Table des matières

Table des matières **1**

Liste des figures **3**

Liste des tableaux **5**

Liste des algorithmes **6**

1 Introduction aux architectures parallèles **1**

Introduction . . . 1

1.1 Définition . . . 1

1.2 Pourquoi les Architectures Parallèles ? . . . 1

1.2.1 Problèmes de vitessee . . . 2

1.2.2 Problèmes de packing . . . 2

1.2.3 Problèmes liés aux débits mémoire . . . 2

1.3 Avantages des architectures parallèles . . . 3

1.3.1 Problèmes des architectures parallèles . . . 3

1.4 Classification des architectures parallèles . . . 3

1.4.1 Classification de Flynn . . . 3

1.4.2 Classification de Raina . . . 5

1.5 Sources de parallélisme . . . 6

1.5.1 Parallélisme de données . . . 7

1.5.2 Parallélisme de contrôle . . . 7

1.5.3 Parallélisme de flux . . . 7

1.6 Evaluation de la performance . . . 7

1.6.1 Approches classique . . . 8

1.6.2 Approches moderne . . . 9

Conclusion . . . 9

2 Réseaux d'interconnexion **10**

Introduction . . . 10

2.1 Pourquoi les RIs ? . . . 10

2.2 Mesurer la performance des RIs . . . 10

2.3 Catégories des RIs . . . 11

2.3.1 Réseaux d'interconnexion statiques . . . 11
2.3.2 Réseaux d'interconnexion reconfigurables . . . 14
2.4 Techniques de switching . . . 24
2.4.1 Store & forward . . . 24
2.4.2 Circuit switching . . . 24
2.4.3 Virtual cut through switching . . . 24
2.4.4 Wormhole routing . . . 25
Conclusion . . . 26

Conclusion et perspectives **27**

Bibliographie **30**

Annexe A Installation de la bibliothèque MPI **32**
Introduction . . . 32
A.1 C'est-quoi MPI ? . . . 32
A.1.1 Historique de MPI . . . 33
A.1.2 Types de données MPI . . . 34
A.1.3 Modes de Communication . . . 34
A.1.4 Fonctions de base . . . 35
A.2 Dev-C++ et MPI . . . 38
A.3 MPJ (MPI for Java) et Eclipse . . . 45
Conclusion . . . 47

Liste des figures

1 Introduction aux architectures parallèles **1**

1.1 Architecture SISD [8]. . . . 4
1.2 Architecture SIMD [8]. . . . 4
1.3 Architecture pipeline [8]. . . . 5
1.4 Architecture MIMD [8]. . . . 5
1.5 Classification des architectures MIMD selon Raina [12]. . . . 6
1.6 Exemples sur le parallélisme de données [1]. . . . 7
1.7 Exemples sur le parallélisme de contrôle [1]. . . . 7

2 Réseaux d'interconnexion **10**

2.1 Exemples de RIs statiques, a) Réseau de bus global, b) Réseau en anneau, c) Réseau en arbre, c) Réseau en mesh [3]. . . . 12
2.2 Exemples de RIs statiques, a) Réseau hypercube, b) Réseau complètement connecté [3]. . . . 12
2.3 Illustration de connexion fixe en cluster à 16 processeurs [3]. . . . 13
2.4 Illustration de connexion fixe en cluster à 16 processeurs [3]. . . . 15
2.5 Différents états d'un commutateur 2×2, a) État droit, b)État croisé [3]. . . . 16
2.6 Réseau Clos généralisé [3]. . . . 16
2.7 Réseau Clos généralisé [3]. . . . 17
2.8 Algorithme de routage du réseau Benes [3]. . . . 18
2.9 Après une itération de l'algorithme récursif [3]. . . . 19
2.10 Résultat final de l'algorithme récursif [3]. . . . 19
2.11 Exemple de réseau Omega [3]. . . . 20
2.12 Algorithme de routage du réseau Omega [3]. . . . 20
2.13 Exemple de bloquage dans le réseau Omega [3]. . . . 21
2.14 Réglage du deuxième étage du réseau Omega [3]. . . . 22
2.15 Exemple de routage réussi pour le réseau Omega [3]. . . . 22
2.16 Exemple de routage non réussi pour le réseau Omega [3]. . . . 23
2.17 Exemple de réseau Baseline [3]. . . . 23
2.18 Isomorphisme entre réseau Baselinr et réseau Omega [3]. . . . 24

A Installation de la bibliothèque MPI **32**

1.1 Illustration d'un communicateur global pour 07 processus. . . . 36
1.2 Choix des options du compilateur. . . . 38

1.3 Fenêtre des options du compilateur. 39
1.4 Choix de l'onglet **Directories**. 39
1.5 Choix du répertoire bin 40
1.6 Ajouter le répertoire bin choisit. 40
1.7 Fenêtre des options du projet. 41
1.8 Choix des fichiers lib. 41
1.9 Exemples de programme avec MPI. 42
1.10 Lancer l'exécution avec le chemain complet de **mpirun**. 42
1.11 Choix des propriétés système. 42
1.12 Choix des paramètres système avancées. 42
1.13 Fenêtres des variables système. 43
1.14 Sélection de la variables Path. 43
1.15 Fenêtres de modification de la variable Path. 43
1.16 Lancer l'exécution sans le chemain complet de **mpirun**. 44
1.17 Lancer l'exécution d'**Eclipse** en tant qu'administrateur. 45
1.18 Création d'un nouveau projet Java. 45
1.19 Choisir l'onglet Libraries. 46
1.20 Ajouter mpj.jar au nouveau projet Java. 46
1.21 Choisir le package du programme. 47
1.22 Choisir la fenêtre de configuration. 47
1.23 Choisir les paramètres à passer au programme MPI. 48

Liste des tableaux

1 Introduction aux architectures parallèles **1**
1.1 Classification de Flynn [8]. 3
1.2 Les différentes formes de caractères selon la position dans le mot [13]. 7

2 Réseaux d'interconnexion **10**
2.1 Comparaison entre les différentes Techniques de switching [6]. 24
2.2 Comparaison entre les différents RIs [3]. 26

A Installation de la bibliothèque MPI **32**
1.1 Types de données MPI. 34
1.2 Association des fonctions MPI avec les modes de communication. 34

Liste des algorithmes

Chapitre 1

Introduction aux architectures parallèles

Introduction

Pour un grand nombre d'applications essentiellement dans le domaine scientifique (prévisions météorologiques, analyse des flux aérodynamiques etc.), la puissance de calcul requise nécessite le développement de calculateurs parallèles très puissants. Cependant la puissance crête des machines parallèles n'est presque jamais atteinte, et ceci pour des raisons se situant à tous les niveaux de l'exploitation : applicatif, logiciel de base et matériel.

L'objectif de ce chapitre consiste à introduire et à présenter un état de l'art sur les architectures parallèles .

1.1 Définition

Les ordinateurs parallèles sont des machines qui comportent une architecture parallèle, constituée de plusieurs processeurs identiques, ou non, qui concourent au traitement d'une application. La performance d'une architecture parallèle est la combinaison des performances de ses ressources et de leur agencement. (latence, débit)

1.2 Pourquoi les Architectures Parallèles ?

Précédemment, la diminution du temps de calcul est réalisée en effectuant plus vite chaque (ensemble d') opération(s). Par contre, En calcul parallèle elle est faite exécutant simultanément plusieurs opérations (tâches) [8]. Les systèmes mono-processeurs possèdent les limites suivantes :

- Problèmes liés à la vitesse;
- Problèmes liés au packaging;
- Problèmes liés aux débits mémoire.

Nodes	Neighbors	Accumulated delay	Accumulated service cost	Best routes with Skyline	Ranking routes with Euclidean distance	Best parent	Second parent
2	1	2	11	1-2	1-2	1	2 5
	3	9	12				
	4	18	37				
	5	15	24				
3	1	6	3	1-3 2-3	1-3 2-3	1	2 5
	2	5	12				
	5	9	24				
	6	24	9				
4	2	10	28	2-4	2-4	2	7
	7	13	38				
	8	17	48				
5	2	9	23	3-5	3-5	3	2 5
	3	8	15				
	6	23	20				
	7	16	33				
	9	10	29				
6	3	15	8	3-6 9-6	3-6 9-6	3	9 5
	5	16	20				
	9	14	22				
7	4	11	34	4-7 5-7	5-7 4-7	5	4
	5	12	21				
	8	16	37				
8	4	13	38	4-8 7-8 9-8	7-8 9-8 4-8	7	9 4
	7	14	31				
	9	18	27				
9	5	9	17	5-9 6-9	5-9 6-9	5	6
	6	20	10				
	8	23	33				
10	8	20	36	9-10	9-10	9	8
	9	13	22				

1.2.1 Problèmes de vitessee

Il est bien connu que plus la fréquence de transmission augmente, plus les phénomènes de diaphonie (influence d'une ligne sur une autre) et de réflexion sont deviennent critiques au niveau des connexions métalliques (les changements d'impédances créent une surtension ou une sous-tension suivant le type de réflexion), limitant de ce fait les interconnexions. De plus les problèmes de dissipation de chaleur dans les circuits intégrés sont aussi liés à la vitesse de transmission.

1.2.2 Problèmes de packing

La puissance dissipée est une grandeur directement liée au taux d'intégration des transistors. Ainsi, pour limiter la puissance dissipée, les constructeurs sont obligés de diminuer la tension de commutation des logiques. D'autres problèmes liés au packaging concernent la limitation du brochage puisque plus de la moitié de la chaleur dissipée par un composant est le fait des plots d'entrée-sorties.

1.2.3 Problèmes liés aux débits mémoire

L'évolution des performances des processeurs s'accompagne aussi d'un écart croissant entre l'unité centrale et la mémoire principale, les temps d'accès à la mémoire n'évoluant plus lentement que les performances des processeurs. Diverses améliorationsbsur le plan architectural ont été apportées pour pallier cet écart : l'utilisation systématique de caches, des mots mémoires plus larges, l'entrelacement des adresses mémoire, ...etc.

1.3 Avantages des architectures parallèles

Les architectures parallèles offrent plusieurs avantages telles que:

- L'amélioration des performances de calcul

- L'accroissement de la taille des problèmes à résoudre
- La résolution de nouveaux problèmes pas de limite de mémoire

1.3.1 Problèmes des architectures parallèles

Malgré les avantages de ces architectures, elle souffrent de quelques problèmes, à savoir:

- La remise en question des concepts d'algorithmique classique basés sur le principe de la machine séquentielle.
- Diversité des modèles d'architectures parallèles.
- Difficulté de la programmation des machines parallèles.

1.4 Classification des architectures parallèles

La grande diversité des machines parallèles a donné lieu, à plusieurs taxonomies parmi lesquelles on peut citer celle de Flynn et celle de Raina. Cependant, parmi les critères de différenciation liés au architectures parallèles, on peut noter principalement :

- comment les multiples noeuds sont connectés (topologie);
- le modèle d'exécution (SIMD, MIMD, etc.) :si chaque unité de traitement a son propre flux d'instructions;
- si les unités de traitement possèdent leur propre mémoire privée ou partagent un même espace mémoire (mémoire partagée, distribuée ou virtuellement partagée).

1.4.1 Classification de Flynn

M.J. Flynn a proposé une classification pour l'organisation d'un système informatique en fonction du nombre d'instructions et d'éléments de données manipulés simultanément (voir le tableau 1.1).

Tableau 1.1: Classification de Flynn [8].

Flux d'**I**nstructions / Flux de **D**onnées	**S**ingle	**M**ultiple
Single	SISD	MISD
Multiple	SIMD	MIMD

1.4.1.1 SISD

Dans ce cas, une instruction est exécutée sur une donnée selon l'architecture de machine séquentielle proposée par Von Neumann [10]. La figure 1.1 présente une illustration de cette architecture.

1.4.1.2 SIMD

Une machine SIMD (Single Instruction Multiple Data) est une machine qui exécute à tout instant une seule instruction, mais qui agit en parallèle sur plusieurs données, on parle en général de "parallélisme

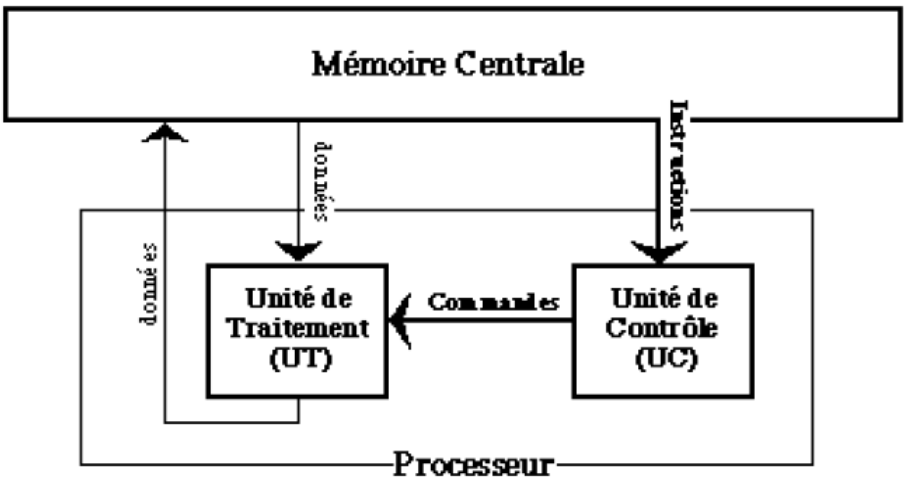


Figure 1.1: Architecture SISD [8].

de données". Les machines SIMD peuvent être de plusieurs types, par exemple, parallèles ou systoliques.

Les machines systoliques sont des machines SIMD particulières dans lesquelles le calcul se déplace sur une topologie de processeurs, comme un front d'onde, et acquiert des données locales différentes à chaque déplacement du front d'onde (comportant plusieurs processeurs, mais pas tous en général comme dans le cas (1)) [10, 5]. La figure 1.2 présente une illustration de cette architecture.

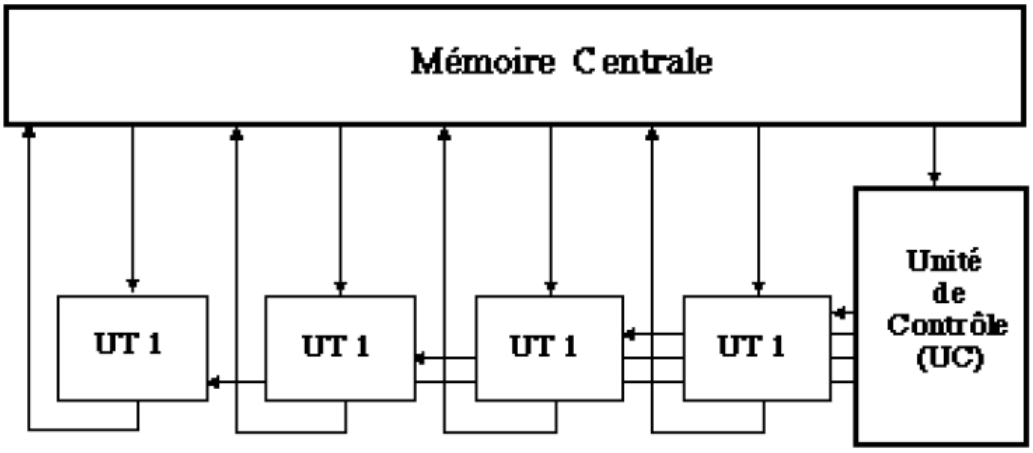


Figure 1.2: Architecture SIMD [8].

1.4.1.3 MISD

Une machine MISD (Multiple Instruction Single Data) peut exécuter plusieurs instructions en même temps sur la même donnée. Cela peut paraître paradoxal mais cela recouvre en fait un type très ordinaire de micro-parallélisme dans les micro-processeurs modernes: les processeurs vectoriels et les architectures pipelines [10, 5]. La figure 1.3 présente une illustration de cette architecture.

1.4.1.4 MIMD

Ici,nous avons plusieurs processeurs, où chaque processeur exécute une instruction différente pour des données différentes.Pour simplifier la programmation, on exécute la même application sur tous les

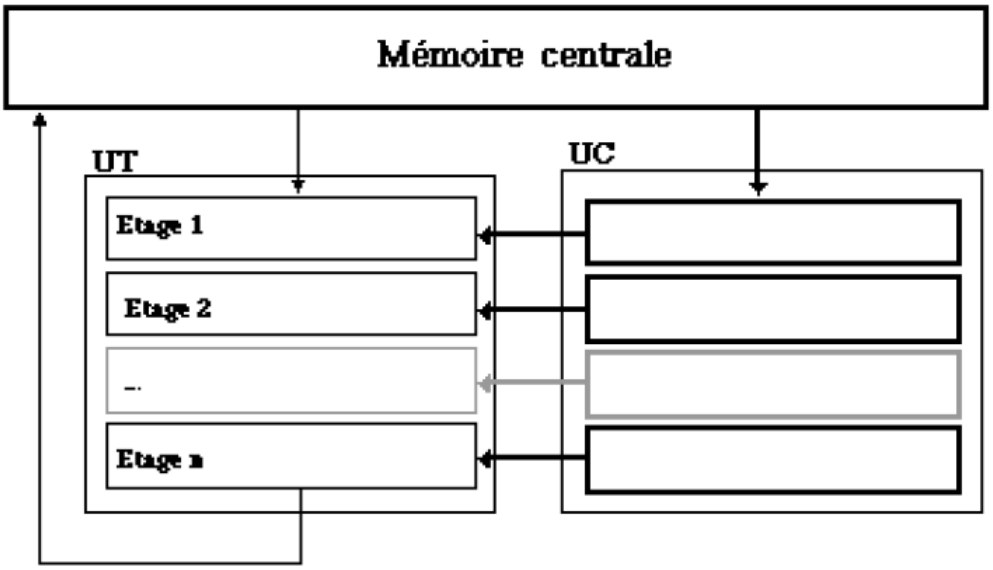


Figure 1.3: Architecture pipeline [8].

processeurs. Ce mode d'exécution est appelé : SPMD (Single Program Multiple Data). On a plusieurs types d'architecture possibles [5, 10, 8]:

1. Mémoire partagée (Sequent etc.);
2. Mémoire locale + réseau de communication (Transputer, Connection Machine) - Système réparti.

La figure 1.4 présente une illustration de cette architecture.

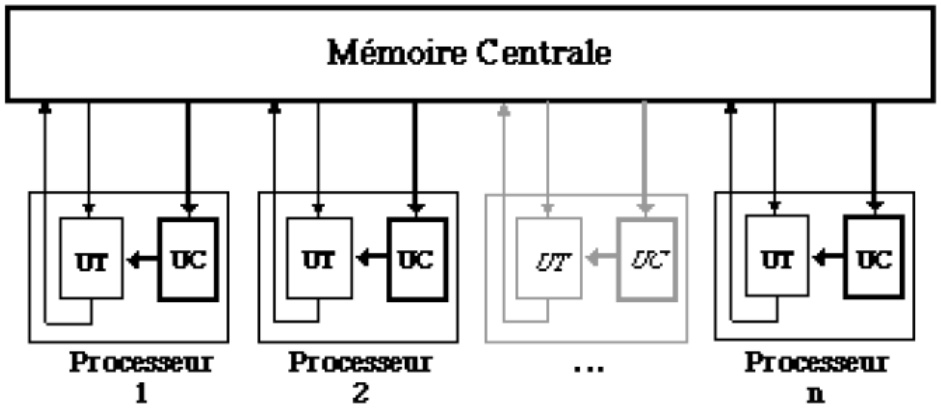


Figure 1.4: Architecture MIMD [8].

1.4.2 Classification de Raina

Elle présente une sous-classification des machines avec une architecture MIMD. Cette classification prend en compte deux critères de l'architecture mémoire, comme suit [12]:

1. l'organisation de l'espace d'adressage:

   (1) SASM (Signle Address space, Shared Memory): mémoire partagée;

(2) DADM (Distributed Address space, Distributed Memory): mémoire distribuée, sans accès aux données distantes. L'échange de données entre processeurs s'effectue nécessairement par passage de messages, au moyen d'un réseau de communication;

(3) SADM (Single Address space, Distributed Memory):mémoire distribuée, avec espace d'adressage global, autorisant éventuellement l'accès aux données situées sur d'autres processeurs.

2. le type d'accès mémoire mis en oeuvre:

(1) NORMA (No Remote Memory Access): pas de moyen d'accès aux données distantes, nécessitant le passage de messages;

(2) UMA (Uniform Memory Access): accès symétrique à la mémoire, de coût identique pour tous les processeurs;

(3) NUMA (Non-Uniform Memory Access): les performances d'accès dépendent de la localisation des données ;

(4) CC-NUMA (Cache-Coherent NUMA) : type d'architecture NUMA intégrant des caches;

(5) OSMA (Operating System Memory Access) : les accès aux données distantes sont gérées par le système d'exploitation, qui traite les défauts de page au niveau logiciel et gère les requêtes d'envoi/copie de pages distantes ;

(6) COMA (Cache Only Memory Access) : les mémoires locales se comportent comme des caches, de telle sorte qu'une donnée n'a pas de processeur propriétaire ni d'emplacement déterminée en mémoire.

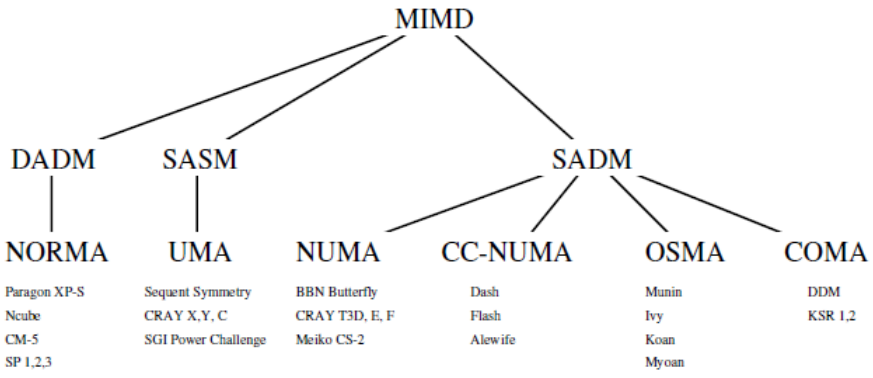


Figure 1.5: Classification des architectures MIMD selon Raina [12].

1.5 Sources de parallélisme

Généralement, on peut distinguer tois sources de parallélisme, à savoir:

1. parallélisme de données,
2. Parallélisme de contrôle,
3. Parallélisme de flux.

1.5.1 Parallélisme de données

il consiste à effectuer la même opération par chaque processeur sur des données différentes (voir figure 1.6).

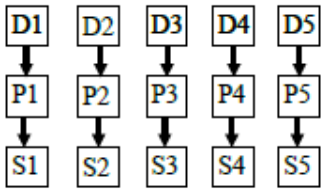


Figure 1.6: Exemples sur le parallélisme de données [1].

1.5.2 Parallélisme de contrôle

Dans ce cas, les opérations sont réalisées simultanément sur plusieurs processeurs. Le programme présente des séquences d'opérations indépendantes qui peuvent être exécutées en parallèle. (voir figure 1.7).

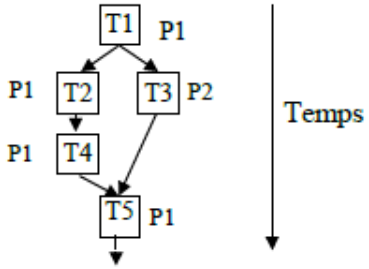


Figure 1.7: Exemples sur le parallélisme de contrôle [1].

1.5.3 Parallélisme de flux

Les opérations sur un même flux de données peuvent être enchaînées (pipeline) (voir le tableau 1.2).

Tableau 1.2: Les différentes formes de caractères selon la position dans le mot [13].

temps	Tâche 1	Tâche 2	Tâche 3	Tâche 4	Tâche 5
t1	Donnée 1				
t2	D2	D1			
t3	D3	D2	D1		
t4	D4	D3	D2	D1	
t5	D5	D4	D3	D2	D1
t6	D6	D5	D4	D3	D2

1.6 Evaluation de la performance

L'évaluation des systèmes parallèles peut être faite selon deux approches:

1. ***Approche classique:*** elle consiste évaluer la performance des machines en utilisant un ensemble de lois et formules prédéfinis.

2. ***Approche moderne:*** elle consiste à évaluer la performance des machines en utilisant un ensemble de programmes appelés benchmarks.

1.6.1 Approches classique

Dans ce cas, afin s'évaluer la performance des machines on utilise un ensemble de lois (tels que loi d'Amdahl et loi de Gustafsun) et formules prédéfinis (telles que celle de l'accélération et celle de l'efficacité) [8].

1.6.1.1 Accélération

L'accélération ou le «speed-up, en anglais» peut être définit comme le rapport entre les temps d'exécution avant et après l'amélioration apportée au matériel. Comme certains dispositifs ne sont pas nécessairement utilisés à chaque instruction du programme, le gain obtenu par une performance accrue, sur une partie de programme, diminue au fur et à mesure qu'on ajoute des améliorations.

Par exemple, un additionneur dix fois plus rapide ne produira pas un speed-up de 10 et une machine avec 1000 processeurs n'exécutera pas chaque travail 1000 fois plus vite.

Considérons un algorithme qui s'exécute sur un ordinateur parallèle comportant p processeurs (identiques) en un temps T_p, et soit T_1 son temps d'exécution séquentiel.

L'accélération est définie par le rapport : $S_p = T_1/T_p$

Généralemnt on a : $1 \leq Sp \leq p$

1.6.1.2 L'Efficacité

L'efficacité d'un algorithme parallèle est le rapport :

$E_p = S_p/p$

1.6.1.3 Loi d'Amdahl

Le temps d'exécution après une amélioration d'un aspect d'une machine est donné par :

$$T_p = \frac{T_{ap}}{F} + T_{np}$$

où:

T_p: Temps d'exécution après amélioration.
T_{ap}: Temps d'exécution affecté par l'amélioration.
F: Temps d'exécution affecté par l'amélioration.
T_{np}: Temps d'exécution non affecté.

Exemple : Supposons que le temps d'exécution d'un certain programme soit 100 sec et que les multiplications représentent 80 sec de ce temps. Quel sera le temps d'exécution après avoir apporté une

amélioration de 5 dans la vitesse de multiplication ?

Temps d'exécution après amélioration = 80 / 5 + 20 = 36 sec

Donc une amélioration globale de 2,77 (100/36) seulement. Donc, on ne peut pas s'attendre à ce qu'une amélioration d'un aspect d'une machine apporte une amélioration proportionnelle au facteur d'amélioration.

1.6.2 Approches moderne

L'évaluation de la performance fait intervenir plusieurs facteurs : la vitesse du traitement proprement dit, la réponse des différents étages de la mémoire (caches), et la vitesse des entrées / sorties. Des programmes de test, appelés benchmarks, permettent d'évaluer les performances des ordinateurs, mais peu d'entre eux sont réellement utilisés de façon standard étant donné la difficulté de couvrir avec quelques tests une vaste gamme d'architectures, de machines et de problèmes à traiter.

Un certain nombre de constructeurs se sont groupés pour former un groupe appelé SPEC (System Performance Evaluation Cooperative) qui a pour but de définir des mesures standards communes. SPEC a défini une dizaine de programmes de test. On les exécute sur une machine à évaluer, puis on compare le temps d'exécution avec un temps de référence.

On calcule un SPECratio pour chaque programme et on fait la moyenne de tous les programmes du test pour obtenir la valeur finale en SPECmarks. Il y a deux classements. Les SPECint, pour mesurer la performance lors de calculs sur des entiers, et les SPECfp pour mesurer la performance en virgule flottante. La dernière mise à jour de ces tests remonte à 1995, de sorte qu'on parle présentement de SPECint95 et de SPECfp95.

Il existe beaucoup d'autres benchmarks, par exemple, les Dhrystones, les Whetstones, les tests Linpack et les Livermore Loops.

Conclusion

Dans ce chapitre, nous avons vu un état de l'art sur les machines parallèles. Pour les quels, nous avons présenté les différents architectures et approches de classification . De plus, nous avons présentés les différents sources de parallélismes. Enfin, nous avons décrit les différentes approches d'évaluation de ces systèmes de calculs.

Chapitre 2

Réseaux d'interconnexion

Introduction

Les processeurs dans les ordinateurs parallèles doivent se communiquer pour résoudre un problème. Cependant, il est nécessaire d'avoir un moyen de communication de haut débit ou un réseau d'interconnexion. La Performance dans les systèmes multiprocesseurs dépend beaucoup sur les processus de communication entre les processeurs et mémoire, périphériques d'E/S, et les autres processeurs. Cependant, le choix du réseau d'interconnexion (RI) approprié est important pour des raisons d'efficacité.

2.1 Pourquoi les RIs ?

Lorsque plusieurs processeurs doivent accéder à une structure de mémoire, des réseaux d'interconnexion sont nécessaires pour router les données [6]:

- des processeurs aux mémoires (accès concurrent une mémoire partagée), ou
- d'un processeur élémentaire (processeur + mémoire) à un autre (pour fournir un service de transmission de messages)

Inévitablement, une bande passante importante est nécessaire pour correspondre à la bande passante combinée des éléments de traitement.

2.2 Mesurer la performance des RIs

Souvent, plusieurs métriques sont utilisés afin de décrire la performance des réseaux d'interconnexion, parmi les on trouve [14, 6]:

- ***La connectivité :*** C'est le degrée, le nombre des noeuds qui sont voisins immédiats d'un noeud, c-à-d, le nombre de nœuds pouvant être atteints en un saut.
- ***Le diamètre :*** le nombre maximal de nœuds par lesquels un message doit passer de la source à la destination. Isl mesure le délai maximum de transmission d'un message d'un processeur à un autre.
- ***La istance moyenne :*** , où la distance entre deux nœuds est définie par le nombre de sauts dans le chemin le plus court entre ces nœuds. La distance moyenne est donnée par:

$$d_{avg} = \frac{\sum_{d=1}^{1}(d \,.\, N_d)}{N-1}$$

où N est le nombre de nœuds, N_d est le nombre de noeuds distants d et r est le diamètre.

- ***La largeur de coupe :*** le plus petit nombre de fils que vous devez couper pour déconnecter le réseau en deux moitiés égales (± 1).

2.3 Catégories des RIs

Les réseaux d'interconnexions peuvent être catégoriser selon les trois critères suivants: topologie, stratégies de routage, et techniques de commutation (switching) [3, 6, 2].

1. **Topologie:** c'est la structure selon laquelle les processeurs individuel sont connectés aux autres éléments. Les deux topologies principales sont: fixe (statique) et reconfigurable.
2. **Stratégies Routage:** sont les procédures utilisées pour changer l'état des commutateurs (switches), et elles jouent un rôle crucial dans la performance des réseaux d'interconnexion multi étages.
3. **Techniques de commutation (Switching):** sont les moyens selon les quels les paquets de données sont maniés durant leur chemin d'un source to à une destination. processor.

Généralement, les réseaux d'interconnexion peuvent être classés en deux grandes classes, comme suit :

1. Réseaux d'interconnexion fixes;
2. Réseaux d'interconnexion reconfigurables;
   (1) Réseau Crossbar;
   (2) Réseaux d'interconnexion multi-étages.

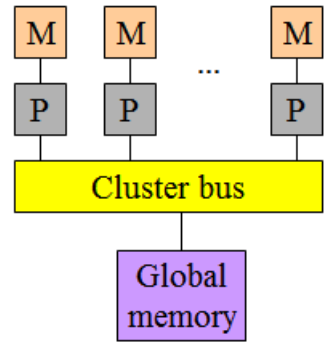
2.3.1 Réseaux d'interconnexion statiques

Les connexions des réseaux d'interconnexion fixes sont des fils de connexion statique et ne peuvent pas changer leurs configuration. Ils ne sont pas flexible comme les systèmes à connexion reconfigurable, ils sont suffisante pour la plupart des demandes des ordinateurs parallèle et moins couteuses. Généralement, les topologies fixes sont plus utilisable pour des problèmes avec des structures de communication bien définis. Les figures 2.1 et 2.2 illustrent des exemples de réseaux statiques [3, 6, 2].

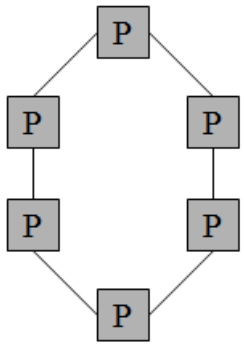
2.3.1.1 Connexion fixe en cluster à 16 processeurs

L'architecture est divisée en 04 clusters. Chaque cluster est composé de 04 processeurs connectés par bus de cluster. Les processeurs dans le même bus de cluster peuvent communiquer entre eux sans affecter les autres clusters; cela maximise le flux de données et minimise les délais. Les passerelles intercluster permettent les transferts entre les clusters. Le mécanisme de communication intercluster communique entre les clusters; Ce réseau peut être un réseau fixe ou reconfigurable [3, 6, 2].

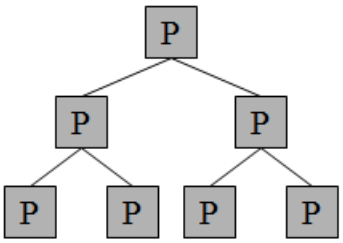
Si une tâche nécessite des processeurs appartenant à plusieurs clusters, le processus suivant est exécuté:



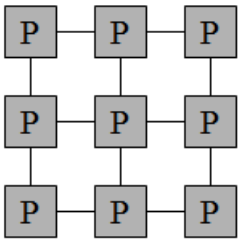
(a) Bus global



(b) Anneau

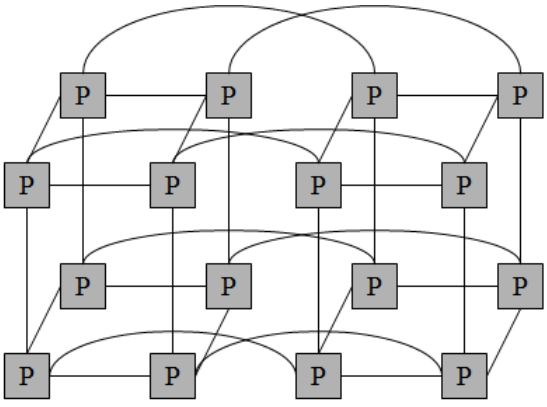


(c) Arbre

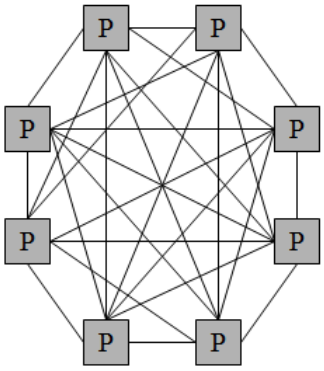


(d) Mesh

Figure 2.1: Exemples de RIs statiques, a) Réseau de bus global, b) Réseau en anneau, c) Réseau en arbre, c) Réseau en mesh [3].



(a) Hypercube



(b) Complètement connecté.

Figure 2.2: Exemples de RIs statiques, a) Réseau hypercube, b) Réseau complètement connecté [3].

1. Un processeur d'un cluster envoie des données et des informations de destination à sa passerelle intercluster via son bus de cluster.
2. La passerelle évalue les informations de destination pour trouver son cluster.
3. Les données sont ensuite envoyées au cluster spécifié via le mécanisme de communication.
4. Enfin, la passerelle de destination envoie les données au processeur de destination.
5. Les processeurs ne se communiquent jamais directement; les processeurs sont libres tandis que les passerelles envoient les données.

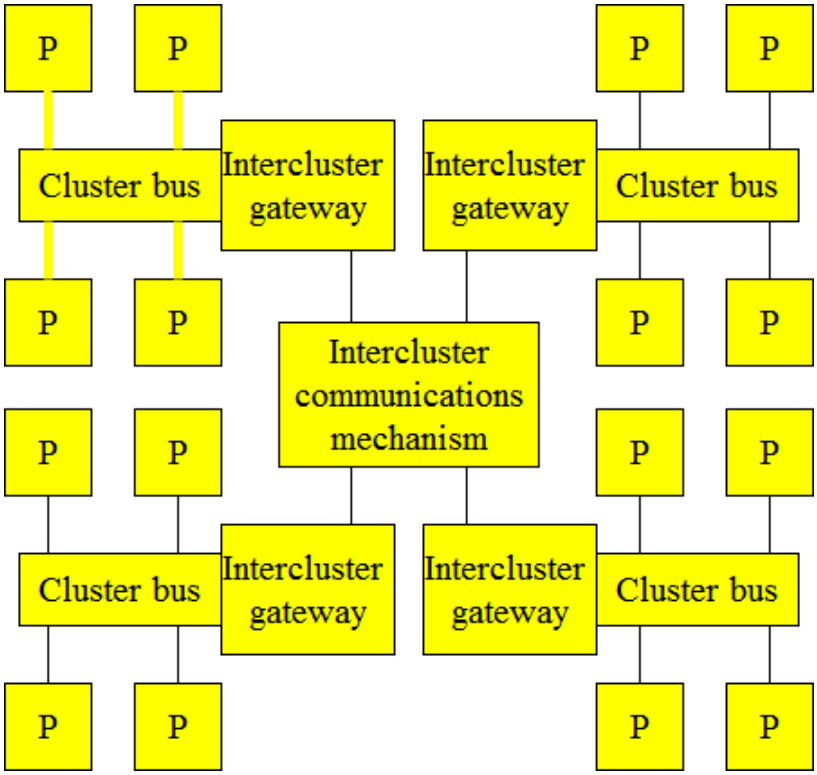


Figure 2.3: Illustration de connexion fixe en cluster à 16 processeurs [3].

2.3.2 Réseaux d'interconnexion reconfigurables

Tant que des tâches différentes nécessite des ressources de traitement différentes, les connexions reconfigurables sont nécessaires. Elles permettent aux configurations dynamiques d'égaler les tâches individuelles ce qui optimise la performance globale du système [3, 6, 2].

il existe deux topologies, à savoir:

- Connexions à commutateurs Crossbar;
- Réseaux d'interconnexion multi-étages (MINs: Multistage Interconnection Networks)

2.3.2.1 Réseaux de permutations

La plupart des réseaux d'interconnexion mutli-étages sont conçus pour réaliser des réseaux de permutation, c'est-à-dire des réseaux avec des connexions 1 à 1 entre leurs entrées et leurs sorties. Les réseaux MINs sont classés en deux classes, comme suit [3, 6, 2]:

1. ***Non-bloquant:*** capable de réaliser n! connexions entre ses n entrées et ses n sorties (tous les combinaisons possibles). Cette classe se divise à son tour en deux sous-classes, à savoir:
   (1) ***Strictement non-bloquant:*** peut changer une connexion sans affecter les autres connexions.
   (2) ***Strictement non-bloquant:*** peut réaliser de nouvelles connexions mais peut devoir rediriger des connexions existantes.
2. ***Non-bloquant:*** ne peut pas réaliser toutes les combinaisons possibles entre ses entrées et ses sorties.

2.3.2.2 Routage des réseaux MINs

Les algorithmes de routage jouent un rôle crucial dans la performance d'un réseau d'interconnexion à plusieurs étages. Les algorithmes de routage lents réduiront considérablement les performances d'un système multiprocesseur. Un réseau MIN peut avoir de nombreux algorithmes de routage parmi lesquels il faut choisir. Mais l'un parmi eux est choisi et implémenté dans le système lors de sa conception [3, 6, 2].

Mais avant d'examiner les algorithmes de routage pour les réseaux d'interconnexion à plusieurs étages, il convient d'introduire des notations pour les permutations [3, 6, 2].

2.3.2.3 Notation de permutation

Une permutation est représentée par une matrice à deux lignes délimitée par des parenthèses. La rangée du haut est la liste des sources et la rangée du bas est la liste des destinations. Par exemple, la permutation droit d'un commutateur serait représentée par (voir la figure 2.5) [3, 6, 2]:

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

et l'état croisé va être représenté par:

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

Le groupe des états réalisable par ce commutateur est:

$$\left\{\begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}\right\}$$

2.3.2.4 Réseau Crossbar

Ce réseau possède n entrées et m sorties; n et m sont généralement les mêmes. Les données peuvent circulées dans les deux directions. Chaque point de croisement (en bleu) peut s'ouvrir ou se fermer pour réaliser une connexion. Toutes les combinaisons possibles entre les entrées et les sorties peuvent être réalisées. Les entrées sont généralement connectées aux processeurs et les sorties connectées à des mémoires, aux E/S ou à d'autres processeurs [3, 6, 2].

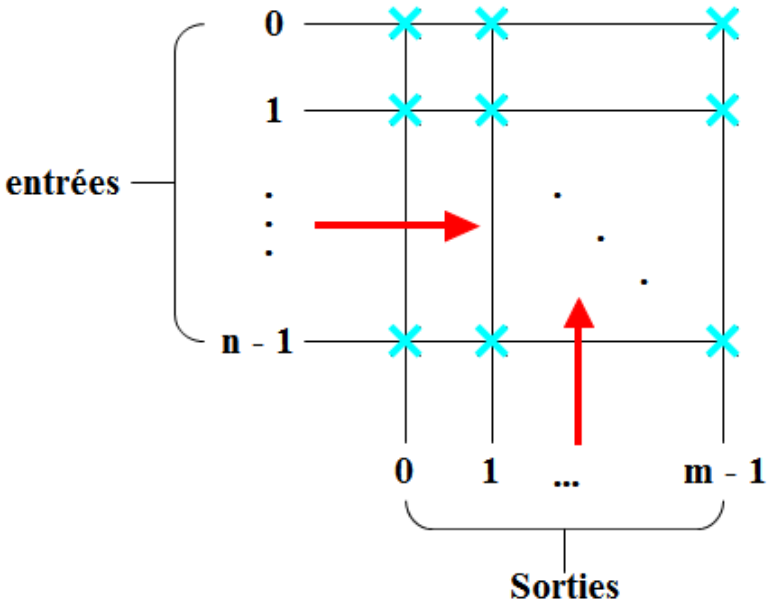


Figure 2.4: Illustration de connexion fixe en cluster à 16 processeurs [3].

Ces commutateurs ont des complexités de $O(n^2)$; doubler le nombre d'entrées et de sorties double également la taille du commutateur. Pour résoudre ce problème, les réseaux d'interconnexion multi-étages ont été développés.

2.3.2.5 Réseau MINs

Ces réseaux utilisent des commutateurs (switches) transversaux plus petits, généralement 2×2, reliés par des liens fixes. Ces commutateurs 2×2 ont deux réglages possibles pour les réseaux de permutation, droit et croisé, comme illustré dans la figure 2.5. Les entrées et les sorties sont connectées 1 à 1. Les réseaux d'interconnexion mutlti-étages réalisent les permutations réseaux désirés entre leurs entrées et sorties en réglant les commutateurs aux états corrects. Les algorithmes de routage permettent de définir les états des commutateurs d'un réseau MIN [3, 6, 2].

Certains réseaux d'interconnexion multi-étages largement utilisés incluent:

- Réseau Clos;
- Réseau Benes;

(a) Droit (b) Croisé.

Figure 2.5: Différents états d'un commutateur 2×2, a) État droit, b)État croisé [3].

- Réseau Omega;
- Réseau Baseline.

2.3.2.5.1 Réseau Clos généralisé

Le réseau est composé de tois étages de commutateurs avec $T = ntimesk$ entrées et sorties. Le 1^{er} étage possède k commutateurs de taille n par m. Le $2^{ème}$ étage possède $(m \times k)$ sur k commutateurs qui reçoivent 1 entrée de chaque commutateur du 1^{er} étage. Le $3^{ème}$ étage possède $k \times m$ par n commutateurs qui reçoivent 1 entrée de chaque commutateur du $2^{ème}$ étage. Si $m \geq n$, le réseau est dit rearrangeablement non-bloquant (voir la figure 2.6) [6, 2].

n, m et k peuvent être changés pour réaliser des complexités entre $O(n\,lg\,n)$ et $O(n^2)$.

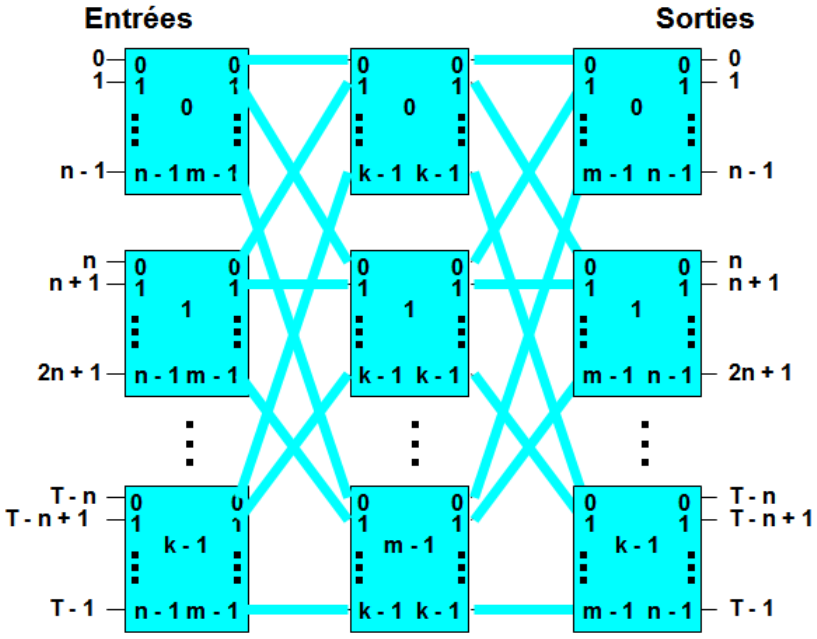


Figure 2.6: Réseau Clos généralisé [3].

2.3.2.5.2 Réseau Benes

Dans cette section, nous allons voir comment créer et comment appliquer l'algorithme de routage d'un réseau Benes.

A. Création d'un réseau Benes

Le réseau Benes est dérivé du réseau Clos en définissant $n = m = 2$ et $k = T/2$ et en décomposant de manière récursive les deux commutateurs centraux de $(T/2) \times (T/2)$. Par exemple, pour créer un réseau Benes de 8×8 , quatre commutateurs de 2×2 sont créés dans le 1^{er} et le $2^{ème}$ étage. En plus, deux commutateurs de 4×4 sont crées dans l'étage centre. Ensuite, une sortie de chaque commutateur du 1^{er} étage est acheminée vers une entrée de chaque commutateur de l'étage centre. Une entrée de chaque commutateur du dernier étage est acheminée vers Une sortie de chaque commutateur de l'étage centre. Enfin, Les commutateurs centraux sont ensuite décomposés en réseaux Benes 4 x 4 en suivant les mêmes étapes (voir la figure 2.7) [6, 2].

Ce réseau est réarrangeablement non-bloquant et possède une complexité de $O(n \lg n)$.

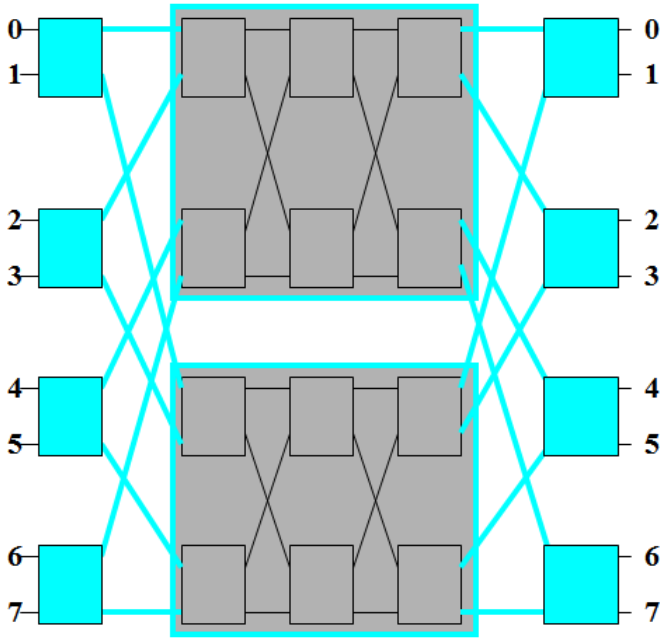


Figure 2.7: Réseau Clos généralisé [3].

B. Algorithme récursif de routage du réseau Benes

C'est une méthode récursive utilisée pour définir les commutateurs d'un réseau de Benes. Rappelons que la structure du réseau de Benes est récursive, composée de deux étages externes de commutateurs et de deux réseaux Benes demi-taille. Cet algorithme prend l'avantage de la récursivité en définissant les commutateurs du réseau de Benes. Il prend la permutation initiale, établit les commutateurs dans les étages extérieurs et génère les permutations à réaliser par les deux sous-réseaux. Ces permutations sont traitées de manière récursive jusqu'à ce que l'ensemble du réseau soit défini. Le temps d'exécution de cet algorithme est $O(n \lg n)$ (voir la figure 2.8) [6, 2].

Pour illustrer, considérons le réseau Benes de 8×8 qui doit réaliser la permutation:

$$\begin{pmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 \end{pmatrix}$$

L'algorithme passe par les étapes suivantes:

1. Définir arbitrairement l'état de n'importe quel commutateur des étages externe.

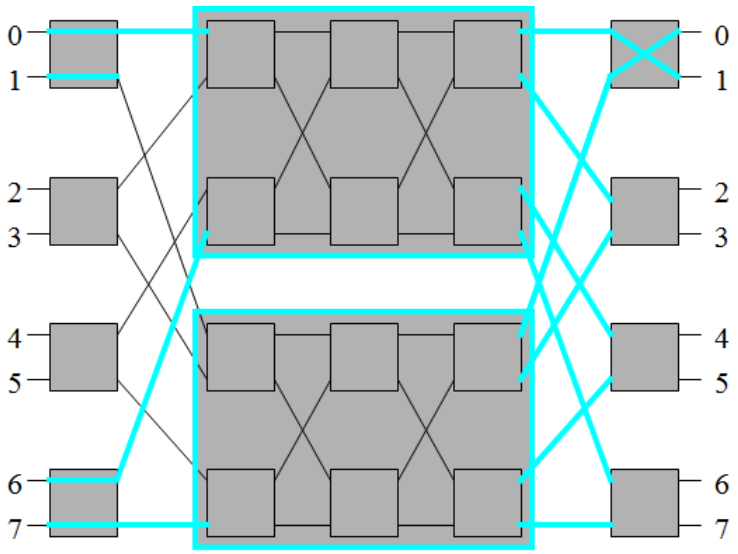


Figure 2.8: Algorithme de routage du réseau Benes [3].

2. Le commutateur le plus élevé du 1^{er} étage est réglé à l'état droit et envoie l'entrée réseau 0 au sous-réseau supérieur.

3. Tantque, chaque commutateur du dernier étage reçoit 1 entrée du sous-réseau supérieur et 1 du sous-réseau inférieur. Etant donné que l'entrée réseau 0 est acheminée vers le sous-réseau supérieur et que cette entrée doit être acheminée vers la sortie réseau 1, le commutateur le plus haut du dernier étage doit être réglé à l'état croisé.

4. La sortie réseau 0 reçoit alors ses données du sous-réseau inférieur, et puisque sa source est l'entrée réseau 7, son commutateur est réglé à l'état droit. Cela nécessite que l'entrée réseau 6 doit être routée vers le sous-réseau supérieur. Si aucun commutateur des étages externes n'est défini, il faut revenir l'étape 1.

Cet algorithme suit la même procédure, effectuant une boucle entre les entrées et les sorties jusqu'à ce que le commutateur d'origine soit atteint.

Après la première itération de l'algorithme, les commutateurs des étages externes vont être réglés comme indiqué par la figure 2.9. Les permutations à réaliser par les sous-réseaux supérieurs et inférieurs sont :

$$\begin{pmatrix} 0 & 2 & 4 & 6 \\ 1 & 3 & 5 & 7 \end{pmatrix} et \begin{pmatrix} 1 & 3 & 5 & 7 \\ 0 & 2 & 4 & 6 \end{pmatrix}$$

respectivement.

En répétant l'algorithme sur les sous-réseaux, les réglages finaux des commutateurs sont illustrés par la fugure 2.10, en fonction de la permutation initiale.

2.3.2.5.3 Réseau Omega

Dans cette section, nous allons voir comment créer et comment appliquer l'algorithme de routage d'un réseau Omega.

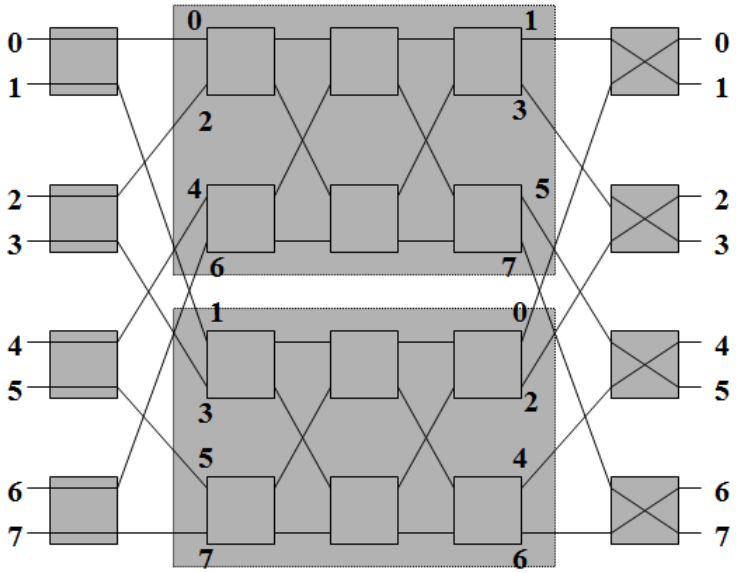


Figure 2.9: Après une itération de l'algorithme récursif [3].

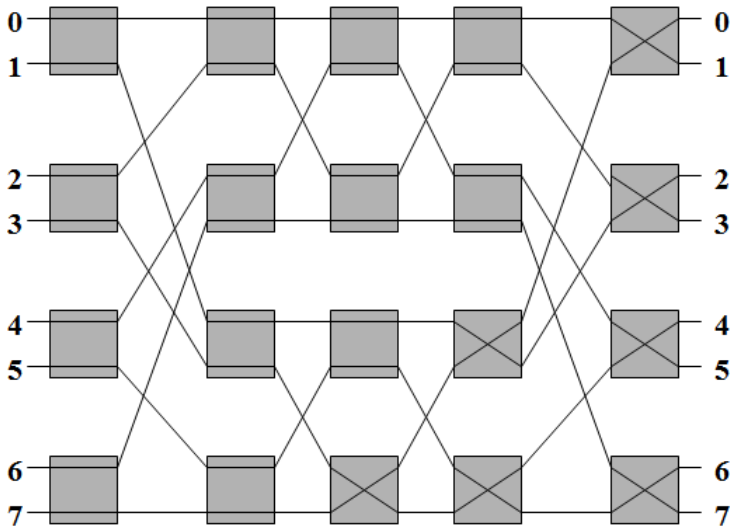


Figure 2.10: Résultat final de l'algorithme récursif [3].

A. Création réseau Omega

Le réseau Omega est composé de quatre commutateurs de 2×2 par étage. Les liens entre chaque paire d'étages sont fixes et identiques. il possède une complexité de $O(n \lg n)$. Pour 8 entrées possibles, il y en a $8! = 403201$ mappages 1 à 1 entre les entrées et les sorties. Mais seulement 12 commutateurs pour un total de $212 = 4096$ réglages. Pour cela, le réseau est bloquant. La figure 2.11 illustre un exemple de ce réseau [6, 2].

B. Algorithme de routage réseau Omega

Contrairement au réseau Benes, qui utilise un algorithme centralisé pour configurer tous ses commutateurs, le réseau Omega utilise une procédure d'auto-routage distribué. Les commutateurs examinent les destinations de leurs données d'entrée et règlent leurs états. Aucun matériel de routage

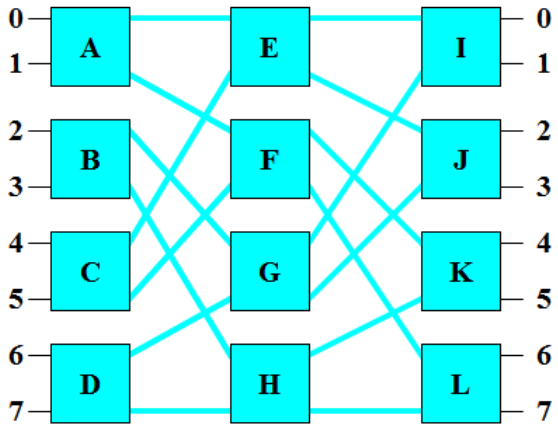


Figure 2.11: Exemple de réseau Omega [3].

central n'est nécessaire. De ce fait, les commutateurs de chaque étage peuvent être configurés en parallèle et le réseau peut être configuré en temps $O(lg\, n)$ [6, 2].

Les quatre commutateurs du 1er étage envoient leurs sorties supérieures aux commutateurs E et G et leurs sorties inférieures aux commutateurs F et H.

Les commutateurs E et G envoient tous les deux leurs sorties aux commutateurs I et J; leurs données ne peuvent atteindre que les sorties réseau 0, 1, 2 et 3.

De même, les données des commutateurs F et H ne peuvent atteindre que les sorties réseau 4, 5, 6 et 7.

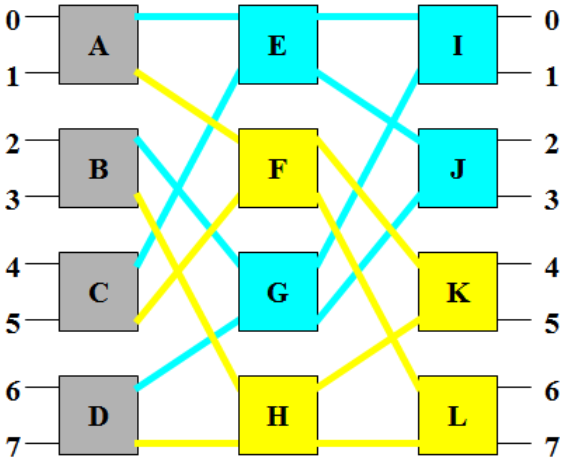


Figure 2.12: Algorithme de routage du réseau Omega [3].

Chaque commutateur du 1^{er} étage doit être réglé de sorte que sa sortie supérieure ait une destination de valeur binaire 000, 001, 010 ou 011, c'est-à-dire qu'elle ait 0 dans la position du premier bit de sa destination. De même, la sortie inférieure de chaque commutateur du 1^{er} étage doit avoir un 1 dans la position du premier bit de sa destination pour atteindre les sorties 100, 101, 110 ou 111.

Par exemple, si l'entrée réseau 0 doit établir une connexion avec la sortie réseau 7 (111), le

commutateur du 1^{er} étage le plus élevé doit se régler à l'état croisé.

Si deux entrées d'un commutateur du 1^{er} étage ont la même valeur à la position du premier bit, le réseau Omega ne peut pas réaliser cette permutation.

Par exemple, si l'entrée réseau 0 possède la sortie réseau 4 et si l'entrée réseau 1 a pour destination la sortie réseau 7, alors le commutateur A est bloqué car les deux 4 (100) et 7 (111) ont le bit 1 dans leur première position de bit (voir la figure 2.13).

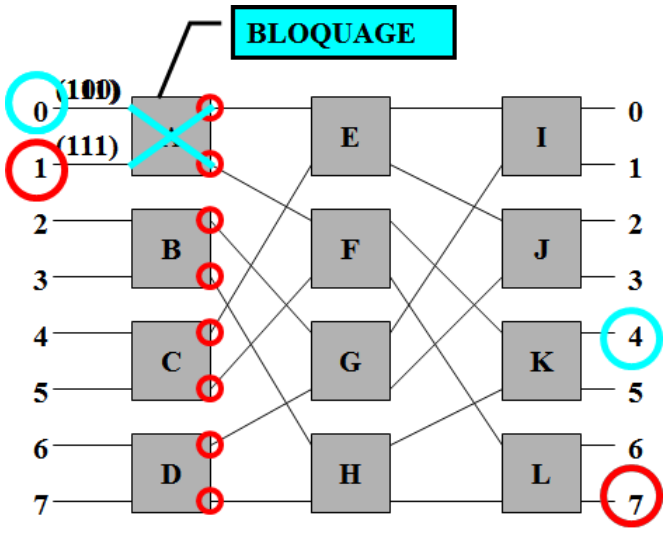


Figure 2.13: Exemple de bloquage dans le réseau Omega [3].

Similairement, le commutateur du $2^{ème}$ étage envoie sa sortie supérieure aux commutateurs I ou K, qui se connectent aux sorties 0 (000), 1 (001), 4 (100) et 5 (101).

Les sorties inférieures peuvent atteindre les commutateurs J ou L, qui peuvent accéder aux sorties 2, 3, 6 et 7 (010, 011, 110 et 111).

Pour le deuxième étage, le deuxième bit de la destination détermine le réglage du commutateur.

De même, le bit le moins significatif de la destination détermine le réglage des commutateurs au $3^{ème}$ étage.

Les sorties du $3^{ème}$ étage sont les sorties du réseau, le dernier étage ne peut pas bloquer une permutation qui a été routée avec succès par les étages précédents.
Les figures 2.15 et 2.16 illstrent des cas de routage réussi et non réussi, respectivement.

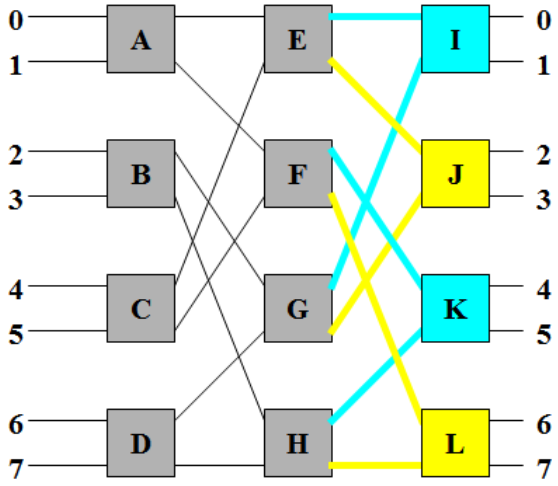


Figure 2.14: Réglage du deuxième étage du réseau Omega [3].

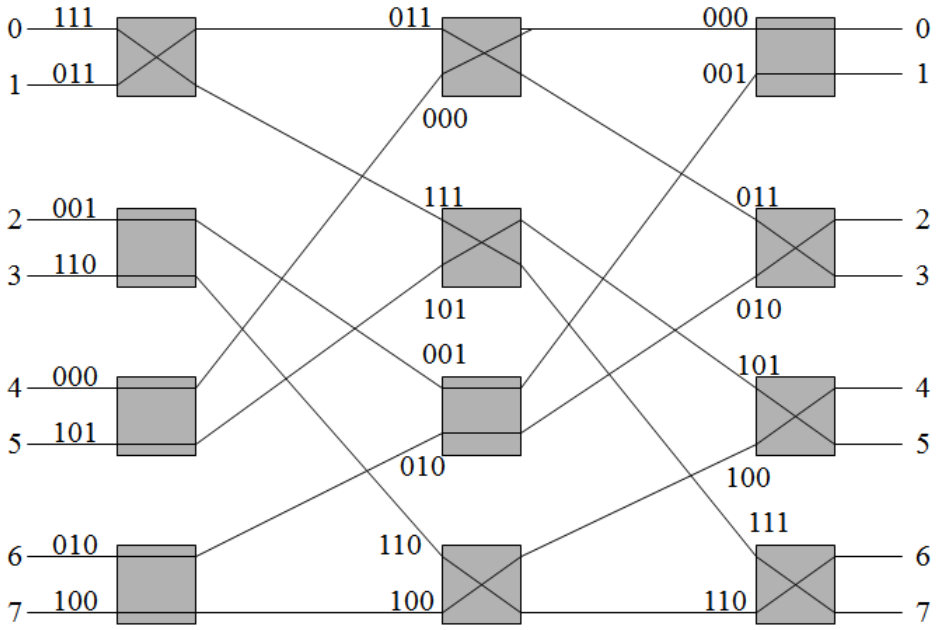


Figure 2.15: Exemple de routage réussi pour le réseau Omega [3].

2.3.2.5.4 Réseau Baseline

Dans cette section, nous allons voir comment créer et comment appliquer l'algorithme de routage d'un réseau Baseline.

A. Création du réseau Baseline

Le réseau Baseline est semblable au réseau Benes, essentiellement la moitié avant d'un réseau Benes. La figure 2.17 montre un réseau de base 8×8. Pour généraliser à un réseau $n \times n$, tout d'abord, il faut créer un étage de $(n/2)$ commutateurs 2×2 . Ensuite, une sortie de chaque commutateur de 2×2 est connectée à une entrée de chaque commutateur de $(n/2) \times (n/2)$. Par la suite, les commutateurs de (n / 2) x (n / 2) sont remplacés par des réseaux de (n / 2) x (n / 2) construits de la même manière [6, 2].

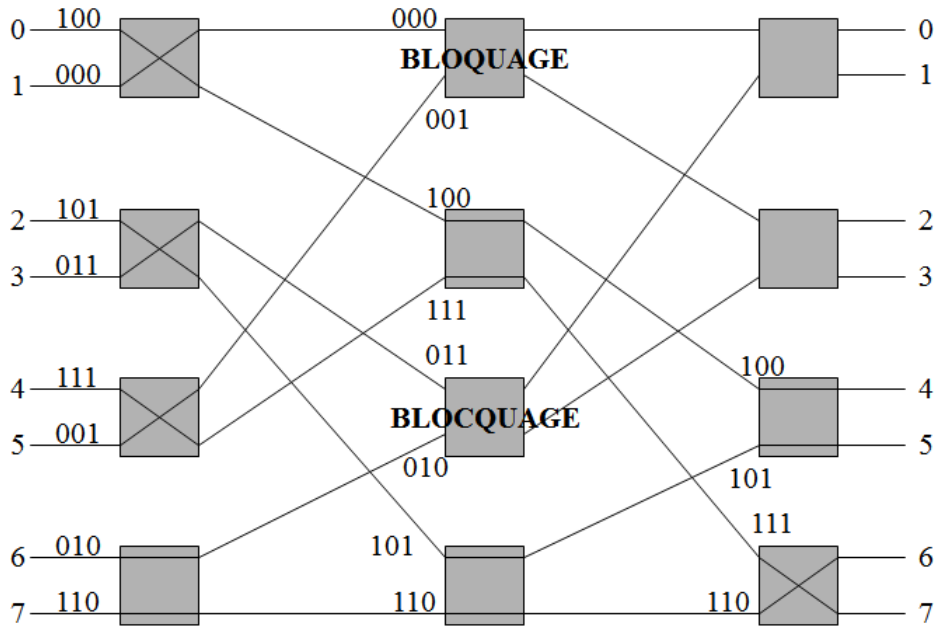


Figure 2.16: Exemple de routage non réussi pour le réseau Omega [3].

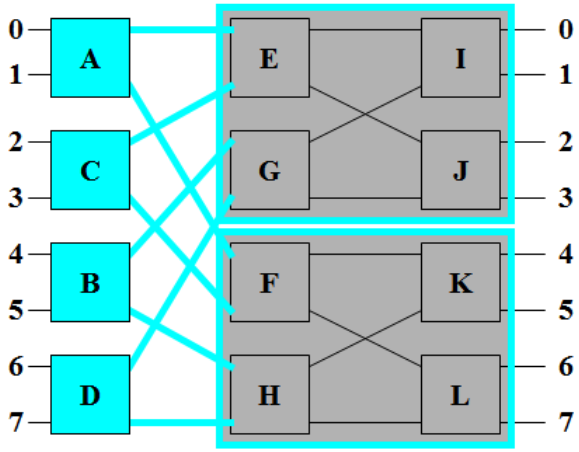


Figure 2.17: Exemple de réseau Baseline [3].

Les réseaux de Baseline et Omega sont isomorphes l'un avec l'autre. En commençant par le réseau Baseline. Si B et C, et F et G sont repositionnés tout en conservant les liens fixes lors du déplacement des commutateurs. Le réseau Baseline se transforme en réseau Omega. Par conséquent, les réseaux Baseline et Omega sont isomorphes (voir la figure 2.18) [6, 2].

B. Algorithme de routage réseau Baseline

Le réseau Baseline utilise le même algorithme de routage utilisé par le réseau Omega.

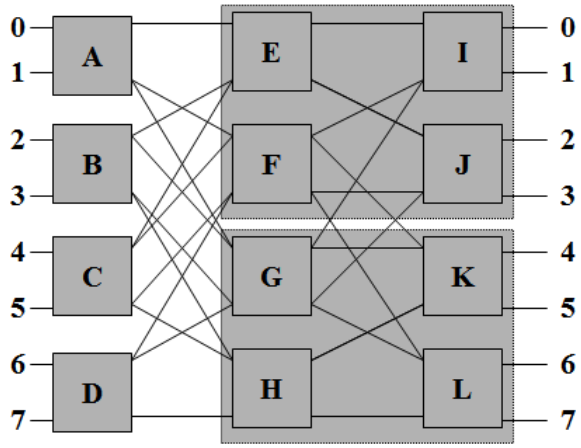


Figure 2.18: Isomorphisme entre réseau Baselinr et réseau Omega [3].

2.4 Techniques de switching

Les techniques de switching sont des méthodes de traitement de paquets de données durant leur transfert d'un processeur source à un processeur de destination. Parmi ces techniques on trouve: stocker et transférer (en anglais: Store and forward), circuit switching, cut through, and wormhole. Le tableaux 2.1 illustre une comparaison entre les diffrentes techniques de switching.

Tableau 2.1: Comparaison entre les différentes Techniques de switching [6].

	Liens	Tampons	Commentaires
Store and Forward	Paquet	Paquet	Tête flit attend pour le reste (queue)
Circuit Switching	Messages	N/A (pas de tampon)	installation et Ack
Virtual Cut Through	Paquet	Paquet	Tête peut avancée
Wormhole	Packet	Flit	HOL
Virtual Channel	Flit	Flit	Entrelacement de flits de differents paquets

2.4.1 Store & forward

lorsqu'un paquet de données atteint un processeur intermédiaire, le paquet est stocké dans une mémoire tampon. Lorsque le canal de sortie suivant est disponible, le paquet est transmis au processeur suivant.

2.4.2 Circuit switching

Tout le chemin via le réseau est réservé avant de transférer le message.

2.4.3 Virtual cut through switching

les paquets de données sont stockés sur des processeurs intermédiaires uniquement si le canal requis suivant n'est pas disponible; sinon, il est transmis immédiatement sans mise en mémoire tampon.

2.4.4 Wormhole routing

Le paquet est divisé en plusieurs parties, avec une partie directrice au cours du chemin. Lorsque cette partie du paquet suit un chemin, les parties restantes suivent selon le mode pipeline. Lorsqu'un canal est utilisé et que la partie principale ne peut pas avancer, elle est bloquée jusqu'à ce que le canal soit libre. Les parties suivantes, plutôt que d'être retirées du réseau, sont mises en mémoire tampon le long du chemin.

Conclusion

Les réseaux d'interconnexion jouent un rôle très important dans la détermination de la performance globale d'un système multiprocesseurs. Pour cela, si ces réseaux ne peuvent pas minimiser le temps de latence des différents messages pour une application particulière, alors, les processeurs vont être fréquemment forcés d'attendre l'arrivée des données. Le tableau 2.2 ci-dessous donne des comparaisons qualitatives entre les différents types de configurations d'interconnexion.

Tableau 2.2: Comparaison entre les différents RIs [3].

Propriétés	Bus	Crossbar	Multi-étages
Vitesse	Faible	Elevée	Elevée
Coût	Faible	élevé	Modéré
Fiabilité	Low	Elevée	Elevée
Configurabilité	Elevée	Faible	Modérée
Complexité	Faible	Elevée	Modérée

Conclusion et perspectives

L'objectif de nos travaux est l'amélioration du taux de reconnaissance d'un système hors-ligne de reconnaissance de l'écriture manuscrite en intégrant l'algorithme de puzzle en post-traitement.

Un texte arabe manuscrit est composé d'un ensemble de caractères qui possèdent une représentation irrégulière, ce qui génère des fluctuations dans les lignes, des espaces irréguliers intra et inter-mots et des chevauchements, caractères mal représentés et des coupures entre les mots et même entre les caractères d'un mot. L'image d'un texte peut être vue comme un jigsaw puzzle à résoudre, où nous devons ajuster les segments afin de restaurer le texte original, et par conséquent, résoudre le problème des coupures, chevauchements et, caractères mal représentés.

Nous avons dégagé de cette étude deux contributions qui peuvent profiter des capacités des algorithmes puzzles en feedback pour surmonter le problème d'irrégularité dans l'écriture cursive manuscrite.

Le problème des coupures ne peut être résolut qu'avec l'utilisation d'une opération de fusion afin de fusionner les différents segments. Par contre, le problème de chevauchements nécessite une opération de division pour qu'il soit résolu. Autant que le problème de caractères mal représentés implique l'utilisation d'un ensemble de règles de morphisme permettant de changer complètement la nature du caractère dans le but de restaurer le caractère original.

La première contribution, se présente par une méthode de segmentation structurelle multi-scans (N-Scans) qui a pour rôle de créer le premier ensemble des segments représentant par la suite l'état initial du jigsaw puzzle. Autant que la deuxième est présentée par un post-processeur puzzle qui possède trois étapes principales : Fusion, Division et Morphisme. En appliquant ces étapes sur l'ensemble initial du puzzle le post-processeur est capable de régénérer des autres sous-ensembles. Ce processus permet de surmonter de manière remarquable les problèmes posés par l'écriture manuscrite en achevant une performance importante de 90.03% de segmentation et un taux de reconnaissance de 96.93%. Les résultats présentés dans la section **??** prouve que la performance du système de reconnaissance peut être améliorée par l'utilisation d'un contrôleur puzzle parce que les résultats de segmentation et les décisions de classificateur SVM sont guidés par l'état objectif du puzzle (les mots modèles de la base) et jugés par le contrôleur du puzzle pour atteindre le meilleur état objectif.

Les perspectives présentées dans ce travail sont nombreuses. L'une des perspectives les plus prometteuses est l'utilisation des étapes du puzzling (Fusion, Division et Morphisme) ou les chevauchements, coupures des segments des caractères et les caractères mal représentés comme une métrique d'évaluation de la qualité d'une écriture, ou dans l'identification des personnes ou de ses

caractères à partir d'écriture (graphologie). Une autre perspective très intéressante consiste à ajouter la détection des points de croisement en utilisant un masque similaire au masque de détection des caractères de nature verticale (voir la section **??**). Ces points vont permettre de surmonter le problème de division de mots avec des caractères complètement enchevêtrés. Par exemple, dans le cas du mot **الحجام** dans l'image  référencée par af03_046, il est impossible de segmenter les caractères **ا**, **ل**, **ح** par l'application d'une opération de division en se basant sur les points de fusion et de division précédemment indiqués. L'ensemble des points de croisements permet de générer d'autres sous-segments (la figure ci-dessous (A)) qui contiennent les caractères corrects.

(A) Segments de caractères corrects

(B) Segments non corrects

Une autre piste d'amélioration intéressante consiste à utiliser une heuristique adaptative tel que PSO afin de rendre le système capable de générer de nouvelles règles permettant d'améliorer l'ensemble du puzzle. Une autre piste non moins intéressante consisterait à étendre le travail à la reconnaissance des écritures manuscrites artistiques telles que la montre la figure ci-dessous.

Bibliographie

[1] G. Abandah et N. Anssari – "Novel moment features extraction for recognizing handwritten arabic letters", *Journal of Computer Science* **5** (2009), no. 3, p. 226.

[2] A. BEKLEY – "Contribution à l'étude des réseaux d'interconnexion des machines parallèles -utilisation des hyperfréquences-", Thèse, Université El-Hadj Lakhdar Batna, 1994.

[3] W. Chan et S.-M. Lee – "Interconnection networks in multiprocessor systems", San Jose State University, Février 2011.

[4] T. GARCIA – "Les bases de mpi (message passing interface)", Atelier T8.AP03, JDEV'2017, February 2017.

[5] E. Goubault – "Introduction aux architectures parallèles", http://www.enseignement.polytechnique.fr/profs/informatique/Eric.Goubault/Cours05html/poly002.html, 2019, [Online; accessed 21-Mars-2019].

[6] N. Honarmand – "Interconnection networks", CSE 610 – Parallel Computer Architectures, Stony Brook University, Fall 2015.

[7] C. Jalel, D. Isabelle, G. Denis, L. Pierre-Francois, L. Dimitri et W. Philippe – "Message passing interface (mpi)", Institut du développement et des ressources en informatique scientifique, Juin 2011.

[8] M. JEMNI – "Introduction aux algorithmes et architectures paralléles", École Supérieure des Sciences et Techniques de Tunis, Février 2004.

[9] W. Kendall – "MPI Tutorial Introduction", http://mpitutorial.com/tutorials/mpi-introduction/, 2019, [Online; accessed 09-Mars-2019].

[10] A. Marchand – "Les architectures parallèles", obspm, France, Fall 2015.

[11] R. Margaret et B. Stephen – "message passing interface (MPI)", https://searchenterprisedesktop.techtarget.com/definition/message-passing-interface-MPI, 2019, [Online; accessed 09-Mars-2019].

[12] F. PELLEGRINI – "Architectures et systèmes des calculateurs parallèles", ENSEIRB, France, Octobre 2003.

[13] H. Schahrazed – *Segmentation de textes en caracteres pour la reconnaissance optique de l'ecriture arabe*, Mémoire, Université El-Hadj Lakhdar Batna, 2007.

[14] B. Semelin – "Programmation parallèle pour le calcul scientifique", Méthodologie M2, Février 2014.

Annexes

Annexe A

Installation de la bibliothèque MPI

Introduction

MPI (Message Passing Interface) est une bibliothèque standardisée et portable développé par un groupe de chercheurs de l'université et de l'industrie pour fonctionner sur une grande gamme d'architectures parallèles. Le standard définie la syntaxe et la sémantique d'un cœur de bibliothèque de routines utiles pour une large éventail d'utilisateurs écrivant des programmes de passage de messages portables en C, C++, Java, Matlab et Fortran. Il existe plusieurs implémentations efficaces et bien testées de MPI, dont beaucoup sont libre (open-source). Ceux-ci ont favorisé le développement d'une industrie parallèle du logiciel, et encouragé le développement d'applications parallèles à grande échelle, portables et évolutives selon le modèle de programmation SPMD (**S**ingle **P**rogram **M**ultiple **D**ata).

L'objectif de cette annexe consiste d'une part à introduire et à présenter la bibliothèque MPI qui permet d'établir une communication entre des processus dans un environnement des processus distribués, et d'autre part à donner les différentes étapes nécessaires pour installer et configurer cette bibliothèque avec Dev-C++ en C++, et Eclipse pour Java.
w

A.1 C'est-quoi MPI ?

Avant les années 1990, la plupart des applications parallèles étaient dans les domaines de la science et de la recherche. Le modèle le plus couramment adopté par les bibliothèques était le modèle de passage de messages. Cela signifie simplement qu'une application passe des messages entre les processus afin d'effectuer une tâche. Comme la plupart des bibliothèques utilisaient actuellement le même modèle de passage de messages, avec seulement quelques différences mineures entre elles, les auteurs des bibliothèques et d'autres se sont réunis à la conférence Supercomputing 1992 pour définir une interface standard permettant le passage des messages - l'interface de passage de messages (MPI: Message Passing Interface). Cette interface standard permettrait aux programmeurs d'écrire des applications parallèles portables pour toutes les principales architectures parallèles. Elle va les permettent également d'utiliser les fonctionnalités et les modèles auxquels ils étaient déjà habitués dans les bibliothèques populaires actuelles. En 1994, une interface complète et une norme ont été définies (MPI-1). A tenir en compte que celle-ci n'est qu'une définition d'interface, et c'est à développeurs de créer des implémentations de l'interface pour leurs architectures respectives [9, 11].

MPI n'est pas reconnu comme norme officielle par un organisme de normalisation tel que IEEE ou ISO, mais il est généralement considéré comme le standard de l'industrie et constitue la base de la plupart des interfaces de communication adoptées par les programmeurs de l'informatique parallèle. L'ancien standard MPI 1.3 (appelé MPI-1) fournit plus de 115 fonctions. La dernière norme MPI 2.2 (ou MPI-2) offre plus de 500 fonctions et est largement compatible avec les versions antérieures de MPI-1. Cependant, pas toutes les bibliothèques MPI fournissent une implémentation complète de MPI-2. Aujourd'hui, le forum MPI est entrain d'élaborer le standard MPI-3 afin d'améliorer l'évolutivité, d'améliorer les performances, d'inclure la prise en charge de multicœurs et de clusters et d' interopérer avec plus d'applications. Aujourd'hui, le forum MPI est entrain d'élaborer le standard MPI-3 afin d'améliorer l'évolutivité, d'améliorer les performances, d'inclure la prise en charge de multicœurs et de clusters et d' interopérer avec plus d'applications. Il existe plusieurs implémentations de MPI, parmi les on trouve: OpenMPI, MPICH, MVAPICH, IntelMPI et DeinoMPI [11, 4, 7].

A.1.1 Historique de MPI

Plusieurs versions de la bibliothèque MPI ont été développer de l'année 1994 jusqu'aujourd'hui, parmi les on trouve [7]:

▶ ***Version 1.0 :*** en juin 1994, le forum MPI (Message Passing Interface Forum), avec la participation d'une quarantaine d'organisations, abouti à la définition d'un ensemble de sous-programmes concernant la bibliothèque d'échanges de messages MPI.

▶ ***Version 1.1 :*** juin 1995, avec seulement des changements mineurs.

▶ ***Version 1.2 :*** en 1997, avec des changements mineurs pour une meilleure cohérence des dénominations de certains sous-programmes.

▶ ***Version 1.3 :*** septembre 2008, avec des clarifications dans MPI 1.2, en fonction des clarifications elles-mêmes apportées par MPI-2.1.

▶ ***Version 2.0 :*** apparue en juillet 1997, cette version apportait des compléments essentiels volontairement non intégrés dans MPI 1.0 (gestion dynamique de processus, copies mémoire à mémoire, entrées-sorties parallèles, etc.)

▶ ***Version 2.1 :*** juin 2008, avec seulement des clarifications dans MPI 2.0 mais aucun changement.

▶ ***Version 3.0 :*** septembre 2012, changements et ajouts importants par rapport à la version 2.2. un meilleur support des applications actuelles et futures, notamment sur les machines massivement parallèles et many cores. Les principaux changements actuellement envisagés :

- communications collectives non bloquantes ;
- révision de l'implémentation des copies mémoire à mémoire ;
- tolérance aux pannes ;
- Fortran (2003-2008) bindings ;
- interfaçage d'outils externes (pour le débogage et les mesures de performance) ;

A.1.2 Types de données MPI

Les données dans un message envoyé ou reçu sont décrites par un triplet : (address, count, datatype), où [8]:

- address : l'adresse (un pointeur);
- count : le nombre d'éléments de type datatype;
- datatype : un nom de type MPI pour les données.

Le tableau 1.1 des exemples de type de données MPI prédéfinis (non exhaustif) et association avec C.

Tableau 1.1: Types de données MPI.

C/C++	MPI
char, unsigned char	MPI_CHAR, MPI_UNSIGNED_CHAR
int, unsigned int	MPI_INT, MPI_UNSIGNED_INT
long, unsigned long	MPI_LONG, MPI_UNSIGNED_LONG
float, double	MPI_FLOAT, MPI_DOUBLE
packet d'octet (void*)	MPI_PACKED

A.1.3 Modes de Communication

Généralement, il existe 04 modes de communication entre les différents processus exécutant en parallèle, à savoir:

1. ***standard :*** l'utilisateur peut modifier la donnée dès le retour du sous-pogramme. Il est à la charge de MPI d'effectuer ou non une recopie temporaire du message. Si c'est le cas, l'envoi se termine avant que la réception ne soit postée.
2. ***synchroneous :*** l'envoi du message ne se termine que si la réception a été postée et la lecture du message terminée. C'est un envoi couplé avec la réception.
3. ***buffered :*** il est éventuellement à la charge du programmeur d'effectuer une recopie temporaire du message. L'envoi est ainsi découplé de la réception.
4. ***ready :*** l'envoi du message ne peut commencer que si la réception a été postée auparavant (ce mode est intéressant pour les applications clients/serveurs).

Il existe une panoplie de fonctions pour effectuer des communications bloquante ou non dans le différent mode. Elle s'utilisent avec les même arguements que MPI_SEND et MPI_RECV, plus un argument "request" pour les fonctions non-bloquantes [14].

Tableau 1.2: Association des fonctions MPI avec les modes de communication.

	Bloquant	Non-bloquant
Standard	MPI_SEND	MPI_ISEND
Synchrone	MPI_SSEND	MPI_ISSEND
Ready	MPI_RSEND	MPI_IRSEND
Buffered	MPI_BSEND	MPI_IBSEND

- Le mode **buffered** nécessite une copie mémoire de plus mais permet de continuer les calculs à coup sûr.
- Le mode **ready** diminue la latence, mais est délicat à utiliser (synchronisation).

A.1.4 Fonctions de base

Dans cette section, nous allons voir quelques fonctions de base de la bibliothèque MPI. Parmi ces fonctions on trouve [14]:

- MPI_COMM_SIZE(comm,size,err): locale. Renvoie size, le nb de process dans comm.
- MPI_COMM_RANK(comm,rank,err): locale. Renvoie rank, le n° du process appelant.
- MPI_COMM_GROUP(comm,group,err): locale! Crée un groupe group avec tout les process de comm.
- MPI_GROUP_INCL(group,n,rank_array,subgroup,err): locale. Crée un groupe subgroup, avec un sélection de n process de group définie par rank_array.
- MPI_COMM_CREATE(comm,subgroup,subcomm,err): collective. Crée l'intra-communicateur subcomm de subgroup.
- MPI_GROUP_SIZE(group,size,err): locale. Renvoie size, le nb de process dans group.
- MPI_GROUP_RANK(group,rank,err): locale. Renvoie rank, le n° du process appelant.
- MPI_GROUP_FREE(group,err): locale. Désalloue le groupe group.
- MPI_COMM_FREE(comm,err): collective. Désalloue le communicateur comm.

A.1.4.1 Tags des messages MPI

Tous les messages envoyés sont accompagnés d'une étiquette ou tag (valeur entière représentant le numéro du message) qui sert au récepteur à identifier un message. Le processus récepteur doit spécifier qu'il attend la réception d'un message en précisant un tag :

1. En donnant une valeur entière
2. En donnant la valeur MPI_ANY_TAG qui correspond à toute les valeurs

A.1.4.2 Communicateur MPI

Lors du lancement d'un programme MPI, tous les processus appartient au même groupe, c'est le communicateur MPI. Le communicateur par défaut ou global est appelé MPI_COMM_WORLD. Chaque processus possède un identificateur dans ce groupe, c'est le rang du processus qui peut prendre une valeur de 0 jusqu'au nombre des noeuds (N) mois un ($ID \in [0, N-1]$). La figure 1.1 présente une illustration[14]
.

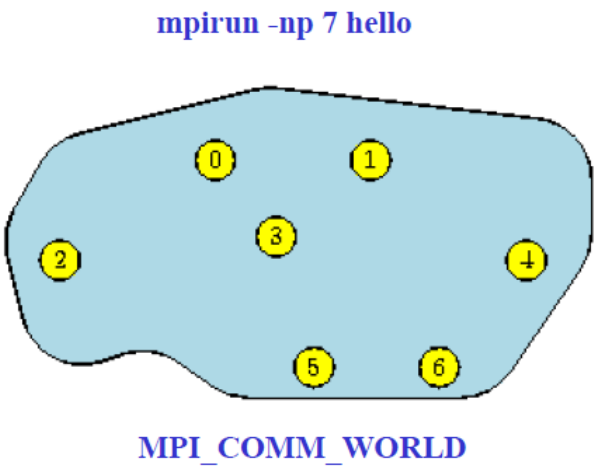


Figure 1.1: Illustration d'un communicateur global pour 07 processus.

A.1.4.3 Types d'appels de fonctions MPI

Généralement, il existe 05 types d'appels des fonctions MPI, à savoir [7]:

1. Local : ne nécessite aucune communication. Par ex. pour générer un objet MPI local.
2. **Non-local :** nécessite l'exécution d'une procédure MPI sur un autre processus. Par exemple, un envoi de message (nécessite la réception).
3. **Bloquant :** à la fin d'un tel appel, le programme appelant peut réutiliser les ressources utilisées dans l'appel: par exemple un buffer.
4. **Non-bloquant :** à la fin d'un tel appel, le programme doit attendre que l'opération se termine avant de réutiliser les ressources utilisées. Par ex: un envoi de message non-bloquant peut libérer le processus émetteur avant que le message ne soit envoyé.
5. **Collectif:** Nécessite l'appel de la même fonction par tous les processus (d'un même groupe de processus). Par exemple broadcast, diffusion d'une valeur d'un processus à tous les autres.

A.1.4.4 Primitive d'envoi de message MPI_Send

L'envoi d'un message de donnée (**buffer**, **count**, **datatype**) à un noeud de rang **dest**, avec le tag **tag** dans le communicateur **comm** est réalisé à l'aide de la primitive:

MPI_Send(***buffer***, ***count***, ***datatype***, ***dest***, ***tag***, ***comm***)

Les arguments sont tous des paramètres d'entrée à la fonction :

- ***buffer:*** adresse du début du buffer (donnée à envoyer);
- ***count :*** nombre de valeurs à envoyer (par rapport au type de données);
- ***datatype :*** type de chaque valeur;
- ***dest :*** rang du processus de destination;
- ***tag :*** étiquette pour identifier le message;
- ***comm :*** communicator: groupe de processus.

► ***Remarques:***

1. Dans le cas le plus simple, on fixe comm à MPI_COMM_WORLD;
2. Entre deux processus, les communications dans MPI sont FIFO.
3. Comme un processus peut recevoir plusieurs messages (et éventuellement plusieurs d'un même émetteur), ***tag*** peut servir à les identifier. Si leur identité est connue (par le contexte du programme), on peut fixer ***tag*** à n'importe quelle valeur lors de l'émission et à MPI_ANY_TAG lors de la réception.
4. L'émetteur place les valeurs à émettre dans des lieux successifs de la mémoire locale avant d'appeler MPI_Send
5. Si on fixe count=0, le message ne contiendra que ses informations de controle source, destination, tag et communicateur et pourra servir à une synchronisation ou à signifier un événement.

► ***Exemple 1:*** envoi d'un entier au processus de rang 0

1. int integer = 31415;
2. int tag = 37;
3. MPI_Send(&integer, 1, MPI_INT, 0, tag, MPI_COMM_WORLD);

► ***Exemple 2:*** envoi d'un tableau de « float »

1. float array [256]; /* plus initialisation */
2. int tag = 39;
3. MPI_Send(array, 128, MPI_FLOAT, 0, tag, MPI_COMM_WORLD);

A.1.4.5 Primitive de réception de message MPI_Recv

La réception d'un message de donnée (buffer, count,datatype) provenant du noeud de rang **dest**, avec le tag **tag** dans le communicateur **comm**, et avec un état de réception status peut être réalisée à l'aide de la primitive:

MPI_Recv (***buffer***, ***count***, ***datatype***, ***dest***, ***tag***, ***comm***, ***status***)

Les arguments buf et status sont des paramètres de sortie, les autres sont des paramètres d'entrée :

1. le ***buffer*** de réception est constitué de ***count*** valeurs successives de type ***datatype*** situées à partir de l'adresse ***buffer***. Si moins de valeurs sont reçues, elles sont placées au début. S'il y en a trop, une erreur de débordement est déclenchée.
2. le message sélectionné est le premier dans la file de réception dont l'étiquette et le rang d'émetteur sont tag et source; on peut fixer tag à MPI_ANY_TAG et/ou source à MPI_ANY_SOURCE pour ne pas tenir compte d'une ou l'autre de ces valeurs.
3. ***status*** est une structure de type **MPI_Status** contenant trois champs **status.MPI_SOURCE**, **status.MPI_TAG** et **status.MPI_ERROR** contenant l'émetteur effectif, l'étiquette et le code d'erreur.

Pour connaître le nombre exact de valeurs reçues, il faut “extraire” cette information de status en lui appliquant la fonction suivante :

int MPI_Get_count(MPI_Status ***status***, MPI_Datatype ***datatype***, int ***count***)

dont les arguments ***status*** et ***datatype*** sont en entrée et ***count*** en sortie. Il faut que ***datatype*** soit le type des valeurs spécifié dans l'appel à MPI_Recv et que ***status*** soit la valeur obtenue.

► ***Exemple 1:*** réception d'un entier du noeud de rang 2:

```
1. int integer;
2. int tag = 37;
3. MPI_Status status;
4. MPI_Recv( &integer, 1, MPI_INT, 2, tag, MPI_COMM_WORLD, &status );
```

► ***Exemple 2:*** réception d'un tableau de « float »:

```
1. float array [256];
2. int tag = 39;
3. MPI_Recv( array, 128, MPI_FLOAT, 2, tag, MPI_COMM_WORLD, &status );
```

A.2 Dev-C++ et MPI

Dans cette section, nous allons voir les différentes étapes nécessaires pour installer la bibliothèque MPI et la configurer avec l'outil d'édition et compilation Dev-C++. Le processus d'installation et de configuration suit les étapes suivantes:

1. Installer Dev-C++;
2. Installer mpich.nt.1.2.5_2 dans C:\MPICH;
3. Créer un projet avec un fichier source.
4. Pour chaque projet Dev-C++ utilisant MPI faire les étapes suivantes :

   (1) Choisir du menu : Tools|Compiler Options :

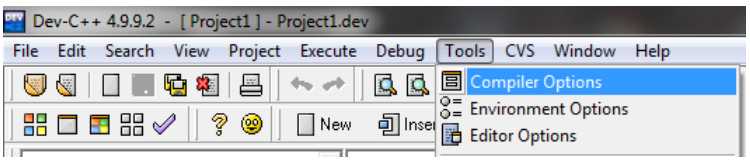


Figure 1.2: Choix des options du compilateur.

La fenêtre suivante s'affiche (voir la figure 1.3):

   (2) Choisir l'onglet **Directories** (voir la figure 1.4).

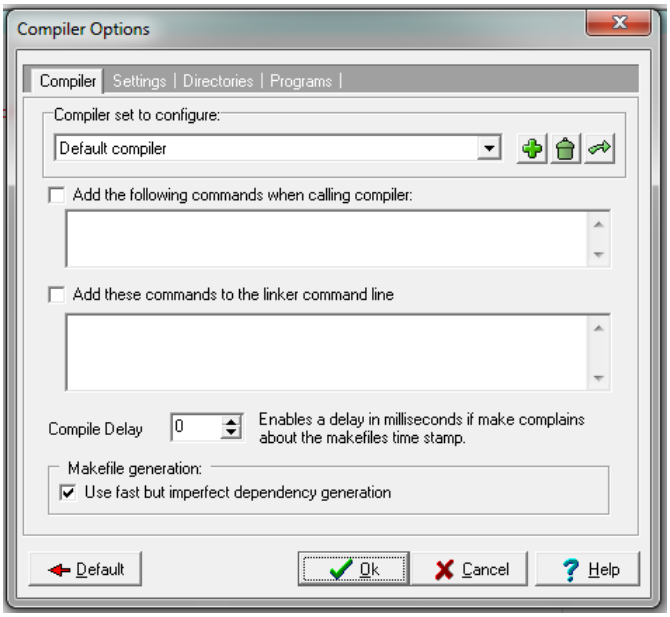


Figure 1.3: Fenêtre des options du compilateur.

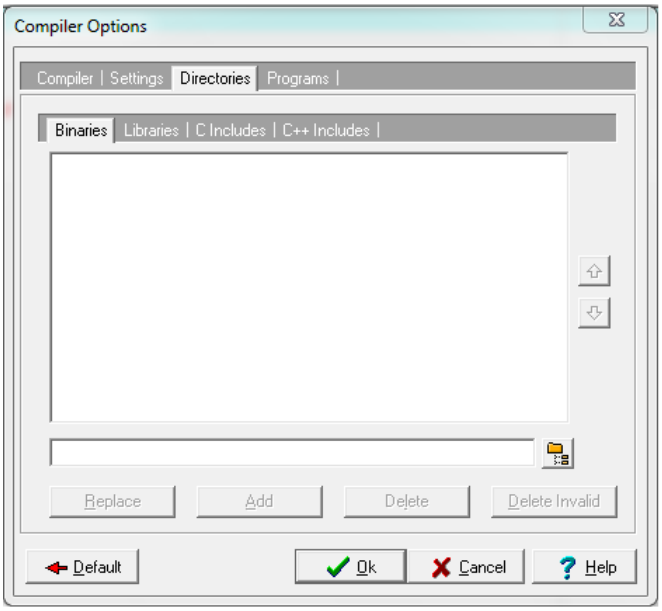


Figure 1.4: Choix de l'onglet **Directories**.

(3) Choisir l'onglet **Binaries** et cliquer sur l'icône dans le cadre rouge pour choisir le **bin** de MPI dans **C:\MPICH\mpd\bin** ensuite cliquer sur **OK** (voir la figure 1.5).

(4) Cliquer sur le bouton Add (voir la figure 1.6).

(5) Choisir l'onglet **Libraries** et cliquer sur l'icône dans le cadre rouge pour choisir le répertoire **lib** de MPI dans **C:\MPICH\SDK.gcc \lib** ensuite cliquer sur **OK**. Cliquer sur le bouton **Add**.

(6) Choisir l'onglet **C Includes** et cliquer sur l'icône dans le cadre rouge pour choisir le répertoire **include** de MPI dans **C:\MPICH \SDK.gcc\include** ensuite cliquer sur **OK**. Cliquer sur le bouton **Add**.

(7) Appuyer sur **Alt+P**, la fenêtre suivante s'affiche (voir la figure 1.7):

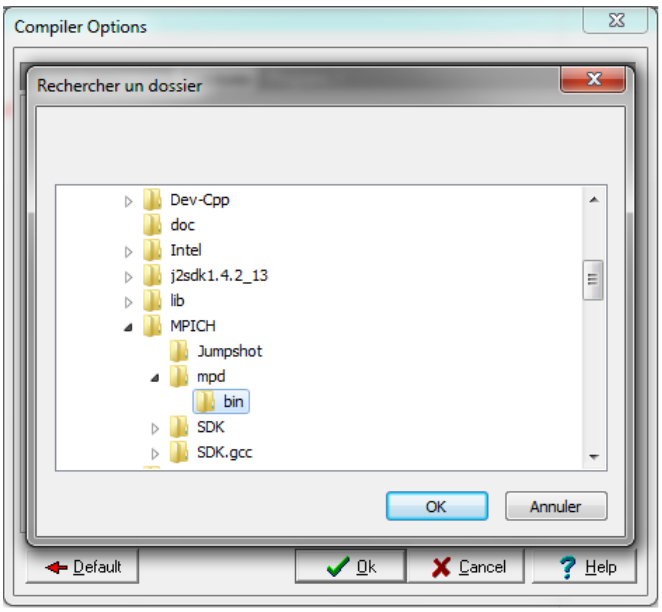


Figure 1.5: Choix du répertoire bin .

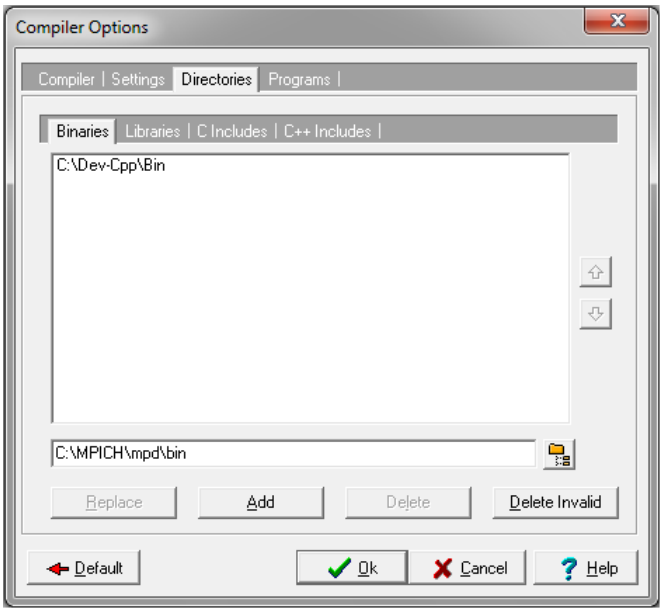


Figure 1.6: Ajouter le répertoire bin choisit.

(8) Choisir l'onglet **Parameters**, et cliquer sur le bouton **Add Library or Object** pour choisir le répertoire **C:\MPICH\SDK.gcc\lib**. Sélectionner tous les fichiers dans le répertoire et Cliquer sur **Ouvrir** ensuite **OK** (voir la figure 1.8).

(9) Actuellement, le projet est prêt à utiliser la bibliothèque MPI. Pour le tester insérer le programme suivant (voir la figure 1.9):

(10) Pour lancer l'exécution du programme on utilise le fichier exécutable **mpirun** dans le répertoire **C:\MPICH\mpd\bin** de la façon suivante (voir la figure 1.10):

(11) Dans la réalité pour simplifié un peut cette commande il vaut mieux ajouter le chemin **C:\MPICH\mpd\bin** dans le path de windows comme suit :

A. Cliquer sur le bouton droit de la sourie sur l'icône de l'ordinateur et choisir **propriétés**

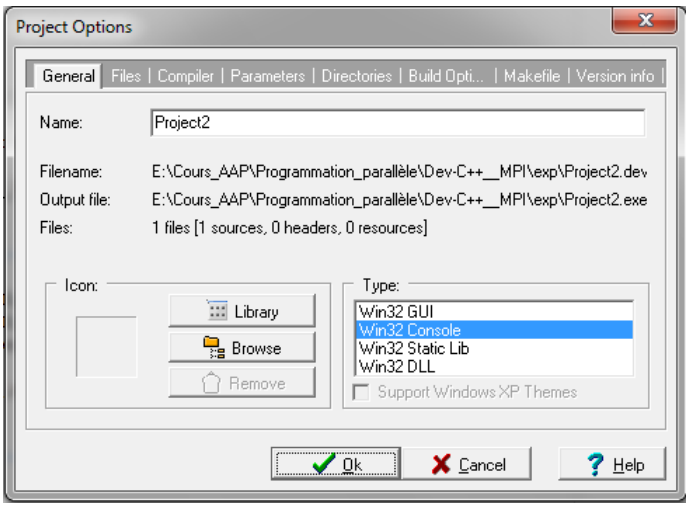


Figure 1.7: Fenêtre des options du projet.

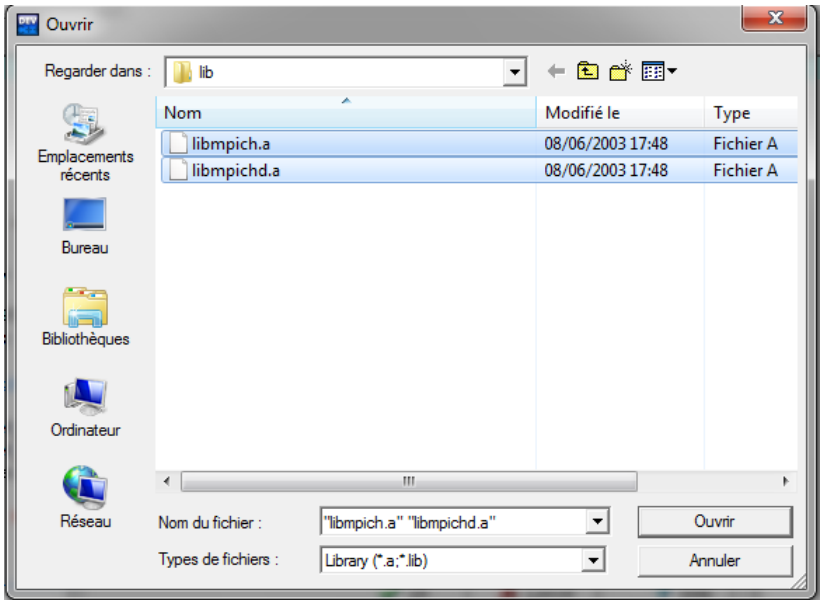


Figure 1.8: Choix des fichiers lib.

(voir la figure 1.11).

La fenêtre représentée par la figure 1.12 s'affiche :

B. Choisir **Paramètres système avancés**, pour afficher la fenêtre suivante (voir la figure 1.13):

C. Cliquer sur le bouton **Variables d'environnement** (voir la figure 1.14).

D. Choisir **Path** parmi les **variables système**, ensuite cliquer sur le bouton **modifier** pour afficher la fenêtre suivante (voir la figure 1.15):

E. Ajouter le chemin **C:\MPICH\mpd\bin** dans la valeur de la variable et cliquer sur **OK** .

F. Maintenant, la commande précédente peut être reformulée comme suit (voir la figure 1.16):

```c
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int main(int argc, char *argv[])
{
  int done = 0, n, myid, numprocs, i;
  int  namelen;
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  MPI_Init(&argc,&argv);

  //------ Main MPI program ------------------

  MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
  MPI_Comm_rank(MPI_COMM_WORLD,&myid);
  MPI_Get_processor_name(processor_name,&namelen);

  printf("Process %d on %s\n",myid, processor_name);


  //-----------------------------------------
  MPI_Finalize();
  printf("test...");
  system("pause");
  return 0;
}
```

Figure 1.9: Exemples de programme avec MPI.

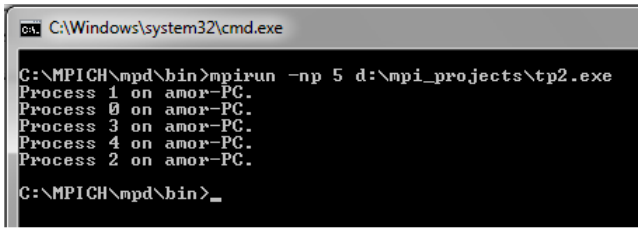


Figure 1.10: Lancer l'exécution avec le chemain complet de **mpirun**.

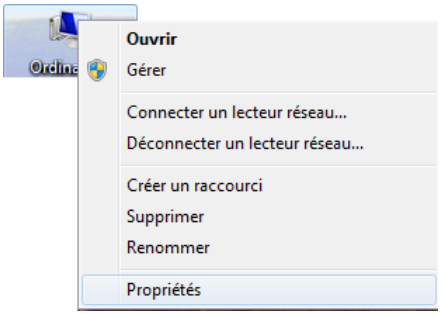


Figure 1.11: Choix des propriétés système.

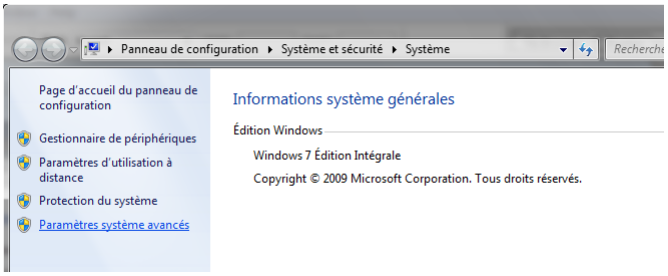


Figure 1.12: Choix des paramètres système avancées.

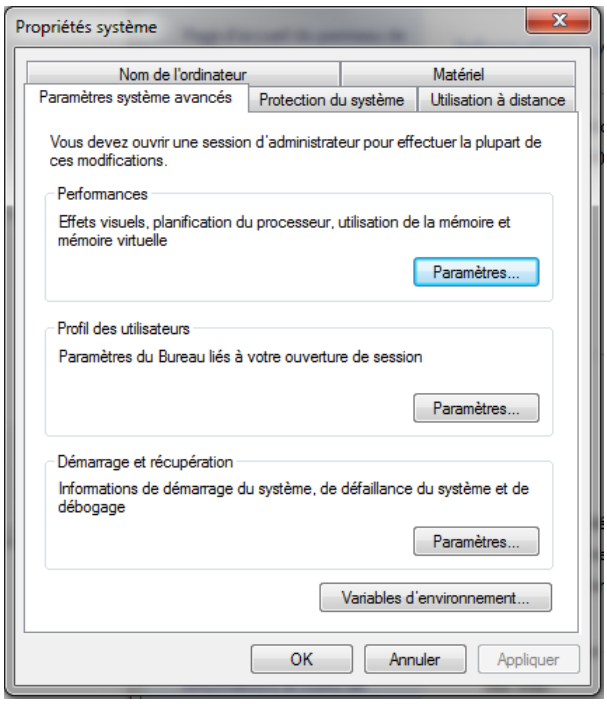


Figure 1.13: Fenêtres des variables système.

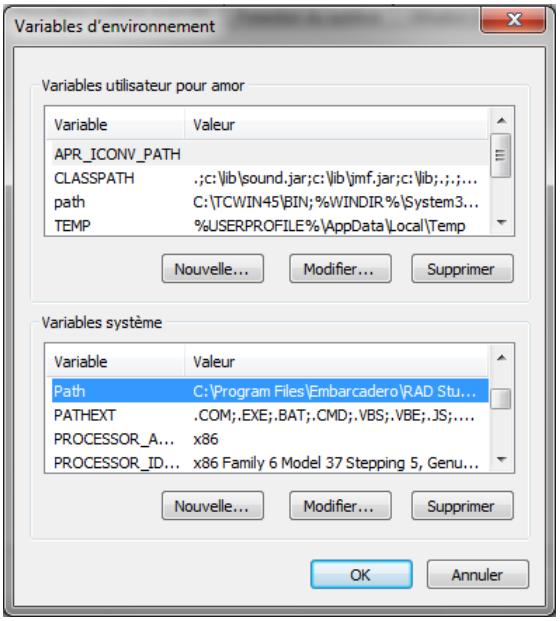


Figure 1.14: Sélection de la variables Path.

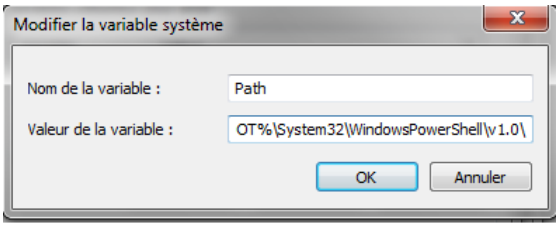


Figure 1.15: Fenêtres de modification de la variable Path.

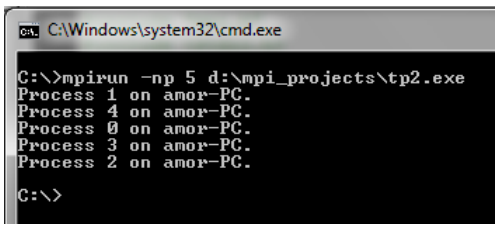


Figure 1.16: Lancer l'exécution sans le chemain complet de **mpirun**.

***Remarque*:**

Cette installation et configuration est testée avec: Dev-C++ 4.9.9.2, mpich.nt.1.2.5_2 et windows 7.

A.3 MPJ (MPI for Java) et Eclipse

Dans cette section, nous allons voir les différentes étapes nécessaires pour installer la bibliothèque MPI pour Java (MPJ) et la configurer avec l'outil d'édition et compilation Eclipse. Le processus d'installation et de configuration suit les étapes suivantes:

1. Décompresser le fichier dans la partition **C**.
2. Installer le **jdk-8u25-windows-i586**.
3. Lancer en tant qu'administrateur **installmpjd-windows.bat** dans le **bin**.
4. Entrer dans Panneau de **Configuration-> Outils Administrative ->Services-> MPJ Daemon** et lancer le service.
5. Ajouter aux variables d'environnement du système la variable MPJ_HOME avec la valeur : **C:\MPJ\mpj-v0_38**.
6. Ajouter à la variable **Path** du bin du **MPJ** : **C:\MPJ\mpj-v0_38\bin**;
7. Copier **MPJ_Express_Debugger_1.0.0.20140211** dans le répertoire **plugins**.
8. Lancer **Eclipse** en tant qu'administrateur (voir la figure 1.17):

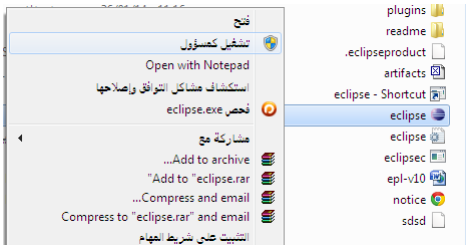


Figure 1.17: Lancer l'exécution d'**Eclipse** en tant qu'administrateur.

9. Créer un nouveau projet java (**Java Project**) avec le **nom MPJ_Test** et cliquer sur **Next**, la fenêtre suivante s'affiche (voir la figure 1.18):

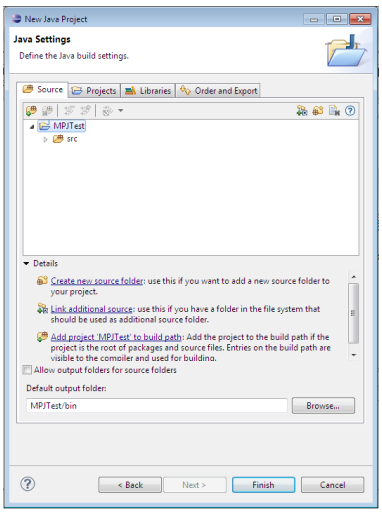


Figure 1.18: Création d'un nouveau projet Java.

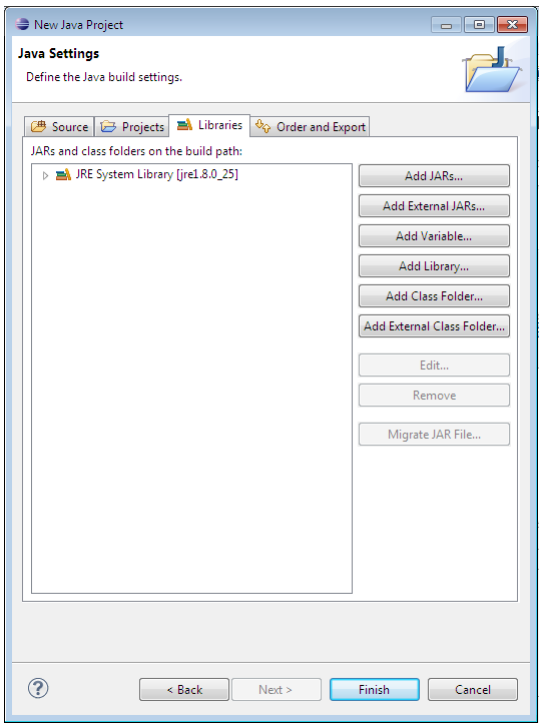


Figure 1.19: Choisir l'onglet Libraries.

10. Choisissez l'onglet Librairies comme suit (voir la figure 1.19):

11. Cliquer sur le bouton **Add External JARs...** et choisissez **mpj.jar** du répertoire **lib** (voir la figure 1.20):

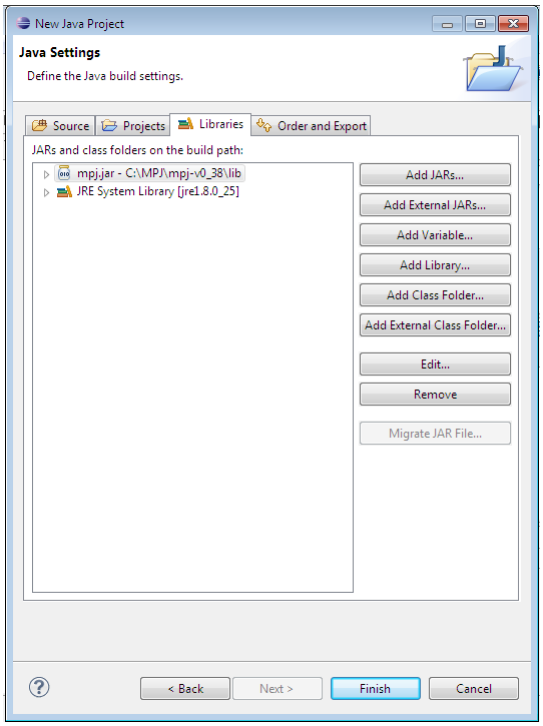


Figure 1.20: Ajouter mpj.jar au nouveau projet Java.

12. Cliquer sur **Finish**, ensuite choisissez le package **MPJ_Test** el cliquer sur **src** (voir la figure 1.21).

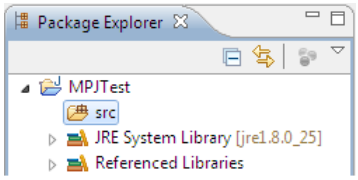


Figure 1.21: Choisir le package du programme.

13. Cliquer sur l'icône pour ajouter une nouvelle classe avec le nom Bonjour et cliquer sur finish.

14. Taper le programme suivant:

```
import mpi.*;
public class Bonjour {
public static void main(String args[]) throws Exception {
MPI.Init(args);
int rang = MPI.COMM_WORLD.Rank();
int nbr_proc = MPI.COMM_WORLD.Size();
System.out.println("Bonjour je suis le processus numero <"+rang+"> parmi "+nbr_proc);
MPI.Finalize();
}
}
```

15. Choisissez le menu **Run->Run Configurations...** et double cliquer sur **MPJExpress Application** (voir la figure 1.22).

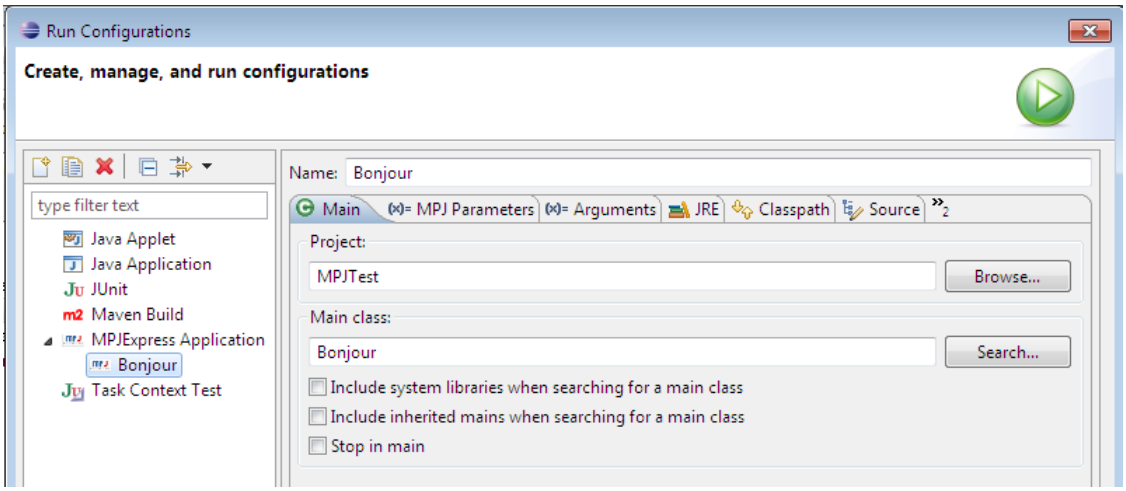


Figure 1.22: Choisir la fenêtre de configuration.

16. Choisissez l'onglet **MPJ Parameters** et choisir un nombre de processus par exemple 5 (voir la figure 1.23):

17. Cliquer sur **Apply** et ensuite **Run** pour exécuter.

Conclusion

Dans cette annexe, nous avons vu une introduction et un survol présentant la bibliothèque de passage de messages MPI. Ce dernier est capable de réaliser une communication entre des différents processus

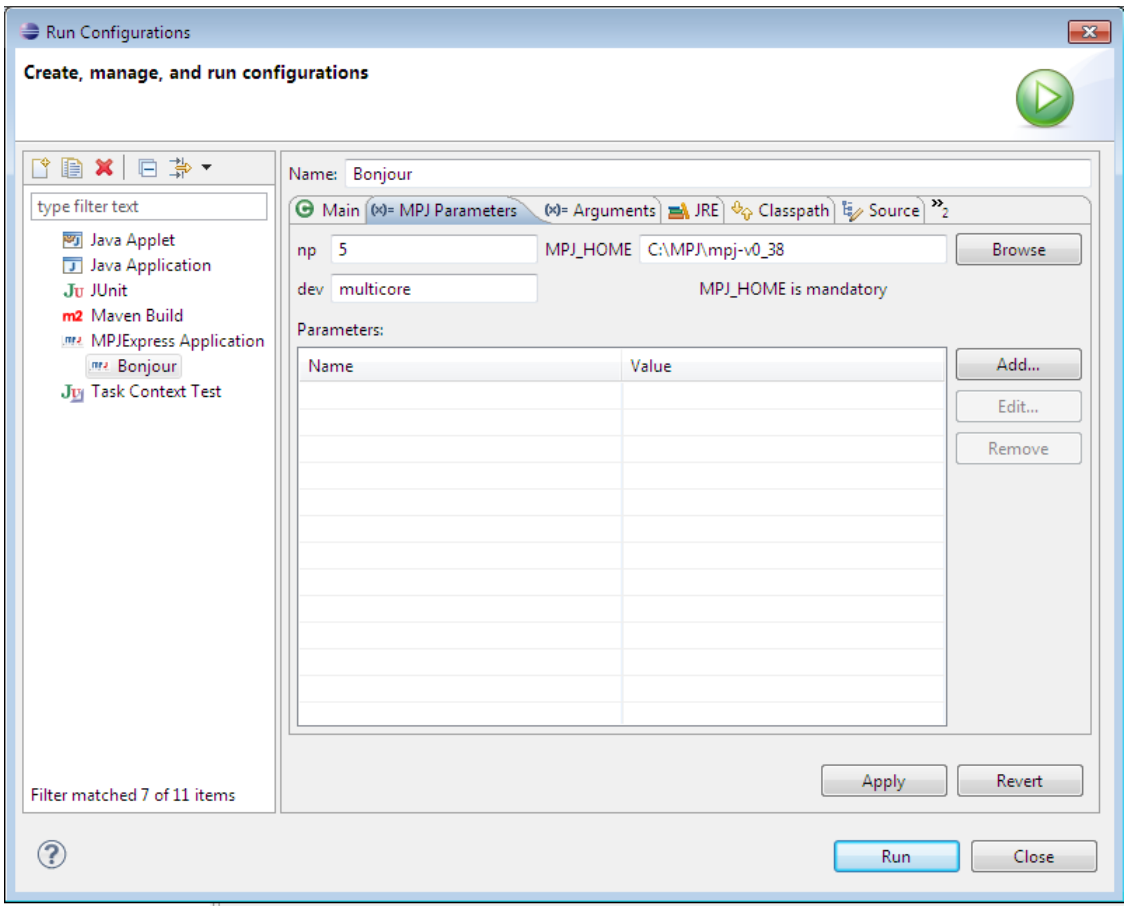


Figure 1.23: Choisir les paramètres à passer au programme MPI.

dans un environnement distribué en suivant le modèle de programmation SPMD. En plus, nous avons donné les différentes étapes nécessaires pour faire une installation et configuration réussi pour les deux environnements de développement Dev-C++ et Eclipse.