

# Algorithmes, Performance, Parallélisme

## Introduction à l'algorithmique parallèle

Frédéric Louergue



Département d'Informatique  
Faculté des Sciences



Master IPVGCA – 2007-2008

## Sources

Éléments de :

- Algorithmique PRAM de Gaétan Hains
- BSP Tutorial Material de Jonathan Hill

## Plan

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

## Plan de la séance

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
- 3 Algorithmes BSP

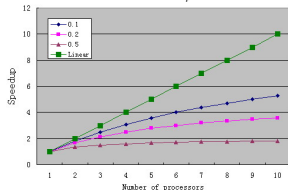
## Quand faut-il paralléliser ?

- Programme séquentiel de complexité exponentielle : inutile
- Programme séquentiel de complexité polynomiale :
  - s'il n'est pas P-complet OK
  - sinon non
- Si seulement une partie du programme est parallélisable ?

## Loi de Amdahl

- $T$  le temps d'exécution séquentielle
- $r$  la portion du programme parallélisable
- $p$  le nombre de processeurs

L'accélération est donnée par :  $\frac{1}{1 - r + \frac{r}{p}}$



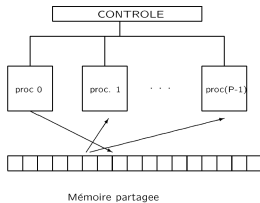
## Plan de la séance

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
- 3 Algorithmes BSP

## Plan de la séance

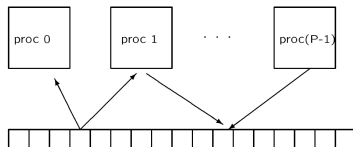
- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
    - Quelques algorithmes
    - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

## Parallel RAM

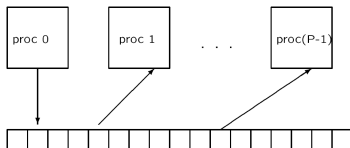


- $p$  machines RAM (avec petite mémoire locale)
- synchronisées
- mémoire partagée : comment se font les accès concurrents ?

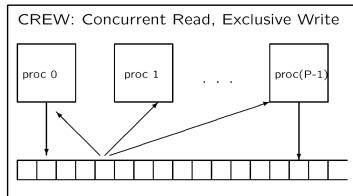
## CRCW – Concurrent Read Concurrent Write



## EREW – Exclusive Read Exclusive Write



## CREW – Concurrent Read Exclusive Write



- Pseudo Pascal/Ada
- Variables locales en minuscules
- Variables de la mémoire partagée dites également variables globales, en majuscules
- Seules opérations autorisées sur les variables globales :
  - `global_read(X,x)` lecture de la variable globale  $X$  et affectation de sa valeur à la variable locale  $x$
  - `global_write(e,Y)` évaluation de l'expression locale  $e$  et écriture de sa valeur dans la variable globale  $Y$ .
- *SPMD = Single Program Multiple Data* c'est-à-dire qu'un seul programme est écrit pour tous les processeurs.
- Un processeur exécute la même instruction (ou groupe d'instructions) que les autres ou ne fait rien.

- Le nombre d'étapes qu'un algorithme PRAM nécessite est son temps de calcul  $T$  pour un nombre  $p$  donné de processeurs.
- Le *travail*  $W$  d'un algorithme PRAM est le nombre total d'opérations effectuées. On a  $W \leq T \times p$
- Le *coût* est  $T \times p$

## Exemple

## Faire la somme des éléments de deux tableaux

- *Entrée* : deux tableaux  $X$  et  $Y$  de même taille  $n$  en mémoire partagée, indicés à partir de 1
- *Sortie* : un tableau  $S$  de même taille  $n$ , indicé à partir de 1, tel que pour tout  $i$  valide,  $S[i] = X[i] + Y[i]$
- *Processeurs* :  $n$  processeurs, identifiés par les entiers de 1 à  $n$ . La variable locale contenant le numéro de processeur est  $i$ .
- Algorithme :
 

```
global_read(X[i],x);
global_read(Y[i],y);
s := x+y;
global_write(s,S[i]);
```
- Temps d'exécution :  $\theta(1)$

## Plan de la séance

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

## Réduction par une opération binaire associative

### Réduction

- *Entrée* : un tableau  $X$  de taille  $n$  indiqué à partir de 1 en mémoire partagée
- *Sortie* : une variable  $S$  en mémoire partagée telle que  $S = X[1] \oplus X[2] \oplus \dots \oplus X[n]$  où  $\oplus$  est une opération binaire associative

### Pourquoi l'associativité ?

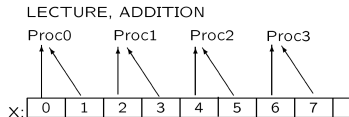
En fait  $X[1] \oplus X[2] \oplus \dots \oplus X[n]$  désigne par ex. :

$$X[1] \oplus (X[2] \oplus \dots (X[n-1] \oplus X[n]) \dots)$$

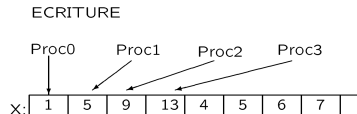
L'associativité permet de dire que c'est aussi :

$$((X[1] \oplus X[2]) \oplus (X[3] \oplus X[4])) \dots (X[n-1] \oplus X[n]) \dots$$

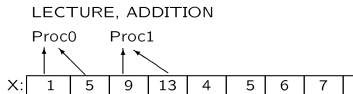
## Exemple pour $n = 8$

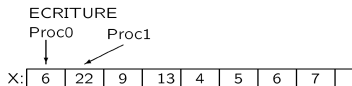


## Exemple pour $n = 8$



## Exemple pour $n = 8$





- Et pour terminer lecture et addition par le processeur 0 puis écriture et enfin écriture dans S
- Pseudo-code ?
- Complexité ?

Processeurs :  $n$  indicés de 1 à  $n$  et désignés par  $i$

```

k := n/2;
while (k>0) do
  if (i<=k) then
    begin
      global_read(X[2*i-1],x);
      global_read(X[2*i],y);
      s := x+y;
      global_write(s,X[i]);
      k := k/2;
    end
  end
end
if i=1 then global_write(s,S);

```

## Complexité

- $T = \log n$
- $p = n$

- *Entrée* : matrice  $M$  carrée de taille  $n \times n$  et un vecteur  $V$  de taille  $n$  en mémoire partagée
- *Sortie* :  $V'$  de taille  $n$  en mémoire partagée
- *Processeurs* :  $p$  qui divise  $n$ , désignés par  $i$
- *Algorithme* :
 

```

global_read(V,v);
global_read(M[(i-1)*r+1:i*r,1:n],b)
-- Compute w = b v
global_write(w,V'[(i-1)*r+1:ir]);

```

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

## Énoncé

Soit un algorithme PRAM de complexité en temps  $T$  et qui effectue un travail  $m$ . Alors il existe un algorithme PRAM équivalent sur  $p$  processeurs et de complexité en temps

$$O\left(\frac{m}{p} + T\right)$$

## Application

On peut appliquer le théorème pour réduire le nombre de processeurs de tout algorithme parallèle (éventuellement même jusqu'à le séquentialiser).

On peut en particulier réduire le coût parallèle.

## Réduction à gros grain

- Pour réduire  $n$  valeurs, on prend  $p \ll n$
- Première étape : chaque processeur réduit en séquentiel  $n/p$  valeurs
- Seconde étape : les  $p$  valeurs obtenues sont réduites par l'algorithme de réduction précédent
- $T = O(n/p) + O(\log p)$  avec  $p$  processeurs

## Présentation des algorithmes

- Plus besoin de préciser le nombre de processeurs
- Juste le travail effectué en parallèle, en général avec une notation de boucle parallèle : `pardo`

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
- 3 Algorithmes BSP

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
    - Somme des préfixes
    - Somme des préfixes sur des valeurs composées
    - Tri

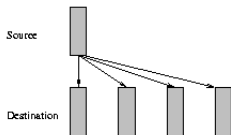
## Spécification

- *Entrée* : une valeur  $v$  dans la mémoire d'un processeur  $i$  donné
- *Sortie* : tous les processeurs doivent avoir cette valeur  $v$  en mémoire

## Diffusion en une super-étape

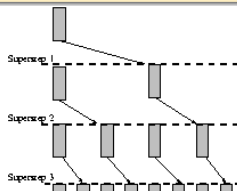
## Algorithme

- un processeur envoie directement à tous les autres une donnée de taille  $s$
- Coût :  $s \times (p - 1) \times g + L$

Diffusion binaire (ou  $d$ -aire)

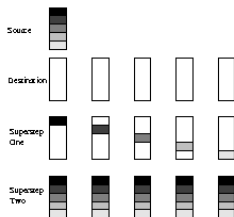
## Algorithme

- les processeurs qui ont la valeur l'envoient à  $d$  autres processeurs
- Coût :  $(s \times d + L) \times \log_d(p - 1)$



## Algorithme

- un processeur envoie directement à tous les autres un morceau de la donnée à envoyer de taille  $s/p$
- ensuite échange total puis recollage des morceaux
- Coût des communications =  $2 \times \frac{s}{p} \times (p-1) \times g + 2 \times L$



- Pour une petite valeur de  $s$  (en fonction de  $p$  et  $g$ ), la diffusion directe est la meilleure
- Pour  $p > 2$  la diffusion en deux phases est plus efficace que la diffusion binaire (ou  $d$ -aire)

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

## Somme des préfixes

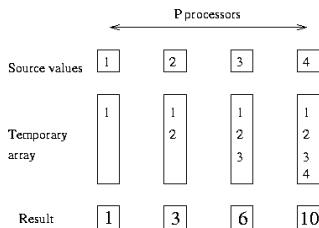
### Sur les listes

- scan: ('a → 'a → 'a) → 'a → 'a list
- $\text{scan} \oplus i_{\oplus} [x_1; \dots; x_n] = [i_{\oplus}; x_1 \oplus x_1; x_1 \oplus x_2; \dots; \oplus_{j=1}^n x_j]$   
où  $\oplus$  est associative et  $i_{\oplus}$  est un neutre pour  $\oplus$
- On appelle parfois scan la fonction qui prend en argument une opération associative et une liste non vide et renvoie la liste renvoyée par le scan précédent mais sans son premier élément.

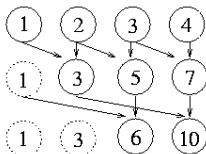
En parallèle : une valeur par processeur, opération associative sans son neutre

- en une super étape
- en  $\log p$  super-étapes

## scan parallèle en une étape



## scan parallèle en $\log p$ étapes



## Plan de la séance

- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

Initial	Rotate left	scan addition in local memory	Rotate right
$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{pmatrix}$

Initial	Rotate	scan vector addition on $n/p$ size chunks	Rotate back
$\begin{pmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \\ 6 & 6 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 1 & 3 & 5 \\ 2 & 4 & 6 \\ 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}$	$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 2 & 6 & 10 \\ 4 & 8 & 12 \\ 3 & 9 & 15 \\ 6 & 12 & 18 \end{pmatrix}$	$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \\ 4 & 8 & 12 \\ 5 & 10 & 15 \\ 6 & 12 & 18 \end{pmatrix}$

## Plan de la séance

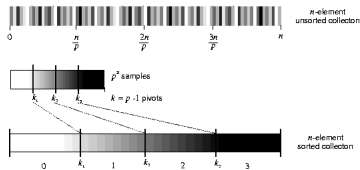
- 1 Parallélisation
- 2 Le modèle Parallel Random Access Machine (PRAM)
  - Le modèle
  - Quelques algorithmes
  - Principe de Brent
- 3 Algorithmes BSP
  - Diffusions
  - Somme des préfixes
  - Somme des préfixes sur des valeurs composées
  - Tri

## Que veut-on faire ?

	Processeur 0	Processeur 1	Processeur 2
Non-triées	7, 3, 6, 2, 0, 2,	5, 9, 1, 6, 3, 4,	8, 8, 9, 0, 5, 1
Triées	0, 0, 1, 1, 2, 2,	3, 4, 4, 5, 5, 6, 6,	7, 7, 8, 9, 9

Entrée : un vecteur parallèle de listes

- Trier localement les listes
- Prendre  $p$  échantillons primaires réguliers
- Échange total des échantillons primaires
- Tri local des échantillons primaires
- Prendre  $p$  échantillons secondaires réguliers par processeur
- La comparaison d'une valeur de la liste locale triée avec les échantillons secondaires donne le processeur sur lequel elle doit être envoyée.
- Tri local (par fusion) des listes obtenues par l'échange précédent



	Process 1	Process 2	Process 3
Data to be sorted	7 3 6 2 0 2,	5 9 1 6 3 4,	8 8 9 0 5 1
Local sort	0 2 2 3 6 7	1 3 4 5 6 9	0 1 5 8 8 9
Pick $p$ samples	0 2 3 3 6 7	1 3 4 5 6 9	0 1 5 8 8 9
Gather samples on process 0	2 3 7 3 5 9 1 8 9		
Sort the samples	1 2 3 3 5 7 8 9 9		
Best pivots	3 7	3 7	3 7
Partition data	0 2 2 3 6 7	1 3 4 5 6 9	0 1 5 8 8 9
Exchange	0 2 2 1 0 1	3 6 3 4 5 6 5	7 9 8 8 9
Merge partitions	0 0 1 1 2 2	3 3 4 5 5 6 6	7 8 8 9 9