

# **CALCUL PARALLELE. TECHNIQUES DE BASE**

*Transparents du cours*

**Tome 1 – Notions fondamentales**

*Rémi COUDARCHER*



Ecole Nationale de l'Aviation Civile  
7, avenue Edouard-Belin  
C.S. 54005  
31055 TOULOUSE cedex 4  
FRANCE  
[www.enac.fr](http://www.enac.fr)



# Calcul parallèle. Techniques de base

*Transparents du cours*

**Tome 1 – Notions fondamentales**

**R. Coudarcher**

*2<sup>ème</sup> édition, refondue*

ENAC Toulouse

2017

## Pour contacter l'auteur

Rémi COUDARCHER

Tél. : 05.62.17.41.54

Email : [Remi.Coudarcher@enac.fr](mailto:Remi.Coudarcher@enac.fr)

Bureau : C123

Département : SINA/INF/SAR

---

**Tirage :** octobre 2017

**Révision :** CPARBA-CTR-200-150211-171010-01

---

# CALCUL PARALLELE. TECHNIQUES DE BASE

Tome 1 - Notions fondamentales

2<sup>ème</sup> édition, refondue

octobre 17

Rémi COUDARCHER



CPA/RBA/CTR-2001/3021/171010-01

Notes :

# PLAN

Notes :

# PLAN

Durée du cours : 10 H (tome 1 : 6 H, tome 2 : 4 H, 2 H de travail personnel individuel)

• Organisation de l'enseignement	<u>4</u>
• Introduction aux architectures et applications parallèles	<u>7</u>
• Architectures des machines parallèles	<u>36</u>
• Réseaux d'interconnexion	<u>85</u>
• Programmation d'un cluster sous MPI	<u>97</u>
• Conclusion	<u>170</u>

CPA/RBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

3

Notes :

# ORGANISATION DE L'ENSEIGNEMENT

CPA/RBA/CTR-2001/32T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

4

Notes :



# Répartition

## Organisation de l'enseignement

- |                                                                                                    |         |
|----------------------------------------------------------------------------------------------------|---------|
| • Cours magistral                                                                                  | 5 x 2 H |
| • Travail personnel                                                                                | 1 x 2 H |
| – Acquisition de connaissances en autonomie à partir des supports de cours et de travaux pratiques |         |
| – Préparation des séances de travaux pratiques                                                     |         |
| • Travaux pratiques encadrés                                                                       | 5 x 2 H |
| • Examen                                                                                           | 0 H     |
| – Travaux pratiques notés                                                                          |         |

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

5

Notes :

# Pré-requis

- Connaissances nécessaires
  - Architectures des calculateurs
  - Chaîne de construction d'un programme
    - Compilation, édition des liens, chargement en mémoire
  - Algorithmique séquentielle
    - Structures de données et structures de contrôle
  - Architectures et protocoles des réseaux informatiques
    - Bases
  - Programmation concurrente et notion de synchronisation
    - Multi-processus (éventuellement multi-threadée)
  - Langage C/C++
    - Pour les travaux pratiques
- Connaissances souhaitables
  - Systèmes d'exploitation multi-tâches
  - Bibliothèque PThreads
  - Rudiments de programmation sous Microsoft Windows et UNIX (ou Linux)
    - Pour les travaux pratiques
  - Analyse numérique
    - Pour les travaux pratiques

Notes :

# INTRODUCTION

Notes :

# INTRODUCTION

Durée du cours : 1 H 30

- Le parallélisme 9
- Typologie parallèle 20
- Intérêts et usages 26
- Exemples 33

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

8

Notes :

# Le parallélisme (1/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Les programmes informatiques traditionnels sont écrits
    - Pour fonctionner sur une seule machine
    - Ayant une seule unité de calcul
      - Un seul processeur (CPU – *Central Processing Unit*)
    - En découpant un problème en une série finie d'instructions
      - Algorithme
    - Les instructions étant exécutées les unes après les autres
      - Séquentiellement
- Une seule instruction est exécutée à un instant donné, c'est un programme séquentiel

CPA/RBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

9

Notes :

# Le parallélisme (2/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Le calcul parallèle est
    - l'exploitation simultanée de multiples ressources de calcul
      - Une seule machine mono-processeur multi-cœurs
      - Une seule machine multi-processeurs (multi-cœurs ou non)
      - De multiples machines mono-processeur (ou multi-processeurs) reliées par un réseau de communication
      - Une (ou plusieurs machines reliées par un réseau) présentant de multiples unités de traitement spécialisées (processeurs graphiques, co-processeurs mathématiques, co-processeurs de traitement du son, processeurs de traitement du signal (DSP – *Digital Signal Processor*), ASIC (*Application-Specific Integrated Circuit*),...)
- ... pour résoudre un problème plus efficacement

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

10

Notes :

# Le parallélisme (3/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Pour exploiter plusieurs CPU
  - Le problème à résoudre est décomposé en plusieurs parties
    - Chaque partie représentant un sous-ensemble du problème
    - Chaque partie est elle-même décomposée en une série d'instructions (algorithme séquentiel)
  - Ces sous-problèmes seront traités simultanément par des CPU différents
    - Ils doivent donc pouvoir être résolus concurremment
    - Ils doivent présenter un haut degré d'indépendance les uns par rapport aux autres

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

11

Notes :

# Le parallélisme (4/11)

[Introduction](#) | [Architectures](#) | [Interconnexion](#) | [Programmation MPI](#) | [Conclusion](#)

- Exemple en traitement d'images (1/3)

- Fonction de seuillage

- Image en niveaux de gris
    - Parcours exhaustif des pixels de l'image
    - Transformation en une image binaire de même dimension
    - Un pixel de l'image résultante
      - = 1 si le niveau de gris du pixel de l'image initial > au seuil préalablement fixé
      - = 0 sinon
  - (filtre « passe-haut »)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

12

Notes :



# Le parallélisme (5/11)

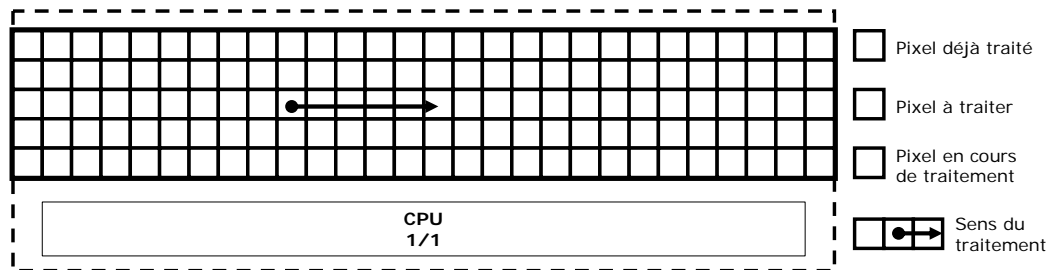
Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Exemple en traitement d'images (2/3)

– Temps de traitement séquentiel :  $T_s = D \times I$

- $D$  = dimension de l'image en nombre de pixels
- $I$  = temps de traitement d'un pixel

⇒  $T_s$  est la somme des temps de traitement de chacun des pixels



CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

13

Notes :

# Le parallélisme (6/11)

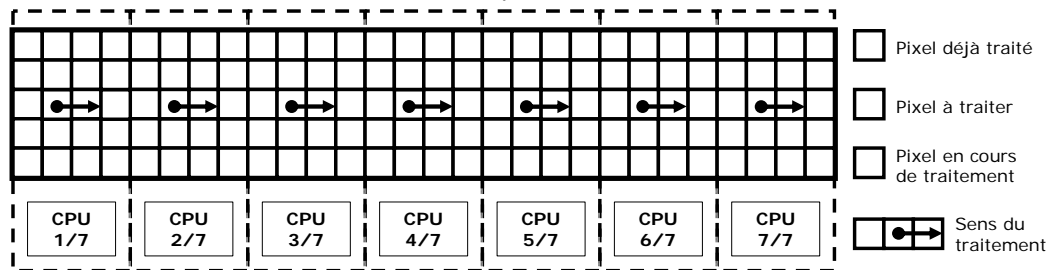
[Introduction](#) | [Architectures](#) | [Interconnexion](#) | [Programmation MPI](#) | [Conclusion](#)

- Exemple en traitement d'images (3/3)

- Temps de traitement parallèle :  $T_p = T_s / N$

- $N$  = nombre de CPU mis en œuvre

⇒ Chaque traitement étant indépendant, régulier et systématique,  $T_s$  est réduit d'un facteur  $N$ , chaque CPU pouvant faire ses traitements en parallèle avec les autres de manière totalement autonome et indépendante



CPA/RBA/CTR-2001-132T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

14

Notes :

# Le parallélisme (7/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Ajouter plus de processeurs/nœuds signifie-t-il toujours plus de puissance ?
  - C'est rarement le cas
    - L'évolutivité d'un système parallèle est la somme de plusieurs facteurs interdépendants
  - Le parallélisme ajoute un problème de coordination des multiples flots d'instructions exécutés simultanément
    - Alors que la séquence imposée des instructions dans chaque flot spécifie déjà
      - Les opérations arithmétiques
      - Les accès mémoire en lecture et en écriture
      - L'adresse de la prochaine instruction à exécuter

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

15

Notes :

# Le parallélisme (8/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Le but recherché par une parallélisation dépend de l'application
  - Les systèmes transactionnels recherchent généralement à augmenter leur bande passante (mise en avant de l'évolutivité potentielle)
  - Les systèmes d'aide à la décision s'intéressent davantage aux temps de réponse (mise en avant de la vitesse des traitements)
- Le système parallèle idéal
  - Accélération linéaire (*linear speedup*)
    - N fois plus de ressources signifie N fois moins de temps mis pour résoudre un même problème
  - Mise à l'échelle linéaire (*linear scaleup*)
    - N fois plus de ressources permet de résoudre des problèmes N fois plus larges dans le même temps

CPA/BBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

16

Notes :

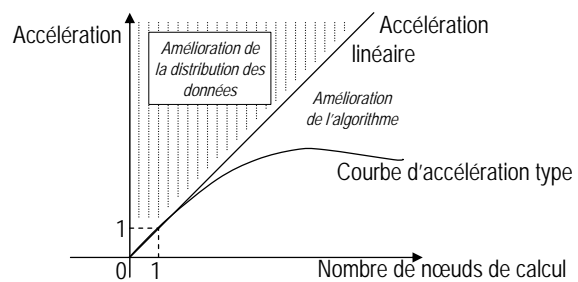
# Le parallélisme (9/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Limitation de l'accélération

- Pour une parallélisation
  - Démarrage/arrêt : initialisation des activités parallèles, synchronisation des résultats
  - Interférences : conflits lors de l'accès à des données partagées
  - Dispersion : le temps d'exécution global est celui de la tâche la plus lente
- Tous ces problèmes augmentent avec l'augmentation du nombre de processeurs/nœuds

- Accélération maximale théorique :  $N$  (*linear speedup*)



CPA/RBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

17

Notes :

# Le parallélisme (10/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Lors de la conception d'algorithmes parallèles
  - Il est nécessaire de repenser la manière d'appréhender le flot d'instructions d'un algorithme lors du passage de la version séquentielle à la version parallèle
    - Identifier les activités qui peuvent être exécutées en parallèle et celles qui ne le peuvent pas
    - Voir les programmes comme des ensembles de tâches interdépendantes
  - La meilleure solution parallèle diffère généralement de manière très importante de la version séquentielle
    - Et des approches différentes dans la façon de concevoir un algorithme parallèle peut conduire à des solutions aussi très différentes

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

18

Notes :

# Le parallélisme (11/11)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Les systèmes parallèles ne sont pas des architectures récentes
  - Elles existent depuis les tous débuts de l'informatique
    - ENIAC (Electronic Numerical Integrator and Computer, Penn's Moore School of Electrical Engineering, 1943)
    - 704 (IBM, 1955)
    - Atlas Computer (première utilisation des mécanismes d'interruptions pour simuler l'exécution concurrente de plusieurs programmes, Kilburn & Howarth, 1961)
    - ILLIAC IV (machine vectorielle, université de l'Illinois, 1965)
    - Le calcul parallèle est passé des architectures à mémoire partagée aux architectures réparties dès la fin des années 1970
  - Aujourd'hui, ils se retrouvent partout
    - Les systèmes temps réel embarqués sont des systèmes parallèles par nature
    - Chaque nouvel ordinateur personnel vendu à travers le monde est au moins un ordinateur à processeur double-cœur

CPA/RBA/CTR-2001-13021-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

19

Notes :

# Typologie parallèle (1/6)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Concurrency

- Deux actions, ou plus, sont en cours d'exécution au même moment sur un seul CPU (mono-threadé)
- Implique ordonnancement et synchronisation des actions
  - Une exécution concurrente est l'entrelacement d'instructions atomiques en provenance de plusieurs flots d'instructions séquentiels (processus)  
(un processus est l'exécution séquentielle d'instructions atomiques)
- Responsabilité du système d'exploitation  
(partage des ressources pour une meilleure utilisation du CPU, de la mémoire,...)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

20

Notes :



# Typologie parallèle (2/6)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Parallélisme (1/2)

- Deux actions, ou plus, s'exécutent simultanément sur des CPU différents
- Demande
  - Une architecture matérielle intrinsèquement parallèle
    - Nécessite autant de CPU que d'actions en cours
    - Coprocesseurs spécialisés
    - Multiplication des ressources (périphériques,...)
  - Des mécanismes de synchronisation entre tâches réparties,
  - Des moyens de communication entre CPU

→ Le modèle de programmation est fonction de l'architecture matérielle choisie et des moyens de communication mis en œuvre

CPRBBACTR-2001-192T-1:171010-01



Notes :

# Typologie parallèle (3/6)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Parallélisme (2/2)

- “A parallel computer is a set of processors that are able to work cooperatively to solve a computational problem”

Foster I., *Designing and Building Parallel Programs*, Addison-Wesley Inc., 1995

CPA/RBA/CTR-2001-02T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

22

Notes :

# Typologie parallèle (4/6)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Répartition (1/2)

- Implique un réseau d'interconnexion
    - Communication par passage de messages (ou par appels de procédures distantes encapsulés)
  - Nombreuses machines autonomes
    - Processeur(s) local(aux), mémoire locale,...
    - Interaction entre elles afin d'atteindre un objectif commun
  - Pannes et dysfonctionnements doivent être pris en charge
    - Qu'ils concernent les machines ou le réseau d'interconnexion
- Défis
- Hétérogénéité, interconnexion extérieure, sécurité, évolutivité, tolérance aux pannes, concurrence, transparence, absence d'horloge globale et commune,...

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

23

Notes :

# Typologie parallèle (5/6)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Répartition (2/2)

- “A distributed system is a collection of independent computers that appear to the users of the system as a single computer”

Tanenbaum A., *Distributed Operating System*, Prentice-Hall, 1994

- “A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable”

Lamport L., *DEC Corp.*, mail interne du 28 mai 1987

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

24

Notes :

# Typologie parallèle (6/6)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Concurrence, parallélisme et répartition
  - Deux threads démarrés par une application
    - Sont définis comme des activités concurrentes par le programme
    - Sont susceptibles d'être exécutés en parallèle
    - Peuvent être répartis sur des machines différentes

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

25

Notes :

# Intérêts et usages (1/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Distribuer les ressources de calcul
  - Evolutivité *scalability*  
(augmenter les performances avec l'augmentation du nombre de ressources disponibles)
  - Exploitation *scavenging*  
(s'adapter aux ressources disponibles en en tirant profit au mieux)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

26

Notes :

# Intérêts et usages (2/7)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Gagner du temps

- Réduction des temps de traitement

(plus de ressources affectées à la même tâche)

- Temps réel pour les systèmes embarqués (ex. : pilotage, conduite et guidage automatiques, robotique)
    - Requêtes dans des bases de données volumineuses (ex. : web, télécommunications)

- Augmentation des performances

(faire, plus rapidement)

- Augmentation du débit

(faire plus, dans le même temps)

CPA/RBA/CTR-2001-192T1:171010-01



Notes :

# Intérêts et usages (3/7)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Résoudre des problèmes plus complexes
  - Puissance de calcul
  - Capacité mémoire
    - Simulations (ex. : nucléaire, prédictions météo, calcul de structures)
  - Espace de stockage
    - Bases de données (ex. : domaine bio-médical (projet MediGrid))
    - Expérimentations avec de larges volumes de résultats (ex. : collisionneurs du CERN)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

28

Notes :



# Intérêts et usages (4/7)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Protéger les ressources
  - Tolérance aux pannes
    - Haute disponibilité (*High-availability*)
    - Si un nœud du réseau de traitement tombe en panne, d'autres nœuds prennent le relai (qu'ils soient déjà actifs ou non)
  - Redondance des traitements
    - Plusieurs nœuds de calcul peuvent être dédiés à la même tâche
    - Vérification des calculs pour des systèmes critiques

CPA/RBA/CTR-2001-13021-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

29

Notes :

# Intérêts et usages (5/7)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Améliorer les performances des processeurs
  - Les contraintes physiques se font de plus en plus sentir
    - Fréquences d'horloges
    - Vitesse de propagation des signaux
    - Finesse de la gravure des circuits
    - Températures de fonctionnement
    - Consommation électrique...
  - Parallélisme au cœur même des processeurs
    - Architectures pipelines
    - Architectures superscalaires
    - Hyper-threading
    - VLIW
    - Multi-cœurs

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

30

Notes :

# Intérêts et usages (6/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Economiser de l'argent (1/2)
  - Même puissance de calcul avec un système moins coûteux
    - *Cluster* (grappe) : mise en œuvre de nombreuses machines peu coûteuses (PC, machines d'anciennes générations) pour atteindre les performances d'un gros système
    - Attention toutefois à la difficulté de mise en œuvre et à l'énergie nécessaire au fonctionnement de l'ensemble (électricité, refroidissement,...)
  - Compromis prix/performance  
(obtenir les meilleures performances pour un prix donné)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

31

Notes :

# Intérêts et usages (7/7)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Economiser de l'argent (2/2)

- Contrairement à une idée communément admise, exploiter plusieurs machines en réseau (ou une seule machine multi-processeurs) ne signifie pas forcément une consommation plus importante d'énergie électrique

- La consommation électrique est plus importante (car plus de machines)
    - Mais pendant un temps plus court (calcul parallèle)

- Economie d'énergie

- Exemple

- *Extrait de* Parallélisme avec la beta 1 de Visual Studio 2010 : mise à l'échelle d'une application C++, *Boucard B., Microsoft France, MSDN, Août 2009.*

- Calcul séquentiel : 54 W pendant 35 s → 0.52 W/h

- Calcul parallèle (processeur dual-core) : 75 W pendant 13 s → 0.27 W/h

CPA/BBA/CTR-2001-192T-1-171010-01



Notes :

# Exemples (1/3)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Le calcul parallèle/réparti est utilisé pour (1/2)
  - Modéliser de nombreux problèmes scientifiques et d'ingénierie
    - Simulations dans les domaines du nucléaire, de la physique des particules, de la matière condensée, des hautes pressions, des turbulences,...
    - Prédiction du temps et du climat, évolutions environnementales, sismologie...
    - Génomique, biophysique, biotechnologies...
    - Chimie moléculaire
    - Aéronautique et aérospatiale
    - Conception de circuits électronique à haute intégration, microélectronique
    - Mathématiques

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

33

Notes :

# Exemples (2/3)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

- Le calcul parallèle/réparti est utilisé pour (2/2)
  - Le développement de nombreuses applications commerciales
    - Stockage et recherche dans des bases de données volumineuses (opérateurs de télécommunications,...)
    - Exploitation pétrolière, pneumatiques,...
    - Imagerie médicale, développement de médicaments,...
    - Rendus graphiques, réalité virtuelle, jeux vidéos,...
    - Modélisations financière et économique
    - Environnements de travail collaboratifs, services web, applications réseau,...

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

34

Notes :

# Exemples (3/3)

[Introduction](#) | Architectures | Interconnexion | Programmation MPI | Conclusion

➤ *Extrait de Putting multicore processing in context: Part One, Brian T., EETimes, Janvier 2006.*

- Pourquoi des processeurs multi-cœurs dans l'électronique grand public ?
  - Evolutivité
  - Division du travail
  - Spécialisation du travail
  - Répondre à la demande en termes d'augmentation des performances
  - Réduire la consommation électrique
    - Les processeurs présentent une consommation électrique plus importante avec l'augmentation de leur fréquence d'horloge
    - Les processeurs dissipent plus de chaleur avec l'augmentation de leur fréquence d'horloge  
(ce qui nécessite une énergie plus importante pour les refroidir)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

35

Notes :

# ARCHITECTURES DES MACHINES PARALLELES

CPA/RBA/CTR-2001/32T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

36

Notes :



# ARCHITECTURES

Durée du cours : 1 H 30

• Où les trouve-t-on ?	<u>38</u>
• Taxinomie et modèles	<u>45</u>
• Organisation de la mémoire	<u>64</u>
• Architectures standards	<u>75</u>
• Portabilité	<u>83</u>

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

37

Notes :

# Où les trouve-t-on ? (1/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Aujourd'hui encore, il est difficile de définir un modèle unique, unifié, de machine parallèle
  - Grande diversité des architectures matérielles parallèles
  - Grande diversité des modèles de programmation parallèle

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

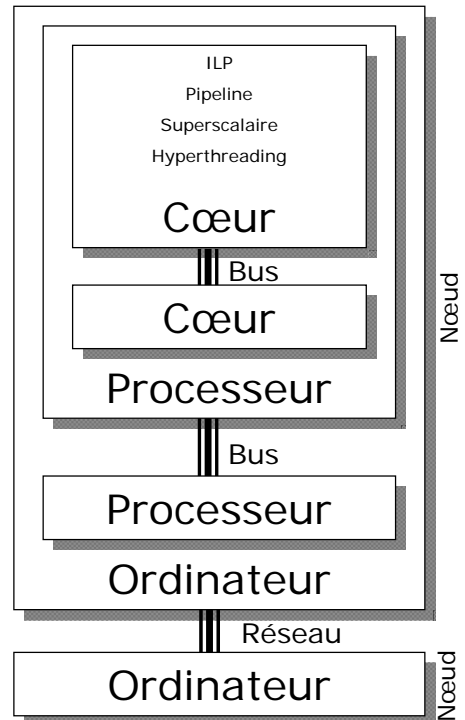
38

Notes :

# Où les trouve-t-on ? (2/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Au cœur d'un processeur
  - *Instruction-level parallelism* (ILP)
    - *Very Large Instruction Word* (VLIW)
  - Pipeline
  - Architecture superscalaire
  - Multi-cœurs
- Par l'usage de plusieurs processeurs au sein d'une même machine
  - *Multi-processing*
  - Certains des processeurs peuvent être spécialisés
    - Son, vidéo, arithmétique flottante,...
- Par l'usage de plusieurs machines
  - Réseau d'ordinateurs (ex. : Beowulf)



CPA/BAC/CTR-2001-192/11-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

39

Notes :

# Où les trouve-t-on ? (3/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Au cœur d'un processeur (1/5)
  - *Instruction-level parallelism* (ILP)
    - *Very Large Instruction Word* (VLIW)
      - 1980 (Fisher et coll.)
      - Le compilateur est responsable de l'identification des instructions pouvant être exécutées en parallèle
      - La complexité de la tâche est transférée du matériel au logiciel
      - Les instructions sont codées sur 256 bits ou plus
      - Ex. : *Explicitly Parallel Instruction Set Computing* – EPIC d'Intel pour l'architecture Itanium (IA-64)
  - Pipeline
    - Les sous-étapes de réalisation de plusieurs instructions traitées en séquentiel sont exécutées en parallèle
    - Problèmes
      - Interdépendances entre les données
      - Gestion des branchements

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

40

Notes :

# Où les trouve-t-on ? (4/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Au cœur d'un processeur (2/5)
  - Architecture superscalaire
    - Plusieurs pipelines travaillent en parallèle
    - Plusieurs instructions peuvent être exécutées en parallèle
      - Appartenant à un sous-ensemble réduit du jeu d'instructions du processeur
      - Seul un nombre très limité de ces instructions peut être exécuté en parallèle car dépendant du nombre de pipelines et d'ALU qui sont disponibles dans l'architecture
    - Chacun des pipelines est spécialisé dans l'exécution d'un sous-ensemble particulier du jeu d'instructions du processeur
      - Par conséquent, seules des instructions qui n'ont pas besoin d'utiliser le même pipeline peuvent être exécutées en parallèle
  - Exécution « *out-of-order* » et prédiction de branchement
    - Techniques pour améliorer l'utilisation des périphériques d'entrées/sorties, des pipelines et des architectures superscalaires

CPA/BAC/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

41

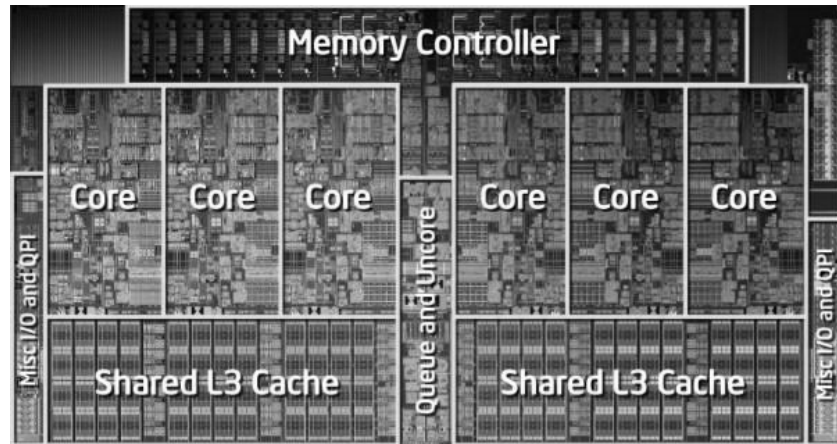
Notes :

# Où les trouve-t-on ? (5/7)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Au cœur d'un processeur (3/5)
  - Architectures multi-cœurs (1/3)

- Chaque cœur est perçu comme un processeur à part entière par le logiciel



Processeur Intel Xeon 5600 Series

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

42

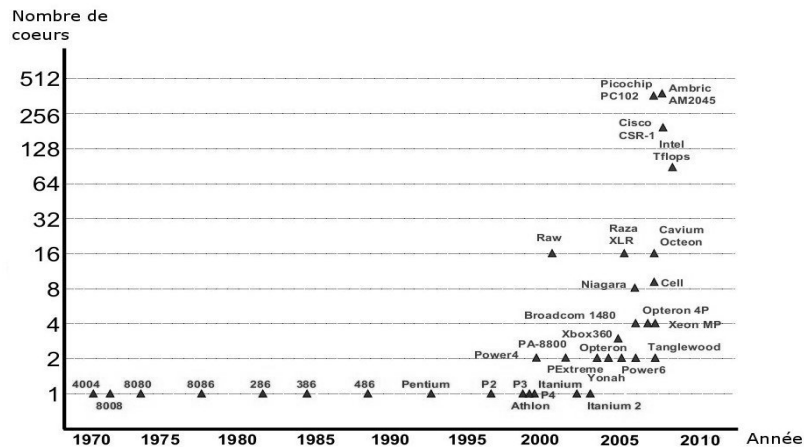
Notes :

# Où les trouve-t-on ? (6/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Au cœur d'un processeur (4/5)
  - Architectures multi-cœurs (2/3)



D'après Exploiting Coarse-Grained Task, Data, and Pipeline Parallelism in Stream Programs,  
C. V. Suresh Babu, Veltech, Multitech Engineering College, p. 2, 2 décembre 2013



Notes :

# Où les trouve-t-on ? (7/7)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion



- Au cœur d'un processeur (5/5)
  - Architectures multi-cœurs (3/3)
    - Comparaison du nombre de « cœurs » disponibles sur une sélection de cartes Nvidia

	GTX TITAN	GTX 780	GTX 770	<b>GTX 760</b>	GTX 660 Ti
<b>Mfg. Process</b>	28nm	28nm	28nm	28nm	28nm
<b>Base Clock</b>	836MHz	863MHz	1046MHz	980MHz	915MHz
<b>Boost Clock</b>	876MHz	900MHz	1085MHz	1033MHz	980MHz
<b>Memory</b>	6GB GDDR5	3GB GDDR5	2GB / 4GB GDDR5	2GB / 4GB GDDR5	2GB GDDR5
<b>Memory Speed</b>	6008MHz (QDR)	6008MHz (QDR)	7010MHz (QDR)	6008MHz (QDR)	6008MHz (QDR)
<b>Memory Bus</b>	384-bit	384-bit	256-bit	256-bit	192-bit
<b>Memory Bandwidth</b>	288.5 GB/s	288.5 GB/s	224.3 GB/s	192.3 GB/s	144 GB/s
<b>CUDA Cores</b>	2688	2304	1536	1152	1344
<b>ROPs</b>	48	48	32	32	24
<b>Texture Units</b>	224	192	128	96	112
<b>TDP</b>	250W	250W	230W	170W	150W
<b>MSRP</b>	\$999	\$649	\$399	\$249	EOL / \$229

CPRBACTR-2001-132T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

44

Notes :



# Modèles (1/19)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion



## • Architecture de von Neumann (1/4)

(rappels)

- 1945
  - Mathématicien américain d'origine hongroise
  - Projet EDVAC
- Description de l'organisation d'un ordinateur électronique générique
  - Depuis, pratiquement tous les ordinateurs suivent ce modèle
- Diffère de celui des machines précédentes par le fait qu'il autorise leur programmation
  - Ses fonctionnalités ne sont pas « pré-câblées »
  - Les premiers ordinateurs électroniques avaient des programmes préétablis
  - Programmes construits à partir d'un jeu d'instructions et enregistrer dans une mémoire (architecture dite « de von Neumann »)

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

45

Notes :

# Modèles (2/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

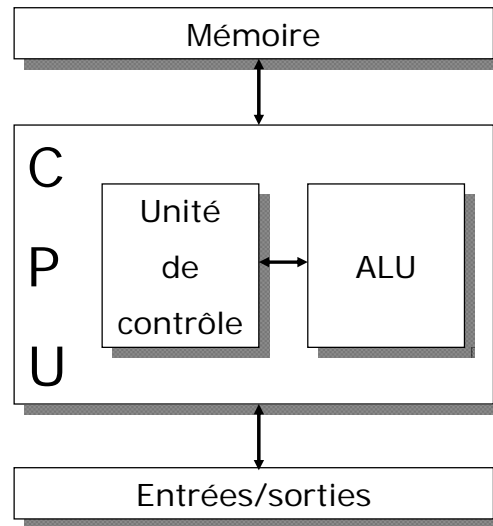


## • Architecture de von Neumann (2/4)

(rappels)

### – 4 composants principaux

- Unité de contrôle
- Unité arithmétique et logique (ALU – *Arithmetic and Logic Unit*)
- Mémoire
- Entrées/sorties



CPRBBACTR-2001-13021-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

46

Notes :

# Modèles (3/19)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion



- Architecture de von Neumann (3/4)

(rappels)

- Unité de contrôle

- Récupère les instructions et les données stockées en mémoire
- Décode les instructions chargées dans les registres du CPU
- Séquence les opérations pour accomplir la tâche demandée (programmée)

- Unité arithmétique et logique (ALU)

- Opérations arithmétiques
- Opérations logiques
- Comparaisons

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

47

Notes :

# Modèles (4/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



## • Architecture de von Neumann (4/4)

(rappels)

### – Mémoire

- Accès aléatoire, en écriture et en lecture (RAM – *Random Access Memory*)
- Mémorise programme (suite ordonnée d'instructions) et données
- Instruction = donnée particulière qui ordonne au CPU de réaliser une opération
- Donnée = information n'ayant aucun sens particulier pour le CPU

### – Entrées/sorties

- Interface avec les périphériques (clavier, écran, stockage de masse, son, capteurs,...)
- Donne accès au monde extérieur (à l'ordinateur)
- Interface avec l'opérateur

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

48

Notes :

# Modèles (5/19)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Taxinomie de Flynn (1/9)
  - 1966
    - L'une des classifications les plus utilisées
  - 4 catégories d'architectures
  - Classification des architectures selon 2 dimensions
    - Quantité d'instructions exécutées en parallèle
    - Quantité de données traitées en parallèle
  - Propriétés de ces dimensions
    - Indépendantes l'une de l'autre
    - 2 états possibles : unique (*single*) ou multiple (*multiple*)

CPA/RBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

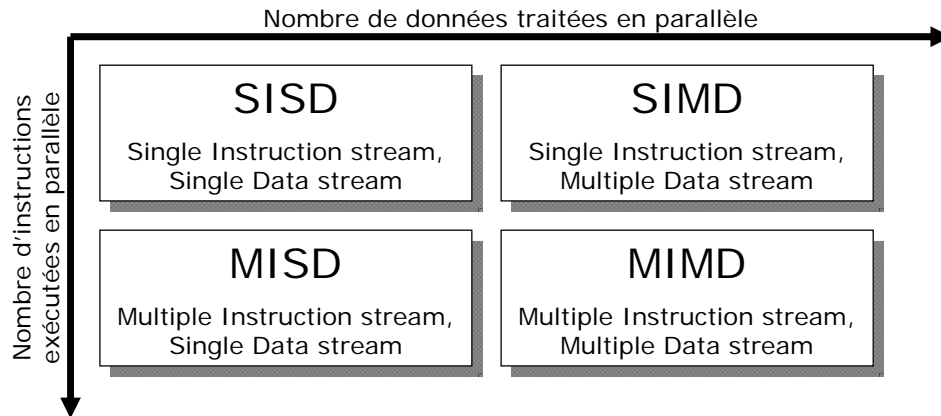
49

Notes :

# Modèles (6/19)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Taxinomie de Flynn (2/9)



CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

50

Notes :

# Modèles (7/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Taxinomie de Flynn (3/9)

(1/2)

### SISD

Single Instruction stream,  
Single Data stream

- Forme d'ordinateur la plus répandue
  - Architecture non parallèle
  - Mono-processeur, simple-cœur, mono-threadée
  - Exécution déterministe d'un programme
- 1 seul flot d'instructions
  - Une seule instruction est traitée par le CPU à chaque cycle d'horloge
- 1 seul flot de données
  - Une seule donnée est manipulée comme entrée à chaque cycle d'horloge
  - Le traitement d'une donnée nécessite une (ou plusieurs) instructions, mais une instruction en cours d'exécution ne peut servir à traiter plusieurs données différentes

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

51

Notes :

# Modèles (8/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Taxinomie de Flynn (4/9)

(2/2)

## SISD

Single Instruction stream,  
Single Data stream

- Exemples

PC, stations de travail,...

UNIVAC1



PDP1



CRAY1

CPA/RBA/CTR-2001-132T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

52

Notes :



# Modèles (9/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Taxinomie de Flynn (5/9)

(1/2)

### SIMD

Single Instruction stream,  
Multiple Data stream

#### – Architecture parallèle

- Ordinateur dit « vectoriel »
- Exécution déterministe d'un programme
- Adaptée aux traitements réguliers (calcul matriciel, traitement d'images, graphisme,...)

#### – 1 seul flot d'instructions

- Plusieurs unités d'exécution fonctionnent en parallèle, mais en exécutant à un instant donné strictement la même instruction (exécution dite « synchrone »)
- Deux mises en œuvre types : matrice de processeurs, pipeline vectoriel (fondamentalement, ce type de machine possède un très grand nombre d'unités de calcul de très faible puissance disposées en matrice)

#### – Plusieurs flots de données

- Chaque unité d'exécution peut traiter un flot de données différent

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

53

Notes :

# Modèles (10/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Taxinomie de Flynn (6/9)

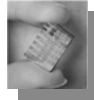
(2/2)

### SIMD

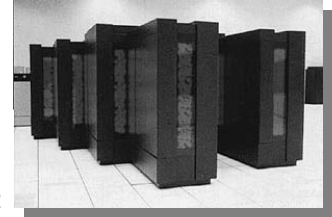
Single Instruction stream,  
Multiple Data stream

### – Exemples

- GPU *Computing*
  - Cartes graphiques (NVIDIA Tesla,...)
  - Processeurs CELL d'IBM/Sony/Toshiba (Sony PlayStation 3)
- Matrices de processeurs
  - ILLIAC IV,
  - MasPar MP-1 & MP-2,
- Pipelines vectoriels
  - IBM 9000,
  - NEC SX-2, CRAY Y-MP &
- Intégré au jeu d'instructions natives de nombreux processeurs
  - IBM/Motorola PowerPC : AltiVec
  - Intel : SSE – *Streaming SIMD Extensions*



Connection Machine CM-2



X-MP



CPA/RBA/CTR-2001-132T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

54

Notes :

# Modèles (11/19)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

## • Taxinomie de Flynn (7/9)

(1/2)

### MIMD

Multiple Instruction stream,  
Multiple Data stream

#### – Architecture parallèle

- La plus générale et souple
- Exécution déterministe ou non
- Cette architecture est capable de se comporter comme une architecture SIMD
- Les unités d'exécution de cette architecture suivent bien souvent le modèle SIMD

#### – Plusieurs flots d'instructions

- Indépendants, 1 par processeur ou cœur
- Synchrones ou non (asynchrone)

#### – Plusieurs flots de données

- Chaque processeur peut traiter une donnée différente de celle traitée par les autres processeurs au même instant

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

55

Notes :

# Modèles (12/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Taxinomie de Flynn (8/9)

(2/2)

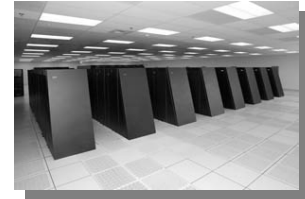
### MIMD

Multiple Instruction stream,  
Multiple Data stream

#### – Exemples

- PC à base de processeurs multi-cœurs
- PC bi-processeurs (ou plus)
- Grappes (*clusters*) de PC ou de stations de travail
- Calcul distribué sur un LAN ou un WAN
- GRID *Computing*
- Ordinateurs parallèles de type SMP (*Symmetric MultiProcessing*)
- IBM Power5
- HP AlphaServer
- Intel IA32
- AMD Opteron

IBM BG/L



CRAY XT3



CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

56

Notes :

# Modèles (13/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



## • Taxinomie de Flynn (9/9)

### MISD

Multiple Instruction stream,  
Single Data stream

#### – Architecture parallèle

- Usages possibles :
  - filtres multiples en fréquence sur un même signal
  - Décryptage d'un code selon plusieurs méthodes/algorithmes simultanément,...

#### – Plusieurs flots d'instructions

#### – 1 seul flot de données

- Une même donnée est envoyée pour traitement vers plusieurs unités d'exécution fonctionnant en parallèle
- Chaque unité peut exécuter des instructions différentes
- Les traitements sont indépendants

#### – Exemples

- Calculateur parallèle expérimental de l'université Carnegie-Mellon (1971) : C.mmp computer

CPA/BBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

57

Notes :

# Modèles (14/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Architecture de type pipeline (1/3)

- Une tâche est scindée en plusieurs étapes
- Chaque étape est exécutée par des unités de traitement différentes et spécialisées appelées « étages » (*stages*), le flot de données traversant l'ensemble de ces étages

→ Augmente le débit de sortie

(la cadence à laquelle les résultats sont obtenus)

- Le temps de traitement pour l'ensemble de l'algorithme n'est pas pour autant réduit car cette organisation ne réduit pas le temps de traitement nécessaire à chaque étape

→ Induit un délai avant l'obtention du premier résultat

- Délai nécessaire au chargement complet du pipeline

Premier ordinateur à utiliser cette technique : IBM Stretch (1958)

CPA/BBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

58

Notes :

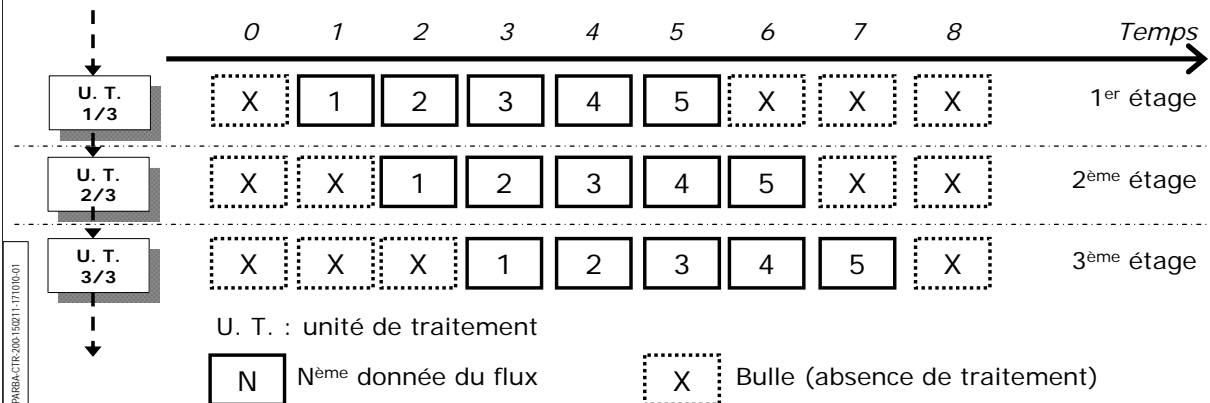
# Modèles (15/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Architecture de type pipeline (2/3)

– N'augmente pas la vitesse du traitement complet d'une donnée

- Augmentation de la cadence de sortie/d'obtention des résultats
- Nécessite un flot d'information (inutile sur une donnée isolée)

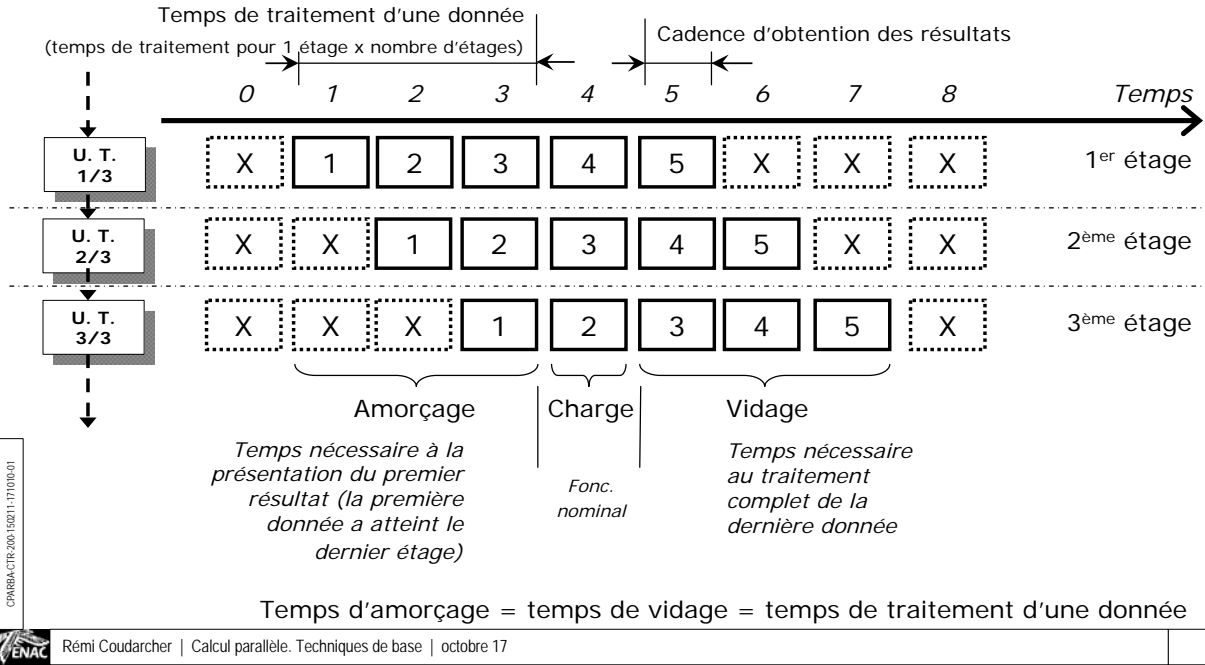


Notes :

# Modèles (16/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Architecture de type pipeline (3/3)



Notes :



# Modèles (17/19)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion



- Architecture systolique (1/3)

(Kung H. T., Carnegie Mellon University, 1980)

- Architecture « flot de données » (*data flow*)

- Conçue pour équilibrer des volumes de calcul important avec des opérations d'entrées/sorties demandantes en termes de bande passante

- Unité de traitement

- Appelée « cellule systolique » (*systolic cell*)
    - Indépendante et autonome vis-à-vis des autres unités
    - Quelques registres rapides
    - ALU
    - Mécanisme d'entrées/sorties simple

- Synchronisation

- Toutes les cellules sont cadencées par la même horloge (changement synchrone d'état)
    - Une synchronisation explicite règle le flux de données (pipeline)
    - Dans chaque intervalle de temps, une cellule exécute invariablement la même séquence (courte) d'instructions

CPA/RBA/CTR-2001-13021-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

61

Notes :

# Modèles (18/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Architecture systolique (2/3)

- “Imagine n simple processors arranged in a row or an array and connected in such a manner that each processor may exchange information with only its neighbors to the right and left. The processors at either end of the row are used for input and output. Such a machine constitutes the simplest example of a systolic array.”

➤ *Extrait de Specification and Verification of Systolic Arrays, Bayoumi M. & Ling N., World Scientific Publishing Co. Pte. Ltd., Singapour, 1999.*

- “Systolic Arrays are regular arrays of simple finite state machines, where each finite state machine in the array is identical...A systolic algorithm relies on data from different directions arriving at cells in the array at regular intervals and being combined.”

➤ *Extrait de VLSI Circuits and Systems in Silicon, Brown A., McGraw-Hill Book Company, Londres, 1991.*

CPANBACTR-2001-02T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

62

Notes :

# Modèles (19/19)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Architecture systolique (3/3)

- Avantages

- Architecture évolutive
      - Il est aisé d'augmenter le nombre de cellules dans le réseau
    - Capable d'émuler une architecture SIMD (opérations vectorielles)
    - Capable d'émuler une architecture MIMD (parallélisme non homogène)
    - Autorise un très grand débit en sortie (bande passante de sortie élevée)

- Inconvénients

- Architecture compliquée à mettre en œuvre
      - Tant au niveau matériel que logiciel
    - Coûteuse
    - Horloge commune
      - Sa fréquence limite la taille du réseau
    - Absence de tolérance aux pannes
      - Le dysfonctionnement d'une seule cellule entraîne le dysfonctionnement de l'ensemble de la machine

CPRABACTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

63

Notes :

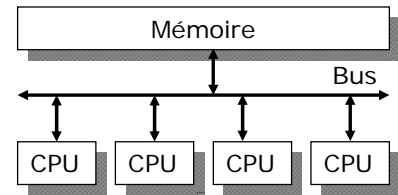
# Mémoire (1/11)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

## • Mémoire partagée (1/3)

SM - *Shared-Memory*

(Modèle mémoire des calculateurs SMP)



### – Mémoire commune à tous les CPU d'une même machine

- Espace d'adressage commun et unique pour tous les CPU
- Temps d'accès constant à la mémoire (UMA – *Uniform Memory Access*)
  - Quelque soit le CPU et la localisation de la donnée
- Si la cohérence des caches est préservée on parle de CC-UMA (*Cache Coherent-UMA*)
  - Importante pour maintenir la cohérence des données en mémoire
  - Gérée au niveau du matériel

### – Accès concurrent et direct

- Bus
- A tout instant, chaque CPU peut traiter une donnée localisée à une adresse différente de celles traitées par les autres CPU
- Tout changement en mémoire réalisé par un CPU est vu par les autres CPU

CPA/BBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

64

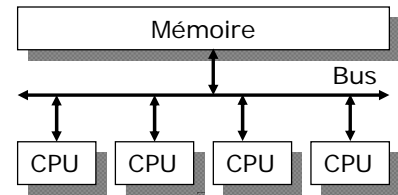
Notes :

# Mémoire (2/11)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Mémoire partagée (2/3)

SM - *Shared-Memory*



- Avantages

- Facilité de programmation
      - Pas de distinction dans les méthodes d'accès aux données en fonction de leur localisation
      - Pas de mise en œuvre de bibliothèques de communication particulières
    - Partage de données direct, rapide et uniforme entre processeurs et entre processus

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

65

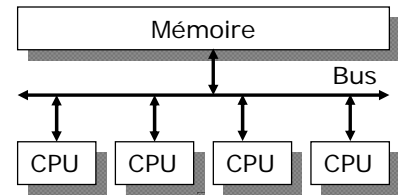
Notes :

# Mémoire (3/11)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Mémoire partagée (3/3)

SM - *Shared-Memory*



- Inconvénients

- Accès concurrent à la mémoire
      - Représente un goulet d'étranglement
      - Evolutivité limitée (le nombre de processeurs ne peut être aussi grand qu'on veut)
    - En raison de l'utilisation de caches pour réduire le trafic sur le bus d'accès à la mémoire, la notion de localisation des données continue d'être importante
    - Il faut faire évoluer la quantité de mémoire avec le nombre de processeurs
      - Attention à la capacité d'adressage totale des processeurs utilisés
    - Le programmeur est responsable de la synchronisation des accès concurrents aux données (utilisation de verrous, sémaphores,...)
    - Architectures chères à produire si le nombre de processeurs est important

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

66

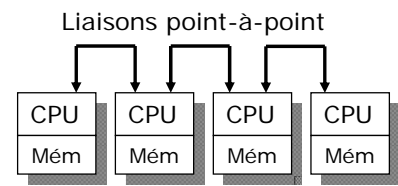
Notes :

# Mémoire (4/11)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

## • Mémoire distribuée (1/3)

DM - *Distributed-Memory*



### – Chaque CPU a sa propre mémoire locale

- Pas d'espace d'adressage global : une adresse mémoire utilisée sur un CPU n'a aucun sens (ne se réfère pas à la même information) sur une autre CPU
- Tout changement dans la mémoire d'un des CPU n'a aucun effet sur le contenu des mémoires des autres CPU (pas de notion de cohérence de cache)
- Temps d'accès à une information non constant (NUMA – *Non-Uniform Memory Access*)

### – L'accès mémoire entre CPU se fait par le réseau d'interconnexion

- Liaisons point-à-point ou bus
- Réseaux classiques de type Ethernet ou liaisons spécialisées/propriétaires
- Attention à la vitesse de communication : l'accès mémoire distant est incomparablement plus lent qu'un accès à la mémoire locale

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

67

Notes :

# Mémoire (5/11)

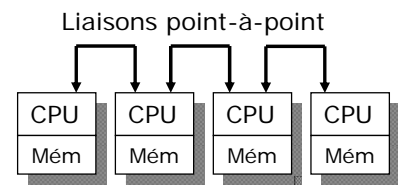
Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

## • Mémoire distribuée (2/3)

DM - *Distributed-Memory*

### – Avantages

- Pas de goulet d'étranglement au niveau de l'accès mémoire
- La quantité de mémoire évolue avec le nombre de CPU
- Chaque processeur a un accès rapide et exclusif à une certaine quantité de mémoire (pas de maintien de la cohérence de cache, pas de conflit d'accès sur bus partagé,...)
- Facilité de mise en œuvre de ce type d'architecture mémoire (ex. : *clusters*)



CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

68

Notes :

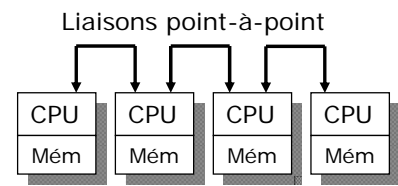


# Mémoire (6/11)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

## • Mémoire distribuée (3/3)

DM - *Distributed-Memory*



### – Inconvénients

- Lorsqu'une donnée n'est pas présente dans la mémoire locale d'un CPU, il est de la responsabilité du programmeur de définir dans quelle mémoire elle se trouve, quand accéder à cette information et comment le faire
  - Synchronisation explicite, maîtrise des moyens de communication, connaissance de la topologie du réseau d'interconnexion (efficacité des algorithmes de distribution de données)
- Changements nécessaires dans l'organisation de certaines SdD applicatives
- Temps d'accès non constants (NUMA)

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

69

Notes :

# Mémoire (7/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Mémoire partagée virtuelle
  - Mémoire physiquement distribuée mais gérée de manière à ce que l'utilisateur ait l'impression d'une mémoire partagée
    - Avantages de programmation de la mémoire partagée, et avantages de mise en œuvre physique de la mémoire distribuée
    - Inconvénient des surcoûts liés au maintien de la cohérence de cache et aux temps d'accès non constants (mais masqués) : CC-NUMA – *Cache Coherent NUMA*
    - Difficulté de gérer la localité des données (problème d'efficacité)
- Mémoire hybride
  - Mémoire à la fois distribuée et partagée
    - Mise en réseau d'architectures SMP

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

70

Notes :

# Mémoire (8/11)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Cohérence de cache (1/4)

- Architectures mono-processeur

- La mémoire cache est utilisée pour accélérer le fonctionnement d'un processeur
    - La mémoire cache est réalisée à partir d'une petite quantité de mémoire très rapide (mais coûteuse)
    - En utilisant cette approche à deux couches d'accès mémoire, si la mémoire cache est écrite, alors mémoire cache et mémoire principale ne contiennent plus la même information (elles sont considérées comme étant dans un état « incohérent »)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

71

Notes :

# Mémoire (9/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Cohérence de cache (2/4)

### – Architectures multi-cœurs (1/2)

- Le schéma de cohérence de cache doit répondre simultanément à des lectures et des écritures sur le cache individuel d'un cœur, tout en maintenant la cohérence avec la mémoire partagée et toutes les mémoires cache des autres cœurs
- Chaque cœur doit voir la mémoire comme une entité globale commune
- Utiliser de la mémoire cache sur un système multi-cœurs tend à séquentialiser l'exécution des processus
- Mises en œuvre principales
  - « Invalidation »
    - » Si un cœur écrit une donnée, toutes les copies sont marquées comme invalides
    - » La détection de cette situation est faite par observation de l'activité sur le bus d'interconnexion entre les cœurs
    - » Une tentative d'accès à une donnée invalide génère une faute de cache et son rechargement
  - « Mise-à-jour »
    - » Si un cœur écrit une donnée, toutes les copies sont écrites également
    - » La cohérence est garantie, aucune faute de cache ne peut se produire
    - » Cette procédure génère plus de trafic sur le bus d'interconnexion que la précédente : elle est généralement moins efficace

CPA/BBA/CTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

72

Notes :

# Mémoire (10/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Cohérence de cache (3/4)

- Architectures multi-cœurs (2/2)

- Des algorithmes plus avancés de gestion des caches existent, mais dérivent des principes précédents
    - Les mécanismes d'invalidation peuvent utiliser des états intermédiaires du cache permettant une gestion plus fine : « modifié », « exclusif », « partagé », « invalide »
    - Les architectures à haut degré de parallélisme ont des politiques différentes selon les niveaux de cache

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

73

Notes :

# Mémoire (11/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Cohérence de cache (4/4)

- Systèmes répartis

(Dans les cas d'usage d'une mémoire virtuellement partagée et de duplication des données)

- Les délais de communication et l'usage de protocoles non-déterministes rendent la cohérence de cache distribuée très difficile à mettre en œuvre en termes de conception et d'exploitation

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

74

Notes :

# Architectures std. (1/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- SMP – *Symmetric MultiProcessing*

- Ensemble de processeurs équivalents dans un seul système
  - Modèle MIMD-SM
  - Pas de processeurs spécialisés (E/S, mathématique, son, graphique, pour exécuter un OS ou les applications,...)
- Les processeurs partagent l'accès à la mémoire centrale par l'intermédiaire d'un bus unique
- Avantages
  - Les performances augmentent avec le nombre de processeurs
  - Modèle de programmation par mémoire partagée classique
  - Redondance intégrée possible
- Inconvénients
  - Augmentation du nombre de processeurs (*scale-up*) limitée par l'existence d'un bus unique (goulet d'étranglement) -> Beaucoup de processeurs peuvent vouloir accéder à la même mémoire en même temps quand il y a beaucoup de processeurs

CPRABACTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

75

Notes :

# Architectures std. (2/8)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Machines massivement parallèles
  - Nombreux processeurs interconnectés
    - Un processeur ou un nœud hôte est responsable du chargement des programmes et des données sur le réseau de processeurs
    - Réseau d'interconnexion haute-performance
    - Importance de la topologie du réseau d'interconnexion dans la conception des algorithmes et leur efficacité (bus, anneau (*ring*), grille 2D (*mesh*), tore 3D (*torus*), hypercube, arbre (*tree*),...)
    - Modèle MIMD
  - Avantages
    - Solution très efficace
    - Réseaux rapides et à temps de latence faible
  - Inconvénients
    - Solution coûteuse
    - Consommation énergétique à surveiller
    - Problème du refroidissement du système (*cooling*)

CPA/BBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

76

Notes :



# Architectures std. (3/8)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- *Clusters* (grappes) (1/6)
  - Ensemble de stations de travail, de PC autonomes ou de calculateurs « rackables » interconnectés par un réseau local (LAN)
    - Modèle MPMD-DM
    - Outils logiciels largement utilisés :
      - MPI
      - PVM
      - OpenMP
      - SLURM (Simple Linux Utility for Resource Management)
      - Moab (répartiteur de charge)
      - ...

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

77

Notes :

# Architectures std. (4/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- *Clusters* (grappes) (2/6)
  - Organisation matérielle générale (1/4)
    - Plusieurs nœuds de calcul
      - Sont les plus nombreux dans un cluster
      - Exécutent les jobs des utilisateurs
    - 1 ou plusieurs nœuds d'administration
      - Ordonnancement des jobs, distribution du système de fichiers, surveillance des ressources,...
    - 1 frontal (éventuellement plusieurs)
      - Point d'accès au cluster pour les utilisateurs
      - Identification et authentification des utilisateurs
      - Pare-feu
      - Accès aux comptes utilisateurs
      - Génération des exécutables (« compilation »)
      - Demande et contrôle d'exécution des jobs

CPA/RBA/CTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

78

Notes :

# Architectures std. (5/8)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- *Clusters* (grappes) (3/6)

- Organisation matérielle générale (2/4)

- Un nœud est une machine dotée de 1 ou plusieurs processeurs, avec ou sans mémoire de masse, sans clavier, ni souris, ni écran
      - L'administration se fait à distance par le réseau d'interconnexion
    - Chaque nœud peut être, ou non, du même type (même configuration,...) que les autres nœuds du cluster
      - Notamment, la génération des exécutables doit tenir compte de l'hétérogénéité des nœuds (compilation croisée)
      - L'ordonnanceur de jobs est aussi amené à tenir compte de ces différences pour placer les bons exécutables sur les bons nœuds (en termes de code machine généré mais aussi de fonctionnalités présentes dans chaque exécutable et demandant donc des ressources spécifiques à un nœud)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

79

Notes :

# Architectures std. (6/8)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- *Clusters* (grappes) (4/6)
  - Organisation matérielle générale (3/4)
    - Des nœuds peuvent être éventuellement regroupés dans des baies s'ils sont « rackables »



Baies du cluster TLCC  
(Tri-laboratory Linux Capacity Cluster),  
Lawrence Livermore National Laboratory,  
USA

CPA/RBA/CTR-2001-132T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

80

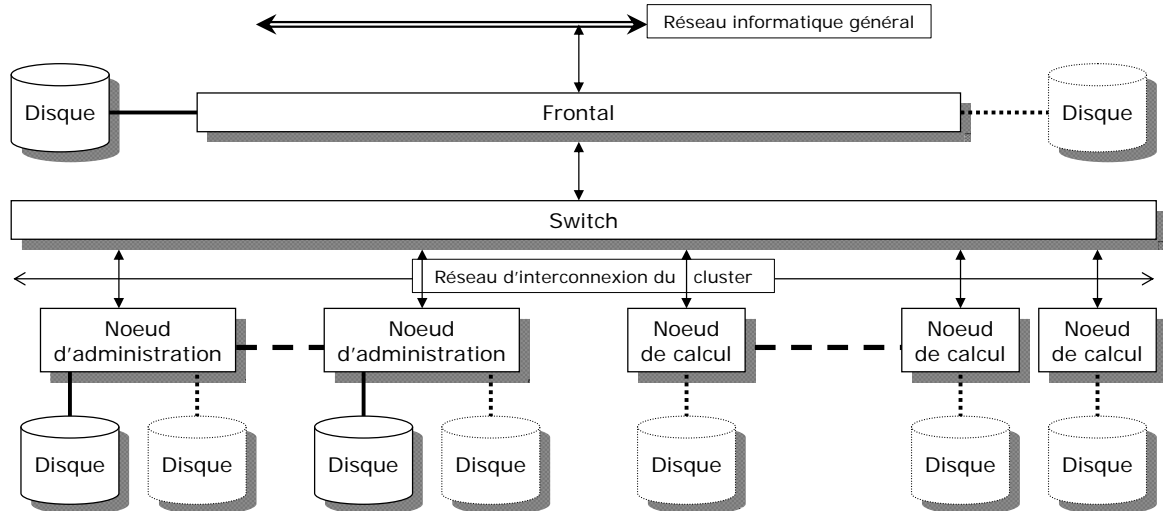
Notes :

# Architectures std. (7/8)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- *Clusters* (grappes) (5/6)

- Organisation matérielle générale (4/4)



CPA/BAC/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

81

Notes :

# Architectures std. (8/8)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- *Clusters* (grappes) (6/6)

- Avantages

- Solution à moindre coût pour du calcul parallèle (tant au niveau matériel que logiciel)
    - Haute disponibilité (*high-availability*)
    - Compatibilité de niveau application sur chaque machine (mono-processeur)

- Inconvénients

- Programmation totalement explicite
    - Administration du réseau de machines peu aisée
    - Consommation énergétique à surveiller

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

82

Notes :

# Portabilité (1/2)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Les architectures matérielles parallèles sont très diverses
  - Topologies, protocoles et technologies des réseaux
  - Processeurs
  - ...
- Hétérogénéité des
  - Systèmes d'exploitation
  - Langages de développement
  - ...

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

83

Notes :

# Portabilité (2/2)

Introduction | [Architectures](#) | Interconnexion | Programmation MPI | Conclusion

- Des efforts ont été fait pour standardiser
  - API (MPI, API de la Multicore Association, POSIX Threads,...)
  - Environnements d'exécution (PVM,...)  
(*Runtimes*)
  - Intergiciels (CORBA,...)  
(*Middleware*)
  - Langages (Java, HPF, OpenMP, OpenCL,...)

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

84

Notes :



# RESEAUX D'INTERCONNEXION

CPA/RBA/CTR-2001/32T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

85

Notes :

# INTERCONNEXION

Durée du cours : 1 H 00

- Communications et architectures des réseaux d'interconnexion
  - Vitesses de communication
  - Graphes d'interconnexion
  - Bus
  - Déterminisme
  - Modèles de communication élémentaire
  - Algorithmes de distribution
  - Notion de routage

CPA/RBA/CTR-2001/3021/171010/01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

86

Notes :

# Réseaux (1/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Pour permettre la communication entre nœuds
  - L'usage d'une mémoire partagée n'est pas viable pour un nombre important de nœuds
  - Une approche plus réaliste est que
    - Chaque nœud possède sa propre mémoire locale, et
    - Les données soient échangées via un réseau d'interconnexion par passage de messages
- Le réseau d'interconnexion joue un rôle central
  - Il détermine notamment pour une bonne part les performances d'ensemble d'un système parallèle ou réparti

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

87

Notes :

# Réseaux (2/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Peu de réseaux d'interconnexion mettent en œuvre  $N^2$  liaisons pour connecter  $N$  nœuds
  - Ils intègrent plutôt des commutateurs (*switches*) afin de router les messages entre nœuds distants
  - Les commutateurs peuvent bloquer ou re-router certains messages lorsque plusieurs de ces derniers nécessitent l'accès à la même liaison au même moment
    - Situation dite de compétition pour la bande passante (*competition for the bandwidth*)

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

88

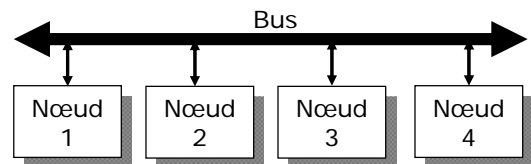
Notes :

# Réseaux (3/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Bus

- Une seule connexion par nœud
- Distance identique entre deux nœuds du réseau, quels qu'ils soient
- Pas de topologie
- Conflits potentiels d'accès au bus (*bus contention*)
  - Seul un nœud peut émettre à un instant donné



CPA/RBACTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

89

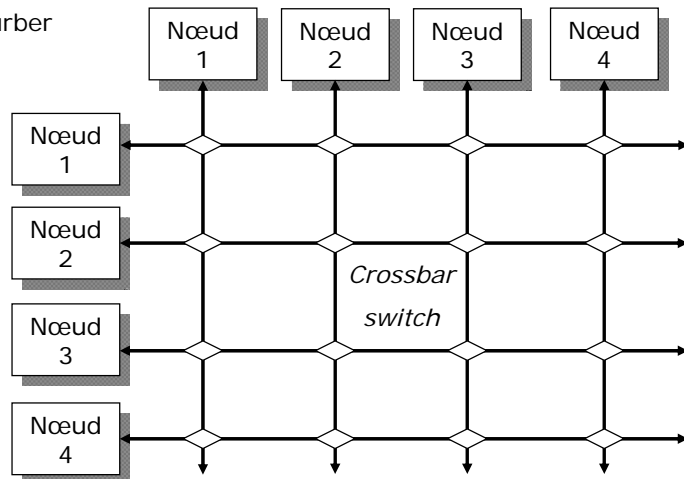
Notes :

# Réseaux (4/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- *Crossbar switch*

- Nombre arbitraire de combinaisons
- Echanges de données sans conflit d'accès (*collision-free*)
  - N'importe quel couple de nœuds peut communiquer sans perturber les autres échanges
- Très coûteux
- Croissance quadratique du réseau avec le nombre de nœuds



CPRBACTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

90

Notes :

# Réseaux (5/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

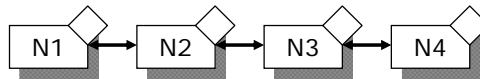
## • Point-à-point (1/3)

– La topologie du réseau d'interconnexion doit être prise en compte dans la conception des algorithmes

- Des algorithmes parallèles efficaces reposent sur une connaissance précise de la topologie du réseau d'interconnexion utilisé
- La résolution d'un même problème peut être faite par des algorithmes parallèles complètement différents en fonction de la topologie du réseau utilisé
- Une métrique caractérisant le média de communication est alors absolument nécessaire

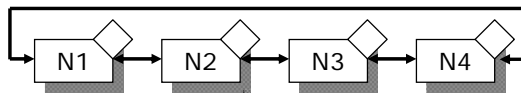
### – Exemples

- Barreau (*bar*)



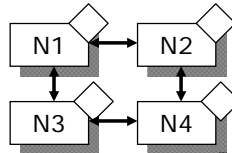
Distance maximale :  $N-1$

- Anneau (*ring*)



Distance maximale :  $N/2$

- Grille 2D (*mesh*)  
(compromis entre coût et connectivité)



CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

91

Notes :

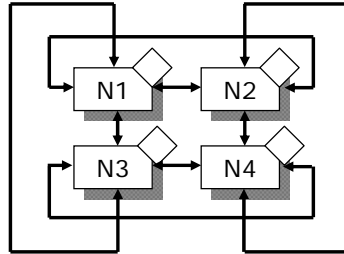
# Réseaux (6/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

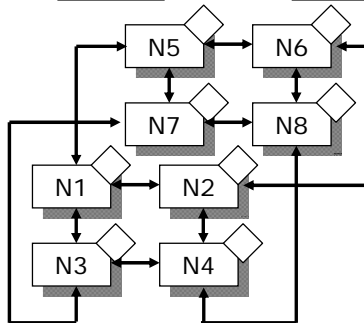
- Point-à-point (2/3)

- Exemples

- Tore 3D (*torus*)  
(compromis entre coût et connectivité)



- Grille cubique



CPA/RBA/CTR-2001-132T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

92

Notes :



# Réseaux (7/10)

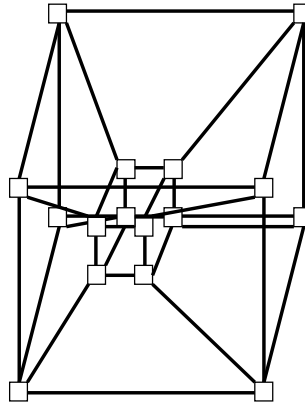
Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Point-à-point (3/3)

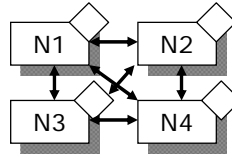
- Exemples

- Hypercube

Un cube dans un cube en reliant  
les sommets  
-> selon la dimension



- Graphe entièrement connecté  
(connectivité maximale,  
réseau le plus puissant,  
mais très coûteux à cause de  
ses  $N(N-1)/2$  connexions)



CPA/RBA/CTR-2001-13021-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

93

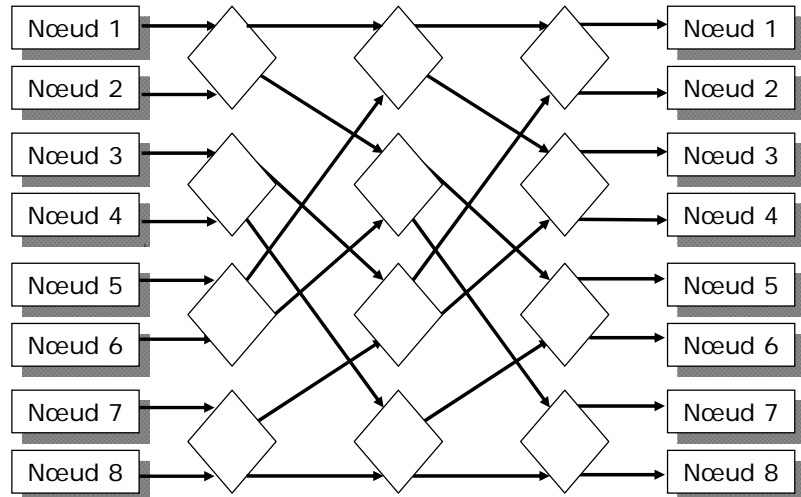
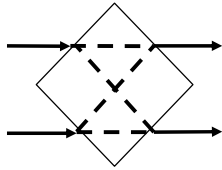
Notes :

# Réseaux (8/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Delta

- Toutes les combinaisons possibles ne peuvent être opérationnelles simultanément
- Coût limité



CPA/RBA/CTR-2001-132T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

94

Notes :

# Réseaux (9/10)

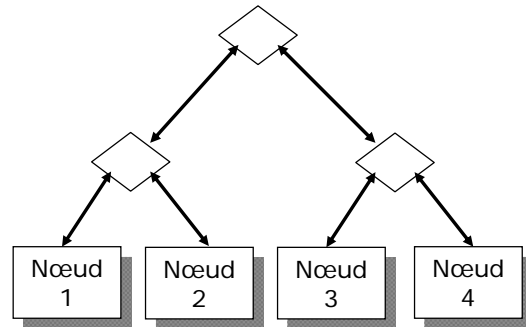
Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- *Fat tree*

- Arbre binaire

- Coût des communications à croissance logarithmique en fonction de l'éloignement des nœuds

- Chaque feuille de l'arbre est un nœud de calcul unique



CPA/RBA/CTR-2001-1302/1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

95

Notes :

# Réseaux (10/10)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Métrique

- Destinée à comparer et à évaluer les réseaux entre eux
- Connectivité (*network connectivity*)
  - Représente le nombre minimum de nœuds ou de liaisons qui doivent s'arrêter de fonctionner pour créer une partition du réseau en 2 (ou plus) réseaux disjoints
  - Mesure la résilience d'un réseau
    - Propriété d'un réseau à continuer de fonctionner malgré la perte de nœuds ou de liaisons
  - Plus la connectivité d'un réseau est grande plus sa capacité de tolérance aux pannes sera élevée
- Diamètre (*network diameter*)
  - Distance la plus grande existante entre deux nœuds
  - Représente le nombre maximum de liaisons qu'il est nécessaire de traverser pour émettre un message vers n'importe lequel des nœuds du réseau en empruntant le chemin le plus court
- Étroitesse (*narrowness, bisection width*)
  - Mesure la congestion du réseau  
(Si on considère 2 partitions P1 et P2 du réseau ; P2 ayant moins de nœuds (NP2) que P1 (NP1) ; soit N le nombre de liaisons entre P1 et P2 ; la valeur maximale de  $NP2/N$  (pour toute partition du réseau) est appelée « étroitesse »)

CPA/BAC/CTB-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

96

Notes :

# PROGRAMMATION D'UN CLUSTER SOUS MPI

Notes :

# CLUSTER SOUS MPI

Durée du cours : 2 H (+ 2 H de travail personnel individuel)

• Modèles de programmation	<u>99</u>
• Programmation par envoi de messages	<u>100</u>
• MPI	<u>108</u>
• Sources de parallélisme	<u>117</u>
• Communications	<u>124</u>
• Synchronisations	<u>162</u>

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

98

Notes :

# Modèles de programmation

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Des modèles de programmation parallèle existent
  - Ils permettent une abstraction des architectures matérielles et de l'organisation de la mémoire
    - Ils ne sont pas spécifiques à un type particulier de machine ou d'organisation mémoire, et
    - Ils peuvent être mis en œuvre sur tout type de matériel sous-jacent
- Modèles les plus communs
  - Mémoire partagée (*shared memory*)
  - *Threads*
  - Passage de messages (*message passing*)
  - Graphes flot de données (GFD) (*dataflow graphs*)

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

99

Notes :

# Par messages (1/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Modèles de programmation parallèle par passage de messages

(*Message-passing*)

- Modèle de programmation parallèle associé à l'usage d'une architecture à mémoire distribuée
  - Standard de fait aujourd'hui
- Approche pour
  - Communiquer des données
  - Partager des données
  - Synchroniser des tâches
- Dans ce paradigme, la ressource à traiter est directement gérée par une seule et unique tâche
  - Lorsqu'une autre tâche veut interroger ou manipuler la ressource, elle envoie un message à la tâche qui la gère

CPA/BBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

100

Notes :



# Par messages (2/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Formes d'implémentations

- Du point de vue du programmeur

- Une bibliothèque de fonctions

- Nombreuses réalisations différentes et incompatibles entre elles depuis 1980

- Du point de vue technique

- Bibliothèque de fonctions utilisateur

- Fonctionnalités intégrées au processeur + API (Transputers,...)

→ Dans tous les cas, mise en œuvre explicite par le programmeur

CPA/RBA/CTR-2001-192T-1:171010-01



Notes :

# Par messages (3/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Caractéristiques

- Les tâches d'une application travaillent en mémoire locale
- Plusieurs tâches peuvent, éventuellement, être en exécution sur le même nœud de calcul
- Echange de données entre tâches par l'intermédiaire de communications
  - Envoi et réceptions de messages selon un protocole applicatif
  - Coopération des tâches entre elles pour établir la communication (un envoi de message suppose normalement la réception de celui-ci par un destinataire)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

102

Notes :

# Par messages (4/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Les processus s'exécutent indépendamment les uns des autres
  - En dehors des périodes de communication
- Les points de communication deviennent des points de synchronisation
  - Synchronisation qui peut être une synchronisation lâche (*loosely synchronization*)
    - Notion de « *loosely coupled sequential processes* »
  - Evite toute hypothèse sur la vitesse relative des processus entre eux
    - Aucun des processus séquentiels ne doit être affecté dans l'exactitude de ses résultats par des variations de temps d'exécution, cela en supposant une mise en œuvre correcte des points de synchronisation

CPA/BBA/CTR-2001-13021-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

103

Notes :

# Par messages (5/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Ce paradigme souffre de problèmes similaires à ceux des sémaphores binaires
  - Inversion de priorité
    - Se produit quand une tâche travaille à partir d'un message de basse priorité et ignore un message de priorité plus élevée dans sa boîte de réception (ou un message provenant indirectement d'une tâche de haute priorité)
  - Interblocages de protocole
    - Survient lorsque deux tâches, ou plus, s'attendent mutuellement pour émettre des messages de réponse

CPRB/BACTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

104

Notes :

# Par messages (6/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Le problème du consensus dans les systèmes asynchrones de passage de messages (1/2)
  - Chaque processus s'exécute à sa propre fréquence
  - Aucune garantie n'est donnée quant au délai de délivrance des messages
  - Si un processus s'arrête (définitivement, suite à un dysfonctionnement), obtenir un consensus dans ce type de système est formellement impossible
    - Les processus ne sont pas en mesure de dire si un autre processus est définitivement arrêté ou juste temporairement isolé du monde extérieur (*i.e.* ses messages sont retardés)
    - En effet, si les processus attendent le processus qui ne répond pas, ils pourraient attendre éternellement si il s'est vraiment arrêté suite à un dysfonctionnement ; Et si au contraire ils prennent une décision, ils pourraient être confrontés au fait que le processus qui pose problème n'ait été que retardé et ait pris entre temps une décision différente

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

105

Notes :

# Par messages (7/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Le problème du consensus dans les systèmes asynchrones de passage de messages (2/2)
  - Il n'existe aucun protocole déterministe pour résoudre le problème d'absence de consensus possible dans les systèmes asynchrones de passage de messages
  - Aucun protocole n'est en mesure de tolérer la panne ne serait-ce que d'un seul et unique processus  
(excepté par l'usage d'un protocole dit « aléatoire » (*randomized protocol*))
  - De leur côté, les systèmes synchrones sont capables de tolérer jusqu'à un tiers de l'ensemble des nœuds du réseau d'interconnexion qui poseraient problèmes

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

106

Notes :

# Par messages (8/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- 4 implémentations majeures
  - PVM
  - MPI
  - LAM
  - OpenMPI

CPA/RBA/CTR-2001-132T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

107

Notes :

# MPI (1/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- MPI – *Message-Passing Interface*
  - Première publication en 1992 par le *MPI Forum* (1/4)
    - MPI-1 : première version (dite « *Part 1* ») en mai 1994
      - Révisée jusqu'en mai 2008 (v. 1.3)
      - *Bindings* pour C, C++ et FORTRAN 90

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

108

Notes :



# MPI (2/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- MPI – *Message-Passing Interface*

- Première publication en 1992 par le *MPI Forum* (2/4)

- MPI-2 : deuxième version (dite « *Part 2* ») en 1996

- Révisée jusqu'en septembre 2009 (v. 2.2)

- Introduit des notions dynamiques dans la gestion des tâches de calcul
          - » Démarrage d'une nouvelle tâche après le lancement de l'application, etc.

- Opérations collectives autorisées entre communicateurs

- Communications unidirectionnelles

- » Mémoire partagée (opérations put/get), etc.

- Gestion des entrées/sorties en parallèle

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

109

Notes :

# MPI (3/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- MPI – *Message-Passing Interface*

- Première publication en 1992 par le *MPI Forum* (3/4)

- MPI-3 : troisième version (dite « *Part 3* ») en septembre 2012

- Plus de 430 routines

- Opérations collectives bloquantes et non-bloquantes

- Amélioration de la prise en charge des communications unidirectionnelles

- Possibilités d'accès à certaines structures de données internes à MPI

- » Pour implémenter des outils de mesure de performance, de débogage,...

- C++ n'est plus supporté nativement

- *Bindings* pour FORTRAN 2003 et 2008

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

110

Notes :

# MPI (4/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- MPI – *Message-Passing Interface*

- Première publication en 1992 par le *MPI Forum* (4/4)

- Principaux contributeurs

- ANL – Argonne National Laboratory, Cray Research, IBM, Intel, Linda, Meiko, Michigan State University, Mississippi State University, NSF – National Science Foundation, PVM – Parallel Virtual Machine, P4 – Portable Programs for Parallel Processors, University of Edinburgh

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

111

Notes :

# MPI (5/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Objectif
  - Création d'une API standardisée pour les architectures parallèles à passage de messages
- Spécifie l'interface et les fonctionnalités attendues
  - Les spécifications sont librement accessibles
  - Ne spécifie pas la manière de réaliser l'implémentation pour une plate-forme particulière
  - Définit la syntaxe et la sémantique pour la portabilité du code source

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

112

Notes :

# MPI (6/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Standard de fait
  - Aussi bien dans le monde industriel qu'académique
- La grande majorité des machines parallèles proposent une implémentation
  - Par contre rares sont celles qui proposent une implémentation complète de la partie 2 des spécifications (MPI-2)
  - Des implémentations « génériques » pour grappes sont largement disponibles
  - Il n'est pas difficile d'implémenter un sous-ensemble de la norme pour une machine prototype

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

113

Notes :

# MPI (7/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Disponible aussi pour architectures à mémoire partagée
  - Utilise alors la mémoire (par recopie) plutôt que le réseau pour des raisons d'efficacité

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

114

Notes :

# MPI (8/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- L'API fournit
  - Des communications point-à-point
  - Des communications collectives
  - L'interfaçage avec les langages C et Fortran
- Caractéristiques
  - Types de données élaborés
  - Gestion de groupes de processus
  - Description de topologies virtuelles
  - Gestion d'erreurs et débogage
  - Interface pour l'analyse de performances

CPRBBA-CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

115

Notes :

# MPI (9/9)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Implémentation MPICH
  - ANL - *Argonne National Laboratory*
  - Implémentation de référence
  - Gratuite et disponible sur de nombreuses plateformes
  - Supports de plusieurs protocoles de communication intra et inter machines
  - Supports pour le débogage

CPA/RBA/CTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

116

Notes :



# Sources (1/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Sources de parallélisme

(Les plus communes)

- Parallélisme de données (*data parallelism*)
- Parallélisme de tâches (*task parallelism*)
- Parallélisme de flux (type pipeline)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

117

Notes :

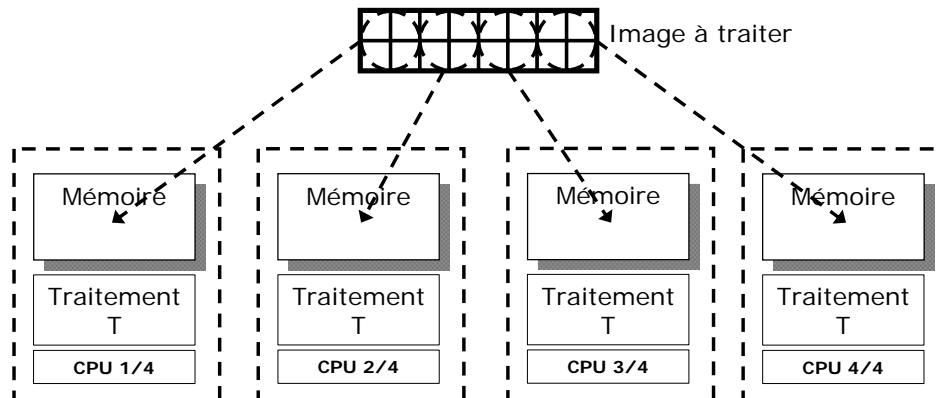
# Sources (2/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Parallélisme de données (1/2)

SPMD – *Single Program Multiple Data*

- Un seul traitement à réaliser
- Mais une partition des données à traiter peut être établie
- Traitement identique en parallèle des partitions



CPA/BBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

118

Notes :

# Sources (3/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

## • Parallélisme de données (2/2)

SPMD – *Single Program Multiple Data*

- A tout instant, toutes les tâches exécutent le même programme
  - Elles peuvent exécuter la même instruction du programme ou des instructions différentes (le « compteur de programme » peut être différent)
- Evolutivité de la parallélisation
  - Bonne
  - Quasi-linéaire avec le nombre de nœuds de calcul mis en œuvre
  - Dépend de la partition possible des données
- Usages types
  - Traitement de structures de données régulières
  - Traitements d'images de bas niveau

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

119

Notes :

# Sources (4/7)

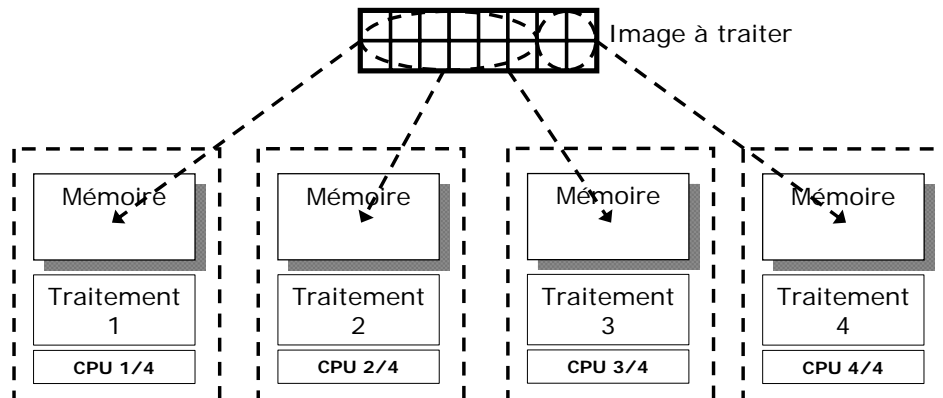
Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Parallélisme de tâches (1/2)

MPMD – *Multiple Program Multiple Data*

- Plusieurs tâches différentes et indépendantes exécutées en parallèle

- Sur les mêmes données ou sur des données différentes



CPA/BBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

120

Notes :

# Sources (5/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Parallélisme de tâches (2/2)

MPMD – *Multiple Program Multiple Data*

- Evolutivité de la parallélisation

- Dépend de la nature et du nombre de tâches à mettre en œuvre

- Usages types

- Traitements concurrents sur les mêmes données
    - Chaînes de traitements dont certains peuvent être parallélisés car indépendants
    - Suivi multi-cibles (ex. : avionique militaire)
    - Traitements d'images de haut-niveau
    - Processeurs dédiés ou spécifiques parmi un ensemble de processeurs généraux

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

121

Notes :

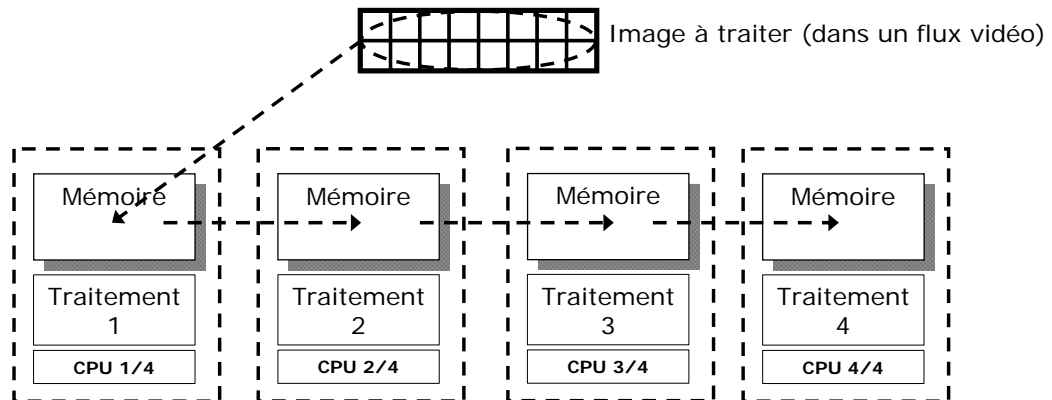
# Sources (6/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Parallélisme de flux (type pipeline) (1/2)

- Séquence de tâches répétée sur un flux de données

- Un traitement sur une donnée particulière peut commencer lorsque le même traitement sur la donnée précédente est terminé (même si l'ensemble des traitements à appliquer à la donnée particulière n'est pas achevé)



CPA/BAC/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

122

Notes :

# Sources (7/7)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Parallélisme de flux (type pipeline) (2/2)
  - Attention au temps d'amorçage
    - Le pipeline ne réduit pas le temps de traitement global d'une donnée
    - Donc, pour certaines tâches à réaliser, le temps d'amorçage pourrait être vu comme un temps de retard à l'obtention d'un premier résultat si le rôle du pipeline a été mal compris et mis en œuvre (ex. : traitement d'un flux vidéo pour la conduite automatique de véhicules)
  - Evolutivité de la parallélisation
    - Dépendante du découpage possible d'un traitement sur une donnée en sous-traitements autonomes
  - Usage type
    - Parallélisme sur un algorithme non parallélisé (ou parallélisable)
    - Traitement des flux de données
    - Diminution du temps d'attente entre deux résultats
      - A condition que le temps d'amorçage du pipeline ne soit pas un problème

CPRABACTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

123

Notes :

# Communications (1/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Une application répartie est avant tout une application communicante
  - Elle met en œuvre de nombreuses communications entre ses tâches
    - Excepté dans de rares cas dits « *Embarrassingly parallel* »
  - Selon des schémas dictés par les algorithmes utilisés
    - Adaptés
      - A l'architecture et la topologie du réseau
      - Aux protocoles de communication (réseau, applicatifs,...)
    - Mis en œuvre
      - Soit par le programmeur lui-même
      - Soit de manière semi-automatique (choix dans une bibliothèque : squelettes algorithmiques, *design patterns*,...)
      - Soit de manière automatique (compilateurs parallélisants,...)

CPA/RBA/CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

124

Notes :



# Communications (2/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Les communications revêtent une importance toute particulière dans une application répartie
  - Elles sont coûteuses en temps et en ressources
    - Par rapport à un accès mémoire (ex. : données partagées entre threads)
  - Elles influencent les performances de l'application et les choix technologiques
  - Elles déterminent le choix de certains des algorithmes qui seront mis en œuvre dans l'application

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

125

Notes :

# Communications (3/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Pour permettre la communication entre nœuds
  - L’usage d’une mémoire partagée n’est pas viable pour un nombre important de nœuds
  - Une approche plus réaliste est que
    - Chaque nœud possède sa propre mémoire locale, et
    - Les données soient échangées via un réseau d’interconnexion par passage de messages
- Le réseau d’interconnexion joue un rôle central
  - Il détermine notamment pour une bonne part les performances d’ensemble d’un système parallèle ou réparti

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

126

Notes :

# Communications (4/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Peu de réseaux d'interconnexion mettent en œuvre  $N^2$  liaisons pour connecter  $N$  nœuds
  - Ils intègrent plutôt des commutateurs (*switches*) afin de router les messages entre nœuds distants
  - Les commutateurs peuvent bloquer ou re-router certains messages lorsque plusieurs de ces derniers nécessitent l'accès à la même liaison au même moment
    - Situation dite de compétition pour la bande passante (*competition for the bandwidth*)

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

127

Notes :

# Communications (5/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Facteurs à prendre en compte (1/7)
  - Coûts des communications
    - Les communications (et les synchronisations) entre les tâches sur un réseau de processeurs ont un coût
    - Une communication entre deux tâches implique forcément un surcoût de mise en œuvre par rapport à un échange par mémoire
      - Temps de communication des données plus important
      - Temps d'établissement de la communication
      - Temps de fermeture de la communication (*shutdown delay*)
      - Risques d'erreurs de transmission et donc temps de correction
    - Utilisent des ressources qui pourraient en partie servir pour du calcul
    - Requièrent habituellement des mécanismes de synchronisation entre les tâches pour les établir et les maintenir : temps de calcul perdu
    - De nombreuses communications peuvent saturer le réseau et faire chuter les performances de l'application

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

128

Notes :

# Communications (6/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Facteurs à prendre en compte (2/7)
  - Temps de latence (*latency*) et bande-passante (*bandwidth*)
    - La taille des messages échangés et leur fréquence peuvent conditionner fortement les performances de l'application
      - Trop de petits messages échangés va faire chuter la bande passante effective (nombre d'octets utiles (de données applicatives) transmis par seconde), les temps de latence (temps mis pour initier une communication) devenant prépondérants
        - Regrouper plusieurs messages dans un seul et unique plus volumineux

CPRMBACTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

129

Notes :

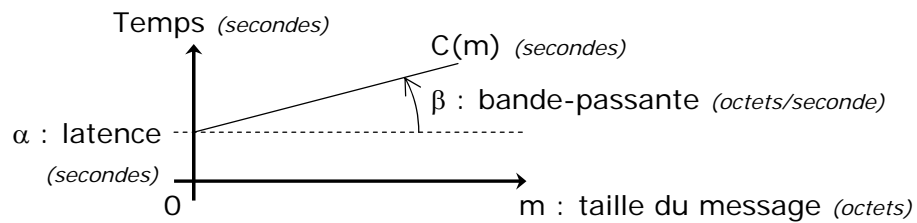
# Communications (7/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

➤ Se reporter à Designing and building parallel programs, Foster I., Chapitre 3.7, Addison-Wesley, 1995,  
pour un modèle de coût de communication plus précis  
(en particulier pour prendre en compte la compétition pour la bande-passante)

- Facteurs à prendre en compte (3/7)

- Modèle simple de coût :  $C(m) = \alpha + m/\beta$



CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

130

Notes :

# Communications (8/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Facteurs à prendre en compte (4/7)
  - Etendue des communications (1/4)
    - Lors de la mise en œuvre d'un schéma de communication, il est incontournable de savoir quelle tâche communique avec quelle autre
    - Deux types de communications possibles
      - Individuelle : implique uniquement et exclusivement deux tâches, l'une servant d'émettrice, l'autre de réceptrice
      - Collective : implique plus de deux tâches, l'une servant d'émettrice, toutes les autres de réceptrices, ou l'une servant de réceptrice, toutes les autres servant d'émettrices

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

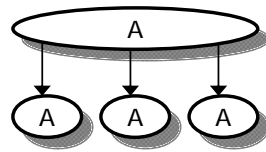
131

Notes :

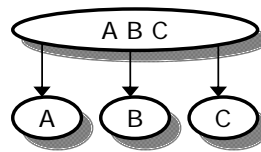
# Communications (9/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Facteurs à prendre en compte (5/7)
  - Etendue des communications (2/4)
    - Communications collectives types (1/3)



*Broadcast*  
(diffusion)



*Scatter*  
(dispersion)

CPA/RBA/CTR-2001-192T-1:171010-01



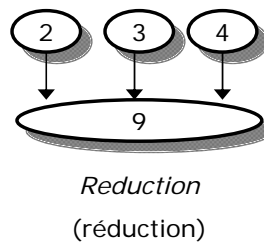
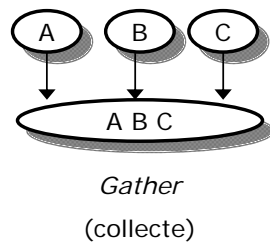
Notes :



# Communications (10/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Facteurs à prendre en compte (6/7)
  - Etendue des communications (3/4)
    - Communications collectives types (2/3)



CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

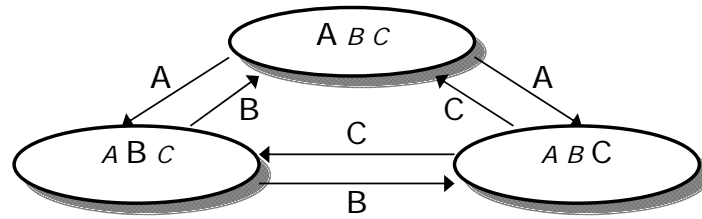
133

Notes :

# Communications (11/11)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Facteurs à prendre en compte (7/7)
  - Etendue des communications (4/4)
    - Communications collectives types (3/3)



« All to all »  
Full exchange  
(échange total)

CPA/RBA/CTR-2001-192T1:171010-01



Notes :

# Communication Modes

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Communication Modes (26)

*From Software Engineering for Real-Time Systems, R. Coudarcher, Lecture of the Specialized Master "Cooperative Avionics", ENAC, release: SERTS-100-061130-070619-02, p. 293–318, November 2007.*

CPA/RBA/CTR-20013021171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

135

Notes :

# Comm. Modes (1/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Synchronous, asynchronous, blocking and non-blocking communication modes do not necessarily correspond to what naïve thinking might indicate

- Notice

- When a point-to-point communication call is made, it is termed posting a send or posting a receive
  - Which means a send or a receive has been started

CPA/BBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

136

Notes :

# Comm. Modes (2/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Synchronous and asynchronous communications (1/6)
  - This essentially refer to different types of send operations
    - Actually, it is not meaningful to talk of synchronous or asynchronous modes in the context of a receive operation
    - Completion of a send operation means by definition that the send buffer can safely be re-used
    - A receive operation completes when a message has arrived
    - The synchronous and asynchronous send operations differ only in one respect: how completion of the send depends on the receipt of the message

CPA/RBA/CTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

137

Notes :

# Comm. Modes (3/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Synchronous and asynchronous communications (2/6)
  - Synchronous communications (1/2)
    - Safest communication mode
    - A synchronous send only completes when the receive has completed
      - When executing a synchronous communication, the calling task is blocked by the scheduler until the task with which the communication has to be established has accepted it
    - A send can be posted (*i.e.*, started) without having a ready matching receive
      - But, in any case, the sender waits until the message has been safely received (a synchronous send operation can complete only if a matching receive has been posted)

CPRBBACTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

138

Notes :

# Comm. Modes (4/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Synchronous and asynchronous communications (3/6)
  - Synchronous communications (2/2)
    - The completion of a synchronous send not only indicates that the send buffer can be reused, but also indicates that the receiver has reached a certain point into its execution, namely that it has started executing the matching receive
      - When a receive is posted, the send will not successfully complete until the matching receive has started to receive the message that was started by the synchronous send operation
    - If the sending task needs to know that the message has been received by the receiving task, then both tasks may use synchronous communication

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

139

Notes :

# Comm. Modes (5/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Synchronous and asynchronous communications (4/6)
  - Asynchronous communications (1/3)
    - Also called “buffered mode”
    - Decouples sender from receiver
    - An asynchronous send always completes (unless an error occurs), irrespective of whether the receive has completed
      - When executing an asynchronous communication, the calling task runs without to be blocked by the scheduler

CPRMBACTR-2001-02T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

140

Notes :



# Comm. Modes (6/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Synchronous and asynchronous communications (5/6)
  - Asynchronous communications (2/3)
    - When a send operation is initiated to a task, it is not always sure if the matching receive is started
      - No assumption should be made in the program about whether the out-going data is buffered by the system or not
      - If a send is buffered then the send may complete before a receive is called
      - If buffer space is unavailable then the message will not be buffered by the system and a matching receive will have to be posted for the send to complete

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

141

Notes :

# Comm. Modes (7/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Synchronous and asynchronous communications (6/6)
  - Asynchronous communications (3/3)
    - An asynchronous send completes once the message has been sent, which may or may not imply that the message has arrived at its destination
      - The message may instead lie “in the communication network” for some time
    - Communicating tasks cannot know when the communication will be established

CPRBBACTR-2001-02T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

142

Notes :

# Comm. Modes (8/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (1/7)
  - Blocking communications (1/3)
    - The sender and/or the receiver blocks until the exchange is finished
    - A blocking send call does not return until the message data has been safely stored away so that the sender is free to access and overwrite the send buffer
      - The message might be copied directly into the matching receive buffer (synchronous mode), or it might be copied into a temporary system buffer (asynchronous mode)
      - Message buffering decouples the send and receive operations. So a blocking send can complete as soon as the message was buffered, even if no matching receive has been executed by the receiver

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

143

Notes :

# Comm. Modes (9/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (2/7)
  - Blocking communications (2/3)
    - A blocking receive call blocks until the data is fully received
      - This call returns only after the receive buffer actually contains the newly received message
      - A blocking receive can complete before the matching send has completed (because the acknowledge may be delayed for a while during its transmission onto the network)
      - Of course, it can complete only after the matching send has started)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

144

Notes :

# Comm. Modes (10/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (3/7)
  - Blocking communications (3/3)
    - In the blocking forms, return from the communication routines implies completion
      - Blocking mode stops the program until the message buffer is safe to use
    - There is almost no synchronization implied by this
      - Typically, an outgoing message is buffered by the system
      - So the sending task can return from a send call when the receiving task has not started receiving the message, or even posted a corresponding receive

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

145

Notes :

# Comm. Modes (11/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (4/7)
  - Non-blocking communications (1/4)
    - Non-blocking communications allow the overlap of computation and communication
      - Typically, large numbers of operations could go on during the time it takes to send a message
    - A non-blocking point-to-point communication typically has the following form
      1. Post the operation and immediately return
      2. Do some other local work (such as computations)
      3. Probe for completion of the communication

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

146

Notes :

# Comm. Modes (12/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (5/7)

- Non-blocking communications (2/4)

- The sender and/or the receiver start the communication which is continued by the OS
      - None of them blocks until the exchange is finished
      - According to them, they can be signalled about the end of the communication operation or check by themselves if it is the case or not

CPRBBACTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

147

Notes :

# Comm. Modes (13/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (6/7)

- Non-blocking communications (3/4)

- Non-blocking refers only to whether the data buffer is available for reuse after the call or not
- A non-blocking send or receive call initiates the operation but does not complete it
  - The send call will return before the message was copied out of the send buffer
  - The receive call will return before the message is fully stored into the receive buffer
  - A separate test to know if the operation has successfully completed is needed to effectively complete the communication (*i.e.*, to verify that the data has been copied out of the send buffer or received into the receive buffer)

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

148

Notes :



# Comm. Modes (14/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Blocking and non-blocking communications (7/7)
  - Non-blocking communications (4/4)
    - A non-blocking synchronous send may complete, if matched by a non-blocking receive, before the test to know if the receive has successfully completed occurs
      - It can complete as soon as the sender knows the transfer will complete, but before the receiver knows the transfer will complete

CPRBBACTR-20013021171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

149

Notes :

# Comm. Modes (15/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Discussion (1/9)

- Synchronous and asynchronous communications (read and receive operations as well) can be either blocking or non-blocking communications
- Blocking means that the routines only return once the communication has completed
  - This is a non-local condition
    - *I.e.*, it might depend on the state of other tasks
- If both sends and receives are blocking operations then the use of the synchronous mode provides synchronous communication semantics
  - A communication does not complete at either end before both tasks rendezvous at the communication

CPRABACTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

150

Notes :

# Comm. Modes (16/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Discussion (2/9)

- Non-blocking communication is analogous to a form of delegation
- A non-blocking send is not necessarily asynchronous
- Non-blocking asynchronous send has no advantage over blocking asynchronous send
- Non-blocking send operations can be matched with blocking receive operations, and *vice-versa*
- A non-blocking send will return as soon as possible, whereas a blocking send will return after the data has been copied out of the sender memory

CPA/RBACTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

151

Notes :

# Comm. Modes (17/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Discussion (3/9)

- If a task executing a blocking synchronous send is “ahead” of the task executing the matching receive, then it will be idle until the receiving task catches up
- Similarly, if the sending task is executing a non-blocking synchronous send, the completion test will not succeed until the receiving task catches up
- Message buffering can be expensive, as it entails additional memory-to-memory copying and it requires the allocation of memory for buffering

CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

152

Notes :

# Comm. Modes (18/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Discussion (4/9)
  - Synchronous mode can therefore be slower than asynchronous mode
    - Synchronous mode is however a safer method of communication
      - The communication network can never become overloaded with undeliverable messages
    - It has the advantage over asynchronous mode of being more predictable
      - A synchronous send always synchronizes the sender and the receiver, whereas an asynchronous send may or may not do so
      - This makes the behaviour of a program more deterministic

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

153

Notes :

# Comm. Modes (19/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Discussion (5/9)
  - Synchronous communications
    - Advantages
      - Safest
      - Portable
      - Send and receive operations order is not critical
      - The amount of buffer space is irrelevant
    - Disadvantages
      - Can incur substantial synchronization overhead

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

154

Notes :

# Comm. Modes (20/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Discussion (6/9)

- Asynchronous communications

- Advantages

- Decouples the send operation from the receive operation
      - An asynchronous send operation guarantees to complete immediately, copying the message to a system buffer for later transmission if necessary
      - No synchronization overhead on the send operation
      - The order of send and receive operations is irrelevant

- Disadvantages

- Additional system overhead incurred by copying to buffer

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

155

Notes :

# Comm. Modes (21/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Discussion (7/9)
  - Blocking communications
    - Advantages
      - Jointly used with synchronous mode, insure task synchronization
    - Disadvantages
      - Computation and communication cannot overlap

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

156

Notes :



# Comm. Modes (22/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Discussion (8/9)

- Non-blocking communications (1/2)

- Advantages

- Non-blocking mode separates communication from computation
      - Avoid deadlocks
      - Decrease synchronization overhead
      - On the non-blocking receive part of the communication, it may avoid system buffering and memory-to-memory copying (as information is provided early on the location of the receive buffer)
      - The completion of a send operation may be delayed for asynchronous mode, and must be delayed for synchronous mode, until a matching receive is posted. The use of non-blocking sends in these two cases allows the sender to proceed ahead of the receiver, so that the computation is more tolerant of fluctuations in the speeds of the two tasks

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

157

Notes :

# Comm. Modes (23/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Discussion (9/9)

- Non-blocking communications (2/2)

- Disadvantages
      - Sometimes, communication completions are uneasy to manage
    - It is best to post non-blocking sends and receives as early as possible, and to do waits as late as possible
      - This way, the program spends as few time as possible in a idle state
      - Computations can overlap as much as possible communications

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

158

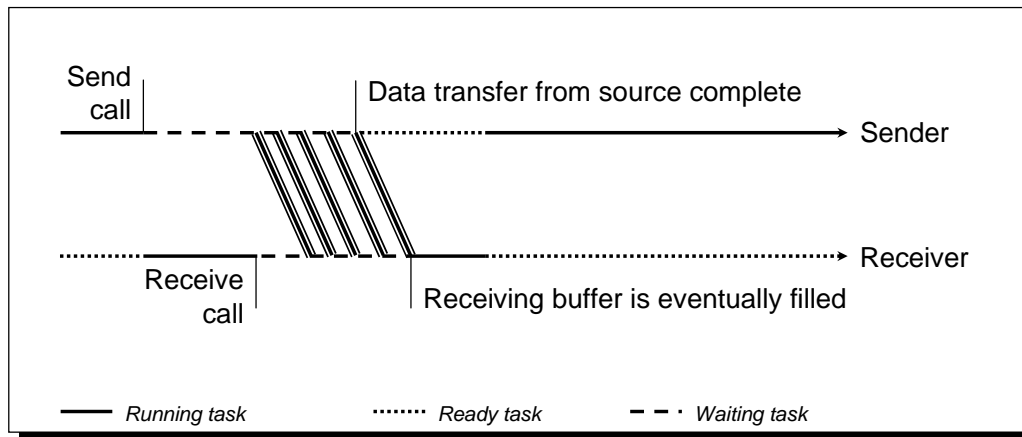
Notes :

# Comm. Modes (24/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Case studies (1/3)

- Blocking synchronous send



CPA/RBA/CTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

159

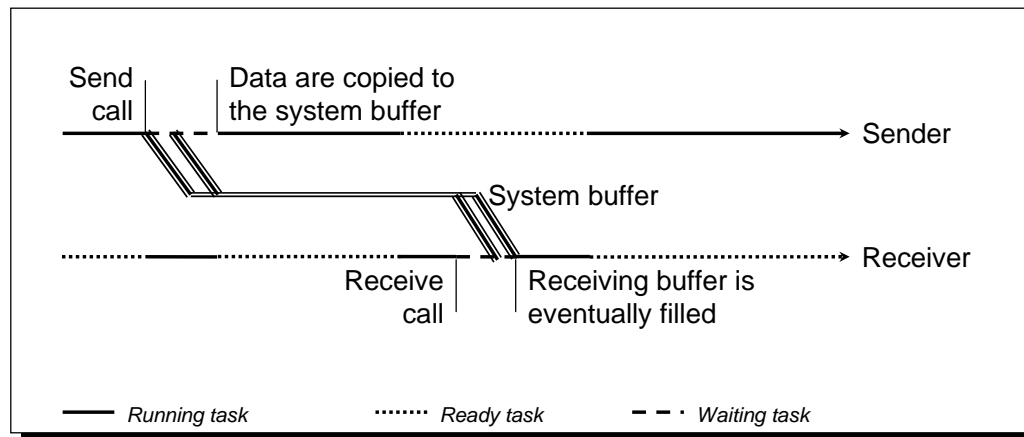
Notes :

# Comm. Modes (25/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Case studies (2/3)

- Blocking asynchronous send



CPRMBACTR-2001-132T-1:171010-01



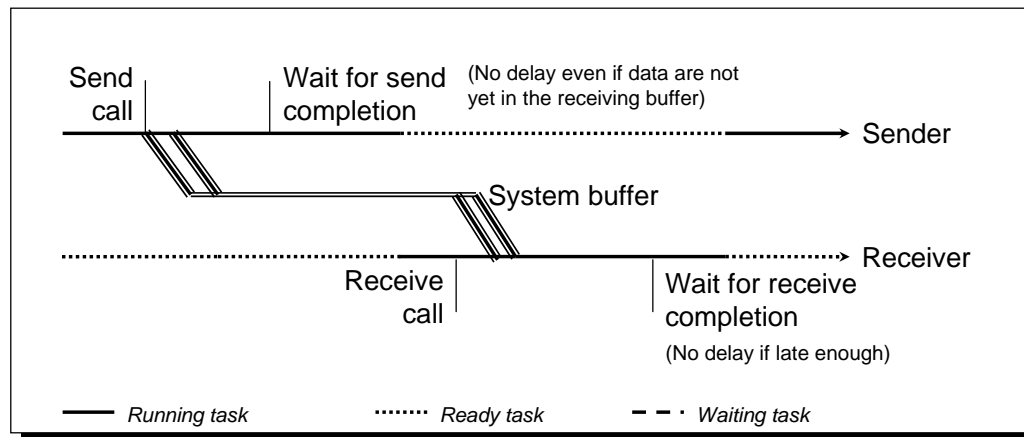
Notes :

# Comm. Modes (26/26)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Case studies (3/3)

- Non-blocking asynchronous send (and non-blocking receive)



CPA/RBA/CTR-2001-132T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

161

Notes :

# Synchronisations (1/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Dépendance de données (1/2)

(*data dependency*)

- Une dépendance existe entre les opérations d'un programme si changer leur ordre d'exécution affecte le résultat du programme
- Une dépendance de données existe lorsque plusieurs tâches ont à manipuler une information physiquement localisée au même endroit

→ Nécessite la synchronisation des traitements interdépendants

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

162

Notes :

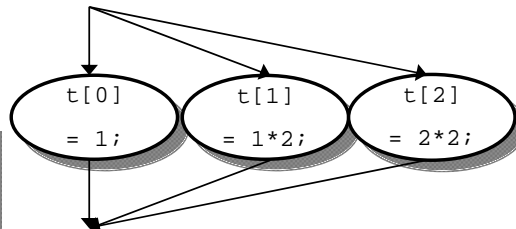
# Synchronisations (2/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Dépendance de données (2/2)  
(data dependency)

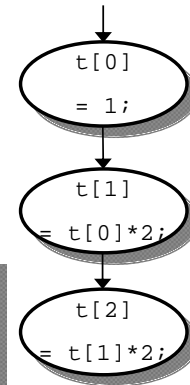
Boucle SANS  
dépendance  
de données

```
int i, t[3];
t[0] = 1;
for (i=1; i<3; i++)
    t[i] = i * 2;
```



Boucle AVEC  
dépendance  
de données

```
int i, t[3];
t[0] = 1;
for (i=1; i<3; i++)
    t[i] = t[i-1] * 2;
```



CPA/RBA/CTR-2001-132T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

163

Notes :

# Synchronisations (3/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Types de synchronisations (1/4)

- Barrière de synchronisation (1/2)

- Point de rendez-vous
      - Toutes les tâches participant à une même opération vont s’y attendre mutuellement
    - On ne franchit pas cette « barrière » tant que toutes les tâches impliquées ne l’ont pas atteinte
      - Les tâches sont bloquées en attendant que les autres tâches aient atteint ce point de synchronisation fort
    - Lorsque le point de rencontre est atteint par toutes les tâches, elles sont toutes débloquées
      - Elles continuent alors leur travail individuel indépendamment les unes des autres
      - Ou une partie commune (séquentielle et centralisée) est mise en œuvre avant que chacune ne poursuive son activité

CPRBBA-CTR-2001-132T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

164

Notes :

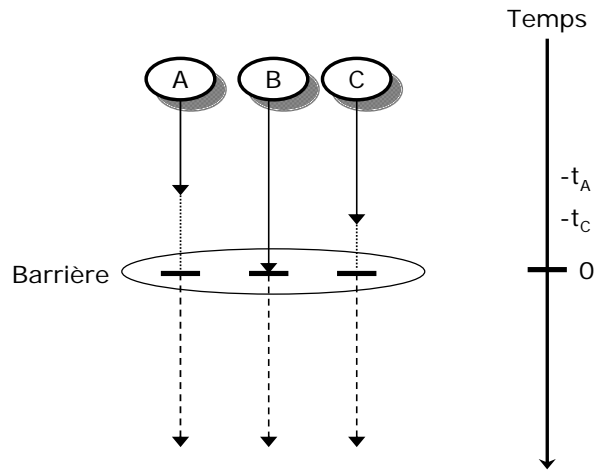


# Synchronisations (4/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Types de synchronisations (2/4)

- Barrière de synchronisation (2/2)



CPA/RBA/CTR-2001-192T1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

165

Notes :

# Synchronisations (5/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Types de synchronisations (3/4)

- *Spinlock*

- Abstraction logique sous la forme d'une structure de données particulière utilisée pour réaliser une synchronisation répartie
    - Similaire à un mutex
    - Mais nécessite l'existence d'un mécanisme matériel réalisant une opération « *test and set* » atomique
  - La tâche n'est alors pas bloquée durant sa tentative pour obtenir une ressource et reste dans un état d'attente active

➤ *Extrait de Putting multicore processing in context: Part Two, Brian T., EETimes, Mars 2006.*

- Bonnes pratiques de l'usage des spinlocks
    - Réduire la fréquence de verrouillage qu'il est possible d'obtenir
    - Verrouiller uniquement l'accès à la ressource, et non pas la routine entière
    - Utiliser plusieurs verrous pour accéder à des sections différentes de structures de données volumineuses
    - Aucune activité potentiellement bloquante (ex. : E/S synchrones) après le verrouillage d'un spinlock et ce jusqu'à son relâchement
    - Réaliser de multiples accès à une même donnée en ne la verrouillant qu'une seule et unique fois (éviter un nouveau verrouillage à chaque accès consécutif)
    - En vue de l'obtention de plusieurs verrous, obtenir le verrou le plus demandé pour la ressource la plus sollicitée en dernier

CPRBBA-CTR-2001-192T-1-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

166

Notes :

# Synchronisations (6/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion



- Types de synchronisations (4/4)
  - Communications synchrones
    - Se reporter aux modes de communication
  - Verrou et sémaphore répartis
    - Non traités dans ce cours

CPA/RBA/CTR-2001/3021/171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

167

Notes :

# Synchronisations (7/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Surcoût dû aux synchronisations

(*synchronization overhead*)

- Fait référence aux phases d'attente et de communication entre tâches interdépendantes attendant des résultats partiels en provenance d'autres tâches afin de continuer leur exécution

- A proprement parler, le surcoût dû aux synchronisations n'est pas le temps d'attente sur un sémaphore, un mutex ou un *spinlock* afin d'acquérir une ressource, mais celui causé par une tâche qui ne peut pas continuer son exécution tant qu'un résultat intermédiaire ne lui est pas fourni par une autre tâche

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

168

Notes :

# Synchronisations (8/8)

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Situation de concurrence (*race condition*)
  - Apparaît lorsque deux tâches, ou plus, réalisent une opération dont le résultat dépend de facteurs temporels imprévisibles
- Partage de ressources (*resource sharing*)
  - Une situation de concurrence peut apparaître lorsque l'accès, de la part de plusieurs tâches, à une ressource partagée n'est pas correctement synchronisé
    - Une solution est de permettre aux tâches d'obtenir un verrou sur la ressource

CPA/RBA/CTR-2001-192T-1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

169

Notes :

# CONCLUSION

Notes :

# Le mot de la fin

Introduction | Architectures | Interconnexion | Programmation MPI | Conclusion

- Pour paraphraser Clay Breshears  
dans son introduction au chapitre 4 de son livre *The Art of Concurrency*

Les programmations parallèle et répartie  
sont encore plus un art qu'une science

CPA/RBA/CTR-2001-192T1:171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

171

Notes :

# ACRONYMES

## A

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- ALU                    Arithmetic and Logical Unit
- ASIC                Application-Specific Integrated Circuit



# C

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- CISC                      Complex Instruction Set Computer
- CPU                      Central Processing Unit

CMRBA-CTR-AD-8201-1710-001



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

174

# D

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- DFD                      Data Flow Diagram
- DMA                      Direct Memory Access
- DSP                      Digital Signal Processor

CMRBA-CTR-AD-8201-1710-001



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

175

# F

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- FIFO First In First Out
- FPGA Field Programmable Gate Array
- FSM Finite State Machine

CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

176

# G

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)


- GCC GNU Compiler Collection
- GPGPU General-Purpose computation on GPU
- GPU Graphics Processing Unit
- GRAFCET GRAphe Fonctionnel de  
Commande Etape Transition


CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

177

I	
<a href="#">Acronymes</a>   <a href="#">Glossaire</a>   <a href="#">Bibliographie</a>   <a href="#">Sites Web</a>	
<ul style="list-style-type: none"> <li>– IEEE</li> <li>– I/O</li> <li>– IPC</li> </ul>	<p>Institute of Electrical and Electronics Engineers</p> <p>Input/Output</p> <p>InterProcess Communications</p>
<small>CMRBA-CTR-AD-8201-1710-0-01</small> 	<small>Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17</small>
	178

L	
<a href="#">Acronymes</a>   <a href="#">Glossaire</a>   <a href="#">Bibliographie</a>   <a href="#">Sites Web</a>	
<ul style="list-style-type: none"> <li>– LAN</li> </ul>	<p>Local Area Network</p>
<small>CMRBA-CTR-AD-8201-1710-0-01</small> 	<small>Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17</small>
	179

# M

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- MFLOPS Million of Floating-Point Operation Per Second
- MIPS Million of Instructions Per Second
- MMU Memory Management Unit
- MPMD Multiple Program Multiple Data
- MSC Message Sequence Chart

CMRBA-CTR-AD-5021-1710-0-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

180

# N

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- NUMA Non-Uniform Memory Access

CMRBA-CTR-AD-5021-1710-0-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

181

# O

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- OS Operating System
- OSE Open System Environment
- OSF Open Software Foundation
- OSI Open System Interconnection

CMRBA-CTR-AD-8201-T10R01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

182

# P

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- PC Personal Computer
- PCB Process Control Block
- POSIX Portable Operating System Interface
- PSD Program Structure Diagram

CMRBA-CTR-AD-8201-T10R01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

183

# R

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- RAM Random Access Memory
- RISC Reduced Instruction Set Computer
- RFC Request For Comments
- ROM Read Only Memory
- RPC Remote Procedure Call

CMRBA-CTR-200-5021-17100-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

184

# S

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- SCO Santa Cruz Operations
- SDL Specification and Description Language
- SMP Symmetric Multi-Processing
- SPMD Single Program Multiple Data
- SRT Shortest Response Time
- STD State Transition Diagram

CMRBA-CTR-200-5021-17100-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

185

# T

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)

- TCB Task Control Block
- TCP/IP Transmission Control Protocol/Internet Protocol

CMRBA-CTR-AD-5021-T10R01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

186

# U

[Acronymes](#) | [Glossaire](#) | [Bibliographie](#) | [Sites Web](#)


- UDP User Datagram Protocol

CMRBA-CTR-AD-5021-T10R01




Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17


187

W		
Acronymes   Glossaire   Bibliographie   Sites Web		
– WAN	Wide Area Network	
<div> <div>CMRBA-CTR-AD-8201-17100-01</div> <div>  R�mi Coudarcher   Calcul parall�le. Techniques de base   octobre 17 </div> <div>188</div> </div>		

GLOSSAIRE		
<div> <div>CMRBA-CTR-AD-8201-17100-01</div> <div>  R�mi Coudarcher   Calcul parall�le. Techniques de base   octobre 17 </div> <div>189</div> </div>		



Glossaire		
Acronymes   <a href="#">Glossaire</a>   Bibliographie   Sites Web		
<ul style="list-style-type: none"> <li>• Glossary (22)           <p><i>From Software Engineering for Real-Time Systems, R. Coudarcher, Lecture of the Specialized Master "Cooperative Avionics", ENAC, release: SERTS-110-061130-080922-01, p. 248–267, November 2008.</i></p> </li> </ul>		
CMABA-CTR-AD-8021-T10R01	 Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	190

A		
Acronymes   <a href="#">Glossaire</a>   Bibliographie   Sites Web		
<ul style="list-style-type: none"> <li>• Asynchronous cancelability           <ul style="list-style-type: none"> <li>– Thread's ability to receive a cancellation request at any time (and not only at cancellation points)</li> </ul> </li> <li>• Asynchronous signal           <ul style="list-style-type: none"> <li>– Signal that is the result of an event that is external to the process and is delivered at any point in a thread's execution when such an event occurs</li> </ul> </li> </ul>		
CMABA-CTR-AD-8021-T10R01	 Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	191

## C (1/3)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Cancel**
  - Mechanism by which one thread requests termination of another thread (or itself)
- **Cancellable routine**
  - Routine in which a cancel may be delivered
- **Cancellation point**
  - Specific routine (usually belonging to a thread library) that, when called, can determine whether a cancel is pending, and if so, can deliver the cancel

CMRBA-CTR-AD-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

192

## C (2/3)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Cluster**
  - “Virtual” parallel machine designed from common and standard elements, generally at low cost
    - PCs or workstations
    - Communication networks such as Ethernet
    - Standard communication protocols
    - Operating systems such as UNIX, Linux or Windows
- **Computing node**
  - Dedicated circuit, processor or machine, autonomous, able to process data
  - Interconnected by a communication network to other nodes in order to make parallel processing

CMRBA-CTR-AD-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

193

## C (3/3)

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Condition variable**
  - Object that allows a thread to block its own execution until some shared data reaches a particular state

CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

194

## D

Acronyms | [Glossaire](#) | Bibliographie | Sites Web


- **Dispatching**
  - The dispatcher starts and stops the tasks, *i.e.*, it implements the schedule
    - To be strict, the distinction between scheduling and dispatching has to be made, with dispatching being the simplest of the two operations
- **Dynamic memory**
  - Memory that is allocated by the programme as a result of a call to some memory management function, and that is referenced through pointer variables


CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

195

E		
Acronymes   <a href="#">Glossaire</a>   Bibliographie   Sites Web		
<ul style="list-style-type: none"> <li>• <b>Embarrassingly parallel</b> <ul style="list-style-type: none"> <li>– Describes a problem that is parallelizable in a manner so clear and simple as its implementation is immediate</li> <li>– Its parallelization generally presents only a very limited or no overhead</li> </ul> </li> <li>• <b>Exception</b> <ul style="list-style-type: none"> <li>– Object that describes an error condition</li> </ul> </li> <li>• <b>Exception scope</b> <ul style="list-style-type: none"> <li>– Block of code where exceptions are handled</li> </ul> </li> </ul>		
CMRBA-CTR-200-50201-17100-01	 Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	196

G		
Acronymes   <a href="#">Glossaire</a>   Bibliographie   Sites Web		
<ul style="list-style-type: none"> <li>• <b>Grain</b> <ul style="list-style-type: none"> <li>– Qualitative measurement of ratio calculations and communications               <ul style="list-style-type: none"> <li>• Coarse grain                   <ul style="list-style-type: none"> <li>Treatments are relatively long between two communications (parallelization in the form of processes or full applications communicating with each other)</li> </ul> </li> <li>• Fine grain                   <ul style="list-style-type: none"> <li>Treatments are relatively short between two communications (parallelization in the form of threads or at instruction level)</li> </ul> </li> </ul> </li> </ul> </li> </ul>		
CMRBA-CTR-200-50201-17100-01	 Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	197

# L

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Lifetime**
  - Length of time memory is allocated for a particular purpose

CMRBA-CTR-200-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

198

# M (1/2)

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Massively parallel**
  - System comprising a large number of processing units
    - Hundred or thousand units of magnitude
    - Each unit generally exhibits limited performance
- **Multi-core**
  - CPU built around several "CPU" on the same chip

CMRBA-CTR-200-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

199

# M (2/2)

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Multithreaded programming**
  - Division of a programme into multiple threads that execute concurrently
- **Mutex**
  - Meaning: MUTual EXclusion
  - Is an object that multiple threads use to ensure the integrity of a shared resource (most commonly shared data) that they access by allowing only one thread to access it at a time

CMRBA-CTR-AD-5021-T10R01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

200

# N

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Non recursive mutex**
  - Mutex that can be locked exactly once by a thread
  - If a thread tries to lock the non recursive mutex again without first unlocking it, the thread receives an error instead of deadlocking
- **NP-complete**
  - The complexity of the problem increases exponentially with the number of constraints involved

CMRBA-CTR-AD-5021-T10R01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

201

# P (1/2)

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Parallel overhead**
  - Time spent into task coordination
    - And during which no actual processing work is done
  - Origins
    - Start up time and stop time of a task
    - Synchronization operations
    - Data communications
    - Operations added by compilers, libraries, operating systems, etc., to manage the parallelism
- **Predicate**
  - Boolean expression that defines a particular state of shared data
  - Threads wait on a condition variable for shared data to enter the defined state

CMRBA-CTR-200-50201-17100-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

202

# P (2/2)

Acronyms | [Glossaire](#) | Bibliographie | Sites Web

- **Pre-emption**
  - OS' ability to replace the running task by the next-to-run (i.e. ready) task
  - With a non pre-emptive OS, tasks run to completion
- **Priority inversion**
  - Occurs when interaction among three or more threads blocks the highest-priority thread from executing until after the lowest priority thread can execute

CMRBA-CTR-200-50201-17100-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

203

# R

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Recursive mutex**
  - Mutex that can be locked more than once by a given thread without causing a deadlock
  - The thread must unlock the recursive mutex the same number of times that it locked the recursive mutex before another thread can lock the mutex
- **Répartition de charge (*load balancing*)**
  - Distribution de la charge de travail entre plusieurs nœuds de calcul pour équilibrer la charge totale à tout instant

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

204

# S (1/5)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Scalability**
  - Ability of a hardware or software system to improve its performance in proportion to the number of processors used
  - Contributing factors
    - Bandwidth of the communication medium with memory
    - Communication network between the computing nodes
    - Algorithm choices
    - Overhead factor reduction
- **Scheduling**
  - Determining the order and the timing (*i.e.*, the “schedule”) with which tasks should be run
    - To be strict, the distinction between scheduling and dispatching has to be made, with dispatching being the simplest of the two operations

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

205



## S (2/5)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Scheduling policy attribute**
  - Attribute that describes how the thread is scheduled for execution relative to the other threads in the programme
- **Scheduling precedence**
  - The set of specifications of threads and the scheduling algorithm that, in combination, determine which thread will be allowed to run when a scheduling decision is made
  - Scheduling decisions are made when a thread becomes ready to run (for example, when a mutex on which it was waiting is unlocked or a condition variable on which it was waiting is signalled or broadcast), or when a thread is blocked (for example, when it attempts to lock a locked mutex, or when it waits on a condition variable)

CMRBA-CTR-AD-50211-1710-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

206

## S (3/5)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Scheduling priority attribute**
  - Attribute that specifies the execution priority of a thread, expressed relative to other threads in the same policy
- **Scope**
  - Areas of a programme where code can access memory
- **SMP (Symmetric Multi-Processing)**
  - Parallel architecture with identical processors sharing a common memory and common access to all resources
- **Software interrupt handler**
  - A routine that is executed in response to an interrupt generated by the operating system or equivalent support software

CMRBA-CTR-AD-50211-1710-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

207

## S (4/5)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Speedup**

- Very commonly used indicator to reflect the performance of a parallelization
- The acceleration produced (and measured) of the speed of execution of a program as it is parallel compared to its sequential form is given by the ratio:

$$\frac{\text{Sequential completion time}}{\text{Parallel completion time}}$$

- **Stack memory**

- Memory that is allocated from a thread's stack area at run time by code generated by the language compiler, generally when a routine is initially called

- **Static memory**

- Any variable that is permanently allocated at a particular address for the life of the programme

CMRBA-CTR-AD-5021-1710-001



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

208

## S (5/5)

Acronymes | [Glossaire](#) | Bibliographie | Sites Web

- **Synchronous signal**


- Signal that is the result of an event that occurs inside a process and is delivered synchronously with respect to that event


CMRBA-CTR-AD-5021-1710-001



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

209

T (1/2)		
Acronymes   <a href="#">Glossaire</a>   Bibliographie   Sites Web		
<ul style="list-style-type: none"> <li>• Task <ul style="list-style-type: none"> <li>– Autonomous section of a processing</li> <li>– Process or thread</li> </ul> </li> <li>• Thread <ul style="list-style-type: none"> <li>– Single, sequential flow of control within a program</li> <li>– Within a single thread, there is a single point of execution</li> </ul> </li> <li>• Thread-reentrant <ul style="list-style-type: none"> <li>– Routine that functions normally despite being called simultaneously or sequentially in different threads</li> </ul> </li> </ul>		
<small>CMRBA-CTR-AD-5021-T10R01</small> 	Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	210

T (2/2)		
Acronymes   <a href="#">Glossaire</a>   Bibliographie   Sites Web		
<ul style="list-style-type: none"> <li>• Thread-safe <ul style="list-style-type: none"> <li>– Routine that can be called simultaneously from multiple threads without risk of corruption</li> </ul> </li> <li>• Thread-specific data <ul style="list-style-type: none"> <li>– User-specified fields of arbitrary data that can be added to a thread's context</li> </ul> </li> <li>• Time-sharing or Time-slicing <ul style="list-style-type: none"> <li>– Mechanism that ensures that every thread is allowed time to execute by pre-empting running threads at fixed intervals</li> </ul> </li> </ul>		
<small>CMRBA-CTR-AD-5021-T10R01</small> 	Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	211

# BIBLIOGRAPHIE

## Bibliographie (1/6)

Acronymes | Glossaire | [Bibliographie](#) | Sites Web

- **A 6-Regular Torus Graph Family with Applications to Cellular and Interconnection Networks**  
Iridon M. and Matula D. W., Journal of Graph Algorithms and Applications, Vol. 6(4), p. 373–404, 2002.
- **A Survey of Parallel Computer Architectures**  
Duncan R., Survey and Tutorial Series, Computer, IEEE, p. 5–16, February 1990.
- **A Taxonomy for Computer Architectures**  
Skillicorn D. B., Computer, IEEE, Vol. 21(11), p. 46–57, November 1988.
- **Attack the Parallel Worlds of Parallel Programming**  
Stewart D., Embedded Systems Design, June 2008.
- **Cache Coherence**  
Torrellas J., Course CS533, 2003.
- **Cache Coherence Techniques for Multicore Processors**  
Marty M. R., Ph.D. Dissertation in Computer Sciences, University of Wisconsin, Madison, 2008.
- **CERN Courier**  
CERN, Vol. 51(8), p. 7, IOP Publishing Ltd., octobre 2011.
- **Communications dans les réseaux de processeurs**  
de Rumeur J., Preface from Quinton P., Coll. ERI, Masson, Paris, September 1994.

# Bibliographie (2/6)

Acronymes | Glossaire | [Bibliographie](#) | Sites Web

- **Communicating Sequential Processes**  
Hoare T., 1985.
- **Cooperating Sequential Processes**  
Dijkstra E. W., 1965.
- **Data Structures in the Multicore Age**  
Shavit N., Communications of the ACM, Vol. 54(3), p. 76–84, March 2011.
- **DEC OSF/1 Network Programmer's Guide**  
Digital Equipment Corp., Part Number: AA-PS2WC-TE, August 1994.
- **DEC OSF/1 Programming with ONC RPC**  
Digital Equipment Corp., Part Number: AA-Q0R5A-TE, February 1994.
- **Designing and building parallel programs**  
Foster I., Addison-Wesley, 1995.
- **Distributed Operating Systems**  
Tannenbaum A. S., Prentice Hall, 1995.
- **Embedded Multiprocessors: Scheduling and Synchronization**  
Sriram S. and Bhattacharyya S. S., Marcel Dekker, New York, 2000.
- **Graphes, algorithmes, logiciels**  
Minoux M. and Bartnik G., Dunod informatique, Bordas, Paris, November 1986.

Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

214


# Bibliographie (3/6)


Acronymes | Glossaire | [Bibliographie](#) | Sites Web

- **Implementing Remote Procedure Calls**  
Birell A. D. and Nelson B. J., ACM Transactions on Computer Systems, vol. 2(1), p. 39-59, February 1984.
- **Introduction aux algorithmes et architectures parallèles : grilles, arbres et hypercubes**  
Leighton F. T., Traduction de Fraigniaud P. et Fleury E., International Thomson Publishing, Vuibert, January 1998.
- **La communication sous UNIX**  
Rifflet J.-M., Mc Graw-Hill, 1990.
- **Le calcul haute-performance**  
La Recherche, Supplément #393, January 2006.
- **Les réseaux**  
Pujolle G., Eyrolles.
- **Les superordinateurs dans le secteur aéronautique français**  
Karadimas G., Nouvelle revue aéronautique et astronautique, n° 2, June 1994.
- **Lightweight Remote Procedure Call**  
Bershad B. N. et al., ACM Transactions on Computer Systems, vol. 8(1), p. 37–55, February 1990.
- **Linearly Extendible ARM (LEA) – A Constant Degree Topology for Designing Scalable and Cost Effective Interconnection Networks**  
Alam J., Kumar R. and Khan Z., Ubiquitous Computing and Communication Journal, Vol. 5(2), June 2010.

Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

215

Bibliographie (4/6)		
Acronymes   Glossaire   <a href="#">Bibliographie</a>   Sites Web		
<div>COMBAC-CTR-200-50201-1710-01</div> <div></div>	<ul style="list-style-type: none"><li>• <b>L'ordinateur et le cerveau</b> Von Neumann J., Coll. Champs, Flammarion, janvier 1996.</li><li>• <b>Mathematical theory of connecting networks and telephone traffic</b> Benes V. E., Vol. 17, Elsevier, 1965.</li><li>• <b>Organisation et conception des ordinateurs. L'interface matériel/logiciel</b> Patterson D. and Hennessy J., Chapter 7 – La hiérarchie des mémoires, p. 458–539, Dunod, Paris, 1994.</li><li>• <b>Organisation et conception des ordinateurs. L'interface matériel/logiciel</b> Patterson D. and Hennessy J., Chapter 9 – Les machines parallèles, p. 604–660, Dunod, Paris, 1994.</li><li>• <b>Parallel Programming: An Axiomatic Approach</b> Hoare C. A. R., LNCS Language Hierarchies and Interfaces, Vol. 46, Chapter 1: Concurrency, p. 11–42, Springer-Verlag, 1976.</li><li>• <b>Patterns for Parallel Programming</b> Mattson T. G., Sanders B. A. and Massingill B. L., Software Patterns Series, Addison-Wesley Professional, 2004.</li><li>• <b>Putting multicore processing in context: Part One</b> Brian T., EETimes, January 2006.</li><li>• <b>Putting multicore processing in context: Part Two</b> Brian T., EETimes, March 2006.</li><li>• <b>Resources, Concurrency and Local Reasoning</b> O'Hearn P. W., Queen Mary, University of London, 2005.</li></ul>	
	Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	
	216	

Bibliographie (5/6)		
Acronymes   Glossaire   <a href="#">Bibliographie</a>   Sites Web		
<div>COMBAC-CTR-200-50201-1710-01</div> <div></div>	<ul style="list-style-type: none"><li>• <b>Sequencing Tasks in Multiprocess Systems to Avoid Deadlocks</b> Shoshani A. and Coffman E. G., Proceedings of the 11th Annual Symposium on Switching and Automata Theory SWAT'70, IEEE Computer Society, Washington DC, USA, 1970.</li><li>• <b>Software Techniques for Shared-Cache Multi-Core Systems</b> Tian T. and Shih C.-P., Intel Software Network, October 2011.</li><li>• <b>Symmetry and Similarity in Distributed Systems</b> Johnson R. E. &amp; Schneider F. B., Proc. 4th ACM Symposium on Principles of Distributed Computing, p. 13–22, 1985.</li><li>• <b>Systolic Arrays Processors</b> McCanny J., McWhirter J. &amp; Swartzlander E., Prentice Hall International Ltd., Hertfordshire, 1989.</li><li>• <b>The Art of Concurrency</b> Breshears C., O'Reilly Media Inc., May 2009.</li><li>• <b>The Performance and Scalability of Distributed Shared Memory Cache Coherence Protocols</b> Heinrich M. A., Ph.D. Dissertation, Computer Systems Laboratory, Stanford University, October 1998.</li><li>• <b>The Synchronous Languages Twelve Years Later</b> Benveniste A. et Coll., Proceedings of the IEEE, 2003.</li></ul>	
	Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17	
	217	

# Bibliographie (6/6)

Acronymes | Glossaire | [Bibliographie](#) | Sites Web

- **The World Wide Web. Everything you (n)ever wanted to know about its servers**  
Vass J. et al., IEEE Potentials, p. 33-37, October/November 1998.
- **Théorie des réseaux (graphes)**  
Kuntzmann J., Dunod, Paris, 1972.
- **Thinking and Programming in Parallel**  
Metaxas P. T., CS331 Class notes, Computer Science Department, Wellesley College, Wellesley, MA, USA, Edition 0.81, 1997.
- **Topological Properties of Hierarchical Interconnection Networks: A Review and Comparison**  
Abd-El-Barr M. and Al-Somani T. F., Journal of Electrical and Computer Engineering, Vol. 2011, Art. ID 189434, Hindawi Publishing Corp., 2011.
- **Validity of the single processor approach to achieving large scale computing capabilities**  
Amdahl G. M., Proc. AFIPS Spring Joint Computer Conf. 30 p. 483-485, Atlantic City, N. J., April 1967.
- **Very High Speed Computing Systems**  
Flynn M., Proc. of the IEEE, Vol. 54, p. 1901-1909, 1966.

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

218


## SITES WEB


CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

219

Sites Web (1/4)		
Acronymes   Glossaire   Bibliographie   <u>Sites Web</u>		
<p><b>Foundations</b></p> <ul style="list-style-type: none"> <li>• <b>The Importance of Data Locality in Distributed Computing Applications</b> Szalay et al. <a href="http://pcbunn.cacr.caltech.edu/Workflows-NSF-Szalay.htm">http://pcbunn.cacr.caltech.edu/Workflows-NSF-Szalay.htm</a></li> </ul> <p><b>History of Parallel Computing</b></p> <ul style="list-style-type: none"> <li>• <b>Historical development of parallel systems</b> Nucciarone J., Introduction to Computational Science Tools, Penn State Computational Science Faculty, 2008 Fall Seminar Series <a href="http://www.csci.psu.edu/seminars/fallnotes/historyHPC.pdf">http://www.csci.psu.edu/seminars/fallnotes/historyHPC.pdf</a></li> <li>• <b>The History of the Development of Parallel Computing</b> Wilson G., October 1994. <a href="http://parallel.ru/history/wilson_history.html">http://parallel.ru/history/wilson_history.html</a></li> </ul> <p><b>Internet</b></p> <ul style="list-style-type: none"> <li>• <b>La création d'ARPANET et son évolution vers l'Internet</b> ISOC – Internet SOCIety. <a href="http://www.isoc.org/internet/history">http://www.isoc.org/internet/history</a></li> </ul>		
<small>CMRBA-CTR-AD-5021-1710-01</small> 	<small>Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17</small>	<small>220</small>

Sites Web (2/4)		
Acronymes   Glossaire   Bibliographie   <u>Sites Web</u>		
<p><b>Miscellaneous</b></p> <ul style="list-style-type: none"> <li>• <b>Does the Concurrency Development is green?</b> Vernié E., Microsoft France, MSDN, February 2009. <a href="http://blogs.msdn.com/devpara/archive/2009/02/16/does-the-concurrency-development-is-green.aspx">http://blogs.msdn.com/devpara/archive/2009/02/16/does-the-concurrency-development-is-green.aspx</a></li> <li>• <b>Parallélisme avec la beta 1 de Visual Studio 2010 : mise à l'échelle d'une application C++</b> Boucard B., Microsoft France, MSDN, August 2009. <a href="http://msdn.microsoft.com/fr-fr/visualc/dd981007.aspx">http://msdn.microsoft.com/fr-fr/visualc/dd981007.aspx</a></li> <li>• <b>TOP500 Project</b> <a href="http://www.top500.org">http://www.top500.org</a></li> </ul> <p><b>MPI – Message Passing Interface</b></p> <ul style="list-style-type: none"> <li>• <b>Specifications</b> MPI Forum. <a href="http://www-unix.mcs.anl.gov/mpi">http://www-unix.mcs.anl.gov/mpi</a></li> <li>• <b>The Message Passing Interface (MPI) standard</b> Math. &amp; Comp. Science Div., Argonne National Lab. <a href="http://www.mcs.anl.gov/research/projects/mpi/index.htm">http://www.mcs.anl.gov/research/projects/mpi/index.htm</a></li> </ul>		
<small>CMRBA-CTR-AD-5021-1710-01</small> 	<small>Rémi Coudarcher   Calcul parallèle. Techniques de base   octobre 17</small>	<small>221</small>



# Sites Web (3/4)

Acronymes | Glossaire | Bibliographie | [Sites Web](#)

## Network Programming

- **Beej's Guide to Network Programming**  
<http://www.ecst.csuchico.edu/~beej/guide/net>
- **BSD Sockets: A Quick and Dirty Primer**  
<http://world.std.com/~jimf/papers/sockets/sockets.html>
- **RFC1180: A Tutorial to TCP/IP**  
<http://www.faqs.org/rfc/rfc1180.txt>
- **UREC**  
CNRS.  
<http://www.urec.cnrs.fr>

## Operating Systems

- **POSIX**  
<http://www.opengroup.org>

CNRS-CTR-AD-5021-17106-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

222

# Sites Web (4/4)

Acronymes | Glossaire | Bibliographie | [Sites Web](#)

## Parallel and Distributed Computing Projects

- **SETI@home**  
Berkeley University.  
<http://setiathome.berkeley.edu>

## Von Neumann Architecture

- **First Draft of a Report on the EDVAC**  
Von Neumann J., Moore School of Electrical Engineering, University of Pennsylvania, June 1945.  
(Reedited by Godfrey M. D., Information Systems Laboratory, Electrical Engineering Department, Stanford University, Stanford, California, November 1992.)  
<http://www.virtualtravelog.net/entries/2003-08-TheFirstDraft.pdf>
- **John Von Neumann, né Johann von Neumann**  
Larousse encyclopédique, 2008.  
<http://www.larousse.fr/encyclopedie/ehm/Von%20Neumann/181003>
- **Von Neumann, l'architecte de l'informatique**  
M. Q., Les Echos, April 2008.  
<http://www.lesechos.fr/info/comm/300253961.htm>

CNRS-CTR-AD-5021-17106-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

223

# ANNEXES

## Annexes

• Task Concepts	<u>226</u>
• Data Dependency	<u>251</u>
• Multi-threaded Application Design	<u>261</u>
• Conception	<u>277</u>

# Task Concepts

- Task, Process and Thread Concepts (21)

*From Software Engineering for Real-Time Systems, R. Coudarcher, Lecture of the Specialized Master "Cooperative Avionics", ENAC, release: SERTS-100-061130-070619-02, p. 20–39, November 2007.*

- Threads (2)

*Excerpt from Software Engineering for Real-Time Systems, R. Coudarcher, Lecture of the Specialized Master "Cooperative Avionics", ENAC, release: SERTS-100-061130-070619-02, p. 339–340, November 2007.*

- Task states (1)

*Excerpt from Software Engineering for Real-Time Systems, R. Coudarcher, Lecture of the Specialized Master "Cooperative Avionics", ENAC, release: SERTS-100-061130-070619-02, p. 341, November 2007.*

CMABA-CTR-200-5021-171010-01



# Process and Thread (1/21)

- Task

- The task term can be used interchangeably with process or thread (to refer to one or the other)
  - But process cannot stand for thread and thread cannot stand for process as they are different technical entities
- Task management is a primary job of the OS
  - Task management for an RTOS is a bit more critical than for a general purpose OS
    - If a real-time task is created, it has to get the memory it needs without delay
    - And that memory has to be locked in main memory in order to avoid access latencies due to swapping

CMABA-CTR-200-5021-171010-01



# Process and Thread (2/21)

- Process (1/4)

- Is an instance of a computer program that is being executed
  - A program itself is just a passive collection of instructions, and therefore just a part of a process
- Several processes can be associated with the same program
  - Each would execute independently
- Basically, a process is an execution path through one or more programs
  - Execution may be interleaved with other processes
  - The process has an execution state and a dispatching priority

CMRBA-CTR-200-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

228

# Process and Thread (3/21)

- Process (2/4)

- It consists of the following resources
  - An "image" of the executable machine code associated with a program
  - The memory which contains the executable code, process-specific data (including a call stack to keep track of active subroutines) and a heap holding data generated at run time
  - Operating system descriptors of resources that are allocated to the process, such as file descriptors ("handles")
  - A process state, also called the process context, such as the content of the processor's registers, the physical memory addressing,...

CMRBA-CTR-200-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

229

# Process and Thread (4/21)

- Process (3/4)
  - The OS holds most of the information about active processes in data structures called Process Control Blocks (PCB)
    - Or Task Control Blocks (TCB)
  - Processes can communicate using mechanisms provided by the OS
    - These mechanisms enable processes to interact in safe and predictable ways

CMRBA-CTR-200-5021-171010-01

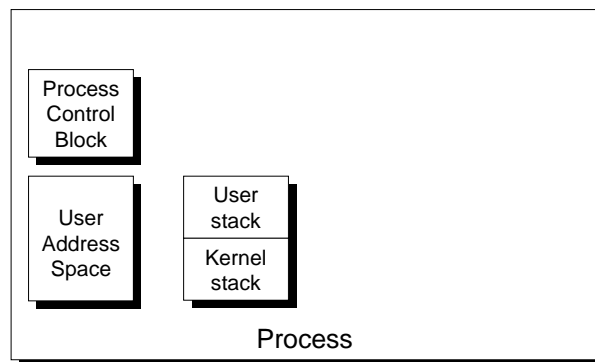


Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

230

# Process and Thread (5/21)

- Process (4/4)
  - Single-threaded process model



CMRBA-CTR-200-5021-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

231

# Process and Thread (6/21)

- Thread (1/8)
  - In computer science, a “thread” is short for a “thread of execution”
  - A thread is an independent flow of control within a process
  - Threads are a way for a programme to split itself into several simultaneously running tasks

CMABA-CTR-200-5021-171016-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

232

# Process and Thread (7/21)

- Thread (2/8)
  - A thread has:
    - Its own execution state (especially its own program counter (PC) and a copy of the other processor’s registers content)
    - Its own execution stack (*i.e.* stack pointer) and some per-thread static storage for local variables
    - Access to the memory address space and resources of its process
      - It shares global data such as global variables and file descriptors
    - Scheduling properties (such as policy or priority)

CMABA-CTR-200-5021-171016-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

233

# Process and Thread (8/21)

- Thread (3/8)

- Threads compared with processes

- Processes are address space independent tasks, not threads
      - Threads of a process share the same address space, *i.e.*, the process' address space
      - Threads of a process share the process' resources (file handles, sockets, device handles,...)
      - A threads only own a stack and a copy of the CPU's registers
    - Processes have a wide state information data structure, not threads
      - Threads of a process share most part of the process' state information
    - Multiple threads of a process share the same program code, OS resources (such as memory and file access) and OS permissions (for file access) as the process they belong to

CMABA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

234

# Process and Thread (9/21)

- Thread (4/8)

- Benefits of threads vs. processes

- Less time is needed to create or terminate a thread than a process
      - Because threads inherit of the current process address space
    - Less time is needed to switch between two threads within the same process
      - Partly because the newly created thread uses the current process address space
    - Less communication overheads are encountered
      - Communicating between the threads of one process is simple because threads share everything and address space in particular. Data produced by one thread are immediately available to all others within the same process
    - Context switching between threads of a process is faster than context switching between different processes

CMABA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

235

# Process and Thread (10/21)

- Thread (5/8)
  - A process can have several threads of execution
    - A process with multiple threads is referred to as a multi-threaded process
    - A process that has only one thread is referred to as a single-threaded process
    - Multi-threaded processes have the advantage that they can perform several tasks concurrently without the overhead needed to create a new process and handle synchronised communication between processes
  - Threads can be seen as lightweight processes
    - Their context is smaller than the context of processes
    - Most of the overhead has already been accomplished through the creation of its process

CMABA-CTR-200-5021-171016-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

236

# Process and Thread (11/21)

- Thread (6/8)
  - Application area
    - Used to express high-level parallelism within a programming language
    - Used to handle slow I/O devices while other threads within a process remain unblocked
    - Used to perform less important operations in background
    - Used to support time-critical applications (RTA) by speeding up the response time for asynchronous events *via* signal handling

CMABA-CTR-200-5021-171016-01



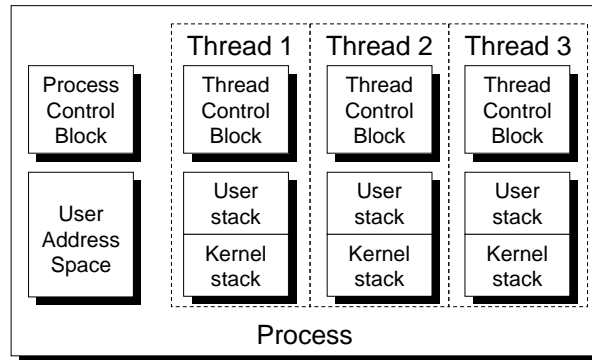
Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

237



# Process and Thread (12/21)

- Thread (7/8)
  - Multiple-threaded process model



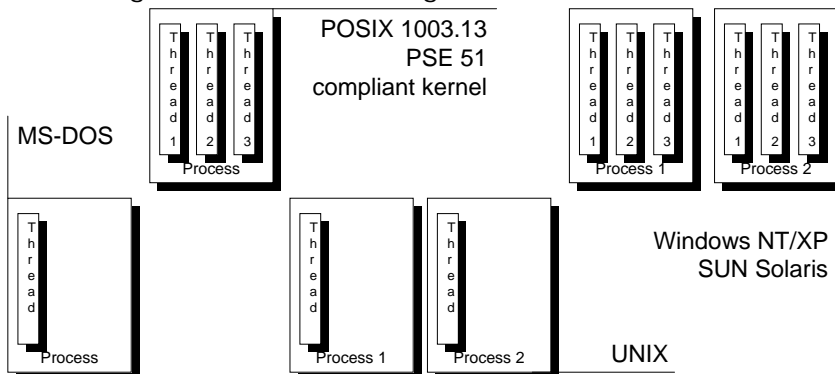
CMRBA-CTR-200-50201-171010-01

Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

238

# Process and Thread (13/21)

- Thread (8/8)
  - Depending of the OS, support to process or thread executions can be given or not, according to four combinations



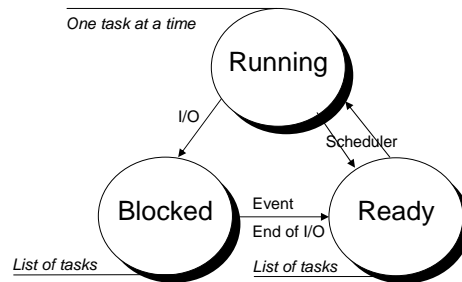
CMRBA-CTR-200-50201-171010-01

Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

239

# Process and Thread (14/21)

- Task states summary (1/6)
  - Processes and threads go through various task states which determine how the task is handled by the OS kernel



CMRBA-CTR-200-50201-171030-01

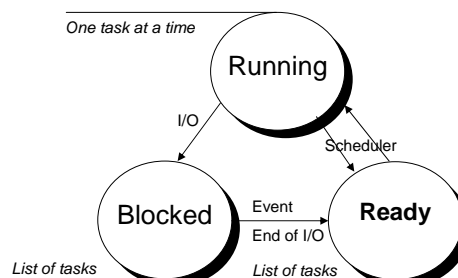
Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

240

# Process and Thread (15/21)

- Task states summary (2/6)

- Creation of a task
  - When a task is created (and loaded into memory), its state is set by the scheduler to READY



CMRBA-CTR-200-50201-171030-01

Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

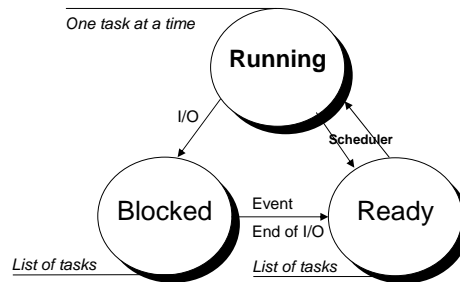
241

# Process and Thread (16/21)

- Task states summary (3/6)

- Activation of a task

- When a task, which have all the resources it needs (except the processor), is eventually assigned to the processor by the scheduler, its state is set to RUNNING (the processor is then executing the task's instructions)
    - A context switch is performed



CMRBA-CTR-200-50201-171030-01



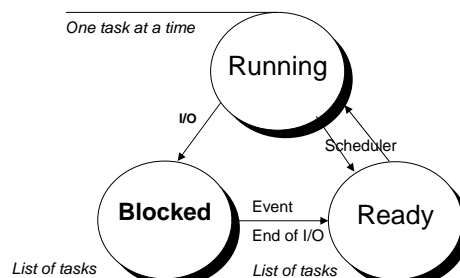
# Process and Thread (17/21)

- Task states summary (4/6)

- I/O operation

- When a task performs an I/O operation, the processor is given to another task until the I/O operation ends up. Its state is set to BLOCKED

- A context switch is performed



CMRBA-CTR-200-50201-171030-01

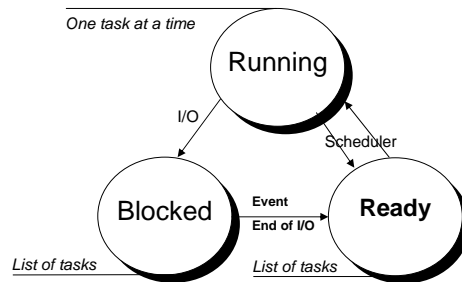


# Process and Thread (18/21)

- Task states summary (5/6)

- I/O operation (end)

- When a task has performed an I/O operation, the kernel sets the task state to READY

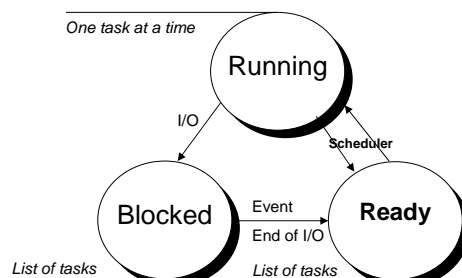


# Process and Thread (19/21)

- Task states summary (6/6)

- Pre-emption

- If the kernel is pre-emptive, it has the possibility to suspend the running task in order to run another one (typically the one with a higher priority level). The state of the running task is set to READY
    - A context switch is performed



# Process and Thread (20/21)

- Memory space summary (1/2)

- The memory space allocated to a process holds

- A stack
      - *I.e.*, local variables, function parameters,...
    - Static data
      - Data of whom the number and the size/type are known at compile time and which will never change at execution time
      - *I.e.*, global variables
    - Dynamic data
      - Data of whom the number or the size are unknown at compile time and which may change at execution time
      - *I.e.*, runtime allocations
    - The code (the instruction stream)
    - An execution context (PCB)

CMRBA-CTR-200-50201-171010-01

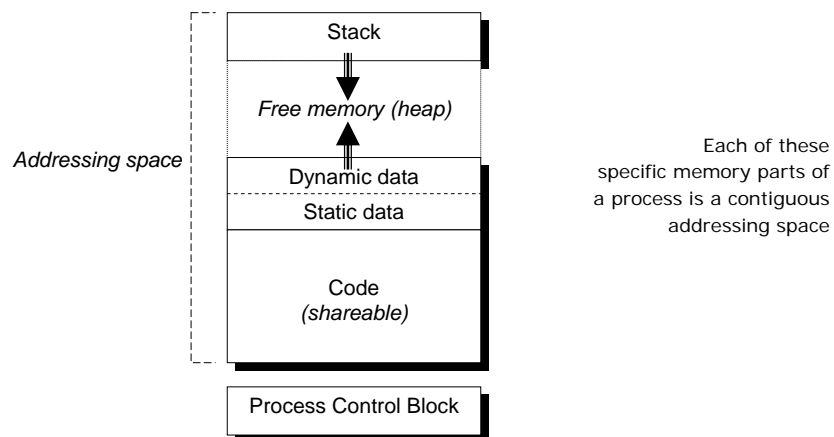


Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

246

# Process and Thread (21/21)

- Memory space summary (2/2)



CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

247

# Threads (1/2)

- Benefits of multithreading a code
  - Improve application responsiveness
    - Any program in which activities are not dependant upon each other can be redesigned so that each activity is defined as a thread. In this way each of them is not supposed to wait for others
  - Use multiprocessor architectures more simply
  - Improve programme structure
    - Many programs are more efficiently structured as multiple independent units of execution instead of as a single, monolithic thread. Multithreaded programs can be more adaptive to variations in user demands than single threaded programs
  - Use fewer system resources than using processes
    - Programs that use several processes that access common data through shared memory are applying more than one thread of control. However, each process has a full address space and operating system state. The cost of creating and maintaining this large amount of state information makes each process much more expensive than a thread in both time and space

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

248

# Threads (2/2)

- Implementation of threads can be carried out as
  - A kernel implementation
    - Where all functionality is part of the OS kernel
    - This type of implementation simplifies control over thread operations and signal handling but adds the overhead of entering and leaving the kernel at each call
  - A library implementation
    - Where all functionality is part of the user program and can be linked in
    - This type of implementation can be more efficient since it does not have to enter the OS kernel but it complicates signal handling and some thread operations
  - A mixture of the above

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

249

# Task States

- I/O operation (blocked task)
  - When a task performs an I/O operation, the processor is given to another task until the I/O operation end up
    - Its state is set to BLOCKED
  - In early systems, tasks would often “poll”, or “busy wait”, while waiting for requested input (such as disk or keyboard input). During this time, the task was not performing useful work but still maintained complete control of the CPU. With the advent of interrupts and pre-emptive multitasking, these I/O bound tasks could be “blocked”, or put on hold, pending the arrival of the necessary data, allowing other tasks to use the CPU. As the arrival of the requested data would generate an interrupt, blocked tasks could be guaranteed a timely return to execution

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

250

# Data Dependency

- Notion d'interblocage (7)
- The dependence relation at the loop level (2)

*Excerpt from Methodology, Tools and Techniques to Parallelize Large-Scale Applications: A Case Study, Kirkegaard K. J. et al., Intel Technology Journal, Vol. 11(04), p. 323–331, November 2007.*
- Thread safe application (6)

*Excerpt from Methodology, Tools and Techniques to Parallelize Large-Scale Applications: A Case Study, Kirkegaard K. J. et al., Intel Technology Journal, Vol. 11(04), p. 323–331, November 2007.*

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

251

## Interblocage (1/7)

- *Deadlock*
- Condition impliquant
  - Une ou plusieurs tâches
  - Et un ensemble d'une ou plusieurs ressources
- Condition dans laquelle
  - Chacune des tâches est bloquée en attente d'une des ressources
  - Et toutes les ressources sont détenues par les tâches
  - De telle sorte qu'aucune des tâches ne peut poursuivre des traitements

CMRBA-CTR-AD-5021-1710-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

252

## Interblocage (2/7)

→ Deux tâches, ou plus, tentent de verrouiller les mêmes sémaphores binaires et se mettent à attendre indéfiniment ceux qu'elles n'ont pu verrouiller les premières

(Si aucun délai d'attente (*timeout*) n'a été défini)

- Ceci crée un graphe de dépendance cyclique
- Ce scénario simple d'interblocage survient lorsque deux tâches verrouillent deux sémaphores l'un à la suite de l'autre, mais dans un ordre inverse

CMRBA-CTR-AD-5021-1710-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

253



## Interblocage (3/7)

- Exemple

- Deux threads sont en interblocage lorsque chacun tente de verrouiller un mutex qui est déjà verrouillé par l'autre

(l'introduction de threads et d'objets de synchronisation supplémentaires entraîne des configurations d'interblocage encore plus complexes)

## Interblocage (4/7)

- Conditions nécessaires et suffisantes

(Coffman E. G., 1971)

- Au nombre de 4

- Les ressources ne sont pas partageables
- Aucune préemption n'est autorisée d'une tâche ayant acquis une ressource
  - Elle doit relâcher volontairement et par elle-même la ressource
- Une tâche ayant déjà acquis une ressource est en mesure de demander à en acquérir une autre
- Une référence circulaire existe dans laquelle une tâche est en attente d'une ressource détenue par la suivante dans la chaîne, la dernière tâche attendant une ressource détenue par la première dans la chaîne (condition d'attente circulaire)

## Interblocage (5/7)

- Les deux principales voies de gestion
  - Prévention
    - Prendre des précautions lors de la conception de l'application
    - Appliquer des contraintes strictes d'usage des ressources partagées
  - Détection et correction à l'exécution
    - Des délais d'attente (*timeouts*) peuvent être définis lors des tentatives d'accès à des ressources partagées
    - Dans la plupart des cas, les tâches qui resteraient bloquées sont purement et simplement arrêtées

CM08-01-CTR-200-5021-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

256

## Interblocage (6/7)

- Quelques manières simples de réduire les risques d'interblocage (1/2)
  - Lier une ressource partageable virtuelle à une ressource réelle non partageable (par file d'attente par exemple)
  - Forcer les tâches à libérer toutes leurs ressources avant d'en demander une nouvelle
  - Restreindre l'allocation de ressources exclusivement à la phase de démarrage

CM08-01-CTR-200-5021-171010-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

257

# Interblocage (7/7)

- Quelques manières simples de réduire les risques d'interblocage (2/2)

- La condition d'attente circulaire peut être évitée en associant un niveau à chaque ressource et en interdisant à toute tâche qui détient déjà une ressource d'accéder à une autre qui aurait un niveau supérieur

- Dans ce cas, prendre soin d'associer les valeurs de niveau faible à des ressources très convoitées

→ Ces règles ne sont pas applicables dans tous les cas, et quand elles le sont, elles peuvent réduire considérablement l'efficacité de l'application

CMRBA-CTR-AD-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

258

# Loop Level (1/2)

- 3 categories

1. If an iteration of a loop writes to a memory location that is later read in another iteration of the loop, we say that the second iteration is flow-dependent on the first iteration
  - Equivalent to Read-After-Write (RAW)
2. If the first iteration reads from a location that is later modified in another iteration of the loop, we say that the second iteration is anti-dependent on the first iteration
  - Equivalent to Write-After-Read (WAR)
3. Two iterations of a loop are output-dependent on each other if both write to the same memory location
  - Equivalent to Write-After-Write (WAW)

CMRBA-CTR-AD-5021-1710-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

259

## Loop Level (2/2)

- A loop that contains no dependence relations can be parallelized
  - On the other hand, parallelizing a loop that contains any of the 3 possible dependence relations may cause invalid results
    - However, it can be shown that if a loop contains only anti-dependence and output-dependence relations, it can be parallelized with the proper code change
- Data dependence relations are often called hazards or data races

CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

260

## Thread Safe App. (1/6)

- For each category of global data, a method for making the data thread safe could be applied
- Global data with dependence relations
  - First, try to rewrite the code to eliminate the data dependence
  - If it is not possible, apply locks to synchronize the access to the global data
- Global data with restricted scope
  - This category consists of global data that could have been declared as constant or as stack variables
  - If it is possible to rewrite global data to be constant or as stack variables they become thread safe automatically

CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

261

## Thread Safe App. (2/6)

- Global data with data dependence relations can be useful categorized into 4 sub-categories
  - Synchronized
  - Mutable
  - Persistent
  - Transient

CMRBA-CTR-APP-50201-T10R001



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

262

## Thread Safe App. (3/6)

- Synchronized category
  - Includes the global data that require locks for controlled synchronized accesses
  - It is not possible to privatize such data without extensive changes
  - Ex.: I/O operation, heap allocation management

CMRBA-CTR-APP-50201-T10R001



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

263

## Thread Safe App. (4/6)

- Mutable category
  - Contains the global data that generally are defined before a parallelized loop, but they may be modified by an iteration of the loop
    - Only to be reset to the original value before the next iteration
  - They are privatized by creating a thread-private copy of the data for each of the iterations
    - This has the additional advantage that there is no longer a need to restore values for the next loop iteration, if data were modified
    - Furthermore, this helps improve maintainability of the code by eliminating the code necessary to restore values

CMRBA-CTR-APP-50201-171016-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

264

## Thread Safe App. (5/6)

- Persistent category
  - Comprises the global data that are defined and used in each thread but do not have any cross iteration dependence relations
  - The lifetime of such a global data spans the entire thread
    - They are allocated and initialized after thread creation and freed before thread termination

CMRBA-CTR-APP-50201-171016-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

265

## Thread Safe App. (6/6)

- Transient category
  - Consists of the global data that are defined and used only within a certain phase of the threaded region
    - They are allocated on entry to a phase and freed on exit from that phase
    - In general, the transient state is allocated on the stack and is assigned and accessed through a thread-private pointer

CMRBA-CTR-APP-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

266

## Multithreaded App. Design

- 8 Simple Rules for Designing Multithreaded Applications (9)

*Excerpts from The Art of Concurrency, Breshears C., Chapter 4, O'Reilly Media Inc., p. 73–80, May 2009.*

CMRBA-CTR-APP-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

267

## 8 Simple Rules (1/9)

- Rule 1  
Identify truly independent computations
  - You can't execute anything concurrently unless the operations that would be executed can be run independently of each other

CMRBA-CTR-200-5021-T10R01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

268

## 8 Simple Rules (2/9)

- Rule 2  
Implement concurrency at the highest level possible
  - Placing concurrency at the highest possible level around a hotspot is one of the best ways to achieve that all-important coarse-grained division of work to be assigned to threads

CMRBA-CTR-200-5021-T10R01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

269



## 8 Simple Rules (3/9)

- Rule 3 (1/2)

Plan early for scalability to take advantage of increasing numbers of cores

- Scalability is the measure of an application's ability to handle changes, typically increases, in system resources (e.g., number of cores, memory size, bus speed) or data set sizes
  - In the face of more cores being available, you must write flexible code that can take advantage of different numbers of cores

CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

270

## 8 Simple Rules (4/9)

- Rule 3 (2/2)

Plan early for scalability to take advantage of increasing numbers of cores

- Designing and implementing concurrency by data decomposition methods will give you more scalable solutions
  - Task decomposition solutions will suffer from the fact that the number of independent functions or code segments in an application is likely limited and fixed during execution
  - After each independent task has a thread and core to execute on, increasing the number of threads to take advantage of more cores will not increase performance of the application
  - Since data sizes are more likely to increase than the number of independent computations in an application, data decomposition designs will have the best chance for scalability

CMRBA-CTR-200-50201-171000-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

271

## 8 Simple Rules (5/9)

- Rule 4

Make use of thread-safe libraries wherever possible

- If your hotspot computations can be executed through a library call, you should strongly consider using an equivalent library function instead of executing handwritten code
  - Even for serial applications, it's never a good idea to “reinvent the wheel” by writing code that performs calculations already encapsulated by optimized library routines
- Check the library documentation for the thread-safety of any library you are using within concurrent execution
- Be sure the routines are reentrant
  - If this is not possible, you will need to add synchronization in order to protect access to shared resources

CMRBA-CTR-200-55-01-1710-0-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

272

## 8 Simple Rules (6/9)

- Rule 5

Use the right threading model

- If threaded libraries are insufficient to cover all the concurrency of an application and you must employ user-controlled threads, don't use explicit threads if an implicit threading model (e.g., OpenMP, Intel Threading Building Blocks) has all the functionality you need
  - Explicit threads do allow for finer control of the threading implementation
  - However, they imply much more work and the more complex the implementation, the easier it will be to make a mistake and the harder it will be to maintain such code later

CMRBA-CTR-200-55-01-1710-0-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

273

## 8 Simple Rules (7/9)

- Rule 6  
Never assume a particular order of execution
  - Execution order of threads is nondeterministic
    - Controlled by the OS scheduler
    - There is no reliable way of predicting the order of threads running from one execution to another, or even which thread will be scheduled to run next
  - Data races are a direct result of this scheduling nondeterminism
    - Code that relies on a particular order of execution among threads that is enforced through nothing more than positive thinking may be plagued by problems such as data races and deadlock

CMRBA-CTR-200-5021-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

274

## 8 Simple Rules (8/9)

- Rule 7  
Use thread-local storage whenever possible or associate locks to specific data
  - You should actively seek to keep the amount of synchronization to a minimum
    - Synchronization is overhead that does not contribute to the furtherance of the computation, except to guarantee the correct answers are produced from the parallel execution of an application
    - Synchronization is a necessary evil

CMRBA-CTR-200-5021-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

275

## 8 Simple Rules (9/9)

- Rule 8  
Dare to change the algorithm for a better chance of concurrency
  - If you cannot easily turn a hotspot into threaded code, you should consider using a suboptimal serial algorithm to transform
    - Rather than the algorithm currently in the code

CMRBA-CTR-200-50201-1710001



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

276

## Conception (1/7)

- Lors de la conception d'algorithmes parallèles (1/5)
  - Une grande attention doit être portée à
    - Obtenir un haut niveau de concurrence
    - Tendre vers une évolutivité linéaire
      - Si le nombre de tâches évolue avec la taille du problème, l'algorithme que vous concevez doit être capable de résoudre des problèmes plus grands avec plus de processeurs/cœurs/nœuds
    - Privilégier la localité des données
      - Les algorithmes doivent traiter en priorité des données présentes dans la mémoire localement attachée au processeur sur lequel ils sont exécutés

CMRBA-CTR-200-50201-1710001



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

277

## Conception (2/7)

- Lors de la conception d'algorithmes parallèles (2/5)
    - Maximiser l'usage des processeurs
      - Ils doivent être chargés au maximum de leurs capacités
    - Minimiser les communications
    - Les coûts de communication influencent très fortement les performances parallèles
      - En raison du temps nécessaire pour attendre l'arrivée d'une donnée
      - En raison du fait que, sur certaines machines, il est nécessaire d'interrompre les calculs afin d'envoyer ou de recevoir des données
        - Utiliser moins de messages pour la même quantité de données (réduction du coût fixe d'amorçage d'une communication)
        - Transmettre moins de données (augmentation de la « granularité »)
- Le surcoût dû aux communications (*communication overhead*) peut-être réduit, dans certains cas, en dupliquant données comme calculs

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

278

## Conception (3/7)

- Lors de la conception d'algorithmes parallèles (3/5)
  - Décomposition (1/3)
    - Etape (importante, essentielle) de découpage d'un programme en tâches concurrentes individuelles et d'identification des interdépendances entre elles
    - Pour un même problème, on peut, en général, suivre des approches de décomposition différentes
      - Décomposition par tâche (*task parallelism*)
      - Partition des données à traiter (*data parallelism*)
      - Suivant le flot de données (*data-flow*)  
(on se préoccupe dans ce cas de la manière dont les données circulent entre les tâches → les aspects de synchronisation et de communication sont ici mis en lumière)

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

279

## Conception (4/7)

- Lors de la conception d'algorithmes parallèles (4/5)

- Décomposition (2/3)

- Différentes décompositions offrent différents bénéfices
      - Décomposition en tâches : facilité de programmation
        - » Mais difficilement évolutif puisque le nombre de tâches nécessaire (voire optimal) est, dans la plupart des cas, indépendant des ressources disponibles
      - Partition des données : très bonnes capacités d'évolution en fonction des ressources, prise en compte des questions de performance
      - Flot de données : les dépendances de données (*data precedence*) sont clairement mises en lumière

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

280

## Conception (5/7)

- Lors de la conception d'algorithmes parallèles (5/5)

- Décomposition (3/3)

- Décomposer les calculs et les données en petites unités
      - Une bonne partition permet de regrouper les calculs avec les données qu'ils ont à traiter : principe de localité (*locality principle*)
      - Positionner les tâches concurrentes sur des nœuds différents (processeurs/cœurs)
        - » Par mesure d'efficacité
      - Positionner les tâches effectuant des communications volumineuses dans un voisinage proche (dépend de la topologie du réseau d'interconnexion)
        - » Ou sur le même nœud (évaluer le ratio communications/calculs pour prendre la décision)

CMRBA-CTR-200-50201-171010-01



Rémi Coudarcher | Calcul parallèle: Techniques de base | octobre 17

281

# Conception (6/7)

## • Problème de la localisation (1/2)

(algorithmie répartie)

- La problématique de la localisation est une propriété des systèmes répartis
  - *i.e.*, chaque nœud de traitement est directement relié (en point-à-point) à seulement un nombre fixe et limité d'autres (rarement la totalité des autres nœuds)
  - Par principe de fonctionnement des processeurs actuels, les calculs sont forcément réalisés sur des données localement présentes dans la mémoire attachée au processeur les exécutant
  - Émettre des messages à destination de nœuds de calcul éloignés est coûteux en temps
- Les traitements doivent malgré tout être conçus de manière à ce que des résultats partiels obtenus sur des nœuds différents concourent à produire un résultat d'ensemble cohérent qui réponde au problème
  - Cela notamment en dépit d'une topologie particulière limitant les interconnexions

CM06-CTR-200-5021-1710-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

282

# Conception (7/7)

## • Problème de la localisation (2/2)

(algorithmie répartie)

- Corollaire
  - Les calculs doivent être prévus pour manipuler le plus possible des données locales
  - Les problèmes de « symétrie » des nœuds de calcul doivent être éliminés (*symmetry-braking*)
    - La plupart des fonctions ne peuvent être calculées de façon répartie par des processeurs « anonymes »
    - Elle implique, par exemple, l'utilisation d'identifiants pour chaque processeur

CM06-CTR-200-5021-1710-01



Rémi Coudarcher | Calcul parallèle. Techniques de base | octobre 17

283

# Pour contacter l'auteur

M. Rémi Coudarcher

Email : Remi.Coudarcher@enac.fr  
Tél. : 05.62.17.41.54  
Bureau : C123  
Département : SINA/INF/SAR

Adresse postale  
Ecole nationale de l'aviation civile  
7, avenue Edouard-Belin - C.S. 54005  
31055 TOULOUSE cedex 4  
FRANCE  
[www.enac.fr](http://www.enac.fr)

ENAC

Rémi Coudarcher | Calcul parallèle, Techniques de base | octobre 17

284

Email : Remi.Coudarcher@enac.fr  
Tél. : 05.62.17.41.54  
Bureau : C123  
Département : SINA/INF/SAR

Ecole nationale de l'aviation civile  
7, avenue Edouard-Belin - C.S. 54005  
31055 TOULOUSE cedex 4  
FRANCE  
[www.enac.fr](http://www.enac.fr)





