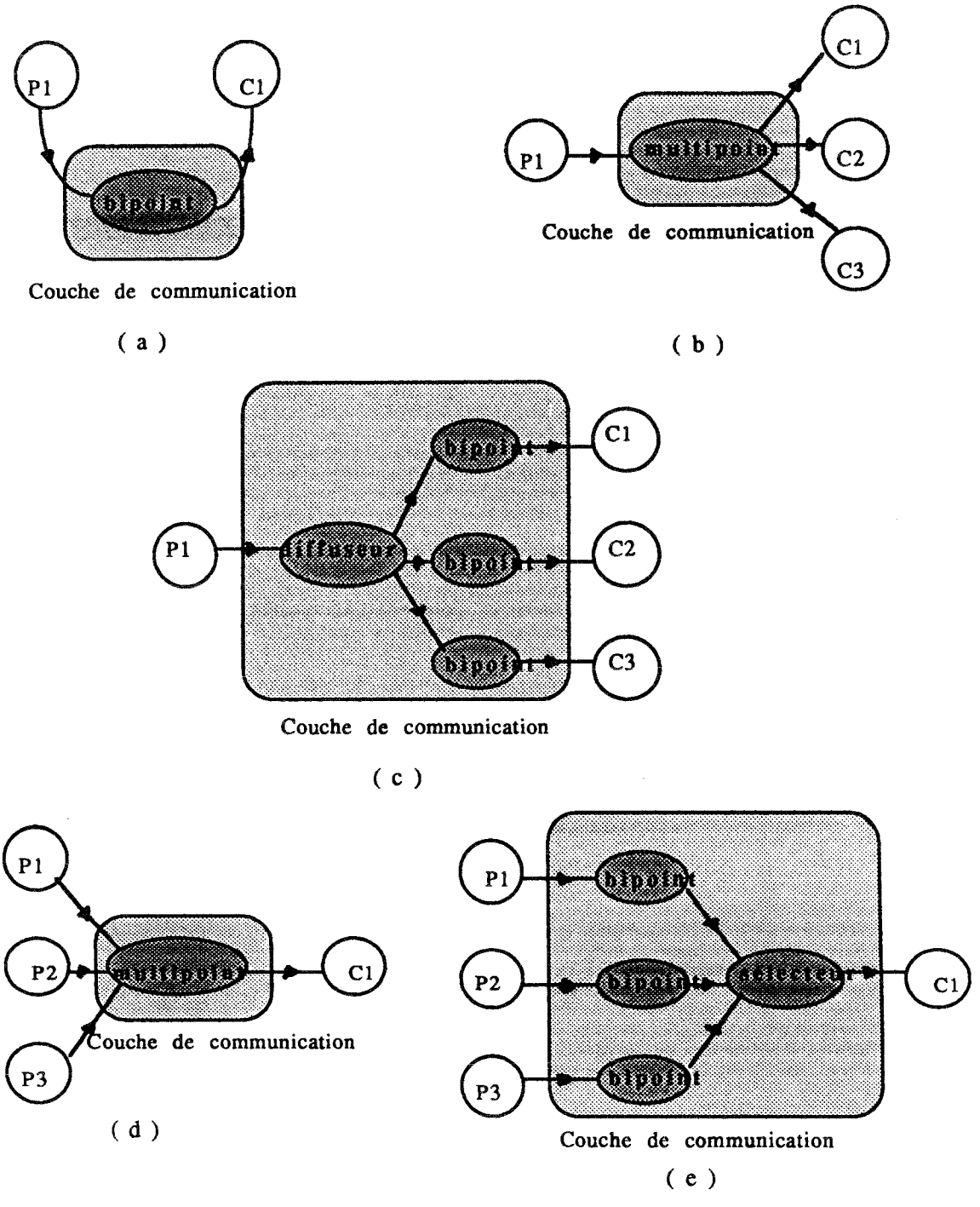
4. **Le module "sélecteur"** : C'est un module utilisé dans le cas d'un objet de communication de topologie convergence. Il sera lié au consommateur et aux différentes tâches associées aux ensembles indexés de l'objet.
   Avec chaque requête de consommation, il reçoit l'identificateur de l'ensemble indexé duquel le consommateur souhaite consommer. Selon la valeur de cet identificateur il choisit la tâche appropriée et il lui transmet la requête de consommation. Ensuite il reçoit de cette tâche un message, le message à consommer, et il le transmet au consommateur. La figure 2-e montre la couche de communication dans le cas d'un objet de communication de topologie convergence.

La programmation de ces modules est donné à l'annexe A.
La figure 2 montre les tâches de communication associées aux différents topologies d'objet de communication.
Pour générer la couche de communication d'une application parallèle, il suffit de déduire les tâches de communication associées à chaque objet de communication de l'application parallèle et d'unir ces différentes tâches.

Les différents processus de l'application parallèle sont écrits par l'utilisateur et ils ne sont pas modifiable par la fonction d'implantation.

Comme nous l'avons précisé, les opérations de communication sont exécutées par la couche de communication. Le déclenchement ou l'activation de ces opérations est fait par les processus

de l'application parallèle.



La couche de communication des différents topolgies d'objet de communication

Figure 7.2

L'activation d'une opération de production d'un message à un ensemble indexé est réalisé par le processus producteur qui envoie le message produit à la tâche de communication correspondant à l'ensemble indexé.
L'activation d'une opération de consommation d'un message d'un ensemble indexé est réalisé : par le processus consommateur qui envoie une requête de consommation à la tâche de communication correspondant à l'ensemble indexé, et la consommation du message se réalise lors de la réception du message en réponse à la requête de consommation.

Pour clarifier les idées nous allons donner les instructions exécutées par les processus de l'application parallèle et celles exécutés par la tâche de communication pour satisfaire les opérations de communication :

- **L'opération de production** :
  Le **producteur** active la production par l'instruction :
  chan_out_word (mess, out_ports[i]);
  où mess est le message produit
  La **tâche** de communication réalise l'opération de production par les instructions :
  chan_in_word (&mess, in_ports[j]);
  ajouter(mess, EI);
  { EI est l'ensemble indexé concerné par la production.
  Ajouter : est une opération définié sur l'ensemble indexé. Elle permet d'ajouter un message à cet ensemble.
  La sortie [i] du producteur est lié à l'entrée [j] de la tâche de communication }

- **L'opération de consommation** :
  Le **consommateur** active la consommation et il consomme un message par les instructions :
  chan_out_word (req, out_ports[i]);
  chan_in_word (&mess, in_ports[i]);
  La **tâche** de communication réalise l'opération de consommation par les instructions :
  chan_in_word (&req, in_ports[j]);
  mess = retirer (EI);
  chan_out_word (mess, out_ports[j]);
  { EI est l'ensemble indexé concerné par la consommation.
  Retirer : est une opération définié sur l'ensemble indexé. Elle permet de retirer un message de cet ensemble.
  Les entrée/sortie [i] du producteur sont liés aux entrée/sortie [j] de la tâche de communication }

Les processus de l'application parallèle ainsi que les tâches de communication sont décrits par des modules en C-Parallèle. La différence des noms, processus et tâche, est employé dans cette description pour montrer la différence entre leur rôle.

Nous avons décrit la couche de communication sans tenir compte du placement des processus sur les sites de l'application répartie. En effet, les tâches de cette couche ne dépendent pas du placement de l'application parallèle, (la couche est composée des mêmes tâches quelque soit le placement des processus).
Evidement le placement des processus affecte le placement des tâches de communication sur les différents sites. Les tâches de communication générées à partir des modules : "bipoint"

et "multipoint" sont placées sur les sites de placement de leur ensemble indexé. Les tâches de communication générées à partir de module "diffuseur" sont placés sur le site de leur producteur, et celles générées à partir de module "sélecteur" sont placés sur le site de leur consommateur.

7.1.3 Exemples

Nous allons donner les images réelles exécutables sur la machine T-NODE des applications parallèles citées comme exemples dans le chapitre 1. Dans ces images nous ne considérons pas le placement physique des différents processus ni celui de différentes tâches de communication, car comme nous l'avons cité ces images sont indépendantes du placement des applications parallèles.

Exemple 1

Comme premier exemple, nous prenons l'application parallèle de calcul de pgcd de quatre nombres. L'image réele exécutable sur la machine T-NODE de cette application est composée : des processus de l'application ($P, P_1, P_2, P_3, P_4,\ Q$), et d'une couche de communication décrite ci-dessous, figure 3.
La topologie de l'objet de communication SP est la diffusion, et il est composé de cinq ensembles indexés. Les tâches de communication correspondant à cet objet sont : une tâche de type "diffuseur" nommée *diffus_sp*, et cinq tâches de type "bipoint" nommées : *bipoint_sp0, bipoint_sp1, bipoint_sp2, bipoint_sp3, bipoint_sp4.*
Une tâche de communication de type "bipoint" correspond à chaque objet SR_i. Cette tâche, nommée *bipoint_sr*$_i$, est liée au processus P_i pour la production et au processus Q pour la consommation.
De même, une tâche de communication, *bipoint_sc*$_i$ de type "bipoint" correspond à chaque objet SC_i. Et chacune de ces tâches est lié aux processus : P_i, Q.
Finalement nous trouvons la tâche *bipoint_sc* qui correspond à l'objet SC et elle est liée aux processus : Q pour la production, et P pour la consommation.
La programmation des processus ($P, P_1, P_2, P_3, P_4,\ Q$) est donnée dans l'annexe B.

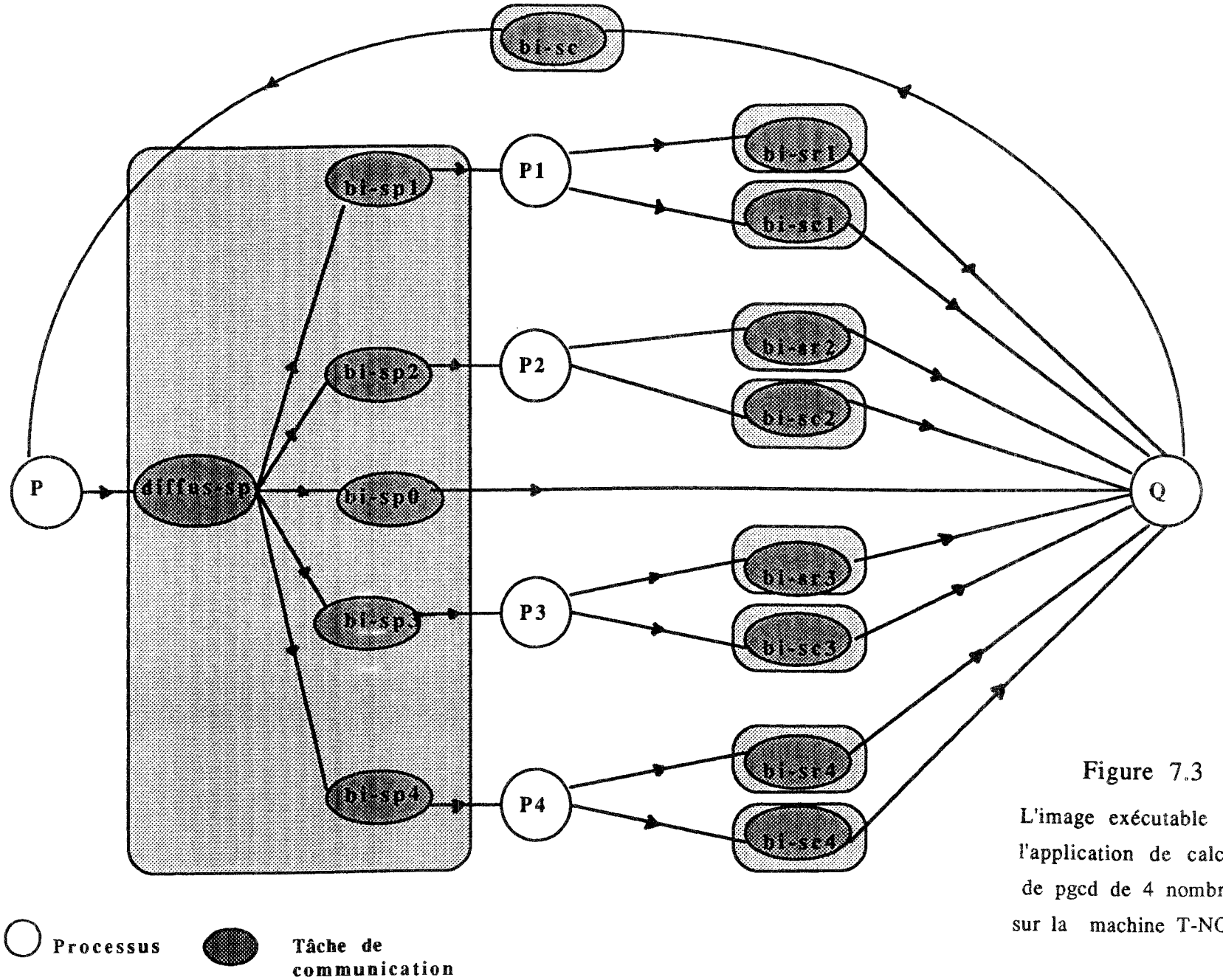


Figure 7.3

L'image exécutable de l'application de calcul de pgcd de 4 nombres sur la machine T-NODE

Exemple 2

Nous reprenons l'exemple 2 du chapitre 1 pour générer sa couche de communication.
Dans cette application nous distinguons deux objets de communication de topologie diffusion ($N1, N2$), et deux objets de topologie bipoint ($F1, F2$).

La couche de communication correspondant a l'objet $N1$ est composée de trois tâches : *diffuseur_N1* de type diffuseur, *bipoint_N11* et *bipoint_N12* de type bipoint. Et celle qui corresponde à l'objet $N2$ est composée aussi de trois tâches : *diffuseur_N2* de type diffuseur, *bipoint_N21* et *bipoint_N22* de type bipoint.
La couche de communication correspondant à chacun des objets $F1, F2$, est une tâche de type bipoint (*bipoine_F1*, *bipoint_F2*).

L'image exécutable de cette application parallèle sur la machine T-NODE est composée des processus de l'application et d'une couche de communication qui n'est que l'union des différentes tâches de communication précisées précédement.

Figure 4, montre les différents processus et les différentes tâches de cette image.
La programmation de différents processus est donnée dans l'annexe C.

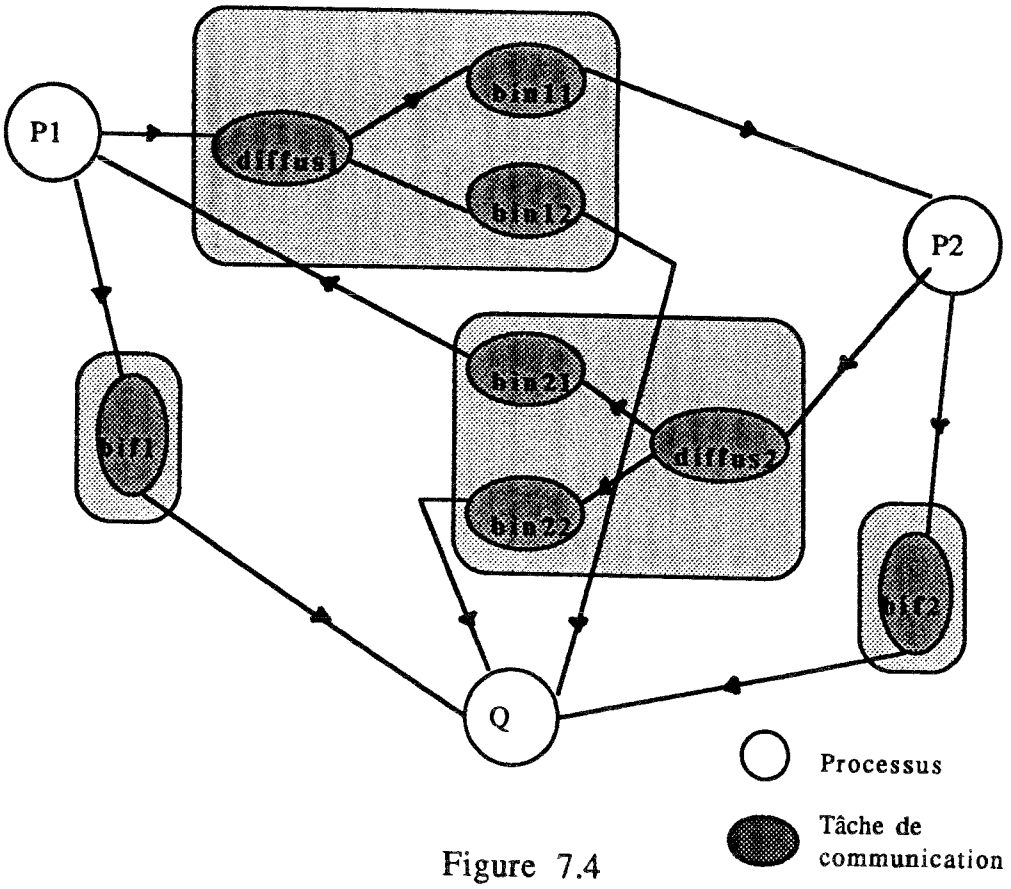


Figure 7.4

7.1.4 Conclusion

Pour définir l'application répartie exécutable sur la machine T-NODE, il a fallu considérer l'environnement du travail disponible sur cette machine ainsi que les concepts fondamentaux de la répartition. Ces concepts ont inspirés le développement de la couche de communication.

Le lecteur peut constater que le passage de la spécification de l'application répartie à son image exécutable sur la machine T-NODE est simple à réaliser. Et à part le problème du placement de processus sur les transputers, ce passage peut être réalisé automatiquement. Cette remarque sera confirmée par le chapitre suivant qui décrit le système du passage, nommé COMPIL.

Avant de développer le système COMPIL, précisons la limite de notre modèle de couche de communication et par conséquent celle du système : nous n'avons pas considéré lors du développement le problème de routage. En fait ce problème sort de cadre de cette étude, et il existe de nombreuses solutions à ce problème.

7.2 La description du système COMPIL

Le systèm COMPIL est la réalisation pratique de la fonction d'implantation sur la machine T-NODE. Il génére l'application répartie exécutable sur la T-NODE qui correspond à une application parallèle donnée.

Nous avons défini, dans la partie 2, la fonction d'implantation comme étant le moyen de passage de la spécification de l'application parallèle à la spécification d'une application répartie correspondant. Dans l'application répartie nous pouvons distinguer : les processus de l'application parallèle et ses tâches de communication. Le rôle de notre système COMPIL est de :

1. réaliser la fonction heuristique de placement,
2. générer la couche de communication,
3. de créer les liaisons entre les processus de l'application et les tâches de communication,
4. allouer les processus et les tâches de communication aux sites (transputers).

Dans l'environnement de programmation 3l, le placement de modules de l'application parallèle et la description de leur liens sont définis dans le fichier de configuration. Le troisième et le quatrième rôle de notre système se traduisent donc par la **création du fichier de configuration de l'application répartie.**

Un autre fichier nécessaire pour l'exécution des programmes parallèles dans l'environnement 3l est un fichier make qui précise : la dépendance entre les différents modules du programme, la compilation et l'édition de liens de ces modules dans l'environnement 3l. Notre système crée aussi ce fichier.

7.2.1 Présentation de système COMPIL

Le système COMPIL est un outil interactif d'aide à l'implantation. Il est basé, comme la fonction d'implantation, sur des règles heuristiques. Il propose une solution, qu'il estime une bonne, à l'utilisateur et il évalue cette solution.

L'utilisateur du système peut intervenir pour :

- modifier une solution proposée par le système,
- proposer une solution,
- prendre la décision : qui consiste à choisir une solution parmi les solutions possibles.

Lors de différentes interventions de l'utilisateur, le système évalue les solutions proposées, mémorise ces différentes solutions avec leurs évaluations et il permet la visualisation de ces informations. L'interaction entre l'utilisateur et le système se fait par l'intermédaire de question/réponse.

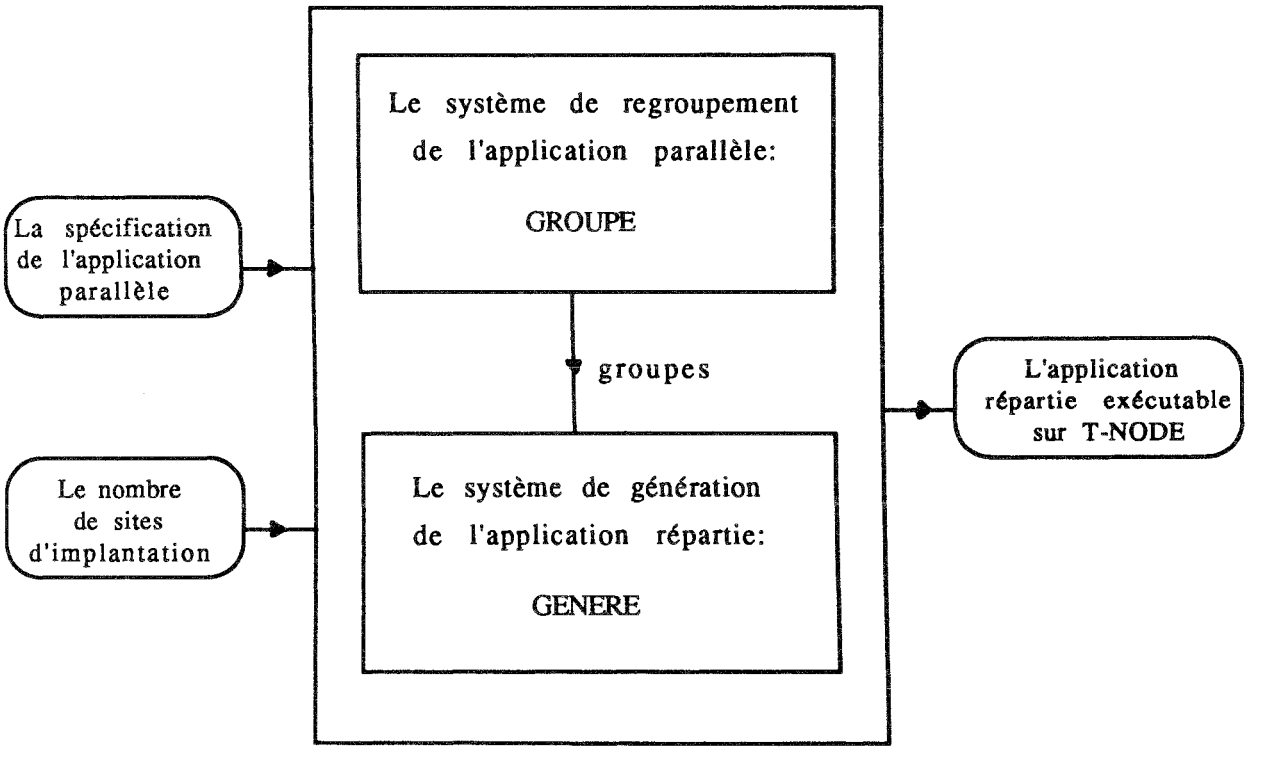
L'**entrée** du système est un fichier qui décrit la spécification de l'application parallèle et il précise le nombre de transputers utilisés pour l'implantation de l'application sur la machine T-NODE.

Les **sorties** du système sont : le fichier de configuration et le fichier make de l'application répartie.

Le système COMPIL est divisé, comme la fonction d'implanatation, en deux sous-systèmes, schéma 1. Cette division a été faite pour séparer la partie heuristique du système de sa partie automatique.
Le premier système est le système GROUPE : la réalisation pratique de la fonction de placement sur la machine T-NODE. Son objectif est de regrouper les éléments de l'application parallèle. Il constitue la partie heuristique de COMPIL et il permet l'intervention de l'utilisateur.

Le deuxième est le système GENERE : la réalisation pratique de la fonction de dérivation sur la machine T-NODE. Son objectif est la génération de l'application répartie. Ce système est un système automatique qui ne nécessite pas l'intervention de l'utilisateur.

Il est possible de rendre le système GROUPE un système automatique en acceptant comme regroupement le regroupement proposé par le système. Le système ne garantit pas que cette solution est la meilleure.



Le sytème COMPIL

Schéma 1

Le système GROUPE

Afin de déterminer un regroupement convenable au choix de l'utilisateur du système, nous avons développé le système heuristique interactif GROUPE. Dans ce système nous pouvons classer les modules, selon leur rôles :

- Modules de Lien : Déduire les liens existant entre les processus de l'application parallèle (voir chapitre 4).
- Modules de Choix : Choisir un regroupement. Le choix se fait par le système ou par l'utilisateur.
- Modules d'Evaluation : L'évaluation d'un regroupement.
- Modules d'Interface : Gérer l'interface entre l'utilisateur et le système.

Ces différentes classes sont en interaction, schéma 2, montre les interactions possibles entre les classes de modules.

L'utilisateur intervient pendant le deroulement du système et surtout pour prendre la décision finale. Son rôle sera précisé lors de description de différents modules.

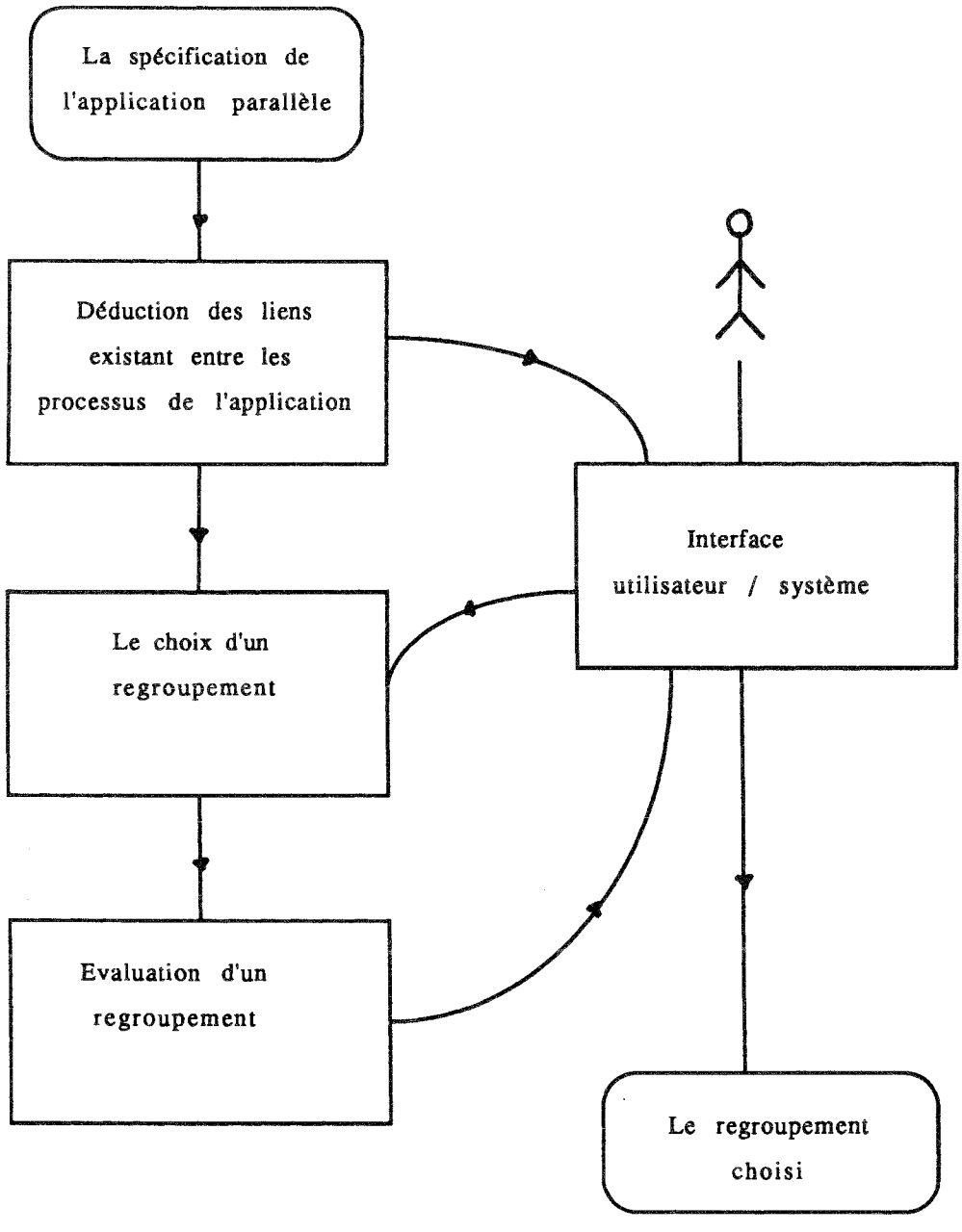
Avant de développer les différents modules précisons les entrées / sorties de système GROUPE.

L'entrée de système GROUPE L'entrée du système est un fichier qui décrit la spécification de l'application parallèle est qui précise le nombre de transputers à utiliser pour l'implantation de cette application.

Le fichier qui décrit la spécification de l'application parallèle contient précisement les informations suivantes :

- Le nombre de processus et d'objets de communication de l'application parallèle ainsi que le nombre de transputer d'implantation.
- Chaque processus est donné par son nom.
- Chaque objet de communication est décrit par :
  - son nom,
  - sa topologie,
  - le nom de son type de communication.
- Liens en production : Chaque lien précise le producteur d'un objet de communication et le numéro de son canal de sortie utilisé pour la production de messages dans cet objet de communication.
- Liens en consommation : Chaque lien précise le consommateur d'un objet de communication et le numéro de son canal d'entrée utilisé pour la consommation de messages de cet objet de communication.

La sortie de système GROUPE Le résultat de ce système est un regroupement des éléments de l'application parallèle. Ce regroupement est un ensemble de "n" groupes, où "n" est le nombre de transputers d'implantation. Chaque groupe est composé d'un ensemble de processus et d'un ensemble d'ensemble indexé.



Les interactions entre les différents modules de système GROUPE

Schéma 2

La déduction de liens Les rôles de modules de cette classe sont :

- donner des valeurs numériques aux facteurs de dépendance : C_k, f_{ijk} de l'application parallèle (voir chapitre 4),
- déduire les liens existant entre les différents paires de processus (voir chapitre 4).

Les valeurs des facteurs C_k : L'**utilisateur** doit préciser la relation d'ordre suffisante pour son application : " as< " ou " op< ".
Le **système** détecte si les types de communication utilisès dans l'application parallèle sont ordonnés totalement, partielement ou qu'il ne sont pas ordonnés par cette relation.
Selon le résultat de cette détection et à l'aide de l'utilisateur, le système donne des valeurs numériques aux facteurs C_k.

Les valeurs des facteurs f_{ijk} : Le **système** déduit les relations de communication existant entre les processus de l'application parallèle (voir chapitre 4), et il interroge l'utilisateur pour connaitre les priorités de ces relations de communication.

La connaisance de valeurs numériques des facteurs C_k, f_{ijk} permet au système :

- d'exprimer les liens entre les processus par des valeurs numériques,
- de pouvoir comparer les liens de dépendances des processus.

Le choix d'un regroupement Le système choisit un regroupement de composants de l'application parallèle et il le propose au utilisateur. Dans le cas où le nombre de transputers utilisés est inférieur au nombre de processus, la proposition faite par le système regroupe les processus les plus liés.

Evaluation Le système évalue chaque regroupement proposé par l'utilisateur ou choisi par lui même.
L'évaluation d'un regroupement consiste à calculer les valeurs des fonctions d'évaluation présentées dans le chapitre 4 : $\mathcal{F}, \mathcal{G}, \mathcal{D}, \mathcal{L}$

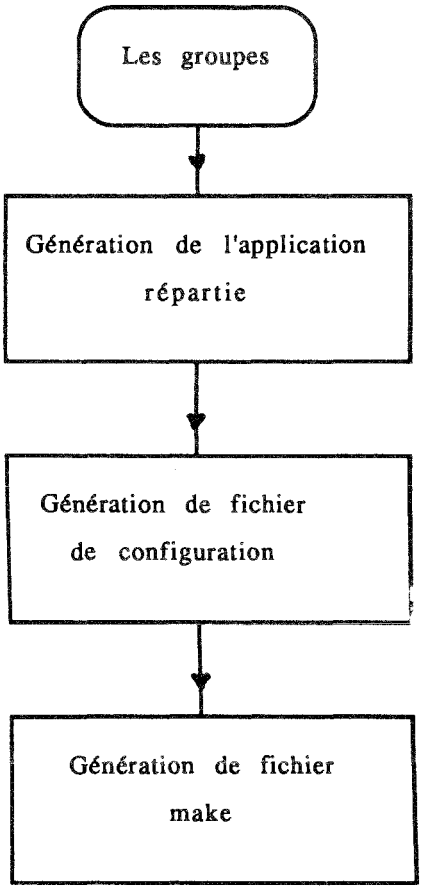
Interface utilisateur/système Le système offre une interface permettant l'interaction entre l'utilisateur et le système. Cette interface met à la disposition de l'utilisateur les possibilités de dialogue suivantes :

- la proposition d'un regroupement,
- la permutation des processus d'un regroupement,
- le déplacement de certains ensembles indexés d'un regroupement,
- la mémorisation ou pas de regroupements acceptables,
- la visualisation de regroupements mémorisés,

- la visualisation d'évaluations des regroupements mémorisés,
- la prise d'une décision : qui consiste à choisir un regroupement parmi les regroupements mémorisés.

Le système GENERE

C'est un système automatique composé de trois parties qui s'exécutent séquentiellement, schéma 3. Il utilise le résultat du système précédent ainsi que la spécification de l'application parallèle.



Les différents modules de système GENERE

Schéma 3

Nous allons décrire brièvement le fonctionnement de chaque partie du système.

Génération de l'application répartie Pour chaque objet de communication de l'application parallèle, le système génére les tâches de communication correspondant. Cette génération consiste à :

- Choisir les modules types de tâches de communication : (bipoint, multipoint, diffuseur, sélécteur) selon la topologie de l'objet de communication.
- Nommer les tâches de communication par des noms dérivés de nom de module type.
- Préciser le nombre de canaux d'entrée et le nombre de canaux de sortie de chaque tâche de communication, ceci en prenant en compte la spécification de l'objet de communication.
- Créer les liaisons entre les tâches de communication de l'objet et ses processus producteurs et consommateurs.

Pour gérer les opérations de communication de l'ensemble indexé, la tâche considère le type de communication associé à cet ensemble.

Le système GENERE détermine aussi le placement des processus et des tâches de communication sur les transputers. Il ne fait aucune distingution entre les différents transputers, et il considère le regroupement choisi par l'utilisateur.

Les règles d'allocation appliquées par le système sont les suivantes :

1. Les processus de chaques groupe et les tâches de communication associées aux ensembles indexés de ce groupe seront alloués au même transputer (cf : la procédure d'allocation de chapitre 5).
2. Une tâche de communication de type diffuseur, qui est associé à un objet de communication de toplogie diffusion, est alloué au même transputer que le producteur de l'objet.
3. Une tâche de communication de type sélécteur, qui est associé à un objet de communication de toplogie convergence, est alloué au même transputer que le consommateur de l'objet.

Les différentes informations déduites par cette partie sont mémorisés dans des tables. Nous citons celles qui seront utilisés lors de la création de fichier de configuration et de fichier make.

- **La table des processus** : Chaque élément de cette table correspond à un processus de l'application répartie. Il est composé : du nom du processus, d'un identificateur du processus, du nombre de canaux d'entrées du processus, du nombre de canaux de sortie du processus, et de l'espace mémoire nécessaire pour l'exécution du processus.
- **La table de transputer** : Chaque élément de cette table correspond à un transputer de l'application répartie. Il est composé : du nom du site, d'un identificateur du site, de l'état des liens physiques du site (utilisé ou non).
- **La table de placement** : Chaque élément de cette table détermine le placement d'un processus de l'application répartie sur un de ses transputers.

- **La table des liaisons logiques** : Chaque élément de cette table repésente une liaison unidirectionnelle entre deux processus de l'application répartie.
- **La table des liens physiques** : Chaque élément de cette table repésente un lien physique bidirectionnele entre deux transputers.
- **Une table pour chaque module type de tâche de communication** : Cette table indique le nombre et les noms de tâches de communication utilisées d'un type donné de module.

La création de fichier de configuration Cette partie consiste à parcourir les tables : de processus, de transputer, de placement, la table de liaisons, et la table de liens physiques; et écrire leurs information sous le format d'un fichier de configuration.

La création de fichier make Cette partie consiste à parcourir les tables : de processus, et les différentes tables de tâche de communication; et écrire leurs information sous le format de fichier make.

7.2.2 Exemples

Nous allons donner quelques exemples d'utilisation de système COMPIL. L'objectif de chaque exemple est de montrer un aspect du système. Un seul exemple, exemple 2, se trouve dans ce document. Les autres exemples se trouvent dans le document de la programmation.

Exemple 1

L'objectif de cet exemple est de montrer la génération de la couche de communication et de préciser l'entrée/sortie du système.
L'application parallèle considérée est l'exemple 2 de chapitre 1. Le nombre de transputers d'implantation est 3, qui est égale au nombre de processus. Le regroupement de processus n'est pas, donc, nécessaire pour cette application.

Exemple 2

L'objectif de cet exemple est de montrer une session de dialogue entre le système et l'utilisateur. L'application parallèle est celle de l'exmple 1, mais le nombre de transputers d'implantation est 2. Un regroupement de processus est nécessaire dans ce cas.

Plusieurs regroupements ont été proposés par l'utilisateur. Nous remarquons l'équivalence de certains regroupements (regroupement : 0, 3), cela est dû au symetrie de processus P1, P2 vis-à-vis le processus Q.

Exemple 3

L'objectif de cet exemple est multiple :

- donner l'image exécutable de l'application répartie de calcul de pgcd de 4 nombre (cf : exemple 1 de chapitre 3),
- montrer un autre mode d'exécution du système. Le mode mi-automatique,
- avoir l'équité de charge des transputers.

Exemple 4

C'est une série d'utilisations du système pour la même application parallèle. L'objectif de cette série est de montrer l'effet des facteurs C_k, f_{ijk} sur : la dépendance entre les processus, le choix fait par le système et sur l'évaluation de regroupement.
Les valeurs de dépendances sont dans la table d'accumulation qui donne pour chaque paire de processus une valeur qui évalue leur dépendance.
Dans ces exemples, nous pouvons éliminer l'éffet des facteurs C_k (resp : f_{ijk}) en leurs associant la même valeur.

Chapitre 8

Conclusion

Pour conclure ce document nous proposons de revenir sur les objectifs de cette thèse et de faire le point sur les problèmes auquels nous avons apporté une réponse, sur ceux qui restent ouverts et sur les perspectives possibles de ce travail.

Notre objectif était d'apporter une contribution à la définition et à l'étude de techniques d'implantation de programmes parallèles sur des architectures réparties.

Afin de généraliser et de formaliser ces techniques nous avons été amené à définir l'abstraction des différents concepts intervenant lors de l'implantation de programmes parallèles sur les architectures réparties.

Les programmes parallèles constituent le concept logique de cette implantation. Ils sont présentés comme des processus communicants via des objets de communication. Chaque objet de communication est caractérisé par un type de communication qui décrit une relation de synchronisation entre les processus.
L'application parallèle est une vue abstraite de ces programmes, ses processus sont décrits par leurs potentiels d'entrée-sortie via les objets de communication.

Les architectures réparties constituent le concept physique de l'implantation. Ce sont des architectures multiprocesseurs à mémoire distribuée (DMPC). Leur abstraction est définie par un système réparti composé d'un ensemble de sites connectés par des liaisons.

Les définitions de l'application parallèle d'une part et du système réparti d'autre part, nous ont permis de définir formellement l'application répartie : l'implantation d'une application parallèle sur un système réparti.

Précisons que l'abstraction de ces différents concepts nous a permis de :

- concentrer notre étude sur l'aspect communication de programmes parallèles et d'architectures réparties. C'est l'aspect fondamental dans notre travail,
- formaliser ces concepts et mettre en évidence leurs caractéristiques principales indépendamment de leurs aspects physiques ou techniques,
- formaliser le mécanisme de l'implantation.

La répartition d'une application parallèle nécessite le placement de ses composants sur les sites du système réparti. Les critères originaux de placement retenus dans ce travail sont fondés sur l'étude précise des objets de communication et des contraintes de synchronisation et de nombre de messages qu'ils induisent.
Le choix des critères de placement nous a amené à approfondir l'étude des caractéristiques des types de communication et à les comparer du point de vue de l'asynchronisme. (Cette étude et les différents critères de placement sont présentés dans le chapitre 4.)

Notre approche en ce qui concerne le placement des tâches est spécifique au langage LESP. Elle est donc complèmentaire aux études traditionnelles concernant ce problème.

Pour un placement (donné) des composants de l'application parallèle sur les sites d'un système réparti, la dérivation de l'application répartie est automatique et unique. Les procédures des différentes étapes de dérivation sont développées dans le chapitre 5.

L'exécution des applications parallèles dans un environnement réparti implique une exécution à distance de certaines opérations. En conséquence une nouvelle interprétation a été associée à ces opérations. Cette interprétation, comme nous l'avons vu au chapitre 5, tient compte du placement des différents composants, impliqués par cette opération, sur les sites du système réparti.

L'implantation d'une application parallèle sur un système réparti est présentée formellement dans ce document. Elle a été développée pour être le guide d'un système d'assistance à la conception des applications réparties. Le système COMPIL est la réalisation concrête de ce système sur la machine T-NODE (machine parallèle à base de transputers). Il génére l'application répartie exécutable sur la T-NODE qui correspond à une application parallèle donnée.

Lors du développement du système COMPIL nous avons constaté que :

- la formalisation de notre étude (première et deuxième parties) est une étape générale, nécessaire pour la réalisation.
- l'environnement matériel et logiciel de la machine n'ont pas d'effet sur la formalisation. En fait nous l'avons développé avant de connaitre l'environnement de la machine T-NODE.
- l'exécution d'une application parallèle dans un environnement réparti nécessite l'introduction d'une couche système qui interprète les opérations de communication et doit être invisible pour l'utilisateur.

Perspectives

Une perspective envisagable à ce travail est d'élargir les aspects considérés en matière d'application parallèle et d'architecture répartie.
En ce qui concerne l'application parallèle nous pouvons prendre en compte le contexte des processus (le corps des processus) et leur fréquence d'accès aux objets de communication. Cette considération peut modifier les règles de regroupement et de placement des processus sur les sites du système réparti. L'évaluation de la fréquence d'accès d'un processus à un objet de communication peut également affiner les règles d'association d'objet de communication aux processus.
En ce qui concerne l'architecture répartie, nous pouvons considérer le cas de réseaux de communication non complétement maillés. Cela implique de modifier des règles heuristiques de placement et la formulation de ses fonctions d'évaluation.

L'introduction de nouvelles règles heuristiques va modifier les critères d'évaluation de placement. La charge d'un site sera évaluée, par exemple, en fonction du nombre d'ensembles indexés et du nombre de processus placés sur le site, tout en considérant le contexte des processus.

Nous pouvons aussi envisager d'évaluer les règles heuristiques introduites en réalisant quelques mesures. Nous comparons le temps d'exécution d'une application répartie obtenue en applicant les règles heuristiques avec le temps d'exécution d'une application répartie obtenue sans tenir compte de règles heuristiques. Cette évaluation pourra compléter notre étude et rendre nos règles heuristiques plus convaincantes. De plus elle nous permettra de favoriser une règle heuristique par rapport aux autres.

Il sera possible aussi d'intégrer nos règles heuristiques dans un algorithme traditionnel de placement tel que l'algorithme de Bokhari [Bok81b]. Cette intégration permettra d'une part de bénéficier de la puissance et de la rapidité d'un tel algorithme et d'autre part d'appliquer les règles heuristiques spécifiques au langage LEPSC.

Enfin, dans la version actuelle du système COMPIL, le passage du programme parallèle à sa formalisation est à la charge de programmeur. Il sera possible dans un avenir proche d'étendre le système COMPIL pour :

- vérifier la correction de la formalisation de l'application parallèle,
- déduire automatiquement la formalisation de l'application parallèle à partir de l'algorithme.

Bibliographie

[ABM88] S. Aggarwal, D. Barbara, and Z. Meth. A software environement for the specification and analysis of problems of coordination and concurrency. *IEEE .TSE*, 14(3):280–290, March 1988.

[Ada80] Ada. *Formal definition of the ADA programming.* Cii Honewell Bull, 1980.

[AP88] F. André and J. L. Pazat. Le placement de tâches sur les architectures parallèles. *TSI*, 7(4):385-401, 1988.

[Bal90] J. C. Ballegeer. Le système d'exploitation distribué HELIOS. *La lettre de TRANSPUTER et des calculateurs distribués*, (5):33–39, Mars 1990.

[BB86] J. P. Banatre and M. Banatre. An overview of GOTHIC distributed operating system. Technical Report 284, IRISA, January 1986.

[BCS89] A. Billionnet, M. C. Costa, and A. Sutter. Les problèmes de placement dans les systèmes distribués. *TSI*, 8(4):307–337, 1989.

[Ber85] G. Berthomieu. Le langage LCS et son interprète. une implantation expérimentale de CCS. In A. Arnold, editor, *Actes du 1er Colloque* C^3, 1985.

[BMH87] A. Benkiran, J. Miralles, and E. Horlait. CSA, une approche objectale d'un système distribué. Technical Report 191, MASI, Juillet 1987.

[Boc79] Gregor V. Bochmann. Distributed synchronization and regularity. *Computer network*, 3:36–43, 1979.

[Bok79] S. H. Bokhari. Dual processor scheduling with dynamique reassignment. *IEEE Trans. on Soft. Eng.*, SE-5(4):341–349, July 1979.

[Bok81a] S. H. Bokhari. A shortest tree algorithm for optimal assignments across space and time in a distributed processor system. *IEEE Trans. on Soft. Eng.*, SE-7(6):583–589, November 1981.

[Bok81b] Shahid H. Bokhari. On the mapping problem. *IEEE trans. on comp.*, C-30(3):207–214, March 1981.

[Bou90] D. Boudebous. Petri-S: un outil d'aide au prototypage rapide des automatismes industriels décrits par des Réseaux de Petri Interprétes. Thèse de Doctorat de l'Université de Nancy 1, 1990.

[BP89] D. Benhamamouch and G. Plateau. Les problèmes d'allocation de ressources dans les systèmes distribués. Université PARIS-NORD, Février 1989.

[BR83] G. V. Bochmann and M. Raynal. Structured specification of communicating systems. *IEEE Trans. on Computer*, C-32(2):120–133, February 1983.

[BS87] Roland P. Bianchini and John P. Shen. Interprocessor trafic scheduling algorithm for multiprocessor network. *IEEE Trans. on Comp.*, C-36(4):396–409, April 1987.

[Car91] D. Caromel. Programmation parallèle asynchrone et impérative: étude et propositions. Thèse de Doctorat de l'Université de Nancy 1, 1991.

[Cer86] P.Le Certen. Conception et mise en œuvre d'un langage impératif pour la programation parallèle. Thèse de Doctorat de l'Université de Rennes 1, 1986.

[CH74] R. H. Campbell and A. N. Haberman. The specification of process synchronisations by path expression. *LNCS*, 16:89–102, 1974.

[COR81] CORNAFION, editor. *Système informatiques répartis, concepts et techniques.* DUNOD Informatique, 1981.

[CY83] S. Chen and T. Yeh. Formal specification and verification of distributed systems. *IEEE TSE*, SE-9(6):710–722, November 1983.

[DFLP85] Y. Deswarte, J. C. Fafre, J. C. Laprie, and D. Powell. Le projet SATURNE. Technical Report 445, INRIA, Septembre 1985.

[Dub84] E. Dubois. Cadre et méthode de spécification de système d'information fondés sur les types de données. Thèse Docteur Ingénieur en Informatique, Institut National Polytechnique de Lorraine, Mars 1984.

[DV87] Kshitij A. Doshi and Peter J. Varman. Optimal graphe algorithm on a fixed size lineaire array. *IEEE Trans. On Comp.*, C-36(4):460–470, April 1987.

[Efe82] K. Efe. Heuristic models of task assignement scheduling in distributed systems. *IEEE Comp.*, pages 50–56, june 1982.

[EHM84] M. Elizabeth, C. Hull, and R. M. McKeag. Communicating sequential processes for centralized and distributed operating system. *ACM TPLS*, 6(2):175–191, April 1984.

[EJ89] M. C. Eglin and J. Juilland. Interprétation parallèle asynchrone de systèmes d'équations. In AFCET, editor, *In 10 ième séminaire tuniso-français.* AFCET-Université de Tunis, May 1989. Rapport crin 88-R-152.

[Fin79] J. P. Finance. Etude de la construction des programmes: méthodes et langages de spécification et de résolution de problèmes. Thèse d'Etat. Université de Nancy 1, 1979.

[Fis89] S. Fisher. MACH, the new Unix. *Unix world*, pages 64–68, March 1989.

[FJ80] J.P. Finance and J. Jary. Towards a methodlogy to specify and construct concurrent programs. Technical report, CRIN, 1980.

[FR86] R. Fitzgerald and R. F. Rashid. The integration of virtual memory management and interprocess communication in ACCENT. *ACM Trans. on Comp. Syst.*, 4(2):147–177, May 1986.

[Gir87] C. Girault. Analyse des systemes paralléles et distribués, problèmes, méthodes et objectifs. Technical Report 201, MASI, Decembre 1987.

[GRA86] Anna GRAM, editor. *Raisonner pour programmer.* DUNOD informatique, 1986.

[Gui84] M. Guillemont. A comparative study of some distributed systems. *TSI*, 3(1):5–17, 1984.

[HMR85] J. M. Hélery, A. Maddi, and M. Raynal. Controler les tranfert de connaissance dans les applications distribuées: application à la détection de l'interblocage. Technical Report 260, IRISA, Juin 1985.

[Hoa78] C. A. Hoar. Communicating sequential processes. *Com. of the ACM*, 21(8):666–677, August 1978.

[Hoc85] F. R. A. Hockney. Mimd computing in the USA. *Parallel Computing*, 2(2):119–136, 1985.

[HPR88a] J. M. Hélary, N. Plouzeau, and M. Raynal. Calcul d'états globaux remarquables dans un système réparti. Technical Report 396, IRISA, Mars 1988.

[HPR88b] J. M. Hélary, N. Plouzeau, and M. Raynal. A distributed algorithme for mutual exclusion in an arbitary network. *The computer journal*, 31(4):289,295, 1988.

[HPRM87] J. M. Hélary, N. Plouzeau, M. Raynal, and A. Maddi. Parcours et apprentissage dans un rséau de processus communicant. *TSI*, 6(2):127–139, 1987.

[HR88] J. M. Hélary and M. Raynal. Un algorithme distribué d'affectation d'identité distinctes aux sites d'un système réparti anonyme. Technical Report 391, IRISA, Fevrier 1988.

[JDK87] C. Jard and O. Drissi-Kaitouni. Deriving trace checkers for distributed systems. Technical Report 347, IRISA, February 1987.

[JK87] C. Jard and O. D. Kaitouni. Géneration de vérificateurs de traces pour les systèmes distribués. Technical Report 347, IRISA, Février 1987.

[JM85] J. Juilland and M. Marmonier. COMEDIE: outils de développement de système parallèles. Technical report, CRIN, 1985.

[JP85] J. Juilland and G.R. Perrin. Towards a methodology for concurrent programming. In Elsevier Science Publishers, editor, *Int. Conf. on parallel computing*, 1985.

[JR86] J. B. Jones and R. F. Rashid. MACH and matchmarker: Kernel and language support for distributed object oriented systems. In *Proc. of the 1st ACM Conf. on Object-Oriented Programming Systems, Languages, and Applications*, 1986.

[KFJP88] A. Koukam, J. P. Finance, J. Juilland, and G. R. Perrin. Spécification et réalisation des types de communication. In *In proceedings of International Symposium on Software Engineering*, Oran- Algérie, 1988.

[Kon88] J. C. Konig. Un algorithme réparti pour la diffusion d'informations dans un réseau. *TSI*, 7(3):309–314, 1988.

[Kou90] A. Koukam. Dérivation de programmes Ada par transformation de systèmes parallèles fondés sur la communication abstraite entre processus. Thèse de Doctorat de l'Université de Nancy 1, 1990.

[Kra87] S. Krakowiak. La recherche dans la domaine des systèmes répartis. *TSI*, 6(2):2–9, Mars-Avril 1987.

[Kra88] S. Krakowiak. GUIDE: un système d'exploitation réparti. *TSI*, 23(2):92–105, February 1988.

[Kro87] F. Kroger. Abstract modules: Combining algebric and temporal logic specification means. *TSI*, 6(6):559–573, 1987.

[LA87] Soo Young. Lee and J. K. Aggarwal. A mapping strategy for parallel processing. *IEEE trans. on comp.*, C-36(4):433–442, April 1987.

[Lam78] E. Lamport. Time, clocks, and the ordering of events in a distributed system. *Com. of the ACM*, 21(7):558–565, July 1978.

[Lam83] E. Lamport. Specifying concurrent program modules. *ACM TPLS*, 5(2):190–222, April 1983.

[Lan87] G. Le Lanne. Le projet SCORE: les systèmes informatiques répartis temps reel. *TSI*, 6(2):175–179. 1987.

[Laz86] N. Lazrak. Spécification temporelle de relation de communication. Rapport de D.E.A, CRIN, 1986.

[Laz90] N. Lazrak. Verification de propriétés des programmes parallèles. Thèse de Doctorat de l'Université de Nancy 1, 1990.

[Lev84] N. Levy. Outils d'aide a la construction et transformation de types abstraits algebriques. Thèse de Doctorat de l'Université de Nancy 1, 1984.

[Lis88] B. Liskov. Distributed programming in ARGUS. *Com. of the ACM*, 31(3):300–312, March 1988.

[Lo84] V. M. Lo. Heuristique algorithms for task assignment in distributed systems. In *Proc. of the Int. Conf. on Distributed Computing Systems*, pages 30–39, San Francisco, California, May 1984.

[LS83] B. Liskov and R. Scheifler. Guardian and actions: linguistic support for Robust, Distributed Programs. *ACM TPLS*, 5(3):381–404, July 1983.

[Ma84] P. Y. Ma. A modele to solve timing critical application problems in distributed computer systems. *Computer*, 17(1):145–159, Mai 1984.

[Mar88] V. Mary. Heuristic algorithms for task assignement in distributed systems. *IEEE trans. on comp.*, 37(11):57–68, November 1988.

[Mar89] M. Marmonier. Spécification et validation de systèmes de processus communicants. Thèse de Doctorat de l'Université de Nancy 1, 1989.

[Mat87] F. Mattern. Algorithms for distributed termination detection. *Distributed computing*, 2:161–175, 1987.

[Mat88] F. Mattern. Virtual time and global states of distributed systems. Technical Report SFB124-38, The universite of Kaiserslautern, October 1988.

[MB85] F. Mattern and C. Beilken. The distributed programming language CSSA. Technical Report 123, Univ de Kaiserslautern, January 1985.

[Mil88] R. Milner. Operational and algebric semantics of concurrent processes. Technical Report 88-46, ECS-LFCS Report Series, University of Edinburgh, February 1988.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[MLT82] P. R. Ma, E. Y. S. Lee, and M. Tsuchiya. A task allocation model for distributed computing systems. *IEEE Trans. on Computers*, C-31(1):66–78, January 1982.

[Mon87] C. Mongenet. SYSTOL, un logiciel de conception d'algorithme systolique. *TSI*, 6(1):41–55, 1987.

[MW84] Z. Manna and P. Wolper. Synthesis of communicationg processes from temporal logic specification. *ACM TPLS*, 6(1):68–93, January 1984.

[Nai87] M. Naimi. Une structure arborescente pour une classe d'algorithmes distribués d'exclusion mutuelle. Thèse de l'Université de Frache-Comte, Mars 1987.

[NHM+87] J. Nehmer, D. Haban, F. Mattern, D. Wybranietz, and D. H. Rombach. Key concept of the INCAS multicomputer project. *IEEE TSE*, SE-13(8):913–923, August 1987.

[OSS80] J. K. Ousterhout, D. A. Scelza, and P. S. Sindhu. MEDUSA: an experiment in distributed operating system structure. *Com. of the ACM*, 23(2):92–105, February 1980.

[Paz89] J. L. Pazat. Outils pour la programmation d'un multiprocesseur à mémoires distribuées. Thèse à l'Université de Bordeaux 1, Février 1989.

[PB87] D. Polychronopoulos and U. Banerjee. Processor allocation for horizontal and vertical parallelisme and related speedup bounds. *IEEE trans. on comp.*, C-36(4):410–420, April 1987.

[Per85] G. R. Perrin. La communication: un outil pour la spécification, la construction, et la verification de systèmes parallèles. Thèse d'Etat de l'Université de Nancy 1, 1985.

[Per87] G. R. Perrin. Programmation parallèle: point de vue sur les langages et les méthodes. *TSI*, 6(2):103–113, 1987.

[Pet81] J. L. Peterson. *Petri Net theory and the modeling of systems.* Prentice-Hall, 1981.

[PW87] Shlomith S. Pinter and Yvon. Wolfstahi. On mapping processes to processors in distributed systems. *Inter. journal of parallel programming*, 16(1):1–15, 1987.

[Rif86] J. M. Rifflet. *La programmation sous UNIX.* McGram-HILL, 1986.

[RK83] K. Ramamritham and R. Keller. Specification of synchronizing processes. *IEEE .TSE*, SE-9(6):722–732, November 1983.

[RM87] R. Ramesh and S. L. Methndiratta. A methodology for developping distributed programs. *IEEE trans. on soft. Eng.*, SE-13(8):967–976, August 1987.

[RR81] G. Robertson and R. F. Rashid, editors. *ACCENT: A communication oriented network operating system kernel.* ACM, Proceeding of the 8th Symposium on Operating Systems Principles, December 1981.

[RSH79] G. S. Rao, H. S. Stone, and T. C. Hu. Assignment of tasks in a distributed processor system with limited memory. *IEEE Trans. on Computers*, C-28(4):291–299, April 1979.

[SAG+87] M. Shapiro, V. Abrossimov, P. Gautron, S. Habert, and M. Mounchili Makpangou. SOS: un systeme d'exploitation réparti fondé sur les objets. *TSI*, 6(2):166–169, 1987.

[Sch89] A. Schiper. *Concurrent Programming.* North Oxford Acadimic, 1989.

[Sei85] Charles L. Seitz. The Cosmic cube. *Com. of the ACM*, 28(1):22–33, January 1985.

[SS84] J. A Stankovic and I. S. Sidhu. An adaptative biding algorithm for processes, clusters and distributed groups. In *Proc. of the 4e Int. Conf. on Distributed Computing Systems*, pages 49–58, San Francisco, California, Mai 1984.

[Tan88] J. Tankoano. M2C: une approche méthodique pour la conception certifiee des systèmes de commandes des automatisme industriels répartis. Thèse d'Etat. Université de Nancy 1, 1988.

[TM87] Wladyslaw M. Turski and Thomas S.E. Maibaum. *The Specification of Computer Programs.* Addison Wesley Publishing Company, 1987.

[UW87] E. Upfal and A. Wigderson. How to share memory in distributed system. *Journal of the association for computing machinery*, 34(1):116–127, January 1987.

[YH88] Stephen S. Yau and Woumo. Hong. Verification of concurrent control flow in distributed computer systems. *IEEE trans. on soft. eng.*, 14(4):405–417, April 1988.

[Zak84] A. Zakari. FLEXI: Langage de conception d'applications de conduite de procédés industriels. Thèse de Doctorat de l'Université de Nancy 1, 1984.

[ZE88] M. Zaki and M. M. Elboraey. Analysis and reliability models for interconnecting MIMD systems. *The computer journal*, 31(4):304–312, 1988.

[ZGMB84] H. Zimmermann, M. Guillemont, G. Morisset, and J. S. Banino. CHORUS: une architecture pour les systèmes répartis. Technical Report 274, INRIA, Mars 1984.

Exemple

Exécution assistée de système COMPIL sur l'application parallèle de calcul de Factoriel

Fichier de spécification

de l'application

LA SPECIFICATION DE L'APPLICATION PARALLELE
LES PROCESSUS:
% P1 P2 Q

LES OBJETS DE COMMUNICATION:

	NOM	TOPOLOGIE	TYP_COM
%	N1	diffusion	aleatoire
%	N2	diffusion	aleatoire
%	F1	bipoint	egalite
%	F2	bipoint	egalite

LES LIENS EN PRODUCTION:

	LA SORTIE DE PROCESSUS	L'OBJET DE COMMUNICATION
%	(P1, 1)	N1
%	(P2, 1)	N2
%	(P1, 2)	F1
%	(P2, 2)	F2

LES LIENS EN CONSOMMATION:

	L'ENTREE DE PROCESSUS	L'OBJET DE COMMUNICATION
%	(P1, 0)	N2
%	(P2, 0)	N1
%	(Q, 0)	N1
%	(Q, 1)	N2
%	(Q, 2)	F1
%	(Q, 3)	F2

LES SITES D'IMPLANTATION:
% S1 S2

Exécution montrant le dialogue avec l'utilisateur

Ce compilateur permet l'implantation de l'application parallele,
decrite dans le fichier d'entree par sa specification.

L'execution du compilateur peut etre: assistee, mi-automatique ou automatique.

1- Assiste: dans ce mode d'execution l'utilisateur peut choisir le
placement qu'il souhaite.
Le role de compilateur est d'evaluer les placement et de realiser le
placement choisi.

2- Mi_automatique: dans ce mode d'execution l'utilisateur peut
seulement deplacer les objets de communication.
Le placement et le regroupement des processus sont imposes par le compilateur.

3- Automatique: Le regroupement et le placement des differents
elements de l'application parallele sont faits par le compilateur.

Entrez le mode d'execution souhaite (1, 2, 3) ? 1

Voulez vous visualiser la trace de toutes les etapes de l'execution (y/n) ? n

Les types trouves sont:
ALEATOIRE
EGALITE

Quelle est la relation d'ordre suffisante pour cette application ?
(1= op< , 2 = as <): 2
Les types de communication sont ordonnes totalement par la relation as<.
(aleatoire as< egalite)

Donner des valeurs aux facteurs Ck
ALEATOIRE : 1
EGALITE : 3

Les liens trouves sont:
P_CONS: Deux processus, l'un est le producteur d'un objet de
communication et l'autre est un consommateur de meme objet.
C_DIFF: Des consommateurs d'un objet dont la topologie est la diffusion.

Le regroupement des processus lies par ces relations a comme effet:
P_CONS:
1 - Diminuer le nombre de liaisons.
2 - Diminuer le nombre de messages.
C_DIFF:
1 - Diminuer le nombre de liaisons dans certains cas.
2 - Diminuer le nombre de messages dans certains cas.
!!!! Pas tres interresant comme regroupement.

Donner des valeurs aux facteurs fijk
P_CONS: 4
C_DIFF: 1

Les accumulations

proc1	proc2	value
0	2	17
1	2	17
0	1	8

La table des identificateurs

P1	0
P2	1
Q	2
N1	100
N1	101
N1	102
N2	103

```
N2          104
N2          105
F1          106
F2          107
SITE1       200
SITE2       201


  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  0    2
   Les objets :  101    102    106    107

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  1
   Les objets :  104    105

La valeur de la fonction Fonction_Dep = 42

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 2
La valeur de la fonction Nombre_Liais = 17
La valeur de la fonction Fonction_DC  = 25

 Les Differentes Evaluation

Groupe   Charge_Pr  Charge_Ob  Nombre_Liais  Fonction_Dep
  0        1          2          17             25

Vouley vous modifier le regroupement des processus (y/n) ? y
Le nombre des processus du groupe  0 : 2
Les processus : 0 1
Le nombre des processus du groupe  1 : 1
Les processus : 2

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  0    1
   Les objets :  101    102    104    105

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  2
   Les objets :  106    107

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 2
La valeur de la fonction Nombre_Liais = 16
La valeur de la fonction Fonction_DC  = 34

Voulez vous sauvgarder le groupement precedent (y/n) ?  y

Voulez vous deplacer les objets (y/n) ?  y

Quel objet voulez vous deplacer ( 0 = fin ) ? 102

A quel groupe ? 1

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 0
La valeur de la fonction Nombre_Liais = 15
La valeur de la fonction Fonction_DC  = 34

  Les groupes
```

```
 GROUPE   0  ::    nbproc =   2 nbobj =   3
   Les processus :  0    1
   Les objets :  101    104    105

 GROUPE   1  ::    nbproc =   1 nbobj =   3
   Les processus :  2
   Les objets :  106    107    102

Quel objet voulez vous deplacer ( 0 = fin ) ? 105

A quel groupe ? 1

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 2
La valeur de la fonction Nombre_Liais = 14
La valeur de la fonction Fonction_DC  = 34

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   2
   Les processus :  0    1
   Les objets :  101    104

 GROUPE   1  ::    nbproc =   1 nbobj =   4
   Les processus :  2
   Les objets :  106    107    102    105

Quel objet voulez vous deplacer ( 0 = fin ) ? 0

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 2
La valeur de la fonction Nombre_Liais = 14
La valeur de la fonction Fonction_DC  = 34

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   2
   Les processus :  0    1
   Les objets :  101    104

 GROUPE   1  ::    nbproc =   1 nbobj =   4
   Les processus :  2
   Les objets :  106    107    102    105

Voulez vous sauvgarder le groupement precedent (y/n) ?  y

Voulez vous visualiser les differentes evaluations des groupes (y/n) ? y

 Les Differentes Evaluation

Groupe   Charge_Pr  Charge_Ob  Nombre_Liais  Fonction_Dep
  0        1          2          17            25
  1        1          2          16            34
  2        1          2          14            34

Voulez vous visualiser les differents regroupements (y/n) ? y

GROUPEMENT   0 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  0    2
   Les objets :  101    102    106    107

 GROUPE   1  ::    nbproc =   1 nbobj =   2
```

```
   Les processus :  1
   Les objets :  104    105

GROUPEMENT   1 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  0    1
   Les objets :  101    102    104    105

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  2
   Les objets :  106    107

GROUPEMENT   2 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   2
   Les processus :  0    1
   Les objets :  101    104

 GROUPE   1  ::    nbproc =   1 nbobj =   4
   Les processus :  2
   Les objets :  106    107    102    105

Voulez vous prendre une decision (y/n) ?  n

Vouley vous modifier le regroupement des processus (y/n) ? y
Le nombre des processus du groupe  0 : 2
Les processus : 1 2
Le nombre des processus du groupe  1 : 1
Les processus : 0

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  1    2
   Les objets :  104    105    106    107

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  0
   Les objets :  101    102

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 2
La valeur de la fonction Nombre_Liais = 17
La valeur de la fonction Fonction_DC  = 25

Voulez vous sauvgarder le groupement precedent (y/n) ?  y

Voulez vous deplacer les objets (y/n) ?  y

Quel objet voulez vous deplacer ( 0 = fin ) ? 106

A quel groupe ? 1

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 0
La valeur de la fonction Nombre_Liais = 18
La valeur de la fonction Fonction_DC  = 25

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   3
```

```
   Les processus :  1    2
   Les objets :  104    105    107

 GROUPE   1  ::    nbproc =   1 nbobj =   3
   Les processus :  0
   Les objets :  101    102    106

Quel objet voulez vous deplacer ( 0 = fin ) ? 0

La valeur de la fonction Charge_Proc  = 1
La valeur de la fonction Charge_Objet = 0
La valeur de la fonction Nombre_Liais = 18
La valeur de la fonction Fonction_DC  = 25

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   3
   Les processus :  1    2
   Les objets :  104    105    107

 GROUPE   1  ::    nbproc =   1 nbobj =   3
   Les processus :  0
   Les objets :  101    102    106

Voulez vous sauvgarder le groupement precedent (y/n) ?  y

Voulez vous visualiser les differentes evaluations des groupes (y/n) ? y

 Les Differentes Evaluation

Groupe   Charge_Pr  Charge_Ob  Nombre_Liais  Fonction_Dep
  0        1          2          17            25
  1        1          2          16            34
  2        1          2          14            34
  3        1          2          17            25
  4        1          0          18            25

Voulez vous visualiser les differents regroupements (y/n) ? y

GROUPEMENT   0 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  0    2
   Les objets :  101    102    106    107

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  1
   Les objets :  104    105

GROUPEMENT   1 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  0    1
   Les objets :  101    102    104    105

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  2
   Les objets :  106    107

GROUPEMENT   2 :

  Les groupes
```

```
 GROUPE   0  ::    nbproc =   2 nbobj =   2
   Les processus :  0    1
   Les objets :  101    104

 GROUPE   1  ::    nbproc =   1 nbobj =   4
   Les processus :  2
   Les objets :  106    107    102    105

GROUPEMENT   3 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   4
   Les processus :  1    2
   Les objets :  104    105    106    107

 GROUPE   1  ::    nbproc =   1 nbobj =   2
   Les processus :  0
   Les objets :  101    102

GROUPEMENT   4 :

  Les groupes

 GROUPE   0  ::    nbproc =   2 nbobj =   3
   Les processus :  1    2
   Les objets :  104    105    107

 GROUPE   1  ::    nbproc =   1 nbobj =   3
   Les processus :  0
   Les objets :  101    102    106

Voulez vous prendre une decision (y/n) ?  y

Quel regroupement voulez vous garder ? 3

Fonction_Dep Charge_Pr  Charge_Ob  Nombre_Liais  Fonction_DC
  42            1          2          17            25


Voulez vous visuliser les differentes tables (y/n) ? n

 Le fichier de configuration est:      factoriel2.cfg
 Le fichier d'execution est:           factoriel2.m
```

Le fichier de configuration résultant

```
!  Fichier de configuration
!
!  Les processeurs
!
processor  SITE1
processor  SITE2
processor  host
processor  root
!
!  Les liens physiques
!
wire ? host[0]  root[0]
wire ? SITE2[0]  SITE1[0]
!
!  Les processus
!
task  afserver  ins=1   outs=1  data =10K
task  filter    ins=2   outs=2  data =10K
task  iohand    ins=3   outs=3  data =10K
task  P1        ins=1   outs=3  data =10K
task  P2        ins=1   outs=3  data =10K
task  Q         ins=4   outs=4  data =10K
task  diffus    ins=1   outs=2  data =10K
task  gbi_al    ins=2   outs=1  data =10K
task  gbi_al1   ins=2   outs=1  data =10K
task  diffus1   ins=1   outs=2  data =10K
task  gbi_al2   ins=2   outs=1  data =10K
task  gbi_al3   ins=2   outs=1  data =10K
task  gbi_eg    ins=2   outs=1  data =10K
task  gbi_eg1   ins=2   outs=1  data =10K
!
! Les placement
!
place afserver  host
place filter    root
place iohand    root
place P2        SITE1
place Q         SITE1
place P1        SITE2
place diffus    SITE2
place gbi_al    SITE2
place gbi_al1   SITE2
place diffus1   SITE1
place gbi_al2   SITE1
place gbi_al3   SITE1
place gbi_eg    SITE1
place gbi_eg1   SITE1
!
!  Les liens logiques
!
connect ?  afserver[0]   filter[0]
connect ?  filter[0]     afserver[0]
connect ?  filter[1]     iohand[1]
connect ?  iohand[1]     filter[1]
connect ?  P1[1]         diffus[0]
connect ?  diffus[0]     gbi_al[1]
connect ?  P2[0]         gbi_al[0]
connect ?  gbi_al[0]     P2[0]
connect ?  diffus[1]     gbi_al1[1]
connect ?  Q[0]          gbi_al1[0]
connect ?  gbi_al1[0]    Q[0]
connect ?  P2[1]         diffus1[0]
connect ?  diffus1[0]    gbi_al2[1]
connect ?  P1[0]         gbi_al2[0]
connect ?  gbi_al2[0]    P1[0]
connect ?  diffus1[1]    gbi_al3[1]
```

```
connect ?  Q[1]           gbi_al3[0]
connect ?  gbi_al3[0]     Q[1]
connect ?  P1[2]          gbi_eg[1]
connect ?  Q[2]           gbi_eg[0]
connect ?  gbi_eg[0]      Q[2]
connect ?  P2[2]          gbi_eg1[1]
connect ?  Q[3]           gbi_eg1[0]
connect ?  gbi_eg1[0]     Q[3]
! Fin de la configuration ......
```

Le fichier make résultant

```
PATH_C = /3l/tc2v0/exe
PATH_T = /3l/tnt/exe
PATH_BIN = /3l/tnt/bin
factor.app : factor.nwk iohand.b4 P1.b4 P2.b4 Q.b4 gbi_a1.b4 gbi_a11.b4 gbi_a12.b4
        cp $(PATH_BIN)/filter.b4 .
        cp $(PATH_BIN)/gloader.b4 .
        $(PATH_T)/config factor.nwk factor.app
        rm *.b4 *.bin
iohand.b4 : iohand.bin
        $(PATH_C)/t4cstask iohand
P1.b4 : P1.bin
        $(PATH_C)/t8cstask P1
P2.b4 : P2.bin
        $(PATH_C)/t8cstask P2
Q.b4 : Q.bin
        $(PATH_C)/t8cstask Q
gbi_eg.b4 : gbi_eg.bin
        $(PATH_C)/t8cstask gbi_eg
        cp gbi_eg.b4 gbi_eg1.b4
gbi_a1.b4 : gbi_a1.bin
        $(PATH_C)/t8cstask gbi_a1
        cp gbi_a1.b4 gbi_a11.b4
        cp gbi_a1.b4 gbi_a12.b4
        cp gbi_a1.b4 gbi_a13.b4
diffus.b4 : diffus.bin
        $(PATH_C)/t8cstask diffus
        cp diffus.b4 diffus1.b4
iohand.bin : iohand.c
        $(PATH_C)/t8c iohand
P1.bin : P1.c
        $(PATH_C)/t8c P1
P2.bin : P2.c
        $(PATH_C)/t8c P2
Q.bin : Q.c
        $(PATH_C)/t8c Q
gbi_eg.bin : gbi_eg.c
        $(PATH_C)/t8c gbi_eg
gbi_a1.bin : gbi_a1.c
        $(PATH_C)/t8c gbi_a1
diffus.bin : diffus.c
        $(PATH_C)/t8c diffus
factor.nwk : factor.cfg
        $(PATH_T)/T_Config factor
```

ANNEXE A

Les différentes tâches de la couche de communication

```
/*                                                                    */
/*     LA TACHE BIPOINT OU MUL-PROD DE LA COUCHE DE COMMUNICATION     */
/*                                                                    */
#include <thread.h>
#include <chan.h>
#include <sema.h>
#include "gestion.c"

ENS_IND ens_ind, *ptind;
SEMA  buf_free;
CHAN  **in_p, **out_p;

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc, ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports, *out_ports;
{
   extern void recevoir();

   int i, mass;

   sema_init(&buf_free, 1);

   in_p = in_ports;
   out_p = out_ports;
   ptind = &ens_ind;
   Init_EI(ptind);
   thread_create (recevoir,50*sizeof(int),1,1);
   thread_create (recevoir,50*sizeof(int),1,0);
}

void recevoir(i)
int i;
{
   int message, mess;

   for (;;) {
       chan_in_word (&message, in_p[i]);
       sema_wait(&buf_free);
          if ( i == 0 )
            { mess = Ret_Egal(ptind);
              chan_out_word(mess, out_p[0]);
            }
          else Ajouter(ptind,message);
       sema_signal(&buf_free);
      }
 }
```

```
/*                                                                    */
/*     LA TACHE DIFFUSEUR DE LA COUCHE DE COMMUNICATION               */
/*                                                                    */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  int     i, mess;

  /* INITIALISATION */

  mess = 1;

  /* La production des messages selon la demande et les envoyer localement */

  while ( mess != FIN )
    { chan_in_word( &mess, in_ports[0]);
      for (i = 0; i < outs; i++)
        chan_out_word( mess, out_ports[i]);
    }
}
```

```
/*                                                                    */
/*     LA TACHE BIPOINT OU MUL-CONS DE LA COUCHE DE COMMUNICATION     */
/*                                                                    */
#include <thread.h>
#include <chan.h>
#include <sema.h>
#include "gestion.c"

ENS_IND ens_ind, *ptind;
SEMA  buf_free;
CHAN  **in_p, **out_p;

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc, ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports, *out_ports;
{
   extern void recevoir();

   int i, mass;

   sema_init(&buf_free, 1);

   in_p = in_ports;
   out_p = out_ports;
   ptind = &ens_ind;
   Init_EI(ptind);
   thread_create (recevoir,50*sizeof(int),1,1);
   thread_create (recevoir,50*sizeof(int),1,0);
}

void recevoir(i)
int i;
{
   int message, mess;

   for (;;) {
       chan_in_word (&message, in_p[i]);
       sema_wait(&buf_free);
          if ( i != 0 )
            { mess = Ret_Egal(ptind);
              chan_out_word(mess, out_p[i]);
            }
          else Ajouter(ptind,message);
       sema_signal(&buf_free);
      }
 }
```

```
/*                                                                    */
/*     LA TACHE SELECTEUR DE LA COUCHE DE COMMUNICATION               */
/*                                                                    */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  int     i, mess, req, canal;

  /* INITIALISATION */

  req = 1;
  chan_in_word( &canal, in_ports[0])

  while ( canal != FIN )
    { chan_out_word( req, out_ports[canal]);
      chan_in_word( &mess, in_ports[canal]);
      chan_out_word( mess, out_ports[0]);
      chan_in_word( &canal, in_ports[0])
      }
}
```

ANNEXE B

Les différentes tâches de l'application répartie correspondant à l'application parallèle de calcul de PGCD

```
/*                          */
/* Une tache d'interface    */
/*                          */
#include <chan.h>
#include <stdio.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc, ins, outs ;
char * argv[], * envp[] ;
CHAN * in_ports[], * out_ports[] ;

{
   int    nb[10], p, pgcd, i, nombre;

   printf(" Ce programme est configure pour calcule le pgcd de 4 valeurs \n");
   nombre = 4;
   for ( i = 0; i < nombre; i++ )
     { printf("Entrez une valeur: ");
       scanf("%3d",&nb[i]);
       getchar();
     }
   for ( i = 0; i < nombre; i++ )
      chan_out_word ( nb[i], out_ports[2]);
   printf("J'ai envoye les valeurs \n");
   chan_in_word( &pgcd, in_ports[2]);
   chan_in_word( &p, in_ports[2]);
   printf(" La valeur de pgcd est : %3d \n", pgcd );
   printf(" Arret a la valeur premier: %3d \n", p );
 }
```

```
/*                                                  */
/* Une tache qui represente le processus P          */
/*                                                  */
#include <chan.h>
#include "gestion.c"

main (argc, argv, envp, in_ports, ins, out_ports, outs)
int argc, ins, outs ;
char * argv[], * envp[] ;
CHAN * in_ports[], * out_ports[] ;

{

   int    a[10], k, req, p, c, suite[50], max, fini, premier, i;

   p = 2;
   max = 0;
   suite[max] = p;

   for( i = 0; i < outs - 2; i++)
     { chan_in_word ( &a[i], in_ports[0]);
       k = i + 2;
       chan_out_word ( a[i], out_ports[k]);
     }
   chan_out_word (p, out_ports[0]) ;
   c = -1;
   while ( c == -1 )
     { chan_out_word ( req, out_ports[1]);
       chan_in_word ( &c, in_ports[1]);
     }

   while ( ( max < 50 ) && ( c == FALSE ))
     { fini = FALSE;
       while ( fini == FALSE )
         { i = 0;
           premier = TRUE;
           p = p + 1;
           while ( ( i < max ) && ( premier == TRUE ) )
              { if ( p == (( p / suite[i] ) * suite[i]) )
                  premier = FALSE;
                  i = i + 1;
              }
           if ( premier == TRUE )
             { max = max + 1;
               suite[max] = p;
               fini = TRUE;
               chan_out_word (p, out_ports[0]) ;
             }
         }
       c = -1;
       while ( c == -1 )
         { chan_out_word ( req, out_ports[1]);
           chan_in_word ( &c, in_ports[1]);
         }
     }
}
```

```
/*                                                              */
/* Une tache qui represente les processus P1, P2, P3, P4 */
/*                                                              */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  int   aj, ai, ak, c, p, r, req;

  /* INITIALISATION */

  req = 1;
  c = FALSE;       /* c = false */
  chan_in_word ( &ai, in_ports[1]);

  aj = ai;
  p = -1;                                  /* consommer p */
  while ( p == -1 )
    {  chan_out_word( req, out_ports[0]);
       chan_in_word( &p, in_ports[0]);
    }
  while ( p < aj)
    { r = 0;
      ai = aj;
      while ( ai == ( ai/ p ) * p )
        { ai = ai / p;
          r = r + 1;
        }
      chan_out_word( r, out_ports[1]);     /* produire r */
      chan_out_word( c, out_ports[2]);     /* produire c */
      p = -1;                              /* consommer p */
      while ( p == -1 )
        {  chan_out_word( req, out_ports[0]);
           chan_in_word( &p, in_ports[0]);
         }
    }
  c = TRUE;           /* c = true */
  chan_out_word( c, out_ports[2]);
}
```

```
/*                                                */
/* Une tache qui represente le processus Q */
/*                                                */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  int     j, req, p, r[10], cext[10], cc, pgcd, s, fini, i;
  int     facteur, nb_proc;

  /* INITIALISATION */
  req = 1;
  fini = FALSE;
  cc = FALSE;   /* c = false */
  pgcd = 1;
  nb_proc = ( ins - 2 ) / 2;



  p = -1;                                  /* consommer p */
  while ( p == -1 )
    {  chan_out_word( req, out_ports[0]);
       chan_in_word( &p, in_ports[0]);
     }
  chan_out_word (p, out_ports[1]) ;
  chan_out_word ( cc, out_ports[10]);  /* produire cc */
  for( i= 0; i < nb_proc; i++ )              /* consommer ci */
    { cext[i] = -1;
      j = i*2 + 3;
      while ( cext[i] == -1 )
        { chan_out_word( req, out_ports[j]);
          chan_in_word( &cext[i], in_ports[j]);
        }
      chan_out_word ( cext[i], out_ports[1]) ;
    }

  while ( fini == FALSE )
    { for( i= 0; i < nb_proc; i++ )         /* consommer ri */
        { j = i*2 + 2;
          r[i] = -1;
          while ( r[i] == -1 )
            { chan_out_word( req, out_ports[j]);
              chan_in_word( &r[i], in_ports[j]);
            }
         chan_out_word ( r[i], out_ports[1]) ;
         }
       s = r[0];                                  /* Le min */
       for( i= 0; i < 4; i++ )
          if ( r[i] < s ) s = r[i];
       facteur = 1;                             /* la puissanc\e */
       i = 0;
       while ( i < s )
         { facteur = facteur * p;
           i = i + 1;
         }
       pgcd = pgcd * facteur;
  chan_out_word (pgcd, out_ports[1]) ;

  p = -1;
  while ( p == -1 )
    {
      chan_out_word( req, out_ports[0]);
```

```
      chan_in_word( &p, in_ports[0]);
    }
  chan_out_word (p, out_ports[1]) ;
  for( i= 0; i < nb_proc; i++ )                    /* consommer ci \*/
     { cext[i] = -1;
       j = i*2 + 3;
       while ( cext[i] == -1 )
         { chan_out_word( req, out_ports[j]);
           chan_in_word( &cext[i], in_ports[j]);
          }
       chan_out_word ( cext[i], out_ports[1]) ;
      }
    for( i= 0; i < nb_proc; i++ )
       if ( cext[i] == TRUE ) fini = TRUE;
    }
    cc = TRUE;
    chan_out_word ( cc, out_ports[10]);
}
```

ANNEXE C

Les différentes tâches de l'application répartie correspondant à l'application parallèle de calcul de factoriel

```
/*                                                              */
/* Une tache d'interface avec le reseau des transputers         */
/*                                                              */
#include <chan.h>
#include <stdio.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc, ins, outs ;
char * argv[], * envp[] ;
CHAN * in_ports[], * out_ports[] ;

{
   Mess_Temp   n1, n2, f, f1, f2;
   int         nombre, req;

   n1 = 1;
   n2 = 50;
   printf(" Entrez la valeur: ");
   scanf("%3d",&nombre);
   getchar();
   chan_out_word( nombre, out_ports[2]);
   printf(" Pour voir la trace \n");
   chan_in_word( &f1, in_ports[2]);
   chan_in_word( &f2, in_ports[2]);
   chan_in_word( &n1, in_ports[2]);
   chan_in_word( &n2, in_ports[2]);
   printf(" Les valeurs de f1, f2, n1, n2 from Q: %3d %3d %3d %3d \n", f1, f2, n1,
   chan_in_word( &f, in_ports[2]);
   printf(" La valeur du factoriel est: %10d  \n", f );
 }
```

```
/*                                                    */
/* Une tache qui represente le processus P1           */
/*                                                    */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  Mess_Temp  n1, n2, f1;
  int        i, req;

  /* INITIALISATION */

  n1 = 1;
  f1 = 1;
  req = 1;
  chan_out_word( n1, out_ports[1]);     /* produire n1 */
  n2 = 0;                                /* consommer n2 */
  while ( n2 == 0 )
    {  chan_out_word( req, out_ports[0]);
       chan_in_word( &n2, in_ports[0]);
    }
  while ( n1 < n2 )
    { f1 = f1 * n1;
      n1 = n1 + 1;
      chan_out_word( n1, out_ports[1]);     /* produire n1 */
      n2 = 0;
      while ( n2 == 0 )
        {  chan_out_word( req, out_ports[0]);    /* consommer n2 */
           chan_in_word( &n2, in_ports[0]);
         }
    }
  chan_out_word( f1, out_ports[2]);
}
```

```
/*                                                    */
/* Une tache qui represente le processus P2           */
/*                                                    */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  Mess_Temp  n1, n2, f2;
  int        i, req;

  /* INITIALISATION */

  chan_in_word( &n2, in_ports[1]);
  f2 = 1;
  req = 1;
  chan_out_word( n2, out_ports[1]);     /* produire n2 */
  n1 = 0;
  while ( n1 == 0 )
    {  chan_out_word( req, out_ports[0]);   /* consommer n1 */
       chan_in_word( &n1, in_ports[0]);
     }
  while ( n1 < n2 )
    { f2 = f2 * n2;
      n2 = n2 - 1;
      chan_out_word( n2, out_ports[1]);     /* produire n2 */
      n1 = 0;
      while ( n1 == 0 )
        { chan_out_word( req, out_ports[0]);   /* consommer n1 */
          chan_in_word( &n1, in_ports[0]);
        }
    }
  chan_out_word( f2, out_ports[2]);         /* produire f2 */
}
```

```
/*                                          */
/* Une tache qui represente le processus Q */
/*                                          */
#include <chan.h>
#include "gestion.c"

main(argc, argv, envp, in_ports, ins, out_ports, outs)
int  argc,ins, outs;
char  *argv[], *envp[];
CHAN  *in_ports[], *out_ports[];
{
  Mess_Temp  n1, n2, f1, f2, f;
  int        req;

  /* INITIALISATION */
  req = 1;

  /* La consommation des messages selon la demande et les envoyer localement et la

  n1 = 1;
  n2 = 50;

  f1 = 0;                                /* consommer f1 */
  while ( f1 == 0 )
    {  chan_out_word( req, out_ports[2]);
       chan_in_word( &f1, in_ports[2]);
     }
  chan_out_word( f1, out_ports[4]);
  f2 = 0;                                /* consommer f2 */
  while ( f2 == 0 )
    {  chan_out_word( req, out_ports[3]);
       chan_in_word( &f2, in_ports[3]);
     }
  chan_out_word( f2, out_ports[4]);
  n1 = 0;                                /* consommer n1 */
  while ( n1 == 0 )
     {  chan_out_word( req, out_ports[0]);
        chan_in_word( &n1, in_ports[0]);
     }
  chan_out_word( n1, out_ports[4]);
  n2 = 0;                                /* consommer n2 */
  while ( n2 == 0 )
    {  chan_out_word( req, out_ports[1]);
       chan_in_word( &n2, in_ports[1]);
    }
  chan_out_word( n2, out_ports[4]);
  if ( n1 == n2 ) f = f1 * f2 * n1;
  else f = f1 * f2;
  chan_out_word ( f, out_ports[4]);
  chan_out_word( FIN, out_ports[0]);
  chan_out_word( FIN, out_ports[1]);
  chan_out_word( FIN, out_ports[2]);
  chan_out_word( FIN, out_ports[3]);
}
```

NOM DE L'ETUDIANT : IDLEBI NIBAL

NATURE DE LA THESE : DOCTORAT DE L'UNIVERSITE DE NANCY I
EN INFORMATIQUE

VU, APPROUVE ET PERMIS D'IMPRIMER

NANCY, le - 8 JAN. 1991 n° 8

LE PRESIDENT DE L'UNIVERSITE DE NANCY I

UNIVERSITE NANCY I
LE PRESIDENT
M. BOULANGER

Mots Clés

Application parallèle, concurrence, communication, asynchronisme, répartition, système répartis, architecture répartie, formalisation, abstraction, types abstraits, spécification, implantation, placement, règles heuristiques, dérivation, représentation, compilation.