Mobilité, Cours Master 2, 2018, Machines PRAM

Michel Habib
habib@irif.fr
http://www.irif.fr/~habib

Janvier 2018

Plan

Le modèle le plus simple possible du parallélisme.
Basé sur le modèle de machine RAM (Random Acces Machine) : modèle théorique simple et universel de l'ordinateur à mémoire à accès direct.

- Chaque RAM possède sa propre mémoire.
- La communication entre machines RAM, se fait à l'aide d'une unique mémoire commune partagée, qui contient la donnée au départ du calcul et le résultat à la fin.

Parallel Random Access Machine

Du moins tolérant au plus tolérant :

- **EREW** Exclusive Read et Exclusive Write
  Modèle trop stricte
- **CREW** Concurrent Read et Exclusive Write
  Modèle assez réaliste.
- **CRCW** Concurrent Read et Concurrent Write
  Modèle un peu trop laxiste

Les variantes du CRCW

Du moins tolérant au plus tolérant :

Commun Les processeurs écrivent tous la même valeur.

Arbitraire un processeurs arbitraire écrit.

Prioritaire Le processeur de plus grand (resp. petit) numéro parmi ceux qui veulent accéder à un mot mémoire, est le seul à avoir le droit d'écrire.

Variante : Fusion Une fonction commutative et associative est appliquée sur les valeurs des processeurs qui écrivent (et, ou, . . .).

- Prioritaire $\geq$ Arbitraire $\geq$ Commun $\geq$ CREW $\geq$ EREW
- Cela veut dire que tout algorithme qui s'exécute sur une EREW s'exécute aussi sur CREW puis ... et une CRCW prioritaire.
- Les réciproques sont fausses !

On peut simuler une PRAM prioritaire à p processeurs par un PRAM EREW à p processeurs avec un facteur multiplicatif équivalent au tri sur une EREW.

Le modèle CRCW prioritaire est trop puissant

Théorème

On peut calculer en $O(1)$ toute fonction booléenne sur le modèle CRCW arbitraire.

Hypothèses du modèle PRAM

- Les données sont dans les premiers mots de la mémoire commune.
- Chaque processeur à un numéro (identifiant unique).
- À chaque étape du calcul, on considère que tous les processeurs de manière synchrone peuvent faire une lecture–écriture dans la mémoire commune.
- On ne considère pas la possibilité d'une panne.

Complexité d'un algorithme PRAM

- Le temps de calcul $Par(n)$ sur une donnée de taille n.
- Le nombre de processeurs requis $Proc(n)$
- Le travail : Travail(n)= $Par(n)xProc(n)$

Soit $Seq(n)$ le temps de calcul du meilleur algorithme séquentiel, tout algorithme parallèle sur un modèle PRAM vérifie :
$Travail(n) \geq \Omega(Seq(n))$.

Donc $Par(n) \geq \Omega(\frac{Seq(n)}{Proc(n)})$
En cas d'égalité on parle d'algorithme parallèle optimal.

Le premier exemple

En séquentiel :
Donnée : un entier n
Pour i=1 à $\sqrt{(n)}$
Si i divise n alors écrire i

Sur le modèle PRAM, si l'on dispose de $\sqrt{(n)}$ processeurs

Initialisons une variable Résultat à 0
$\forall i$ en parallèle :
Si i divise n alors écrire i dans Résultat

Ce calcul se fait en $O(1)$ sur un modèle CRCW (car il suffit qu'un processeur écrive dans Résultat).

Exercice

Calcul de $n!$ en séquentiel puis en parallèle.

1. $\forall i$ en parallèle, $m(i) = VRAI$
2. $\forall i, j \in [1, n]^2$ $i \neq j$ en parallèle, si $A(i) < A(j)$ alors $m(i) = FAUX$
3. $\forall i$ en parallèle, si $m(i) = VRAI$ alors $Maximum = A(i)$

Analyse

n^2 processeurs
Temps : 3 instructions parallèles donc en $O(1)$
Travail $O(n^2)$
Meilleur algorithme en séquentiel $O(n)$
Algorithme non optimal.
L'algorithme fonctionne sur une CRCW PRAM, car la deuxième instruction exige des lectures et écritures concurrentes.
Mais s'il y a écritures multiples elles sont identiques, donc une CRCW Commun suffit.

la fonction Rang se calcule récursivement :
$Rang(i) = 0$ si $Suivant(i) = NIL$
sinon $Rang(i) = Rang(Suivant(i)) + 1$

Initialisations

$\forall i$ en parallèle,

Si $Suivant(i) = NIL$ alors $Rang(i) = 0$

Sinon $Rang(i) = 1$

Corps de la procédure

Tant qu'il existe i tel que $Suivant(i) \neq NIL$

$\forall i$ en parallèle,

Si $Suivant(i) \neq NIL$ alors

$Rang(i) = Rang(i) + Rang(Suivant(i))$

$Suivant(i) = Suivant(Suivant(i))$

Analyse

n processeurs (autant que d'éléments dans la liste)
Temps $O(logn)$
Travail $O(nlogn)$
Meilleur algorithme en séquentiel $O(n)$
Algorithme non optimal.
L'algorithme fonctionne sur un EREW PRAM

Données : une liste $x_1, \ldots x_n$
Résultat : tous les préfixes de $x_1 \oplus \cdots \oplus x_n$
Notons : $y(1) = x_1$ et $y(k) = y(k-1) \oplus x_k$ les n préfixes

Initialisations

$\forall i$ en parallèle, $y(i) = x_i$

Corps de la procédure

Tant qu'il existe i tel que $Suivant(i) \neq NIL$

$\forall i$ en parallèle,

Si $Suivant(i) \neq NIL$ alors

$y(Suivant(i)) = y(i) \oplus y(suivant(i))$ et

$Suivant(i) = Suivant(Suivant(i))$

Analyse

n processeurs (autant que d'éléments dans la liste)
Temps $O(logn)$
Travail $O(nlogn)$
Meilleur algorithme en séquentiel $O(n)$
Algorithme non optimal.
L'algorithme fonctionne sur un EREW PRAM

Extraction d'éléments d'une liste

On considère une liste L de n éléments colorés en bleu ou en rouge
Il s'agit d'en extraire la sous-liste constituée des éléments bleus.
Le principe est d'utiliser la technique de saut de pointeurs pour déterminer pour chaque élément de la liste, la place du bleu qui le suit dans la liste.

$\forall i$ en parallèle, si $Suivant(i) = NIL$ ou $Couleur(Suivant(i)) = bleu$
alors $Fini(i) = VRAI$ et $Bleu(i) = Suivant(i)$

Tant qu'il existe i tel que $Fini(i) = Faux$
$\forall i$ en parallèle
$Fini(i) = Fini(Suivant(i))$
Si $Fini(i) = VRAI$ alors $Bleu(i) = Bleu(Suivant(i))$
et $Suivant(i) = Suivant(Suivant(i))$

A la fin de l'algo :
les bleus sont accrochés au premier élément de la liste s'il est bleu ou sinon à son successeur bleu.

Analyse

n processeurs (autant que d'éléments dans la liste)
Temps $O(logn)$
Travail $O(nlogn)$
Meilleur algorithme en séquentiel $O(n)$
Algorithme non optimal.
L'algorithme fonctionne sur un EREW PRAM

Calcul de sa racine

On considère une arborescence :
$\forall i$ en parallèle :
Si $Pere(i) = NIL$ alors $racine(i) = i$
Tant qu'il existe i tel que $Pere(i) \neq NIL$
$\forall i$ en parallèle :
Si $Pere(i) \neq NIL$ alors
Si $Pere(Pere(i)) = NIL$ alors $racine(i) = racine(Pere(i))$
$Pere(i) = Pere(Pere(i))$

A la fin de l'algo chaque élément connaît la racine de l'arborescence.
n processeurs (autant que d'éléments dans la liste)
Temps $O(logn)$
Travail $O(nlogn)$
Meilleur algorithme en séquentiel $O(n)$
Algorithme non optimal.
L'algorithme fonctionne sur un EREW PRAM

Tour Eulérien en $O(1)$

On considère un arbre orienté symétrique donc connexe, représenté par ses listes d'adjacences. On construit en $O(1)$ une nouvelle liste d'arcs L .

$\forall$ arc xy en parallèle,
$Suivant(xy)$= l'arc qui suit "circulairement" l'arc yx dans la liste d'adjacence de y
(i.e. si yx est le dernier de la liste on prend le premier de la liste).

L est un parcours Eulérien de l'arborescence.
La procédure Suivant associe à chaque arc de type xy un arc yt. En outre cette fonction est injective.
Elle définit un parcours Eulérien de l'arbre.
Interprétation en séquentiel, cela revient à un parcours en profondeur de l'arbre.

Analyse

m processeurs (autant que d'arcs)
Temps $O(1)$
Travail $O(m)$
Meilleur algorithme en séquentiel $O(m)$
Algorithme optimal.
L'algorithme fonctionne sur un EREW PRAM

Algorithme PRAM de calcul de composantes connexes d'un graphe

Données : un graphe $G = (X, E)$, $|X| = n$, $|E| = m$.
Résultats : les composantes connexes de G.
On suppose disposer de $n + 2m$ processeurs.
Initialisations en parallèle

1. $\forall v \in X$, $Pere(v) \leftarrow v$;
2. $\forall$ arête $uv \in E$ (ou $vu \in E$) : Si $u < v$ alors $Pere(Pere(u)) \leftarrow Pere(v)$;
3. $\forall$ arête $uv \in E$ (ou $vu \in E$) : Si u est un singleton alors $Pere(Pere(u)) \leftarrow Pere(v)$;

Boucle principale en parallèle sur les arêtes ou les sommets
Répéter en parallèle tant que l'étape de contraction modifie qqchose

1. $\forall$ arête $uv \in E$ (ou $vu \in E$) :
   If u est dans une étoile et si $Pere(u) < Pere(v)$
   alors $Pere(Pere(u)) \leftarrow Pere(v)$
2. $\forall$ arête $uv \in E$ (or $vu \in E$) :
   If u est dans une étoile et si $Pere(u) \neq Pere(v)$
   "donc v n'est pas dans une étoile"
   alors $Pere(Pere(u)) \leftarrow Pere(v)$
3. Etape de contraction des arborescences
   $\forall u \in X$, $Pere(u) \leftarrow Pere(Pere(u))$;

Invariant

A chaque étape la structure de données Pere représente une forêt d'arborescences.

Pour terminer, il suffit de se convaincre que les deux tests u est un singleton et u est dans une étoile sont réalisables en $O(1)$ en parallèle.

u est un singleton en $O(1)$

Deux conditions à vérifier :

1. $pere(u) = u$ % signifie que u est racine d'une arborescence %
2. $\forall v \neq u$ en parallèle tester si $pere(v) = u$. % signifie que personne ne pointe sur u %

Exercice :

Montrer que le test u est dans une étoile se réalise en $O(1)$ en paprallèle.

Analyse

$n + 2m$ processeurs
Temps $O(logn)$
Travail $O((n + m)logn)$
Meilleur algorithme en séquentiel $O(n + m)$
Algorithme non optimal.
L'algorithme fonctionne sur une Arbitraire CRCW PRAM

Exercices afin de compléter ceux vus en cours

1. Item le plus fréquent d'une liste
2. Tri d'un tableau
3. Parcours en largeur dans un graphe
4. Calcul d'un plus court chemin dans un graphe
5. Arbre de poids minimum.