Notion de Groupe pour la Bibliothèque PPCM

SANLAVILLE Rémy

Stage de 1ère année effectué au LIP
Laboratoire de l'Informatique du Parallélisme

du 15 Juin 1995 au 1er Août 1995

sous la direction de
SERGE MIGUET
Maître de conférence à l'École Normale Supérieure de Lyon

Table des matières

Remerciements

Voilà une tâche difficile que de remercier les personnes qui m'ont entouré pendant mon stage comme je le voudrais par écrit. En effet, il n'est pas aisé de trouver les mots qui conviennent pour exprimer toute ma reconnaissance; d'ailleurs en existent-ils ? Alors, j'espère que vous ne m'en voudrez pas si je vous dis simplement merci.

Merci à l'équipe PEPSI (N. BERT, J.M NICOD, J.M. PIERSON, S. MIGUET) du LIP à qui je pouvais m'adresser à tout moment et qui m'ont toujours accueillis comme étant quelqu'un de l'équipe. Je voudrais particulièrement dire un plus grand **MERCI** à mon directeur de stage qui m'a encadré et qui a su être à ma disposition malgré tout le travail qu'il avait.

Comme pour tout, la première fois est toujours important. Je peux vous dire que mon premier stage restera un très bon souvenir et l'impression d'avoir apporté quelque chose à un projet. En effet vous m'avez fait totalement confiance, vous m'avez laissé libre d'organiser mon travail et vous avez fait attention à toutes mes remarques (pertinentes ou non). Cela constituera un bagage utile pour la suite de mes études.

Pour finir, je voudrais aussi remercier C. NICHITIU pour ses précieux conseils en LaTeX qui m'ont permis de rédiger ce rapport et N. LEROY qui a bien voulu lire et corriger mes fote d'horthaugrafes.

Enfin, comme vous avez pu le comprendre :

Merci pour tout

Résumé

Ce rapport présente la notion de groupe pour la bibliothèque PPCM. Pour cela nous étudions dans un premier temps ce qu'est une bibliothèque par échanges de messages et pourquoi en existe-t-il. Trois exemples de ce type de bibliothèque : PVM, MPI (qui sont très répandues) et PPCM (qui est créée par PEPSI, une équipe du LIP) sont développés. Ensuite nous introduisons la notion de groupe et décrivons la façon dont nous avons incorporé cette notion à PPCM. Enfin, nous terminons par l'exemple exécuté sans et avec la notion de groupe.

Mots Clés
Groupe, Bibliothèque par échanges de messages, PPCM, PVM, MPI, PLRedistribution.

Chapitre 1

Introduction

1.1 Le parallélisme

1.1.1 Généralités

L'ancêtre de toutes les machines à calcul est sans aucun doute l'abaque (ou boulier) qui apparut en Chine environ au IX^{e} siècle av. JC. Il permet principalement d'effectuer des opérations arithmétiques élémentaires. Il doit cependant être manipulé entièrement à la main. En 1642, Blaise Pascal propose la première machine à calculer autonome. Il faudra attendre cependant les années quarante pour voir apparaître les premiers ordinateurs électromécaniques. Cette évolution a conduit avec les travaux d'Alan Turing au modèle de machine virtuelle de Von Neumann. La figure 1.1 présente les différentes caractéristiques du modèle. Cependant, avec l'évolution de la technique et des algorithmes, ce modèle a fini par être remis en cause. Les progrès technologiques ont permis de résoudre des problèmes plus complexes et plus gourmands en temps de calculs. Parallèlement, de nouveaux problèmes ou des évolutions de modèles comme par exemple en métérologie, sont apparus et demandent de nouvelles puissances de calculs. Or, des limites physiques telles la vitesse de la lumière pour le transport de l'information ou la limite technologique de l'intégration, font apparaître de fortes contraintes dans le modèle de Von Neumann, ce qui est le cas avec le goulot d'étranglement des entrées-sorties. Il a donc fallu proposer de nouveaux modèles d'architectures radicalement différentes afin de pouvoir fournir la puissance de calcul nécessaire à la résolution des nouveaux problèmes. On aboutit ainsi à la notion de parallélisme, c'est-à-dire de simultanéité.

1.1.2 Classification

On se réfère généralement à la classification de Flynn [4] qui permet de comparer les machines parallèles entre elles ainsi que de dégager de grandes tendances. Flynn distingue deux types de parallélisme :

- Parallélisme de *données*
- Parallélisme d'*instructions*

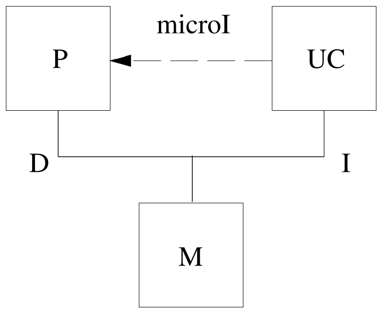

Fig. 1.1 - *Machine virtuelle de Von Neumann ; UC : Unité de commande ; P : Processeur ; M : Mémoire ; D : flot de données ; I : flot d'instructions*

Il propose ainsi quatre grands types de machines parallèles :

- SISD (*Single Instruction stream Single Data stream*) qui est en fait la machine séquentielle dérivée du modèle de Von Neumann.
- SIMD (*Single Instruction stream Multiple Data stream*). Ces machines telles que la MASPAR ou la CM-1, renferment un grand nombre de processeurs simples (par exemple 4 bits). Il n'y a qu'un compteur d'instruction et chaque processeur exécute en même temps la même instruction mais sur des données différentes. Les unités de traitement fonctionnent ici de manière *synchrones*.
- MISD (*Multiple Instruction stream Single Data stream*). Elles possèdent un seul flot de données mais il y a plusieurs unités de traitement qui effectuent des opérations différentes sur ce flot unique. Les réseaux systoliques appartiennent à cette classe de machines.
- MIMD (*Multiple Instruction stream Multiple Data stream*). La Paragon, l'iPSC et la CM-5 sont des exemples de machines MIMD. Sur de telles machines, chaque processeur exécute des instructions indépendantes sur des données indépendantes. Ces machines sont nécessairement *asynchrones* puisque chaque unité de traitement travaille indépendamment des autres.

Les modèles MIMD et SIMD sont les plus répandus pour les machines à vocation généraliste.

De plus, il faut ensuite distinguer deux types d'accès à la mémoire. Soit toute la mémoire est accessible arbitrairement par n'importe quel processeur, et dans ce cas il faut gérer les conflits d'accès, soit chaque processeur dispose de sa propre mémoire locale. Dans le premier type d'accès, on parle de machine parallèle à *mémoire partagée* et dans le second cas, de machine à *mémoire distribuée*. La mémoire distribuée pose de gros problèmes d'interconnexion entre les processeurs pour l'échange efficace d'informations. La figure 1.2 présente des modèles d'architectures SIMD à mémoire partagée ou distribuée. La figure 1.3 propose, quant à elle, des exemples d'architecture MIMD pour les deux types

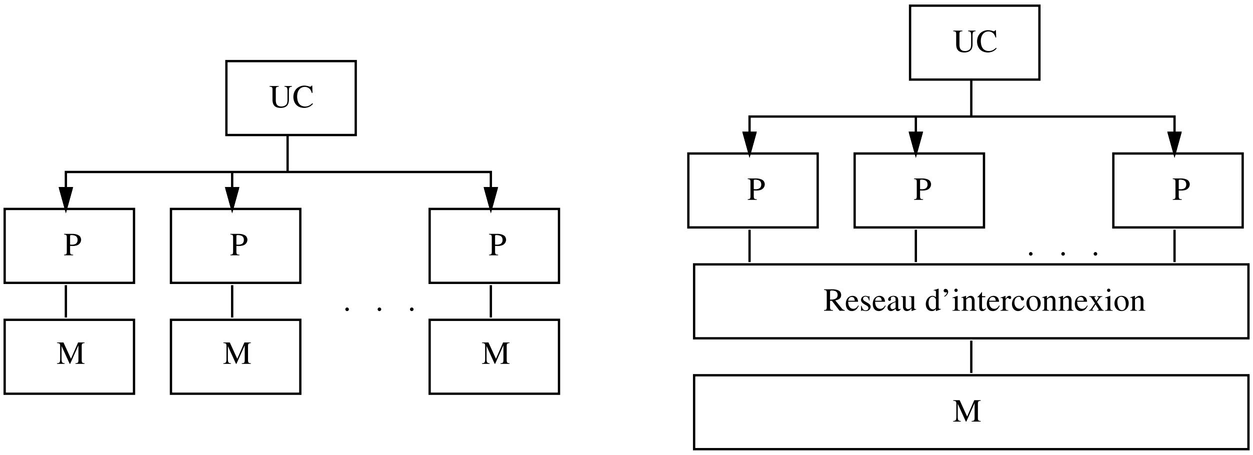

FIG. 1.2 - *Architectures de différentes machines SIMD*

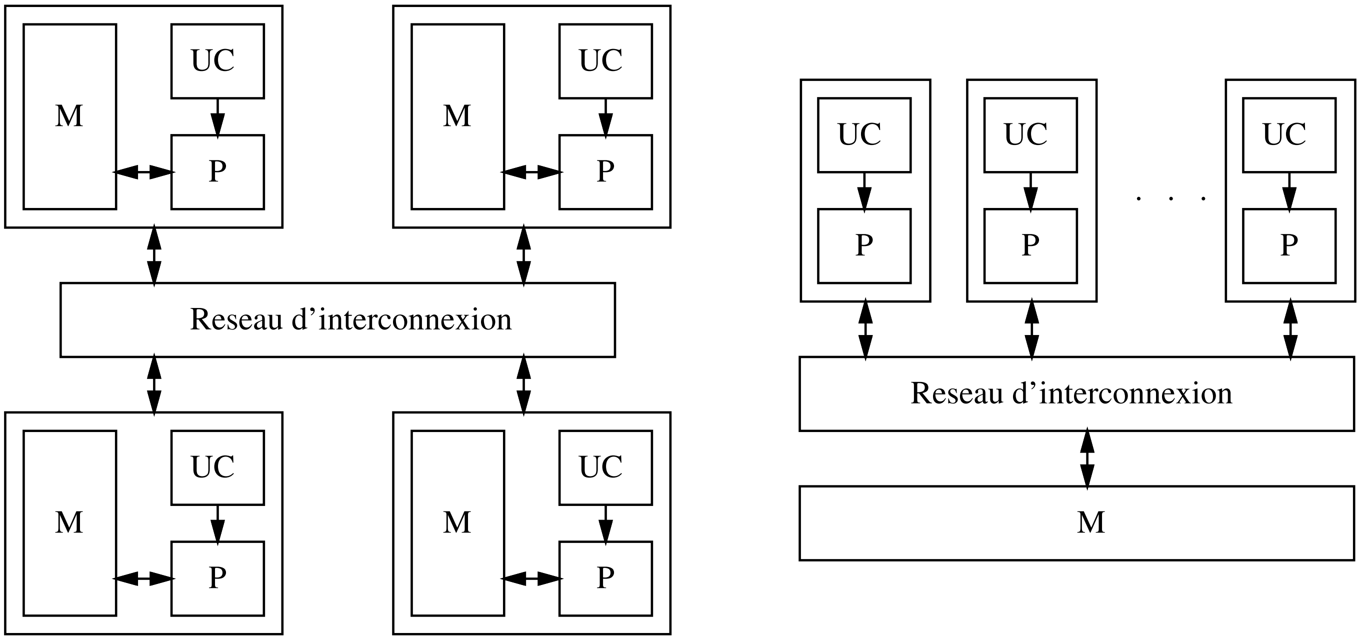

FIG. 1.3 - *Architectures de différentes machines MIMD*

d'accès mémoire.

Enfin, il existe plusieurs façon de relier les processeurs entre eux, on parle alors de topologie d'interconnexion. Les principales topologies sont :

1. Le réseau linéaire
2. L'anneau
3. La grille 2D
4. La grille torique 2D
5. L'hypercube

Les figures 1.4 et 1.5 présentent les diverses topologies classiques d'une machine parallèle.

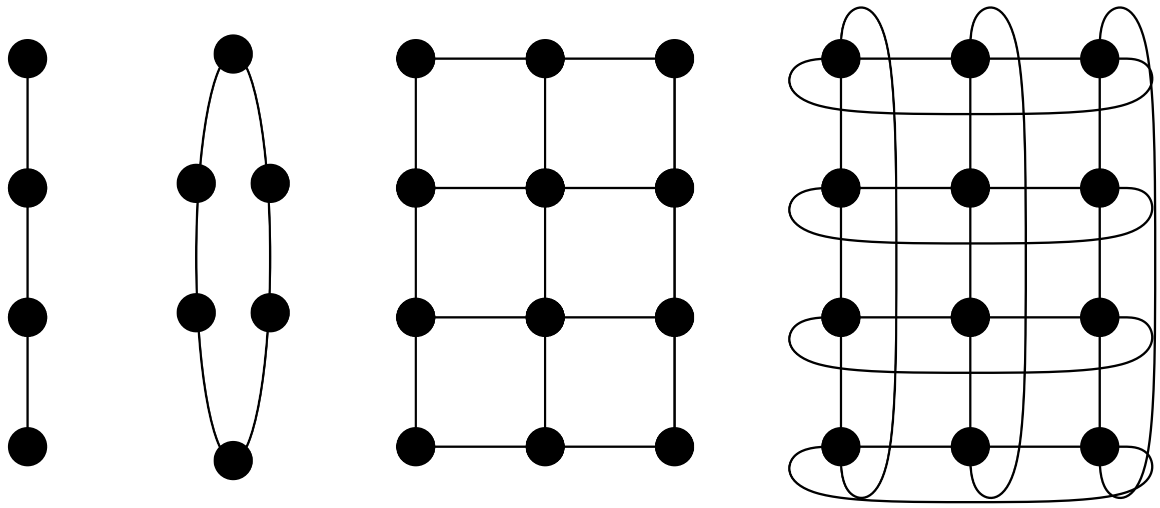

FIG. 1.4 - *Topologies élémentaires : réseau linéaire, anneau, grille 2D, grille torique 2D*

1.2 Les bibliothèques par échanges de messages

Le concept d'informatique distribuée s'est particulièrement développé ces dernières années. La création d'environnements de programmation dans le but de faire fonctionner un réseau d'ordinateurs hétérogènes comme une machine parallèle est devenue nécessaire. En effet, se servir d'un réseau d'ordinateurs pour simuler les dernières machines parallèles actuelles (à mémoire distribuée) conçues par les grands constructeurs comme Cray Research, IBM, Intel et Thinking Machines, rabaisse considérablement le coût des applications parallèles. Il faut cependant veiller à ce que ces applications soient directement portables sur une machine parallèle quelconque, moyennant une recompilation des programmes. Des universtités et des constructeurs ont alors développé des logiciels

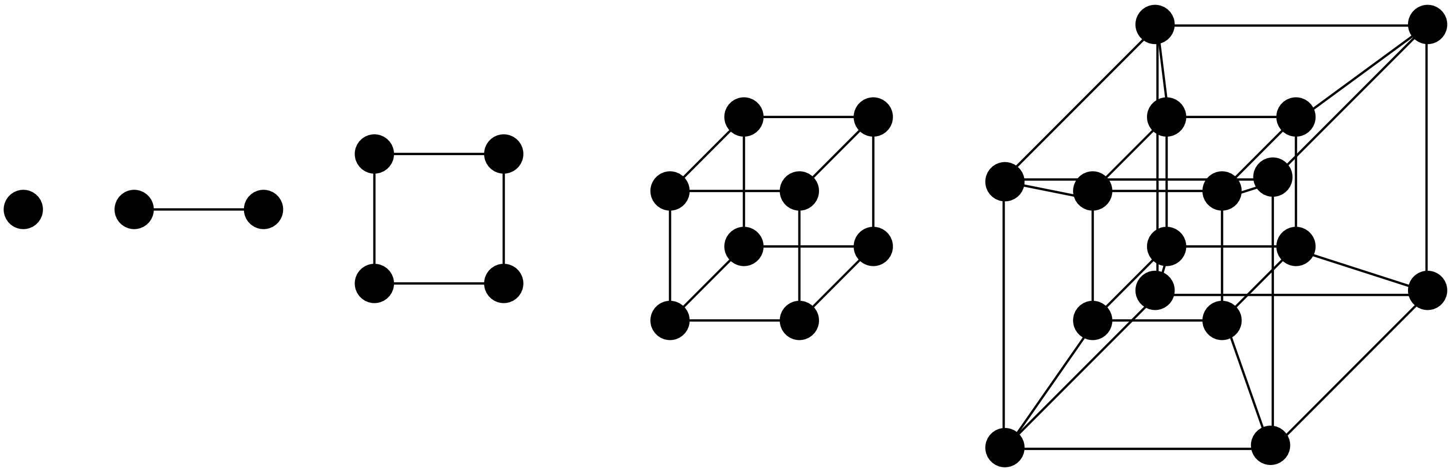

FIG. 1.5 - *Topologie d'hypercube pour les dimensions 1 à 4*

de programmation sur réseaux de machines hétérogènes (parallèles, vectorielles, RISC ...) appelés "Bibliothèques de Communication par Echanges de Messages" ou "Bibliothèques de Message Passing". Parmi les plus utilisées on trouve PVM (Parallel Vitual Machine), PICL (Portable Instrumented Communication Library), P4, Parmacs ou encore MPI (Message Passing Interface).

Chapitre 2

PVM, MPI et PPCM

2.1 PVM : Parallel Virtual Machine

PVM est l'un des environnements de programmation parallèle les plus utilisés actuellement. Développés par des chercheurs de l'université du Tennessee et du "Oak National Laboraty", PVM est facile à mettre en oeuvre.

2.1.1 Caractéristiques

D'après l'article de Manchek [6] et l'article de Geist et Sunderam [5], nous pouvons dégager les grandes lignes de PVM :

- Langage parallèle constitué du langage C et de fonction de communication, accessibles de manière transparente en Fortran et en C++.
- Primitives de communications, de synchronisations par barrière et de gestion des processus [1].
- Placement et activation de processus explicites ou gérés par le système ⇒ facilité pour les systèmes avec réseaux.
- Utilisable avec des machines hétérogènes (CRAY, station HP, IBM, SGI, Sun(s) ...).
- Ensemble d'outils pour le débuggage.
- Grandes similitudes avec MPI ⇒ portage assez facile de PVM vers MPI.
- Gestion de processus : identification des processus (tid [2]), contrôle des processus.
- Envoi de signaux.
- Groupes dynamiques.

1. Cf. Annexe A
2. Numéro unique pour chaque processeur attribué par le démon de PVM

2.2 MPI : Message Passing Interface

MPI est une bibliothèque de communication par échange de messages ayant pour but de rassembler les meilleurs éléments des systèmes déjà existants plutôt que d'en choisir un.

2.2.1 Caractéristiques

Comme le montre C. Calvin [2], les spécificités de MPI sont :

- Communication point à point [1].
- Opérations collectives [1].
- Notion de groupes de processus.
- Notion de contextes associés à une communication.
- Topologie virtuelle de processus.

Ce qui n'existe pas encore et qui sera dans le futur MPI :

- Des fonctions d'entrées/sorties parallèles.
- Opérations explicites de mémoire partagée.
- Des opérations nécessitant une utilisation plus que standard du système d'exploitation, comme par exemple les messages actifs ou l'exécution distante.
- Des outils de construction de programmes.
- Des facilités de debug.
- Des fonctions auxiliaires comme pour la prise de temps.
- Le support explicite de processus légers (au sens Unix).
- Le support pour la gestion des tâches.

Cette bibliothèque est performante mais elle présente l'inconvénient d'être assez lourde d'utilisation.

1. Cf. Annexe A

2.3 PPCM : Portable Parallel Communication Module

Cette bibliothèque est en cours d'élaboration et elle est développée par l'équipe PEPSI du LIP. Elle essaye de regrouper ce qui existe déjà et d'ajouter quelques fonctionnalités comme l'équilibrage de charge, ce qui fait sa grande originalité par rapport aux autres bibliothèques.

2.3.1 Structure/Arborescence

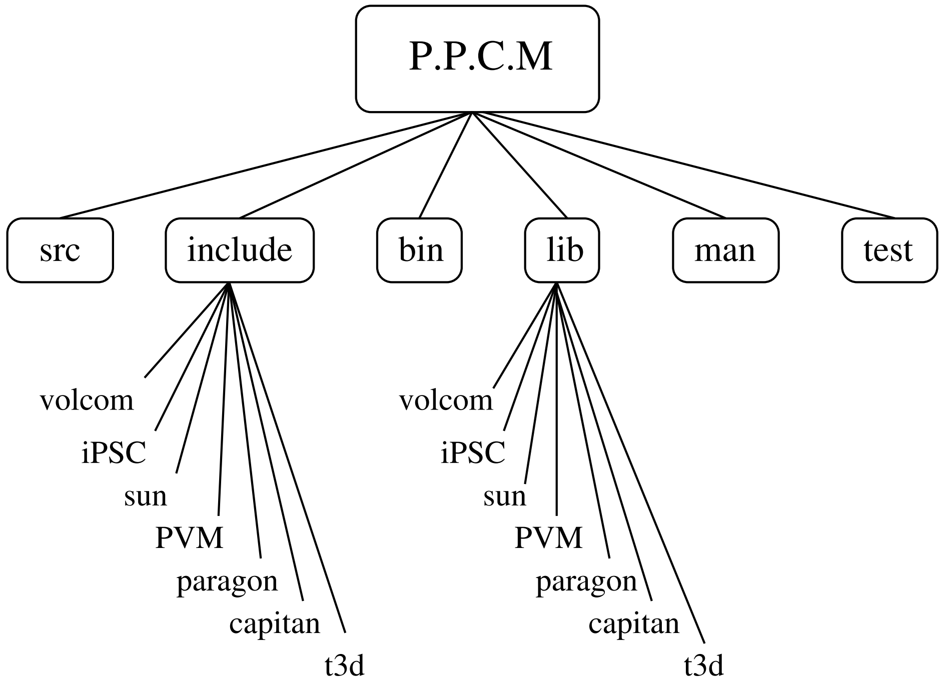

FIG. 2.1 - *Arborescence de PPCM*

Comme le stipule N. Bert [1], nous pouvons représenter l'arborescence de PPCM comme le montre la figure 2.1.

- **src** contient les sources de la librairie PPCM et de PPCMX: les instructions du langage.
- **include** contient les includes, qui déclarent les instructions et définissent les macros-instructions dans le langage de la machine cible.
- **bin** contient les commandes utilisables depuis le shell.
- **lib** est composée de deux répertoires par machine : t3d et t3d.trace. Ces répertoires contiennent le complémentaire des scripts décrits dans *bin*, mais spécifique à chaque machine.
- **man** contient l'aide sur chaque instruction ou macro-instruction.

- **test**. Quelques programmes pour tester soit la validité du portage de PPCM sur une nouvelle machine, soit pour tester et comprendre les instructions du langage.

2.3.2 Caractéristiques

- Fonctions de communications.
- Un environnement de programmation.
- Portabilité sur des machines hétérogènes.
- Gestion des processus.
- Primitives graphiques portables.
- Equilibrage de charge.

Cette bibliothèque à l'avantage d'être efficace et simple d'utilisation. Par contre, elle lui manque une notion importante : la notion de groupes de processeurs.

Chapitre 3

La notion de groupe

3.1 Introduction

Lorsque nous utilisons un modèle de programmation par processus communicants, il est souvent pratique de pouvoir regrouper certains processus ensemble car ils possèdent une caractéristique commune. Par exemple, quelques processus sont chargés d'effectuer un même calcul, alors que les autres processus sont occupés à une autre tâche. Un groupe est communément représenté par un nom unique.

Ainsi avec la notion de groupe, il est possible d'effectuer des diffusions implicites [1]. C'est à dire que l'on ne diffuse plus à une liste de processus, mais à un nom de groupe. On peut aussi, très facilement synchroniser tous les processus d'un groupe par une barrière de synchronisation [1] s'appliquant uniquement à ce groupe.

La gestion des groupes peut être statique, c'est à dire que les groupes sont définis une fois pour toute en début d'exécution des différents processus. On dit qu'elle est dynamique si un processus peut à tout moment rejoindre ou quitter un groupe. Dans ce dernier cas, on doit se poser la question de savoir si la "vision" locale d'un groupe est toujours exacte. En effet, il se peut qu'une diffusion [1] (ou autre schéma global) atteigne plus de processus que prévus (ou moins) si, entre le début et la fin de la communication, le nombre de processus appartenant au groupe considéré a changé. Certaines implantations synchronisent les processus avant toute communication globale à l'intérieur du groupe afin d'éviter ce genre de problème. Sinon, il faut en être conscient, et le prévenir soi-même.

3.2 Avantages

On pourrait se demander pourquoi la notion de groupe est si importante et avantageuse (donc presque indispensable) dans les modèles de programmation par processus communicants. En fait, comme nous l'avons vu dans la section

1. Cf. Annexe A

précédente, elle permet d'affecter une tâche différente pour chaque groupe; ainsi elle est très utile lors de la répartition du travail, ce qui est le cas pour la synthèse d'image.

Lorsque nous n'avions pas la notion de groupe, la répartition de l'image se faisait suivant les lignes : on divise le nombre de lignes par le nombre de processeurs. Donc, plus il y aura de processeurs, plus la région de l'image attribuée sera étirée [1]. Alors qu'avec la notion de groupe, nous pouvons répartir l'image suivant les lignes et les colonnes, ce qui permet d'avoir des régions à peu près carrées. A ce stade, on peut encore se demander pourquoi il est avantageux d'avoir des régions carrées? En fait, la réponse est assez simple. Cela provient de la forme, souvent hexagonale, des dessins représentés dans l'image. Ainsi, dans des applications comme le Z-Buffer, ceci permettrait au processeur d'avoir le maximum d'informations sur l'objet et donc éviter beaucoup d'échanges avec les processeurs voisins ce qui représente un gain de temps important. Dans un certain sens la charge de travail est mieux répartie, avant même d'effectuer l'équilibrage de charge.

3.3 La notion de groupe pour les bibliothèques par échanges de messages

3.3.1 PVM

Une gestion dynamique des groupes de processus est possible avec PVM. En effet, lors de l'exécution d'un programme, des processus peuvent changer de groupe ou faire partie de plusieurs groupes afin de bénéficier d'informations diffusées à l'intérieur même d'un groupe. Les groupes sont utiles : ils peuvent servir pour définir la portée d'une barrière ou d'une diffusion [2]. Une barrière ne peut se faire qu'entre des processus appartenant à un même groupe.

Les processus sont enregistrés comme appartenant à un ou plusieurs groupes de processus. Les fonctions fournies par la bibliothèque libgpvm.a sont :

```
int inum = pvm_joingroup(char *group)
int info = pvm_lvgroup(char *group)

int tid = pvm_gettid(char *group, int inum)
int inum = pvm_getinst(char *group, int tid)

int size = pvm_gsize(char *group)

int info = pvm_gettid (char *group, int count)
```

1. Cf. exemple p14
2. Cf. Annexe A

3.3.2 MPI

Un groupe de processus est un ensemble ordonné de processus où chacun est identifié de manière unique par son rang dans le groupe. Pour un groupe de n processus, le rang va de 0 à n-1. Les groupes de processus peuvent être utilisés de deux manières différentes :

1. La première pour spécifier les processus qui sont utilisés par une communication globale [1], comme une diffusion.
2. La deuxième pour introduire la notion de parallélisme de tâches dans une application.

Bien que le modèle de processus MPI soit statique, les groupes de processus sont gérés dynamiquement, dans le sens où ils peuvent être créés et détruits, et où chaque processus peut appartenir à plusieurs groupes à la fois.

3.3.3 PPCM

En ce qui concerne la bibliothèque PPCM, la notion de groupe se rapproche de celle de MPI. En effet, un groupe de processus est aussi un ensemble ordonné de processus où chacun est identifié de manière unique par son rang dans le groupe. Pour un groupe de n processus, le rang va de 0 à n-1. De plus, comme MPI, bien que le modèle de processus MPI soit statique, les groupes de processus sont gérés dynamiquement, dans le sens où ils peuvent être créés et détruits, et où chaque processus peut appartenir à plusieurs groupes à la fois.

Pour représenter la notion de groupe, il a fallut modifier les fichiers ppcm.h et ring.h [2]. Pour cela, nous avons créé un structure que nous avons appelé PpcmGroup, définie de la manière suivante :

```
typedef struct
  {
      int procnb;
      int position;
      int succ;
      int pred;
  } PpcmGroup;
```

- procnb étant le nombre de processeur dans le groupe.
- position étant le numéro du processeur dans le groupe.
- succ étant le numéro physique (et non du groupe) du processeur suivant.

1. Cf. Annexe A
2. Cf. Annexe B

- pred étant le numéro physique (et non du groupe) du processeur précédent

Nous avons aussi créé deux variables utilisables par le programmeur :

1. **Physical** de type PpcmGroup qui est le groupe qui contient tous les processeurs. Elle représente l'état initiale (comme lorsque nous n'avions pas de groupe).
2. **Courant** qui est un pointeur sur le type PpcmGroup. Elle a pour rôle d'affecter les opérations sur le groupe en cours.

Ensuite, nous avons dû modifier certaines macros :

- PpcmInit() où nous avons rajouté l'initialisation du groupe Physical et où nous l'avons mis comme groupe courant.
- Succ(), Pred(), Position()et ProcNb() qui vont chercher leur information dans la variable GroupeCourant. Les anciennes macros ont étés renommées P_Succ(), P_Pred(), P_Position() et P_ProcNb();

Enfin, il a fallut intégrer des macro :

```
PpcmCreateGroup(pos,pnb,s,p,groupe)
PpcmSetGroup(groupe)
PpcmCheckGroup(groupe)
```

- PpcmCreateGroup permet de créer un nouveau groupe.
- PpcmSetGroup permet de mettre le groupe voulu en tant que groupe courant.
- PpcmCheckGroup vérifie si le groupe est correct.

Ce qui était important lors de l'ajout des groupes dans la bibliothèque, c'est qu'il fallait que toute les autres applications continuent à fonctionner sans pour cela les modifier. Ceci a pu être réalisé grâce à la création d'un groupe Physical et à l'initialisation au préalable du groupe Courant au groupe Physical.

Chapitre 4

Un exemple

4.1 Introduction

Pour notre exemple, nous avons pris comme image un cercle rempli en noir (figure 4.1). Le fichier qui contient l'image est un fichier rasterfile 8 bits. Le but de cet exemple est de répartir au mieux la charge de chaque processeur. Pour calculer la charge de travail, nous comptabilisons le nombre de pixel noir que contient chaque processeur.

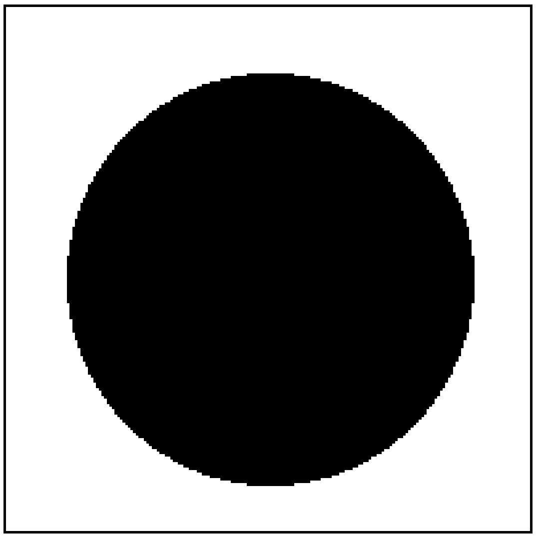

FIG. 4.1 - *Image de départ*

Nous allons étudier deux algorithmes[1] : un sans la notion de groupe et l'autre avec. Cela nous permettra de voir les avantages de la notion de groupe.

Nous pourrons également vérifier que les primitives intégrées dans la bibliothèque fonctionnent correctement (pour la notion de groupe) et en même temps nous assurer du bon fonctionnement de PLREDISTRIBUTION[2].

1. Les listing sont en Annexe D
2. Cf. Annexe C

4.2 Sans la notion de groupe

4.2.1 Méthodologie

- Au début, nous répartissons l'image en région égale pour chaque processeur (figures 4.2 et 4.6).
- Ensuite, nous calculons la charge de travail pour chaque processeur afin de chercher les nouvelles régions pour les processeurs (figure 4.3).
- Enfin, nous redistribuons les lignes entre les processeurs (figures 4.4 et 4.7) et nous recalculons la charge de travail pour chaque processeur (figure 4.5).

4.2.2 Résultats

Grâce aux figures 4.3 et 4.5, nous pouvons constater que l'équilibrage de charge donne des résultats satisfaisants.

Comme nous l'avions prévu, l'image de chaque processeur est "étirée", ce qui ne constitue pas des conditions favorables pour l'utilisation d'algorithmes en synthèse d'image. En effet, comme les dessins contenues dans l'image sont souvent hexagonaux, ils seront divisés entre plusieurs processeurs. Ainsi, lors du traitement d'un dessin, le processeur n'aura pas toutes les informations souhaitées et devra les demander à ses processeurs voisins. Il serait donc plus intéressant que les processeurs possèdent une image presque carrée.

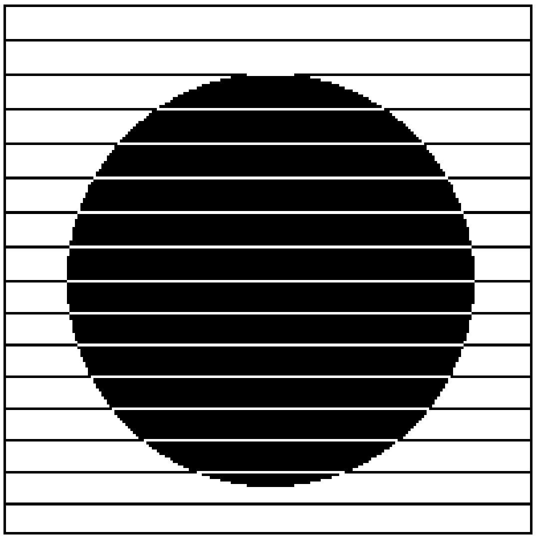

FIG. 4.2 - *Image répartie entre les processeurs*

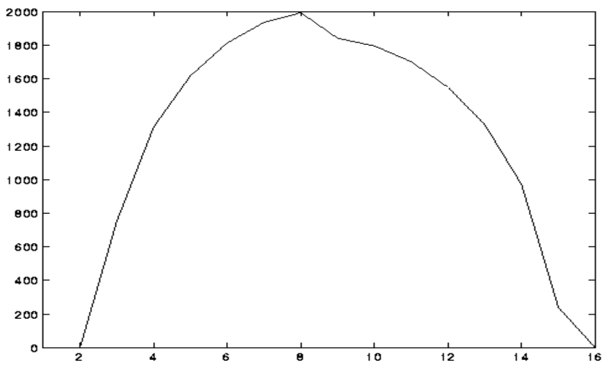

FIG. 4.3 - *Charge de travail au début*

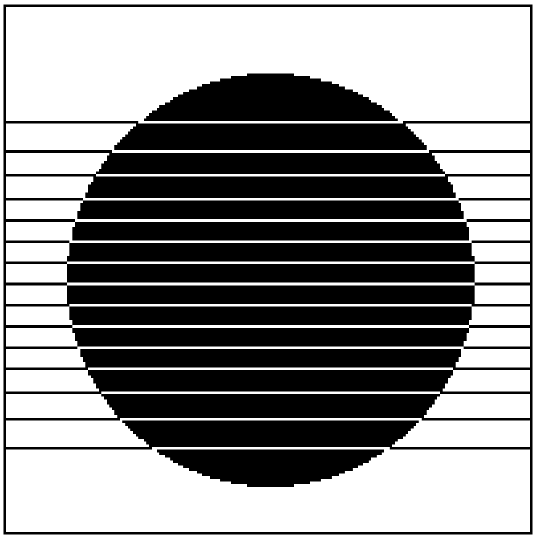

FIG. 4.4 - *Image après redistribution des lignes*

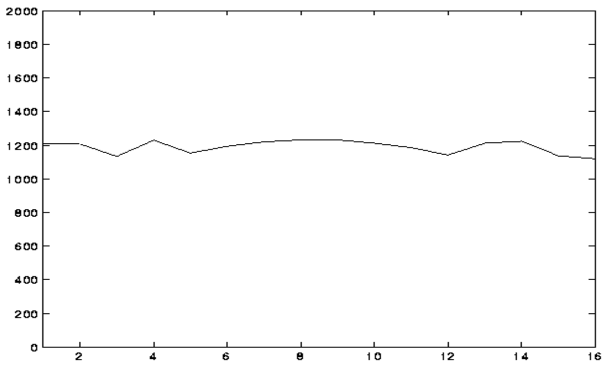

FIG. 4.5 - *Charge de travail après la redistribution des lignes*

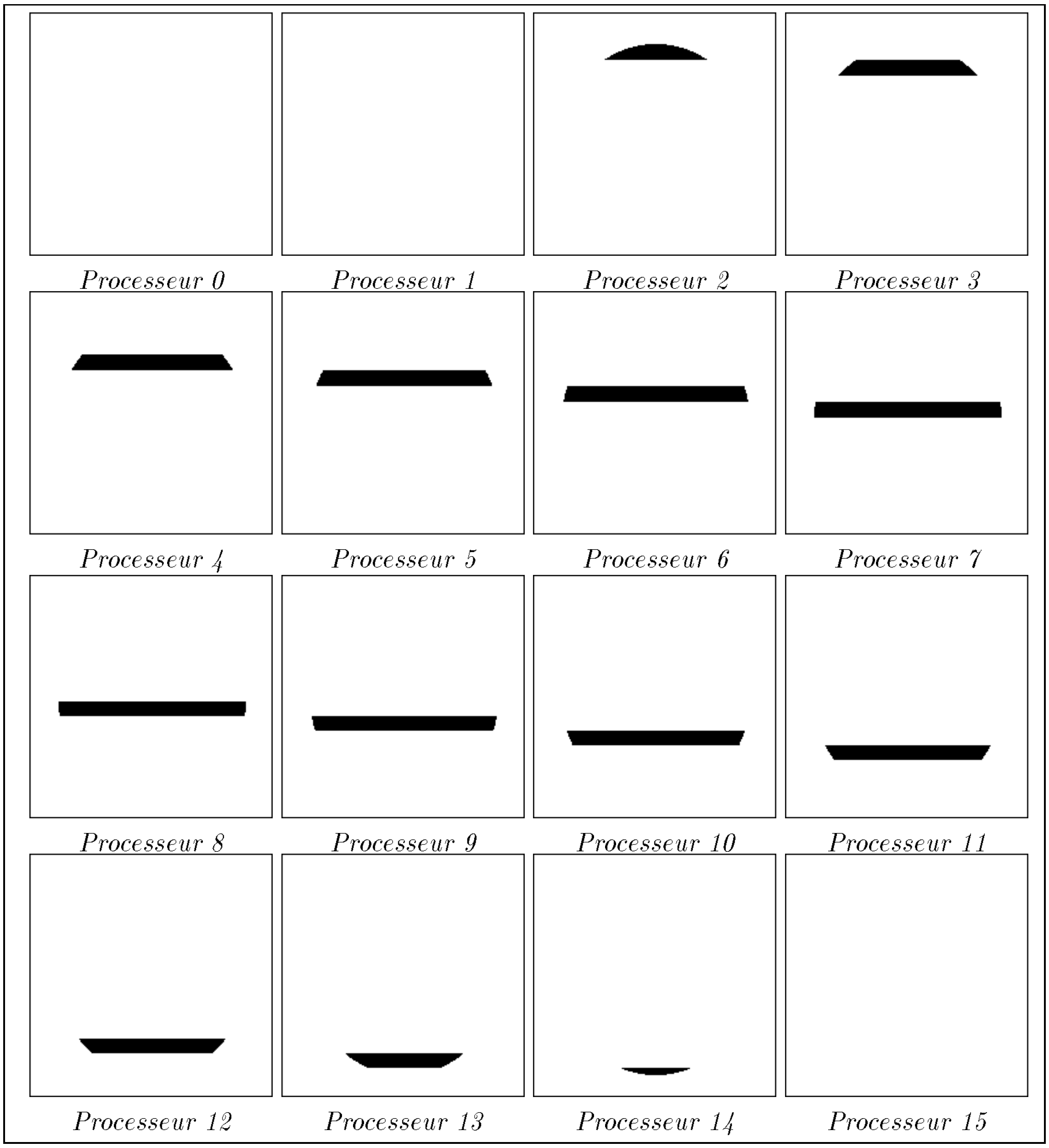

FIG. 4.6 - *Image pour chaque processeur au début sans la notion de groupe*

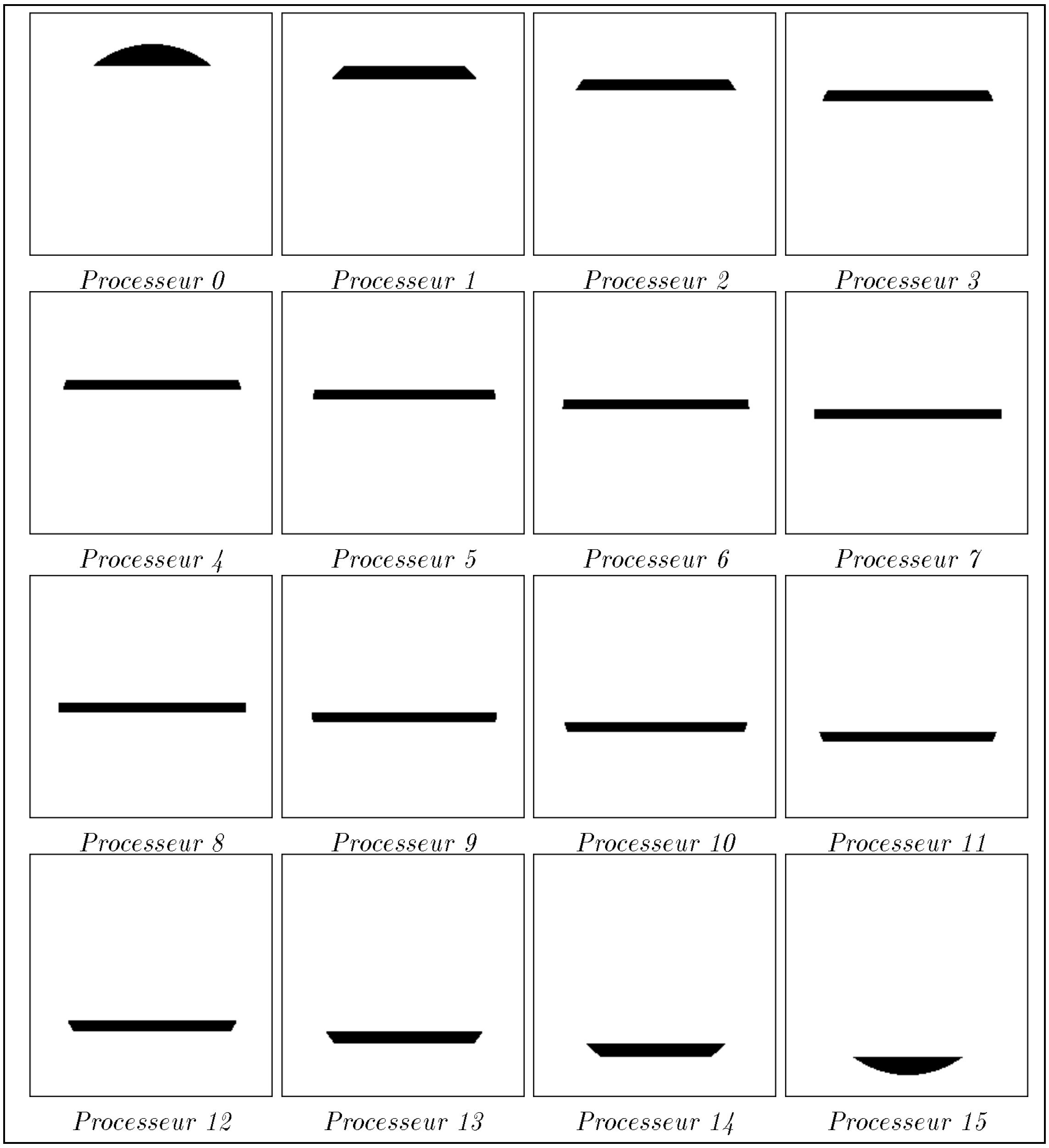

FIG. 4.7 - *Image pour chaque processeur après redistribution des lignes sans la notion de groupe*

4.3 Avec la notion de groupe

4.3.1 Méthodologie

- Au début du programme, nous créons deux groupes : le groupe ligne et le groupe colonne.
- Ensuite, nous répartissons l'image en région égale pour chaque processeur (figures 4.8 et 4.14).
- En premier lieu, nous travaillons sur les lignes.
  - Nous calculons la charge de travail pour les lignes et nous calculons les nouveaux indices des lignes pour le groupe ligne (figure 4.9).
  - Ensuite, nous redistribuons les lignes (figures 4.10 et 4.15).
- En deuxième lieu, nous travaillons sur les colonnes.
  - Nous calculons la charge de travail pour les colonnes et nous calculons les nouveaux indices des colonnes pour le groupe colonne (figure 4.11).
  - Il ne nous suffit plus qu'à redistribuer les colonnes (figures 4.12 et 4.16) et à calculer la charge de travail (figure 4.13).

4.3.2 Résultats

L'algorithme dans ce cas est légèrement différent qu'auparavant. Effectivement, grâce à la notion de groupe, nous redistribuons les lignes et les colonnes alors que, sans la notion de groupe, nous ne pouvons redistribuer que les lignes (ou les colonnes).

Les figures 4.9, 4.11 et 4.13 nous montrent qu'il y a, comme précédemment, une bonne redistribution du travail entre les processeurs.

Comme nous l'envisagions, l'image que possède chaque processeur est non plus "étirée", mais presque carrée grâce à la notion de groupe. Ainsi, ce type de représentation se prêtera mieux aux algorithmes en synthèse d'image comme nous avons pu l'expliquer auparavant (p.15).

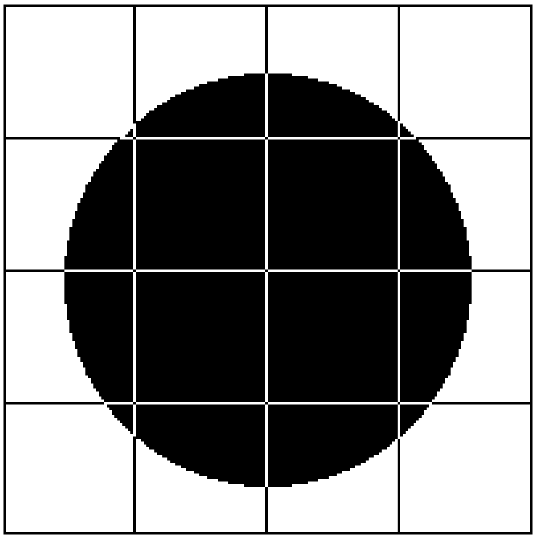

FIG. 4.8 - *Image répartie entre les processeurs*

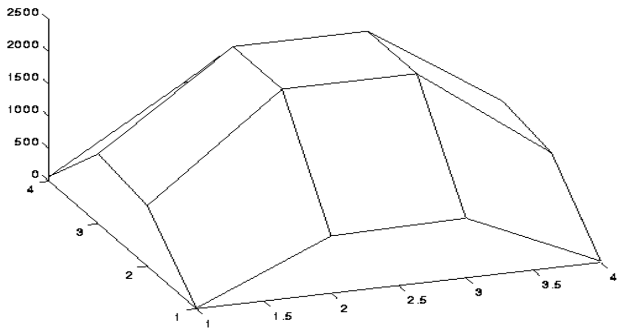

FIG. 4.9 - *Charge de travail au début*

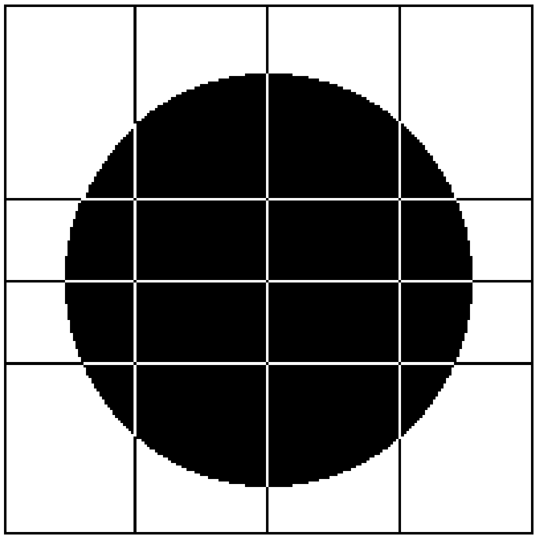

FIG. 4.10 - *Image après redistribution des lignes*

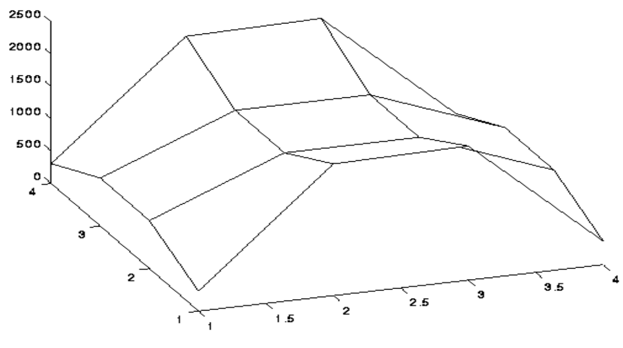

FIG. 4.11 - *Charge de travail après la redistribution des lignes*

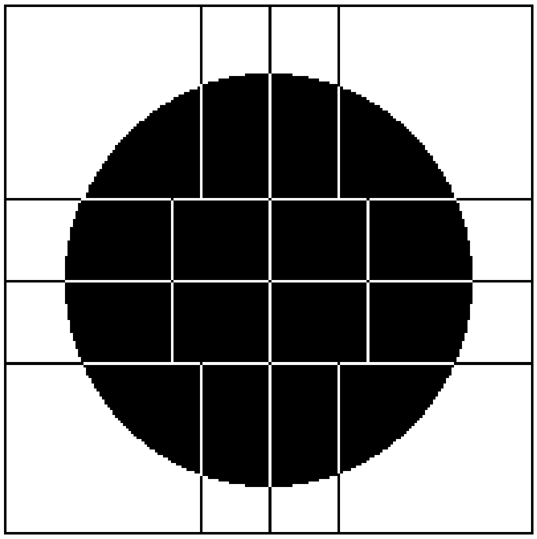

FIG. 4.12 - *Image après redistribution des colonnes*

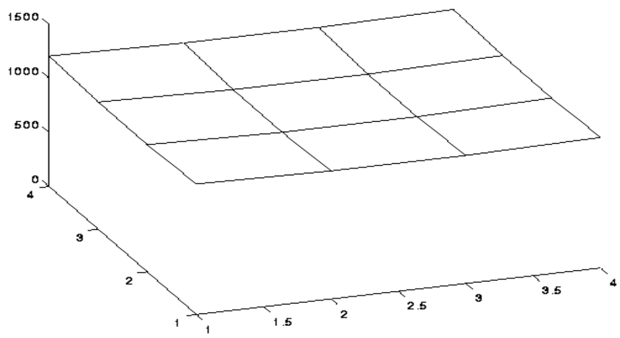

FIG. 4.13 - *Charge de travail après la redistribution des colonnes*