Machines P-RAM et Réseaux de tri

Plan

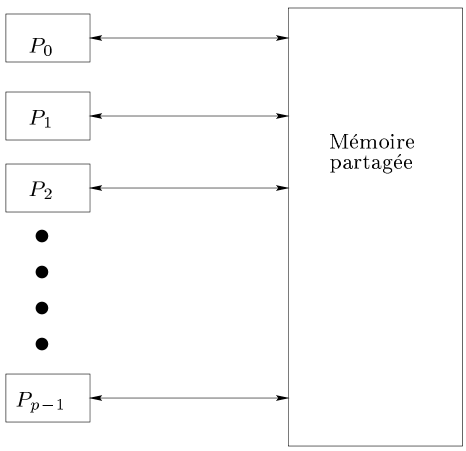
1. Introduction
2. Techniques de pointeurs
3. Simulation
4. Tri sur réseau linéaire

1 Introduction

Un modèle pour le parallélisme ?

- Machine de Turing → coût d'un algorithme séquentiel
- Grande variété des architectures parallèles
- Comment définir le coût d'une communication ?
- **Ignorer les communications ! !**
- Modèle imparfait mais classification des algorithmes parallèles
- Déterminer le parallélisme **maximal** contenu dans un algorithme

Machine P-RAM



- Mémoire centrale partagée (de taille non bornée)
- Nombre de processeurs non borné
- Les processeurs accèdent à n'importe quelle case de la mémoire centrale en une unité de temps
- Accès simultané de plusieurs processeurs à autant de cases mémoire distinctes possible en un seul pas de temps.

Accès concurrent à la même case

CREW (Concurrent Read Exclusive Write)
- Modèle le plus usuel
- Nombre de lectures simultanées d'une même case non borné
- Accès en écriture par un seul processeur à la fois.

CRCW (Concurrent Read Concurrent Write)
- Modèle le plus puissant
- Nombre d'écritures simultanées dans une même case non borné
- Variantes :

1. mode consistant : tous les proc. écrivent la même valeur
2. mode arbitraire : valeur du dernier processeur qui écrit
3. mode fusion : calcul au vol d'une fonction commutative et associative (ET ou un OU booléens, maximum ou somme d'entiers, etc)

EREW (Exclusive Read Exclusive Write)
- Modèle le plus restrictif
- Le plus proche des machines réelles.

2 Techniques de pointeurs

(A) Calcul du rang

Problème

– Liste chaînée L de taille n

– Calculer pour chaque élément i sa distance $d[i]$ à la fin de la liste :

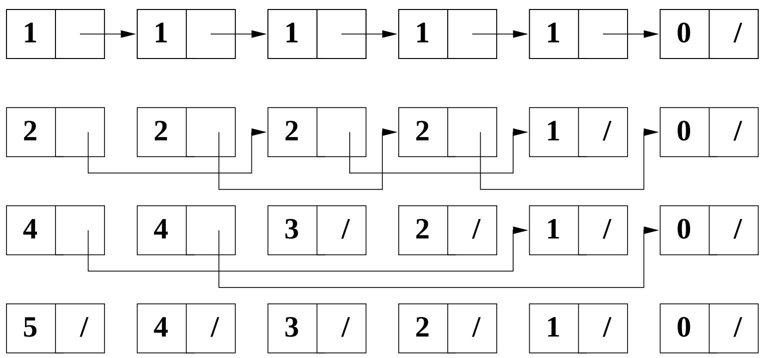
$$d[i] = \begin{cases} 0 & \text{if } next[i] = \text{NIL} \\ d[next[i]] + 1 & \text{if } next[i] \neq \text{NIL} \end{cases}$$

– Séquentiel : propager les valeurs d, temps $O(n)$

– P-RAM : un processeur P_i par élément i, temps $O(\log n)$

Algorithme

Initialisation
  for each processor i in // **do**
    if $next[i] = \text{NIL}$ **then** $d[i] = 0$ **else** $d[i] = 1$
Boucle principale
  while ($\exists$ objet i t.q. $next[i] \neq \text{NIL}$) **do**
    for each processor i in // **do**
      if $next[i] \neq \text{NIL}$ **then** $\begin{cases} d[i] \leftarrow d[i] + d[next[i]] \\ next[i] \leftarrow next[next[i]] \end{cases}$



Correction

– Principe : chaque pas dédouble les listes courantes en listes des objets en positions paires – positions impaires
$\Rightarrow$ Taille de chaque liste réduite de moitié à chaque étape
$\Rightarrow$ Nombre d'itérations égal à $\lceil \log n \rceil$

– A chaque étape, les mises à jour doivent être synchronisées
– Sémantique d'une boucle **forall** parallèle :

$$\begin{array}{l} \textbf{forall } i \textbf{ do} \\ \quad A(i) = B[i] \end{array} \quad \Longleftrightarrow \quad \begin{array}{c} \textbf{for } i = 1 \textbf{ to } n \textbf{ do} \\ A(i) = temp[i] \\ \textbf{for } i = 1 \textbf{ to } n \textbf{ do} \\ temp[i] = B[i] \end{array}$$

– Test de la condition globale :
Remplacer : **while** ($\exists$ objet i t.q. $next[i] \neq$ NIL) **do**
par : **for** étape $= 1$ **to** $\lceil \log n \rceil$

Type de P-RAM : EREW

– Chaque processeur P_i écrit dans $d(i)$ et $next[i]$, en mode exclusif
– P_i et P_j, où $next[j] = i$, accèdent simultanément à $d[i]$ en lecture
– Solution : remplacer $d[i] \leftarrow d[i] + d[next[i]]$
par :
$$temp[i] \leftarrow d[next[i]]$$
$$d[i] \leftarrow d[i] + temp[i]$$

(B) Calcul des préfixes

Problème

– Suite donnée $(x_1, x_2, \ldots, x_n)$

– $\otimes$ opération binaire associative quelconque

– Calculer $(y_1, y_2, \ldots, y_n)$, $y_k == x_1 \otimes x_2 \otimes \ldots \otimes x_k$

Algorithme

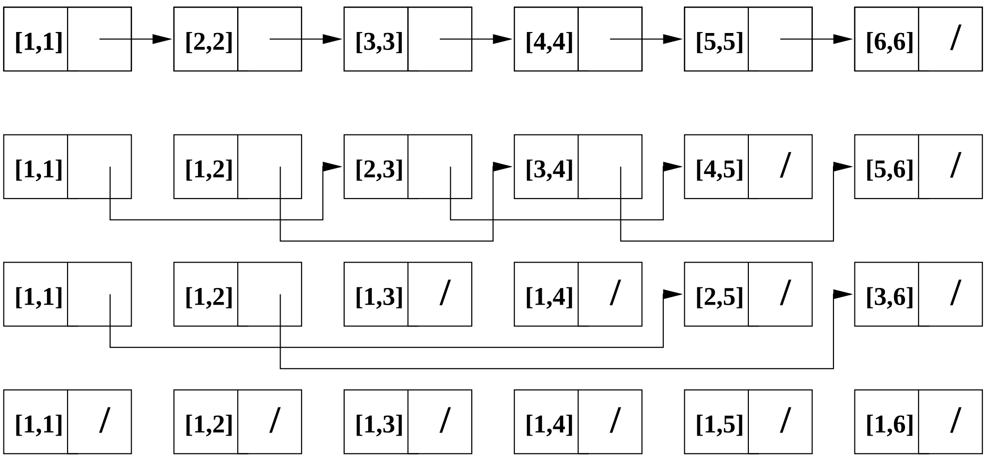
for each processor i in // **do**

  $y[i] \leftarrow x[i]$

  while ($\exists$ objet i t.q. $next[i] \neq$ NIL) **do**

    for each processor i in // **do**

      if $next[i] \neq$ NIL **then** $\begin{cases} y[next[i]] \leftarrow y[i] \otimes y[next[i]] \\ next[i] \leftarrow next[next[i]] \end{cases}$



$[i, j] = x_i \otimes x_{i+1} \otimes \ldots \otimes x_j$ pour $i \leq j$

(C) Tour d'Euler

Problème

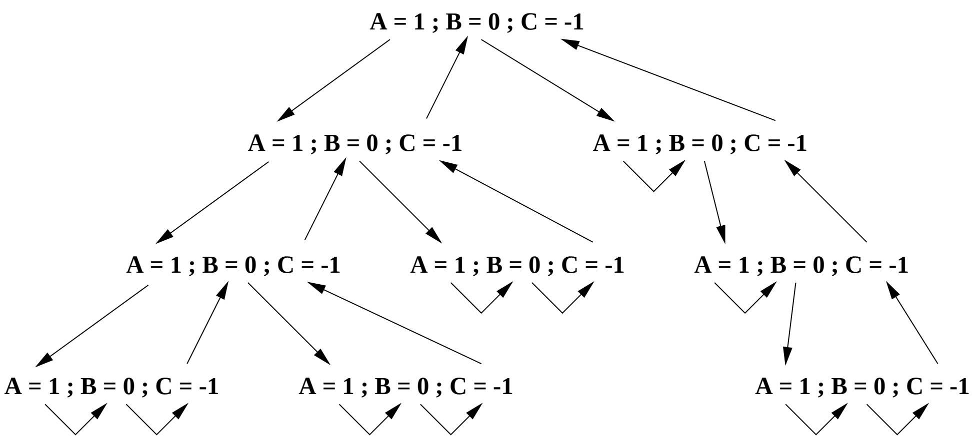
– Dans un arbre binaire à n sommets, calculer la hauteur de chaque sommet, i.e. sa distance à la racine

– Chaque sommet i à trois champs de données $parent[i]$, $left[i]$ et $right[i]$

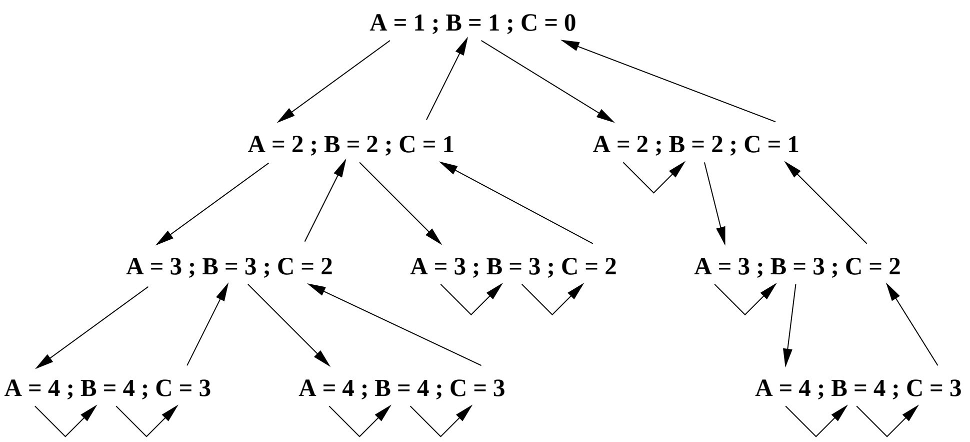
Solution

– Tour d'Euler

– Coût en $O(\log n)$, quelle que soit la hauteur de l'arbre.



(a) Initialisation



(b) Après calcul des préfixes

3 Simulation

Calcul du maximum

- Calcul en temps constant du maximum d'un tableau A de n éléments
- P-RAM CRCW avec $O(n^2)$ processeurs, mode consistant

```
for all i = 1 to n in // do
   m[i] ← TRUE
for all i, j = 1 to n in // do
   if A[i] < A[j] then m[i] ← FALSE
for all i = 1 to n in // do
   if m[i] = TRUE then max ← A[i]
```

Séparation des modèles CRCW et CREW

- Une P-RAM CRCW plus *puissante* qu'une P-RAM CREW ?
- Trouver un algorithme tel que, à nombre de processeurs égal, temps(P-RAM CRCW) < temps(P-RAM CREW)
- P-RAM CRCW de n processeurs : calcul en temps constant de la fusion $\bigotimes_{1\leq i\leq n} e_i$ de n éléments $(e_1, \ldots, e_n)$, où $\otimes$ est une loi associative
- P-RAM CREW de n processeurs : au moins $O(\log n)$ étapes : en temps constant fusion d'au plus un nombre constant d'éléments en une seule valeur.

Séparation des modèles CREW et EREW

– Déterminer si un élément e donné se trouve dans un ensemble de n éléments $(e_1, \ldots, e_n)$ tous distincts.

– Temps constant sur une P-RAM CREW avec n processeurs :
(i) initialiser un booléen à faux : $res \leftarrow$ FALSE
(ii) en parallèle, chaque processeur i compare e à e_i (lectures simultanées) et positionne $res \leftarrow$ TRUE si égalité
NB : les e_i sont différents deux à deux $\rightarrow$ mode écriture exclusive.

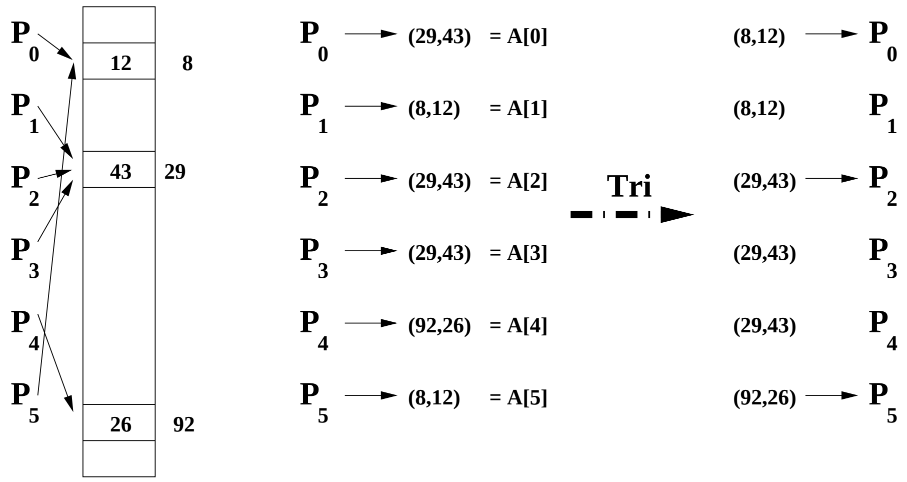
– Temps $\Omega(\log n)$ sur une P-RAM CREW avec n processeurs : comparaisons en temps constant $\rightarrow$ nombre de copies de e de l'ordre de $n \rightarrow O(\log n)$ pas de calcul.

– Diffusion d'une information à n processeurs
$\rightarrow O(\log n)$ étapes en EREW :
le nombre de processeurs "au courant" au plus "double" à chaque étape.

Théorème de simulation

– Tout algorithme sur une machine P-RAM CRCW à p processeurs ne peut pas être plus de $O(\log p)$ fois plus rapide que le meilleur algorithme P-RAM EREW à p processeurs pour le même problème.

Preuve



(a) mode CRCW **(b) simulation EREW**

– Simulation en mode exclusif d'écritures concurrentes, mode consistant

– P_i mode CRCW écrit x_i à l'adresse l_i
$\Rightarrow$ P_i mode EREW effectue l'écriture (exclusive) $A[i] = (l_i, x_i)$

– Tri de p données avec $O(p)$ processeurs EREW en temps $O(\log p)$
$\Rightarrow$ tri de A suivant la première coordonnée

– P_i mode EREW inspecte $A[i] = (l_j, x_j)$ et $A[i-1] = (l_k, x_k)$
et écrit (exclusif) x_j à l'adresse l_j si $l_j \neq l_k$ ou $i = 0$.

Théorème de Brent

- Soit A algorithme comportant un nombre total de m opérations et qui s'exécute en temps t sur une P-RAM (avec un nombre de processeurs indéterminé)
- Alors on peut simuler A en temps $O(\frac{m}{p} + t)$ sur une P-RAM de même type avec p processeurs.
- Preuve : à l'étape i, A effectue $m(i)$ opérations ($\sum_{i=1}^{n} m(i) = m$)
  Simulation avec p processeurs en temps $\lceil \frac{m(i)}{p} \rceil \leq \frac{m(i)}{p} + 1$.

Exemple d'application

- Calcul du maximum sur P-RAM EREW en temps $O(\log n)$
- Idée : arbre binaire, $O(n)$ processeurs
- $n = 2^m$, A de taille $2n$, maximum des n éléments de A en position $A[n], A[n+1], \ldots, A[2n-1]$, résultat dans $A[1]$ après exécution :

$$\textbf{for } k = m-1 \textbf{ downto } O \textbf{ step } -1 \textbf{ do}$$
$$\textbf{for } \mathsf{all}\ j,\ 2^k \leq j \leq 2^{k+1}-1 \ \mathsf{in}\ // \ \textbf{do}$$
$$A[j] \leftarrow \max(A[2j], A[2j+1])$$

- Avec p processeurs, simulation en temps $O(\frac{n}{p} + \log n)$
- Choisir $p = \frac{n}{\log n}$: même temps d'exécution, moins de processeurs !

Travail et efficacité

– P problème de taille n à résoudre

– $T_{seq}(n)$ temps du meilleur algorithme séquentiel (connu)

– Algorithme P-RAM en temps $T_{par}(p)$ avec p processeurs :

Speed-up $S_p = \frac{T_{seq}(n)}{T_{par}(p)}$

Efficacité $e_p = \frac{T_{seq}(n)}{p.T_{par}(p)}$

Travail $W_p = p.T_{par}(p)$

– Le travail se conserve par simulation

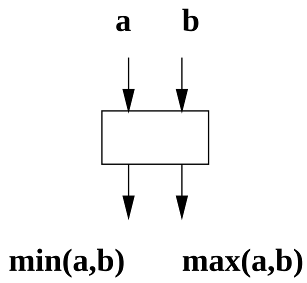
– A algorithme en temps t sur une P-RAM avec p processeurs
$\rightarrow$ simuler A sur une P-RAM de même type avec $p' \leq p$ processeurs, en temps $O(\frac{t.p}{p'})$.

– Algorithme *efficace* $\Longleftrightarrow$ travail de l'ordre de la complexité séquentielle

4 Tri sur réseau linéaire

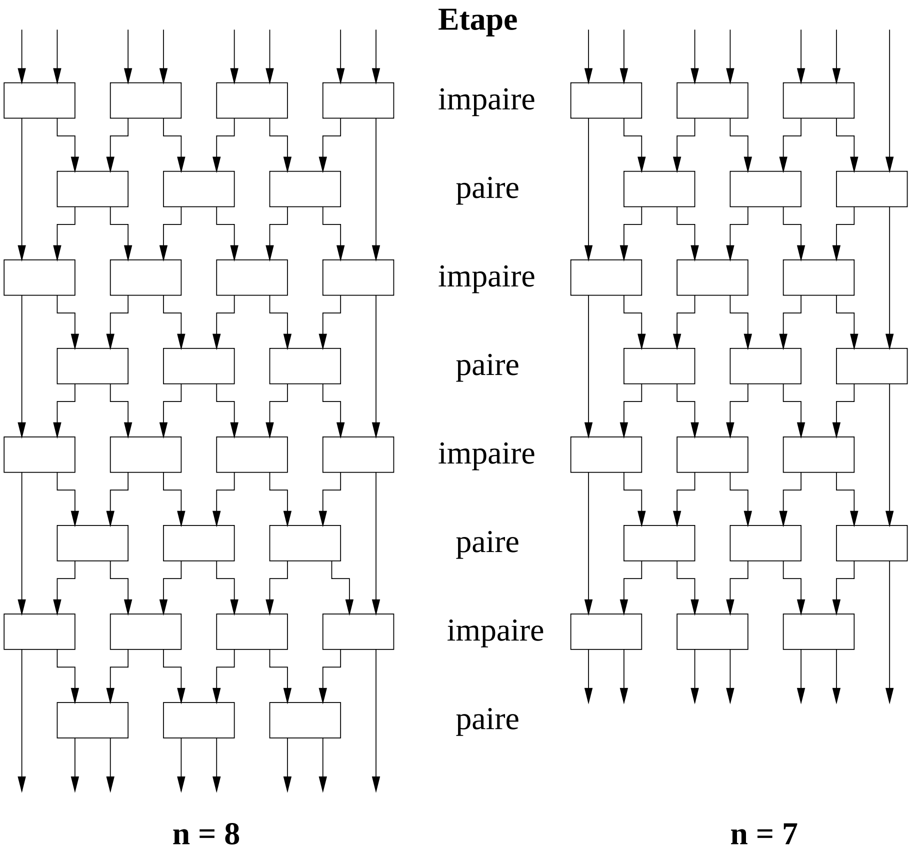
Réseaux de tri

– Modèle de calcul plus réaliste, mais moins général, que les P-RAM
– Disposer et interconnecter des comparateurs afin de trier des listes



– Architecture qui ne dépend que de la taille des listes, et non des valeurs de leurs éléments.

Réseau de transposition pair-impair



- Implante correctement le tri !
- Temps de calcul $t_n = n$
- Comparateurs $p_n = n(n-1)/2$
- Travail $W_n = O(n^3)$

Tri pair-impair sur réseau linéaire de processeurs

– Tri-fusion pair-impair sur réseau linéaire de p processeurs

– Chaque processeur trie sa sous-suite de taille n/p en temps $O(\frac{n}{p}\log n)$

– p étapes d'échanges alternés avec fusion, chacune de temps $O(n/p)$

	P_1		P_2		P_3		P_4		P_5		P_6
init	{8,3,12}		{10,16,5}		{2,18,9}		{17,15,4}		{1,6,13}		{11,7,14}
tri local	{3,8,12}		{5,10,16}		{2,9,18}		{4,15,17}		{1,6,13}		{7,11,14}
impair	{3,5,8}	$\leftrightarrow$	{10,12,16}		{2,4,9}	$\leftrightarrow$	{15,17,18}		{1,6,7}	$\leftrightarrow$	{11,13,14}
pair	{3,5,8}		{2,4,9}	$\leftrightarrow$	{10,12,16}		{1,6,7}	$\leftrightarrow$	{15,17,18}		{11,13,14}
impair	{2,3,4}	$\leftrightarrow$	{5,8,9}		{1,6,7}	$\leftrightarrow$	{10,12,16}		{11,13,14}	$\leftrightarrow$	{15,17,18}
pair	{2,3,4}		{1,5,6}	$\leftrightarrow$	{7,8,9}		{10,11,12}	$\leftrightarrow$	{13,14,16}		{15,17,18}
impair	{1,2,3}	$\leftrightarrow$	{4,5,6}		{7,8,9}	$\leftrightarrow$	{10,11,12}		{13,14,15}	$\leftrightarrow$	{16,17,18}
pair	{1,2,3}		{4,5,6}	$\leftrightarrow$	{7,8,9}		{10,11,12}	$\leftrightarrow$	{13,14,15}		{16,17,18}

– Temps de calcul $O(\frac{n}{p}\log n + n)$

– Travail $O(n(p + \log n))$: **optimal** pour $p \leq \log n$.