Université Pierre et Marie Curie

Note de Cours

HPC

Calcul Haute Performance

Auteur:
Benjamin Bielle

Responsable:
Pierre Fortin

June 23, 2015

Sommaire

1 Cours 1: MPI, une bibliothèque de communication par échange de messages **2**

2 Cours 2: Introduction au parallélisme **3**

3 Cours 7: Introduction au calcul haute performance **8**

4 Cours 8: Parallèlisation automatique, application au parallélisme de contrôle **10**

5 Bonus : Annale **11**

Exercice 2 11

Chapter 1

Cours 1: MPI, une bibliothèque de communication par échange de messages

- Blocs de types avec saut → MPI_Type_Vector.
  Ex : décalage pour envoyer une colonne d'une matrice.
- structure → MPI_Type_Struct.
- zone de données contigues → MPI_Type_Contiguous.

Hétérogénéité

- Tailles différentes des types de données.
- Endianness (little ou big endian).
- Structures → alignement mémoire.

Chapter 2

Cours 2: Introduction au parallélisme

Giga : taille mémoire et puissance de calcul d'un pc personnel.

Exemple:

Un processus à 2GHz, disposant de 2 coeurs, pouvant effectuer 1 FMA (Fused Multiply-Add) ($d = c + a * b$) par cycle; a une performance crête de : $2 * 2 * 2 * 10^9 = 8GFlop/s$.

- Tera : centre de calcul locaux.
- Peta : centre de calcul nationnaux.
- Exa : $\approx$ 2020.

Race Condition

$i = i + 1$ 3 instructions au niveau assembleur.

$$\left.\begin{array}{c}\text{load i}\\ \text{add i 1}\\ \text{store i}\end{array}\right\} \rightarrow \text{pas atomique.}$$

CM: Connection Machine.

Exemple:

Une entreprise devant effectuer l'entretien hebdomadaire des jardins d'une propriété.

1. Éteindre l'alarme.
2. Tondre les pelouses.
3. Tailler les bordures.
4. Désherber les massifs.
5. Vérifier les arroseurs automatiques.
6. Allumer l'alarme.

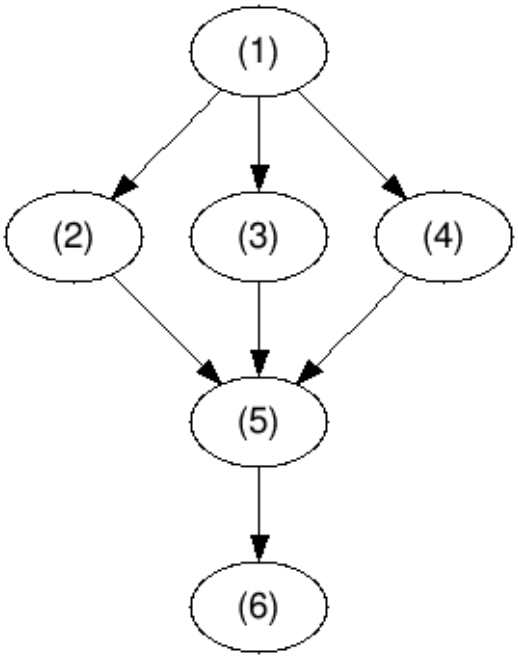

Figure 2.1:

1 chef et 7 employés (E1 - E7), 4 pelouses et 2 massifs. Comment répartir la charge de travail ?

Graphes de dépendances

Les tâches 2, 3 et 4 peuvent être parallèlisées (parallèlisme de contrôle). Subdivision des tâches les plus importantes.

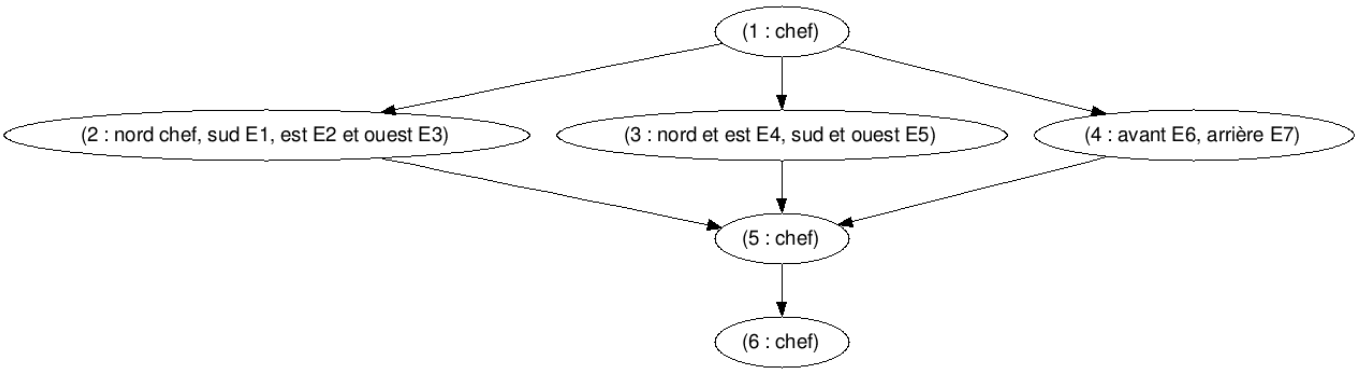

Figure 2.2:

Preuve de la loi d'Amdhal

S(n,p) : accélération parallèle pour un problème de taille de n sur p processeurs?

$\sigma(n)$: temps passé dans la partie non parallélisable du calcul.

$\varphi(n)$: temps passé dans la partie parallélisable du calcul.

C(n,p) : surcoût de la mise en place du parallèlélisme.

Temps de calcul

Séquentielle : $\sigma(n) + \varphi(n)$.
Parallèle : $\sigma(n) + \frac{\varphi(n)}{p} \rightarrow$ équilibrage de charge parfait.

En pratique l'équilibrage de charge est non parfait :

$$\geq \sigma(n) + \frac{\varphi(n)}{p} + C(n,p)$$

d'où

$$S(n,p) \leq \frac{\sigma(n)+\varphi(n)}{\sigma(n)+\frac{\varphi(n)}{p}+C(n,p)}$$

Comme $C(n,p) > 0$

f : fraction intrinsèquement séquentielle ($\frac{\sigma(n)}{\varphi(n)+\sigma(n)}$).
d'où : $\varphi(n) + \sigma(n) = \frac{\sigma(n)}{f} \rightarrow \varphi(n) = \sigma(n) * \frac{1}{f} - 1$
donc

$$S(n,p) \leq \frac{\frac{\sigma(n)}{f}}{\sigma(n) + \frac{\varphi(n)}{p} * \frac{1}{f} - 1} \tag{2.1}$$

$$\leq \frac{\frac{1}{f}}{1 + \frac{\frac{1}{f}-1}{p}} \tag{2.2}$$

$$\leq \frac{1}{f + \frac{1-f}{p}} \tag{2.3}$$

Loi de Gustafson-Barsis (preuve)

On passe en "weak scaling", $\varphi(n)$ dépend aussi de $p\varphi(n,p)$ et d'après l'hypothèse $\varphi(n,p) = \varphi(n)' * p$ alors

$$S(n,p) \leq \frac{\sigma(n) + \varphi(n,p)}{\sigma(n) + \frac{\varphi(n,p)}{p}} \tag{2.4}$$

$$\leq \frac{\sigma(n) + \varphi(n)'}{\sigma(n) + \varphi(n)'} \tag{2.5}$$

S : fraction de temps passé dans le calcul parallèle à effectuer des opérations inter-séquentielles.
1 - S : fraction du temps passé en parallèle.

$$S = \frac{\sigma(n)}{\sigma(n) + \frac{\varphi(n,p)}{p}} \tag{2.6}$$

$$= \frac{\sigma(n)}{\sigma(n) + \varphi(n)'} \rightarrow \text{S constant par rapport à p} \tag{2.7}$$

$$1 - S = \frac{\frac{\varphi(n,p)}{p}}{\sigma(n) + \frac{\varphi(n,p)}{p}} \tag{2.8}$$

$$= \frac{\varphi(n)'}{\sigma(n) + \varphi(n)'} \tag{2.9}$$

D'où

$$\sigma(n) = (\sigma(n) + \varphi(n)') * S \tag{2.10}$$

$$\varphi(n)' = (\sigma(n) + \varphi(n)) * (1 - S) \tag{2.11}$$

Par conséquent

$$S(n,p) \leq \frac{(\sigma(n) + \varphi(n)') * (S + (1 - S) * p)}{\sigma(n) * \varphi(n)'} \tag{2.12}$$

$$\leq S + (1 - S) * p \tag{2.13}$$

$$\leq p + (1 - p) * S \tag{2.14}$$

$$\leq p - (p - 1) * S \tag{2.15}$$

Exemple

$$S = \frac{5}{5 + 95} = 0.05 \tag{2.16}$$

$$f = \frac{5}{5 + 32 * 95} = 0.05 \tag{2.17}$$

Algorithme de Cannon

k = 0 : $P_{i,j} possède le bloc A(i, (i+j) mod 3) et le bloc B((i+j) mod 3, j) par ex : P_{0,2} possède A(0,2) et B(2,2)$.
$k = 1$: $P_{i,j} possède le bloc A(i, (i+j+1) mod 3) et le bloc B((i+j+1) mod 3, j) par ex : P_{0,2} possède A(0,0) et B(0,2)$.

Circulation des blocs de A "vers la gauche" et des blocs de B "vers le haut" sur le tore 2D.

Double buffering

Pour respecter la norme MPI, il faut faire un triple buffering.
Exemple : fonction de coût.
tableau T de taille n (i = 1 to n) et calcul T[i] = f(i) avec une complexité (f(i) = $O(i)$).
Choix de fonction de coût, Coût(i) = i.
On a donc : Coût Total = $\sum_{i=1}^{n} coût(i) = \sum_{i=1}^{n} i = \frac{n*(n+1)}{2}$.

Coût moyen : $\frac{n*(n+1)}{2p}$ $\Rightarrow$ distribution par bloc possible avec des blocs dont le coût est le plus proche possible du coût moyen.

Chapter 3

Cours 7: Introduction au calcul haute performance

Le "Memory Gap", Efficacité d'un code séquentiel (sur un coeur) $\approx 10 - 20\%$ de la crête d'un coeur (coûts des communications, équilibrage de charges, ...).

Causes

- Memory Gap
- SIMD
- FMA
- Pipelines

BLAS 3 : $> 90\%$.

Prédiction de branchement

Calcul du max d'un tableau, Probabilité (a[i] > max) = Probabilité (a[i] > i valeurs rencontrées) = Probabilité (a[i]) est la plus grande parmi i+1 valeurs ($\frac{1}{i+1}$).

Optimisations

fmodulo_sched : réordonnancement des instructions de plusieurs itérations dans les boucles internes.

Mémoire Rapide

- Temps accès mémoire
  $\rightarrow$ registres
- Suffisement grande pour contenir 3 vecteurs
  $\rightarrow$ cache L1

En pratique : $\frac{tm}{tf} > 10$

CPU actuel :

- 1 multiplication flottante 32 bits (sans effet pipeline), 4 cycles.
- 1 accès en mémoire principale $\approx$ 200 cycles, $\frac{tm}{tf} = \frac{200}{4} = 50$

GPU :

- 1 multiplication 32 bits (sans effet pipeline), $\approx$ 22 cycles.
- 1 accès à la mémoire globale $\approx$ 600 cycles, $\frac{tm}{tf} = \frac{600}{22} = 27$

Optimisations implémentées dans les BLAS

- produit matriciel par bloc
- remplissage des pipelines
- déroulage de boucles
- prefetching
- SIMD

Chapter 4

Cours 8: Parallèlisation automatique, application au parallélisme de contrôle

Ordre total

- transitif : $a \leq b$ et $b \leq c \Rightarrow a \leq c$
- anti-symétrique : $a \leq b$ et $b \leq a \Rightarrow a = c$
- total : soit $a \leq b$ soit $b \leq a$

Ordre partiel

- transitif : $a \leq b$ et $b \leq c \Rightarrow a \leq c$
- anti-symétrique : $a \leq b$ et $b \leq a \Rightarrow a = c$
- reflexif : $a \leq a$

Sous-ordre : $a < b \Rightarrow a \rightarrow b$

Preuve de l'unicité

Supposons qu'il existe deux systèmes S et S' et $(i, j) \in \mathbb{N}^2$ tels que : $T_i \prec_S T_j$ et $T_j \prec_{S'} T_i$ alors

$$\left.\begin{array}{l} T_i \prec_S T_j \Rightarrow T_i \rightarrow T_j \\ T_j \prec_{S'} T_i \Rightarrow T_j \rightarrow T_i \end{array}\right\} \rightarrow \text{Contradiction}.$$

Supposons qu'il existe deux systèmes S et S' et $(i, j) \in \mathbb{N}^2$ tels que : $T_i \prec_S T_j$ et $T_j \prec_{S'} T_i$ sont indépendantes dans S'.
Alors $T_i \prec_S T_j \Rightarrow T_i \rightarrow T_j$ et (T_i et T_j) sont non-indépendantes $\rightarrow$ *Contradiction*.

Le dernier cas est symétrique.

Chapter 5

Bonus : Annale

Exercice 2

1°) On utilise un allReduce qui somme sur un processus puis répercute le résultat sur tout les autres processus.
2°) On utilise un équilibrage static car on connaît les "limites" du problème (contraintes). On fait toujours la même chose pour chaque noeud (processus).
3°) p processeurs et p diviseur de n.

```
Donnees d'entree : n, epsilon, r (rang), p (nombre total de procs)

- hc = n/p
- hl = hc + (r > 0? 0 : 1) + (r < p-1 ? 1 : 0)
- allouer les grilles 0 et 1 de taille hl * n
Si (r == ROOT)
   allouer grille de taille n * n
Fin Si

- appel a init() pour initialiser les points des grilles 0 et 1 de taille hl * n
- g = 0
repeat
{
   Si (r-1 existe ET r > 0)
     envoyer la deuxieme ligne de la grille g a r - 1
     recevoir la premiere ligne de la grille g de r - 1
   Fin Si

   Si (r+1 existe ET r < p-1)
     recevoir la deuxieme ligne de la grille g de r + 1
     envoyer l'avant derniere ligne de la grille g a r + 1
   Fin Si

   - Calcul de la grille 1-g a partir des donnees de la grille g
   - Calcul de delta entre les grilles g et 1-g
   - MPI_Reduce() sur les deltas
   - MPI_Broadcast() pour mettre a jour les deltas
   - g = 1-g
} jusqu'a ce que delta < epsilon

- On fait un Gather sur toutes les grilles de g de tous les processus (chaque
  procs envoie hc lignes a partir de sa deuxieme ligne, sauf pour r = 0 qui envoie
  a partir de sa premiere ligne avec comme receveur ROOT.

Si (r == ROOT)
  sauvegarde sur disque de la grille totale
  desallocation de la grille totale
Fin Si

desallocation les grilles 0 et 1
```

4°) Oui, avec des communications non-bloquantes et avec l'algorithme suivant.

```
repeat
{
   - Communications non-bloquantes (MPI_ISend et MPI_IRecv) sur la grille g
   - Calcul des donnees de la grille &-g a partir de celles de la grille g,
     sauf pour la deuxieme ligne et l'avant derniere
   - Attendre la fin des communications (MPI_Wait)
   - Calcul de la deuxieme ligne et de l'avant derniere ligne de la grille
     1-g a partir de la grille g
   - Mise a jour delta
   - g = 1-g
} jusqu'a ce que delta < epsilon
```

5°) Il n'y a pas de communications entre les threads dans un processus MPI, quand les communications ne sont pas recouvertes, on a donc un gain de performance pour un nombre fixé de processus.