16. Approche PCAM de Foster «*Designing Parallel Algorithms*»

INF7235 Programmation parallèle haute performance

Hiver 2017

Aperçu

16.1 Introduction

16.2 Partitioning

16.3 Communication

16.4 Agglomeration

16.5 Mapping

Aperçu

16.1 Introduction

16.2 Partitioning

16.3 Communication

16.4 Agglomeration

16.5 Mapping

Objectifs (en quelques mots)

Objectifs des stratégies de programmation parallèle :

- Minimiser les coûts de synchronisation ou communication
- Distribuer le travail de façon uniforme entre les processeurs

L'approche PCAM de Foster I

- Approche proposée par Ian T. Foster dans son livre «*Designing and building parallel programs*» (1995)
- Approche reprise par la suite dans de nombreux bouquins sur la programmation parallèle
- Livre maintenant disponible en ligne :

  `http://www-unix.mcs.anl.gov/dbpp/text/book.html`
- Ce qui suit est un résumé des principaux éléments du chapitre «*2. Designing Parallel Algorithms*»

L'approche PCAM de Foster II

- Approche conçue principalement pour la programmation par échange de messages, mais de nombreuses idées restent valables pour la programmation avec variables partagées

- Décompose le processus de conception en quatre (4) phases (PCAM), les «détails» liés à la machine étant traités tardivement

Les quatre (4) phases de l'approche PCAM

- *P artitioning* : Les traitements et les données sont décomposées en petites tâches
- *C ommunication* : Les communications requises pour coordonner l'exécution des tâches sont analysées et des structures et algorithmes de communication appropriés sont définis
- *A gglomeration* : Les tâches et communications sont évaluées au niveau performances et coûts et, si nécessaire, les tâches sont combinées en tâches plus grosses
- *M apping* : Les tâches sont associées aux processeurs

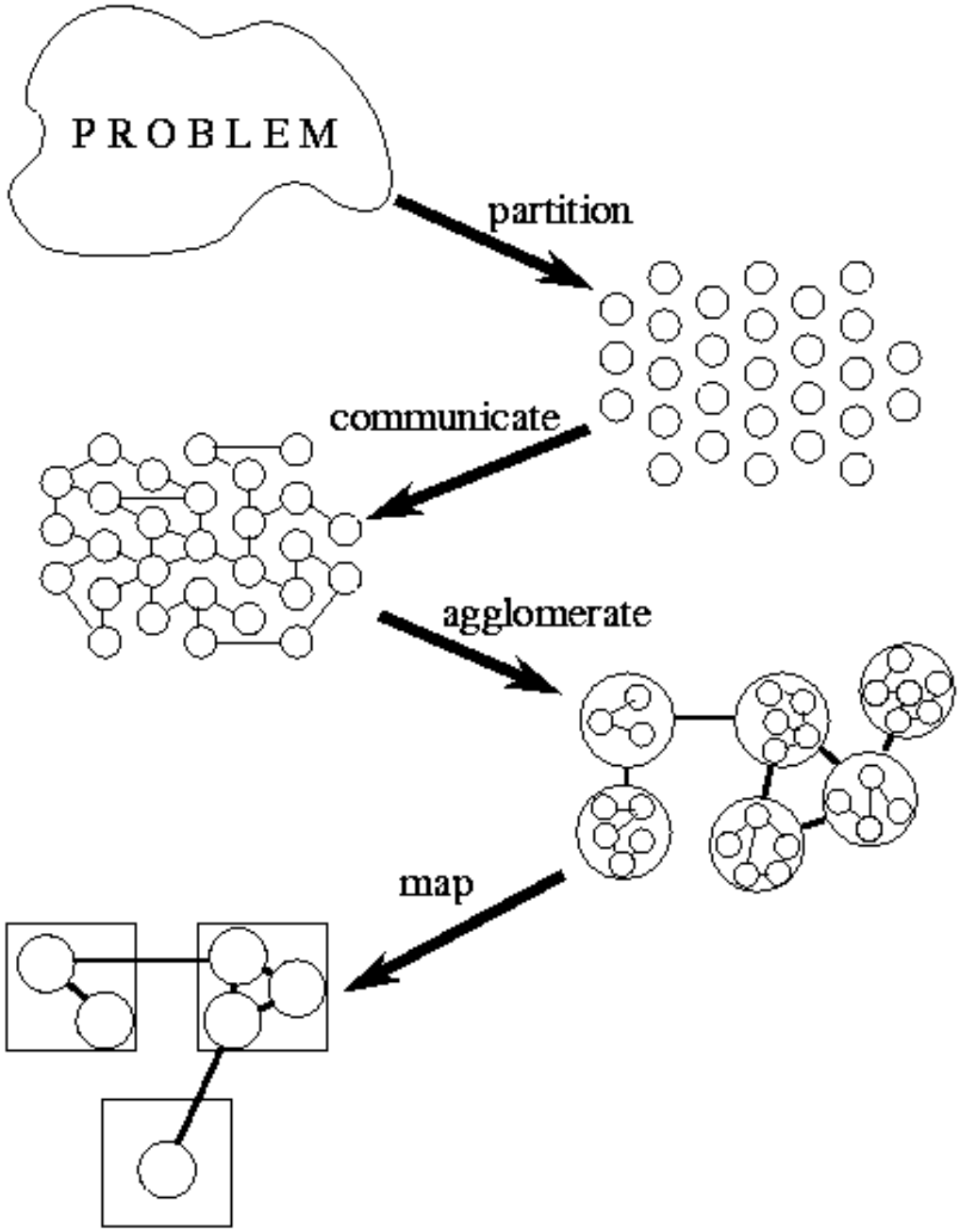


Figure 2.1: *PCAM: a design methodology for parallel programs. Starting with a problem specification, we develop a partition, determine communication requirements, agglomerate tasks, and finally map tasks to processors.*

Aperçu

16.1 Introduction

16.2 Partitioning

16.3 Communication

16.4 Agglomeration

16.5 Mapping

Partitioning

- Objectif = exposer le plus possible les opportunités d'exécution parallèle

$\Rightarrow$ Granularité (très!) fine

- Une bonne partition doit décomposer à la fois...
  - Les calculs qui doivent être effectués
  - Les données qui doivent être traitées par les calculs

Domain vs. Function decomposition

Deux principales approches :

- *Domain decomposition* : On analyse et décompose tout d'abord les données (ou résultats) puis on détermine comment associer les traitements à ces données

- *Functional decomposition* : On décompose tout d'abord les traitements puis on détermine comment associer les données

Note : Il s'agit de techniques complémentaires, pouvant être utilisées pour différentes parties d'un problème

Domain decomposition

Les «données» qui sont décomposées peuvent être...

- Les entrées du programme
- Les sorties du programme (parallélisme de résultat)
- Des données intermédiaires produites par le programme

Heuristique

On met l'accent sur les plus grosses structures de données et/ou celles qui sont accédées le plus fréquemment

Exemple de décomposition du domaine pour une matrice 3D

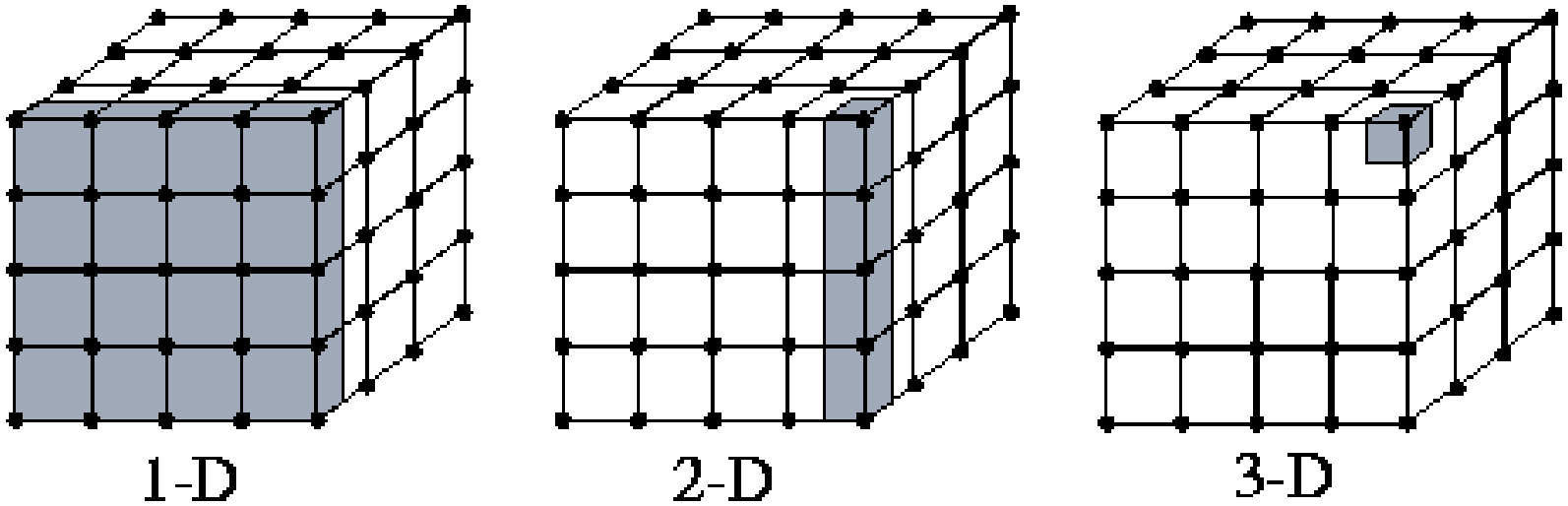


Figure 2.2: *Domain decompositions for a problem involving a three-dimensional grid. One-, two-, and three-dimensional decompositions are possible; in each case, data associated with a single task are shaded. A three-dimensional decomposition offers the greatest flexibility and is adopted in the early stages of a design.*

Functional decomposition

- On décompose les traitements en différentes tâches, les plus indépendantes possible les unes des autres (parallélisme de spécialistes)
- Si elles sont totalement indépendantes, alors la décomposition est terminée
- Si elles sont fortement dépendantes à cause de données qu'elles doivent partager, alors *domain décomposition* possiblement préférable

Exemple de décomposition fonctionnelle

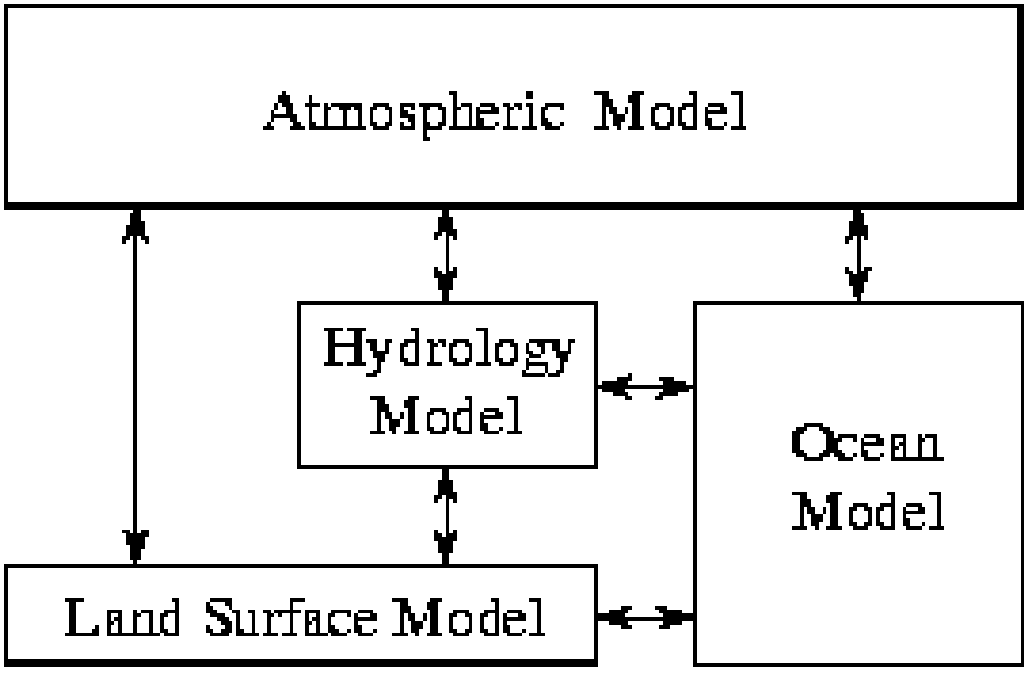


Figure 2.3: *Functional decomposition in a computer model of climate. Each model component can be thought of as a separate task, to be parallelized by domain decomposition. Arrows represent exchanges of data between components during computation: the atmosphere model generates wind velocity data that are used by the ocean model, the ocean model generates sea surface temperature data that are used by the atmosphere model, and so on.*

Liste de vérifications pour le partitionnement

- Est-ce que le nombre de tâches est (au moins un ordre de grandeur) plus grand que le nombre de processeurs?
- Est-ce que la décomposition évite les calculs et/ou structures de données redondants?
- Est-ce que les tâches sont de taille comparable?
- Est-ce que le nombre de tâches augmente lorsqu'on augmente la taille du problème?

Liste de vérifications pour le partitionnement

- Est-ce que le nombre de tâches est (au moins un ordre de grandeur) plus grand que le nombre de processeurs?
- Est-ce que la décomposition évite les calculs et/ou structures de données redondants?
- Est-ce que les tâches sont de taille comparable?
- Est-ce que le nombre de tâches augmente lorsqu'on augmente la taille du problème?

Aperçu

16.1 Introduction

16.2 Partitioning

16.3 Communication

16.4 Agglomeration

16.5 Mapping

Communication I

Remarques préliminaires :

- L'approche de Foster s'applique plus particulièrement au modèle de programmation parallèle par échanges de messages

- Le modèle de communication utilisé par Foster est celui des tâches et canaux de communication :
  - Tâche = un programme, sa «mémoire locale» et une collection de ports de commnunication
  - Canal = Relie des ports entre eux et peut contenir une file de messages

Communication II

Une fois les diverses tâches identifiées, il faut ensuite déterminer les principaux patrons de communication entre ces tâches :

- Locale vs. globale
- Structurée vs. non-structurée
- Statique vs. dynamique
- Synchrone vs. asynchrone

Communication locale vs. globale

- Opération locale de communication $\Rightarrow$ une tâche communique avec un nombre limité d'autres tâches
- Opération globale de communication $\Rightarrow$ un grand nombre de tâches doivent participer

---

Communication locale vs. globale

- Opération locale de communication $\Rightarrow$ une tâche communique avec un nombre limité d'autres tâches
- Opération globale de communication $\Rightarrow$ un grand nombre de tâches doivent participer

---

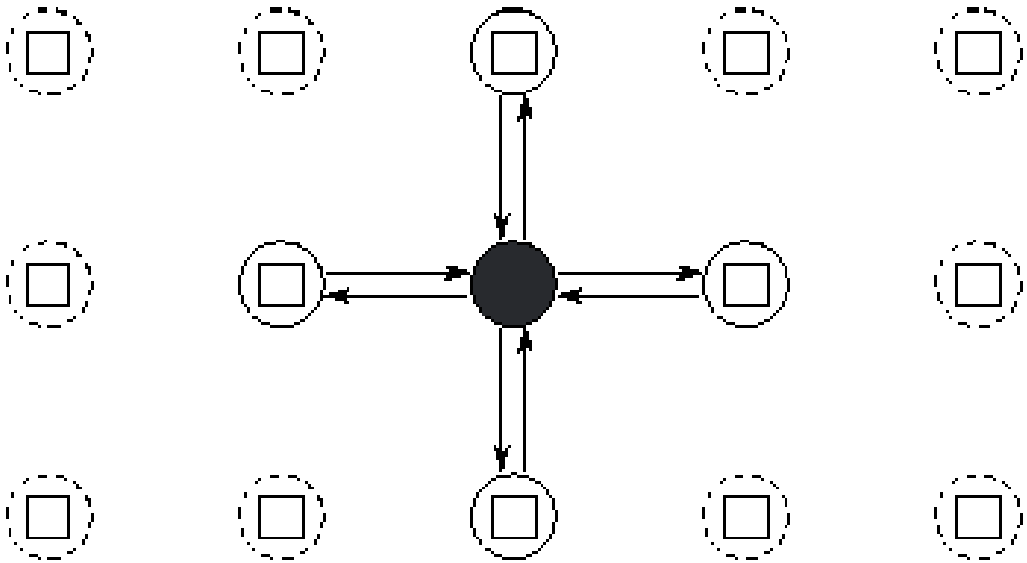


Figure 2.4: *Task and channel structure for a two-dimensional finite difference computation with five-point update stencil. In this simple fine-grained formulation, each task encapsulates a single element of a two-dimensional grid and must both send its value to four neighbors and receive values from four neighbors. Only the channels used by the shaded task are shown.*

Communication locale vs. globale

- Opération locale de communication $\Rightarrow$ une tâche communique avec un nombre limité d'autres tâches
- Opération globale de communication $\Rightarrow$ un grand nombre de tâches doivent participer

---

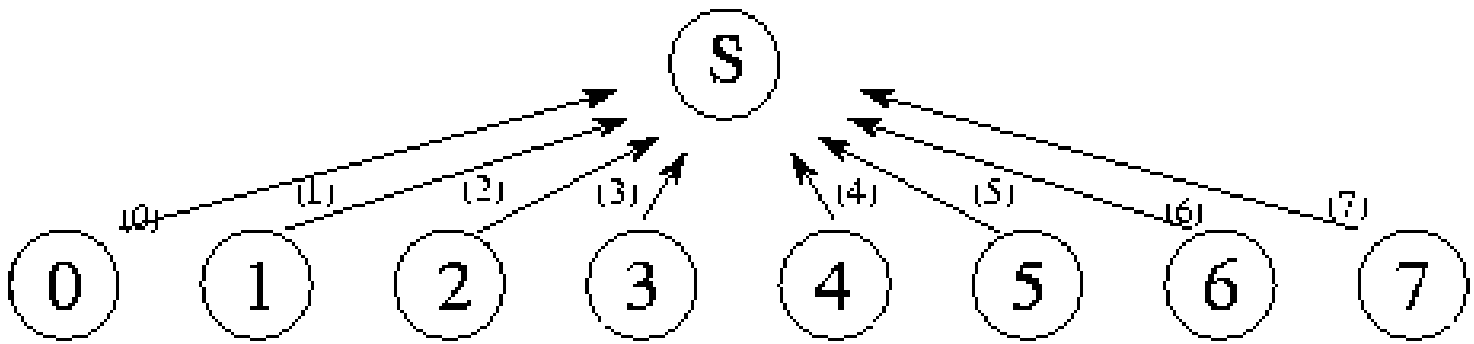


Figure 2.6: *A centralized summation algorithm that uses a central manager task (S) to sum N numbers distributed among N tasks. Here, N=8 , and each of the 8 channels is labeled with the number of the step in which they are used.*

Structurée vs. non-structurée

- Structurée $\Rightarrow$ Les partenaires de communication forment un patron régulier (e.g., grille)
- Non-structurée $\Rightarrow$ Organisation complexe et irrégulière

Structurée vs. non-structurée

- Structurée ⇒ Les partenaires de communication forment un patron régulier (e.g., grille)
- Non-structurée ⇒ Organisation complexe et irrégulière

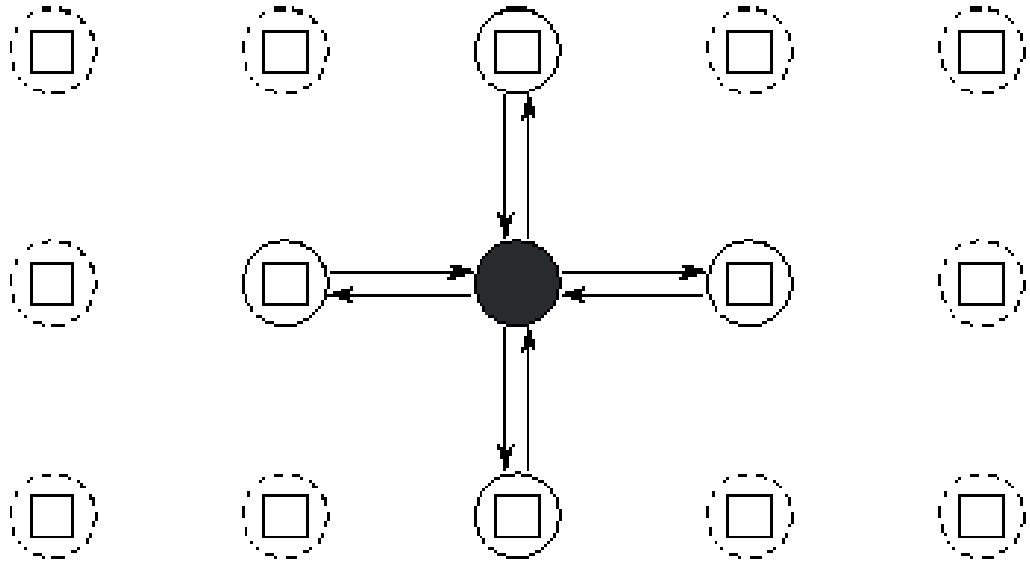


Figure 2.4: *Task and channel structure for a two-dimensional finite difference computation with five-point update stencil. In this simple fine-grained formulation, each task encapsulates a single element of a two-dimensional grid and must both send its value to four neighbors and receive values from four neighbors. Only the channels used by the shaded task are shown.*

Structurée vs. non-structurée

- Structurée $\Rightarrow$ Les partenaires de communication forment un patron régulier (e.g., grille)
- Non-structurée $\Rightarrow$ Organisation complexe et irrégulière

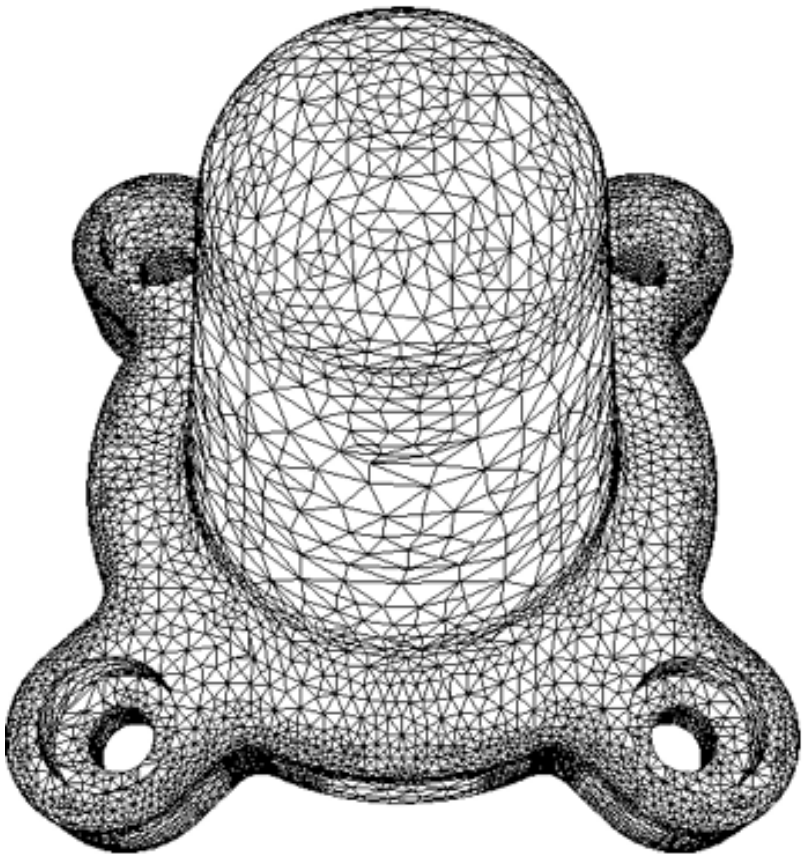


Figure 2.9: *Example of a problem requiring unstructured communication. In this finite element mesh generated for an assembly part, each vertex is a grid point. An edge connecting two vertices represents a data dependency that will require communication if the vertices are located in different tasks. Notice that different vertices have varying numbers of neighbors. (Image courtesy of M. S. Shephard.)*

Statique vs. dynamique

- Statique $\Rightarrow$ Les partenaires de communication sont fixés dès le départ
- Dynamique $\Rightarrow$ Les partenaires de communication changent en cours d'exécution

Synchrone vs. asynchrone

- Synchrone = L'envoyeur et le récepteur doivent s'attendre mutuellement pour que la communication ait lieu
  $\Rightarrow$ Envoi bloquant + Réception bloquante
- Asynchrone = L'envoyeur et le récepteur n'ont pas besoin de s'attendre
  $\Rightarrow$ Envoi non-bloquant + Réception bloquante

Liste de vérifications pour les communications

- Est-ce que toutes les tâches exécutent sensiblement le même nombre de communications?
- Est-ce que chaque tâche ne communique qu'avec un nombre limité d'autres tâches?
- Est-ce que les communications peuvent s'exécuter de façon concurrente?
- Est-ce que les traitements peuvent se faire de façon concurrente?

Aperçu

16.1 Introduction

16.2 Partitioning

16.3 Communication

16.4 Agglomeration

16.5 Mapping

Agglomeration I

- Les deux premières étapes travaillaient à un niveau relativement abstrait
- Les deux étapes subséquentes travaillent à un niveau plus concret

- Objectif = revoir ce qui a été fait dans le but d'obtenir un programme qui s'exécutera de façon efficace sur une machine (ou une *classe* de machines)

- Questions à examiner :
  - Est-il utile de combiner des petites tâches pour réduire le nombre de tâches, augmenter leur granularité et réduire les communications?
  - Est-il utile de dupliquer certaines données (ou certains traitements) pour réduire les communications?

Agglomeration II

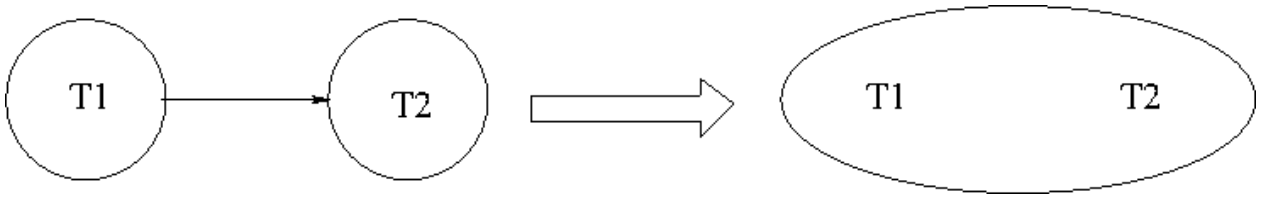
Quelques possibilités :

- On réduit le nombre de tâches, mais il est encore plus grand que le nombre prévu de processeurs
  $\Rightarrow$ On a du travail à faire à l'étape suivante (*mapping*)

- On travaille avec un modèle SPMD (style MPI)
  $\Rightarrow$ On s'assure que le nombre de tâches est égal au nombre de processeurs
  $\Rightarrow$ Rien à faire à l'étape suivante

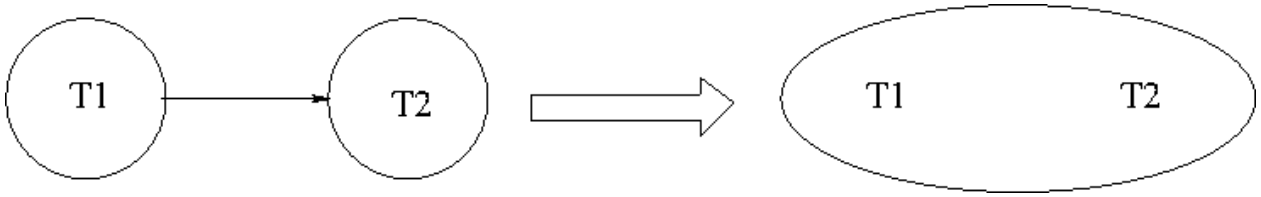
Exemples d'agglomération

On agglomère deux tâches dans un même processus, ce qui élimine complètement une communication

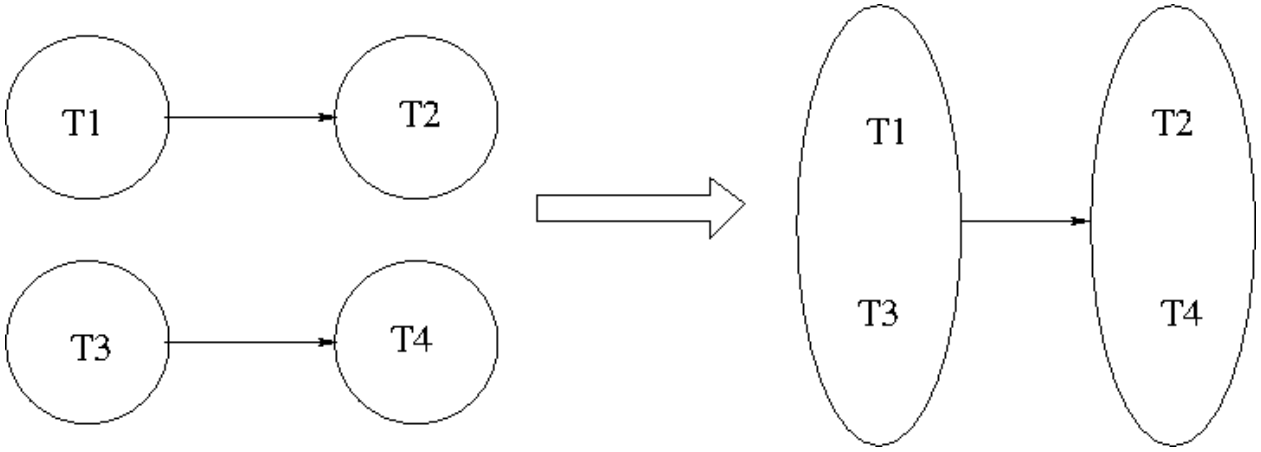


Exemples d'agglomération

On agglomère deux tâches dans un même processus, ce qui élimine complètement une communication



On agglomère partiellement des tâches, ce qui réduit le nombre de messages — bien qu'un message sera (un peu) «plus gros»



Autres exemples

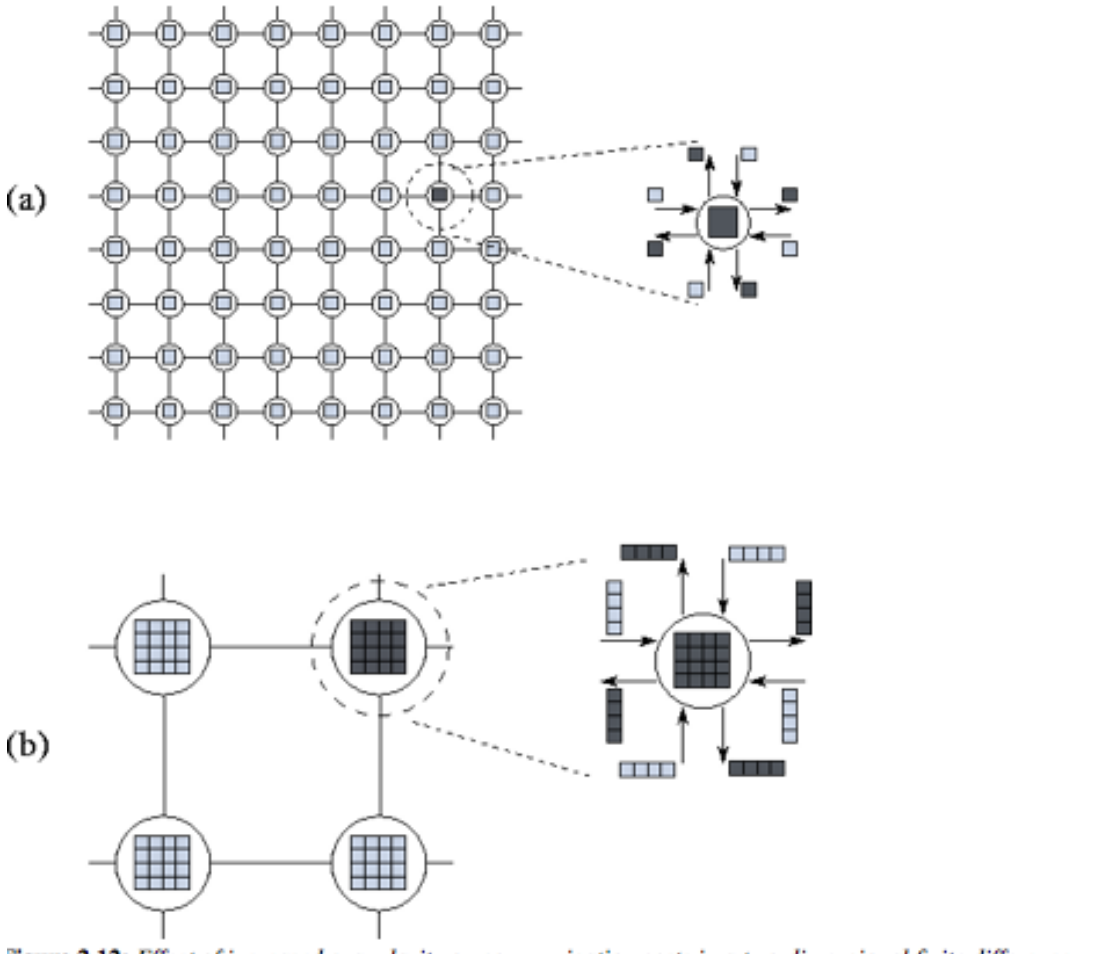


Figure 2.12: *Effect of increased granularity on communication costs in a two-dimensional finite difference problem with a five-point stencil. The figure shows fine- and coarse-grained two-dimensional partitions of this problem. In each case, a single task is exploded to show its outgoing messages (dark shading) and incoming messages (light shading). In (a), a computation on an* 8×8 *grid is partitioned into* $8 \times 8 = 64$ *tasks, each responsible for a single point, while in (b) the same computation is partioned into* $2 \times 2 = 4$ *tasks, each responsible for 16 points. In (a),* $64 \times 4 = 256$ *communications are required, 4 per task; these transfer a total of 256 data values. In (b), only* $4 \times 4 = 16$ *communications are required, and only* $16 \times 4 = 64$ *data values are transferred.*

Les trois objectifs clés

Objectifs, possiblement contradictoires, de l'étape d'agglomération :

- Réduire les coûts des communications
- Conserver une certaine flexibilité quant à la capacité de dimensionnement (*scalability*)
- Préserver la qualité du logiciel (entre autres, modifiabilité)

Les trois objectifs clés

Objectifs, possiblement contradictoires, de l'étape d'agglomération :

- Réduire les coûts des communications
- Conserver une certaine flexibilité quant à la capacité de dimensionnement (*scalability*)
- Préserver la qualité du logiciel (entre autres, modifiabilité)

Les trois objectifs clés

Objectifs, possiblement contradictoires, de l'étape d'agglomération :

- Réduire les coûts des communications
- Conserver une certaine flexibilité quant à la capacité de dimensionnement (*scalability*)
- Préserver la qualité du logiciel (entre autres, modifiabilité)

Liste de vérifications pour l'agglomération

- Est-ce que les coûts de communication ont été réduits?
- Si des calculs ont été dupliqués, est-ce que ça en vaut le coup (le coût!) — pour différentes machines ou tailles de problème?
- Si des données ont été dupliquées, est-ce que ça compromet la capacité de dimensionnement du programme?
- Est-ce que les tâches résultantes sont relativement similaires (calculs et communications)?
- Si des possibilités d'exécution concurrente ont été éliminées, est-ce qu'il en reste suffisamment?

Liste de vérifications pour l'agglomération

- Est-ce que les coûts de communication ont été réduits?
- Si des calculs ont été dupliqués, est-ce que ça en vaut le coup (le coût!) — pour différentes machines ou tailles de problème?
- Si des données ont été dupliquées, est-ce que ça compromet la capacité de dimensionnement du programme?
- Est-ce que les tâches résultantes sont relativement similaires (calculs et communications)?
- Si des possibilités d'exécution concurrente ont été éliminées, est-ce qu'il en reste suffisamment?

Liste de vérifications pour l'agglomération

- Est-ce que les coûts de communication ont été réduits?
- Si des calculs ont été dupliqués, est-ce que ça en vaut le coup (le coût!) — pour différentes machines ou tailles de problème?
- Si des données ont été dupliquées, est-ce que ça compromet la capacité de dimensionnement du programme?
- Est-ce que les tâches résultantes sont relativement similaires (calculs et communications)?
- Si des possibilités d'exécution concurrente ont été éliminées, est-ce qu'il en reste suffisamment?

Point clé du dimensionnement (*scalability*) : Le nombre de tâches devrait augmenter si on augmente la taille du problème

Aperçu

16.1 Introduction

16.2 Partitioning

16.3 Communication

16.4 Agglomeration

16.5 Mapping

Mapping

«Formellement» = Déterminer sur quel processeur chaque tâche va s'exécuter

Pas vraiment un «problème»...

- sur les multi-processeurs (mémoire partagée) où la charge est répartie par le système d'exploitation
- sur les multi-ordinateurs (mémoire distribuée) programmé avec un modèle SPMD

Mapping

«Formellement» = Déterminer sur quel processeur chaque tâche va s'exécuter

Pas vraiment un «problème»...

- sur les multi-processeurs (mémoire partagée) où la charge est répartie par le système d'exploitation (ex., Java)
- sur les multi-ordinateurs (mémoire distribuée) programmé avec un modèle SPMD (ex., MPI)

Mapping

«Formellement» = Déterminer sur quel processeur chaque tâche va s'exécuter

Pas vraiment un «problème»...

- sur les multi-processeurs (mémoire partagée) où la charge est répartie par le système d'exploitation (ex., Java)
- sur les multi-ordinateurs (mémoire distribuée) programmé avec un modèle SPMD (ex., MPI)

Mais :

- Le problème de l'équilibrage de la charge (*load balancing*) entre les processeurs est quand même important

$\Rightarrow$ On le traite (par ex., Java, Ruby) en déterminant comment associer les tâches aux *threads* qu'on va créer :

- Association statique par bloc? statique cyclique?
- Association dynamique avec sac de tâches?