

Cours 5.2: Complexité parallèle

Olivier Bournez

bournez@lix.polytechnique.fr
LIX, Ecole Polytechnique

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Opérations sur une machine RAM

1. $x_i \leftarrow 0$
2. $x_i \leftarrow x_i + 1$
3. $x_i \leftarrow x_i \ominus 1$
4. $x_i \leftarrow x_j$
5. $x_i \leftarrow x_{x_j}$
6. $x_{x_i} \leftarrow x_j$
7. **if** $x_i = 0$ **then** $ctl_state \leftarrow j$ **else** $ctl_state \leftarrow j'$

Subtilité

- Un calcul en temps t se simule en un temps polynomial en t par une machine RAM.
- La réciproque n'est pas vraie, car une machine RAM peut manipuler des entiers très grands en temps unitaire.
 - ▶ Autre façon de mesurer la taille d'une entrée :
 - La taille d'un entier n est $|n| = \lceil \log(n) \rceil$ (le nombre de bits nécessaire pour l'écrire en binaire).
 - Une entrée d'une machine RAM est un mot $w = w_1 w_2 \cdots w_n$ sur \mathbb{N}^* :
 - on mesure sa taille $|w|$ comme $|w| = \sum_{i=1}^n |w_i|$.
- Avec cette mesure, temps polynomial sur RAM correspond bien à un temps polynomial classique (sur structures finies) et réciproquement.

Version effective de la thèse de Church

Théorème

Les modèles suivants se simulent deux à deux en temps polynomial.

1. *Les machines de Turing* *sur un alphabet fini*
2. *Les machines à $k \geq 2$ piles* *sur un alphabet fini*
3. *Les machines RAM* *avec cette convention*
4. *Les algorithmes* *sur une structure finie*

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Le modèle PRAM

Définitions

Séparation des modèles

Théorème de Brent

Travail et efficacité

Algorithmes efficaces et optimaux

Le modèle PRAM

- Un des principaux modèles de machines parallèles :
 - ▶ Différentes machines, ou *processeurs*, interagissent
 1. de façon synchrone
 2. par l'intermédiaire d'une mémoire partagée.
 - ▶ Chaque processeur k
 - correspond à une machine RAM,
 - et possède une mémoire locale : des registres $x_{k,i}$ locaux.
 - ▶ Il y a en plus :
 - une mémoire globale partagée : des registres $x_{0,i}$ partagés.

Formellement

Une machine PRAM (*Parallel Random Access Machine*) correspond à une suite $\Pi_1, \Pi_2, \dots, \Pi_q$ de programmes, chaque programme Π_k étant du type

```
ctl_state ← 0
repeat
  seq
  <instructions>
  endseq
until ctl_state = q_a
```

où $\langle \text{instructions} \rangle$ est une suite finie d'instructions de l'un des types suivants.

Instructions autorisées sur processeur k

Instructions RAM usuelles (sur mémoire locale)

1. $x_{k,i} \leftarrow 0$
2. $x_{k,i} \leftarrow x_{k,i} + 1$
3. $x_{k,i} \leftarrow x_{k,i} \ominus 1$
4. $x_{k,i} \leftarrow x_{k,j}$
5. $x_{k,i} \leftarrow x_{k,x_{k,j}}$
6. $x_{k,x_{k,i}} \leftarrow x_{k,j}$
7.
if $x_{k,i} = 0$ **then** $ctl_state \leftarrow j$
else $ctl_state \leftarrow j'$

Accès en lecture/écriture à la
mémoire partagée :

8. $x_{k,i} \leftarrow x_{0,j}$
9. $x_{k,i} \leftarrow x_{0,x_{k,j}}$
10. $x_{0,i} \leftarrow x_{k,j}$
11. $x_{0,x_{k,i}} \leftarrow x_{k,j}$

Parallélisme ?

- Le nombre q de processeurs est une fonction $q(n)$ de la taille n de l'entrée.
 - ▶ Pour éviter les cas pathologiques, on se restreint au cas *L-uniforme* : la fonction qui à $\mathbf{1}^n$ associe $q(n)$ et les programmes $\Pi_1, \Pi_2, \dots, \Pi_{q(n)}$ est calculable en espace logarithmique.

Évolution

- Une *configuration* $X = (X_1, \dots, X_q)$ est la donnée des états de chacun des programmes.
- On passe d'une configuration X à sa configuration successeur $X' = (X'_1, \dots, X'_q)$ en effectuant simultanément une étape de chaque programme.
- Un programme PRAM termine si chacun des programmes Π_k termine.

Conflits en lecture/écriture ?

1. Le modèle EREW (*Exclusive Read Exclusive Write*) :
 - ▶ lecture et écriture d'un même emplacement mémoire possible que par un processeur à la fois.
2. Le modèle CREW (*Concurrent Read Exclusive Write*) :
 - ▶ lectures simultanées d'un même emplacement mémoire autorisées.
 - ▶ écriture d'un emplacement mémoire partagé possible que par un processeur à la fois.
3. Le modèle CRCW (*Concurrent Read Concurrent Write*) :
 - ▶ lectures et écritures simultanées d'un même emplacement mémoire autorisées.
 - ▶ Résultat d'une écriture simultanée ?
 - ▶ plusieurs variantes :
 - le mode *consistant* : on impose que tous les processeurs qui écrivent simultanément doivent écrire la même valeur.
 - le mode *prioritaire* : on prend la valeur du processeur d'indice le plus petit.
 - ...

EREW \subset CREW \subset CRCW

Le modèle PRAM

Définitions

Séparation des modèles

Théorème de Brent

Travail et efficacité

Algorithmes efficaces et optimaux

CRCW : Calcul du max en temps constant

- sur CRCW avec $\mathcal{O}(n^2)$ processeurs.

```
for  $i \leftarrow 1$  to  $n$  in parallel  
   $m[i] \leftarrow \text{vrai}$   
for  $i, j \leftarrow 1$  to  $n$  in parallel  
  if  $A[i] < A[j]$  then  $m[i] \leftarrow \text{faux}$   
for  $i \leftarrow 1$  to  $n$  in parallel  
  if  $m[i] = \text{vrai}$  then  $\text{out} \leftarrow A[i]$ 
```

- fonctionne même en en mode consistant puisque tout le monde écrit la même valeur, à savoir *faux*.

- plus rapide que tout algorithme séquentiel ($\mathcal{O}(n)$)!!
- $\text{CREW} < \text{CRCW}$:
 - ▶ on ne peut pas résoudre le max en temps constant sur CREW.
 - En temps constant on peut fusionner au plus qu'un nombre constant d'éléments en une seule valeur sur CREW.

CRCW : Opérations logiques en temps constant

- *OU* logique :

```
m ← faux  
for i ← 1 to n in parallel  
    if A[i] then m ← vrai
```

- *ET* logique :

```
m ← vrai  
for i ← 1 to n in parallel  
    if not A[i] then m ← faux
```

CREW : Trouver en temps constant

- déterminer si un élément e donné se trouve parmi n éléments distincts e_1, e_2, \dots, e_n .
- sur CREW avec $\mathcal{O}(n)$ processeurs.

```
res ← faux  
for  $i \leftarrow 1$  to  $n$  in parallel  
  if  $e = e_i$  then  
    res ← vrai.
```

- plus rapide que tout algorithme séquentiel ($\mathcal{O}(n)$) !!
- EREW < CREW :
 - ▶ on ne peut pas résoudre ce problème en temps constant sur EREW.
 - Il faut au moins lire les entrées e_i , et donc $\mathcal{O}(\log(n))$ accès en lecture.

A ressource temps/processeur équivalentes :

$$\text{EREW} < \text{CREW} < \text{CRCW}$$

Tri optimal sur EREW

- Résultat admis :

Proposition

On peut trier p données avec $\mathcal{O}(p)$ processeurs en temps $\mathcal{O}(\log(p))$ avec une machine PRAM de type EREW.

Théorème

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

Preuve

- Preuve dans le cas où les PRAM CRCW opèrent en mode consistant.
- Considérons un pas de l'algorithme CRCW à p -processeurs : on va le simuler en $\mathcal{O}(\log(p))$ étapes d'un algorithme EREW.
- On se concentre sur les accès mémoire.
- Quand le processeur P_i de l'algorithme EREW écrit une donnée x_i à l'adresse l_i en mémoire, le processeur P_i de l'algorithme EREW effectue l'écriture exclusive $A[i] \leftarrow (l_i, x_i)$.
- On trie alors le tableau A suivant la première coordonnée en temps $\mathcal{O}(\log(p))$.
- Une fois le tableau A trié, chaque processeur P_i de l'algorithme EREW inspecte les deux cases adjacentes $A[i] = (l_j, x_j)$ et $A[i - 1] = (l_k, x_k)$, où $0 \leq j, k \leq p - 1$. Si $l_j \neq l_k$, ou si $i = 0$, le processeur P_i écrit la valeur x_j à l'adresse l_j , sinon il ne fait rien.
- Comme le tableau est trié selon la première coordonnée, l'écriture est bien exclusive.

- $EREW^i$ est la classe des problèmes qui peuvent être résolus en temps parallèle $\mathcal{O}(\log^i(n))$ avec un nombre polynomial de processeurs avec une EREW-PRAM :

$$EREW^i = \bigcup_{k \in \mathbb{N}} EREW(n^k, \log^i(n)).$$

- De même :

$$CREW^i = \bigcup_{k \in \mathbb{N}} CREW(n^k, \log^i(n)).$$

$$CRCW^i = \bigcup_{k \in \mathbb{N}} CRCW(n^k, \log^i(n)).$$

$$\text{EREW}^k \subset \text{CREW}^k \subset \text{CRCW}^k \subset \text{EREW}^{k+1}$$

Le modèle PRAM

Définitions

Séparation des modèles

Théorème de Brent

Travail et efficacité

Algorithmes efficaces et optimaux

Effet de la réduction du nombre de processeurs

Théorème (Brent)

Soit A un algorithme comportant un nombre total de m opérations et qui s'exécute en temps t sur une PRAM (avec un nombre de processeurs indéterminé).

Alors on peut simuler A en temps

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

sur une PRAM de même type avec p processeurs.

Preuve

- A l'étape i , l'algorithme A effectue $m(i)$ opérations.
- Par définition, $m = \sum_{i=1}^t m(i)$.
- On simule l'étape i avec p processeurs en temps $\lceil \frac{m(i)}{p} \rceil \leq \frac{m(i)}{p} + 1$.
- On obtient le résultat en sommant sur les étapes i .

Exemple d'application

- Prenons par exemple le calcul du maximum sur une PRAM EREW.
 - ▶ On peut agencer ce calcul en temps $\mathcal{O}(\log(n))$ à l'aide d'un arbre binaire : à l'étape 1, on procède paire par paire avec $\lceil n/2 \rceil$ processeurs, puis on continue avec les maximum des paires deux à deux, etc.
- Que se passe t'il si l'on dispose de moins de $\mathcal{O}(n)$ processeurs ?
 - ▶ Le théorème de Brent nous dit qu'avec p processeurs on peut simuler l'algorithme précédent en temps $\mathcal{O}(\frac{n}{p} + \log n)$ puisque le nombre d'opérations total est $m = n - 1$.

Exemple d'application

- Prenons par exemple le calcul du maximum sur une PRAM EREW.
 - ▶ On peut agencer ce calcul en temps $\mathcal{O}(\log(n))$ à l'aide d'un arbre binaire : à l'étape 1, on procède paire par paire avec $\lceil n/2 \rceil$ processeurs, puis on continue avec les maximum des paires deux à deux, etc.
- Que se passe t'il si l'on dispose de moins de $\mathcal{O}(n)$ processeurs ?
 - ▶ Le théorème de Brent nous dit qu'avec p processeurs on peut simuler l'algorithme précédent en temps $\mathcal{O}\left(\frac{n}{p} + \log n\right)$ puisque le nombre d'opérations total est $m = n - 1$.
 - ▶ Si l'on choisit $p = \frac{n}{\log n}$ processeurs, on obtient à une constante près le même temps d'exécution, mais avec moins de processeurs !

Le modèle PRAM

Définitions

Séparation des modèles

Théorème de Brent

Travail et efficacité

Algorithmes efficaces et optimaux

Quelques définitions

- Soit L un problème de taille n à résoudre, et $T_{seq}(n)$ le temps du meilleur algorithme séquentiel connu pour résoudre L .
- Soit maintenant un algorithme parallèle PRAM qui résout L en temps $T_{par}(p)$ avec p processeurs.

- ▶ *Facteur d'accélération* (*speedup* en anglais) :

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

- ▶ *Efficacité* :

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

- ▶ *Travail de l'algorithme* :

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

Le modèle PRAM

Définitions

Séparation des modèles

Théorème de Brent

Travail et efficacité

Algorithmes efficaces et optimaux

Un algorithme parallèle est dit

■ *efficace* s'il

1. fonctionne en temps parallèle poly-logarithmique (c'est-à-dire en temps $\mathcal{O}(\log^k(n))$)
2. et si son travail est le temps du meilleur algorithme séquentiel connu multiplié par un facteur poly-logarithmique.

■ *optimal*

1. s'il est efficace
2. et que son travail est du même ordre que le travail du meilleur algorithme séquentiel connu.

Rendre optimal un algorithme efficace

Parfois possible. Exemple : calcul du max sous EREW.

■ Astuce :

- ▶ On groupe donc les n entrées en $\mathcal{O}(\frac{n}{\log(n)})$ groupes ayant chacun $\mathcal{O}(\log(n))$ éléments.
- ▶ A chaque groupe, on associe un processeur qui calcule le max de son groupe par l'algorithme séquentiel : il le fait donc en temps $\mathcal{O}(\log(n))$ et on utilise $\mathcal{O}(\frac{n}{\log(n)})$ processeurs.
- ▶ Puis le max des $\mathcal{O}(\frac{n}{\log(n)})$ max par groupe peut être calculé par l'algorithme parallèle initial en temps $\mathcal{O}(\log(n))$ avec $\mathcal{O}(\frac{n}{\log(n)})$ processeurs.
- ▶ L'algorithme obtenu fonctionne en temps parallèle $\mathcal{O}(\log(n))$, avec $\mathcal{O}(\frac{n}{\log(n)})$ processeurs et est optimal.

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Sous menu

- Modèles basés sur les circuits
 - Famille de circuits uniformes
 - Classes NC^i et AC^i
 - Relations avec les PRAM
 - Résumé

- Soit $(C_n)_{n \in \mathbb{N}}$ une famille de circuits booléens, où le circuit C_n possède exactement n entrées et 1 sortie.
- La famille est dite L-uniforme si la fonction qui à $\mathbf{1}^n$ associe la description $\langle C_n \rangle$ du circuit C_n est calculable en espace logarithmique en n .

Notation $\text{CIRCUIT}(f(n), p(n))$

- On dira qu'un langage ou un problème L est dans

$$\text{CIRCUIT}(f(n), p(n))$$

si L est reconnu par une famille de circuits L -uniforme avec

1. une taille en $\mathcal{O}(f(n))$
2. et une profondeur en $\mathcal{O}(p(n))$.

Reformulation du cours 3

Théorème

$$P = \bigcup_{k \in \mathbb{N}} \text{CIRCUIT}(n^k, n^k).$$

Preuve

- C'est (presque) ce que l'on a dit dans le cours 3.
- La petite différence : la notion d'uniformité.
- Il suffit d'observer que la preuve du théorème dans le cours 3 produit bien des circuits L -uniformes.

- Autrement dit, le temps séquentiel est finement relié à la taille des circuits.
- On s'attend à ce que le temps parallèle des circuits soit relié à leur profondeur . . .

Sous menu

Modèles basés sur les circuits

Famille de circuits uniformes

Classes NC^i et AC^i

Relations avec les PRAM

Résumé

Classes NC^i et NC

- Soit $i \geq 1$. On définit :

$$NC^i = \bigcup_{k \in \mathbb{N}} \text{CIRCUIT}(n^k, \log^i(n)).$$

- On définit

$$NC = \bigcup_{i \in \mathbb{N}} NC^i.$$

- La possibilité d'effectuer des *ou* logiques et *et* logiques en temps constant invite à considérer la variante suivante :
 - ▶ AC^i désigne la classe similaire à NC^i mais où l'on autorise les portes portes \vee et \wedge à être appliquées à plus que deux bits.

Notation UCIRCUIT($f(n)$, $p(n)$)

- On dira qu'un langage ou un problème L est dans

$$\text{UCIRCUIT}(f(n), p(n))$$

si L est reconnu par une famille de circuits L -uniforme

- ▶ où les portes \vee et \wedge dans les circuits peuvent être de fanin ≥ 2

avec

1. une taille en $\mathcal{O}(f(n))$
2. et une profondeur en $\mathcal{O}(p(n))$.

Classes AC^i et AC

- Soit $i \geq 1$. On définit :

$$AC^i = \bigcup_{k \in \mathbb{N}} UCIRCUIT(n^k, \log^i(n)).$$

- On définit

$$AC = \bigcup_{i \in \mathbb{N}} NC^i.$$

Proposition

Pour tout i ,

$$\text{NC}^i \subset \text{AC}^i \subset \text{NC}^{i+1}.$$

Corollaire

$$\text{NC} = \text{AC}.$$

Preuve

- $NC^i \subset AC^i$ par définition.
- Maintenant, $AC^i \subset NC^{i+1}$ car toute porte \vee (respectivement \wedge) d'un circuit AC^i de taille $f(n)$ possède au plus un fanin de $f(n)$.
- On peut remplacer chaque telle porte par un arbre binaire de portes \vee (respectivement \wedge).
- En faisant cela systématiquement, on transforme un circuit de $CIRCUIT(f(n), p(n))$ en $CIRCUIT(\mathcal{O}(f(n)), \mathcal{O}(p(n) * \log(f(n))))$.
- La transformation se fait bien espace logarithmique.

Sous menu

Modèles basés sur les circuits

Famille de circuits uniformes

Classes NC^i et AC^i

Relations avec les PRAM

Résumé

Simuler un circuit par une PRAM

Théorème

On peut simuler un circuit de fanin non-borné par des machines PRAM CRCW :

$$\text{UCIRCUIT}(f(n), p(n)) \subset \text{CRCW}(f(n)^2, p(n)).$$

Preuve

- Initialement l'entrée est codée dans les registres partagés 1 à n et un registre $n + i$ est affecté à la porte numéro i du circuit.
- Ces derniers registres sont initialisés à 0 pour une porte \vee et à 1 pour une porte \wedge .
- On associe d'autre part à chaque arc du circuit un processeur.
- A chaque instant, chaque processeur détermine la valeur courante de son arête (u, v) , en lisant le registre $n + u$, et si v est une porte \vee (respectivement \wedge) écrit sa valeur dans l'emplacement $n + v$ si celle-ci vaut 1 (resp. 0).
- Si cette suite d'opération prend k opérations élémentaires sur une RAM, Au temps $k * p(n)$ chaque registre $n + i$ aura la valeur de la porte qu'il représente.

Simuler une PRAM par un circuit

Théorème

On peut simuler une machine PRAM CRCW par un un circuit de fanin non-borné : supposons $f(n)$ et $p(n)$ de complexité propre alors,

$$\text{CRCW}(f(n), p(n)) \subset \text{UCIRCUIT}(\text{poly}(f(n)), p(n)),$$

où poly désigne un polynôme.

Preuve

- On montre que chaque type d'opération élémentaire autorisé dans les instructions d'une machine PRAM (c'est-à-dire essentiellement additionner 1 et réaliser la soustraction \oplus) se réalise par un circuit qui travaille sur les représentations binaires de fanin non-borné de profondeur 2, de taille polynomiale.
- On simule alors l'évolution globale du système par un circuit comme dans la simulation d'un algorithme par un circuit.

(Joli) exercice

Théorème

Tout circuit de $\text{CIRCUIT}(f(n), p(n))$ est équivalent à un circuit de $\text{CIRCUIT}(f(n), p(n))$ de fanout 2.

Corollaire

Pour tout entier k , $\text{NC}^k \subset \text{EREW}^k$.

Sous menu

Modèles basés sur les circuits

Famille de circuits uniformes

Classes NC^i et AC^i

Relations avec les PRAM

Résumé

Résumé

Théorème

Pour tout entier k ,

$$\text{NC}^k \subset \text{EREW}^k \subset \text{CREW}^k \subset \text{CRCW}^k = \text{AC}^k \subset \text{NC}^{k+1}.$$

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Sous menu

Quelques problèmes parallélisables

Addition

Multiplication

Multiplication de matrices booléennes

Clôture transitive

Inversion matricielle et déterminant

Addition

Proposition

Étant donnés a et b en binaire sur n bits, on peut produire le codage binaire de leur somme sur $n + 1$ bits par un circuit AC^0 , et donc NC^1 .

Preuve

- Supposons que a s'écrit $a = a_{n-1}a_{n-2} \dots a_0$, et que b s'écrit $b = b_{n-1}b_{n-2} \dots b_0$.
- Soit c_j la retenue de la j ème position.
- Soit $g_j = a_j \wedge b_j$ (appelé *bit de génération de retenue*) et $p_j = a_j \vee b_j$ bit de *propagation de retenue*).
- En posant $c_{-1} = 0$, on a les récurrences $c_j = g_j \vee p_j c_{j-1}$ et $z_j = a_j \oplus b_j \oplus c_{j-1}$, duquel on tire $c_j = \bigvee_{i \leq j} g_i p_{i+1} \dots p_j$.
- On calcule ainsi dans une première étape tous les g_j et p_j puis tous les produits $g_i p_{i+1} \dots p_j$, puis c_j comme leur union de tous ces produits, et enfin les $z_j = a_j \oplus b_j \oplus c_{j-1}$.
- Cela donne un circuit AC^0 .

Sous menu

Quelques problèmes parallélisables

Addition

Multiplication

Multiplication de matrices booléennes

Clôture transitive

Inversion matricielle et déterminant

Multiplication

Proposition

Étant donnés a et b en binaire sur n bits, on peut produire le codage binaire de leur produit sur $2n$ bits par un circuit NC^1 .

■ Remarque :

- ▶ On sait prouver que la multiplication n'est pas dans AC^0 .

Preuve

- L'algorithme classique pour réaliser une multiplication en binaire ramène le problème à celui d'additionner n nombres de taille $2n$.
- Ce dernier problème peut se résoudre par un circuit de fanin borné par l'astuce "3 pour 2", qui consiste à ramener l'addition de trois nombres sur n -bits à l'addition de deux nombres sur $n + 1$ -bits :
 - ▶ Soient $a = a_{n-1}a_{n-2} \dots a_0$, $b = b_{n-1}b_{n-2} \dots b_0$ et $c = c_{n-1}c_{n-2} \dots c_0$ les trois nombres à additionner.
 - ▶ $a_i + b_i + c_i$ s'écrit en binaire sur deux bits $u_i v_i$.
 - ▶ Les u_i et v_i s'obtiennent par un circuit de taille linéaire et de profondeur constante.
 - ▶ Alors $a + b + c = u + v$, où $u = u_{n-1}u_{n-2} \dots u_0 0$ et $v = v_{n-1}v_{n-2} \dots v_0$.
- L'addition de n nombres, chacun de longueur $2n$ peut alors se résoudre par $\mathcal{O}(\log(n))$ itérations chacune utilisant cette astuce "3 pour 2" pour réduire le nombre de nombres d'un facteur de $2/3$, suivi d'une étape final qui somme deux nombres de $\mathcal{O}(n)$ bits.

Sous menu

Quelques problèmes parallélisables

Addition

Multiplication

Multiplication de matrices booléennes

Clôture transitive

Inversion matricielle et déterminant

Multiplication de matrices booléennes

Proposition

Soient $A = (a_{i,j})$ et $B = (b_{i,j})$ des matrices $n \times n$ à coefficients dans $\{0, 1\}$.

On peut calculer leur produit (booléen) C avec un circuit dans AC^0 et donc dans NC^1 .

Preuve

- C s'écrit $C = (c_{i,j})$ avec $c_{i,j} = \bigvee_{k=1}^n a_{i,k} b_{k,j}$.
- Il suffit de construire un circuit qui à son premier niveau calcule les produits $a_{i,k} b_{k,j}$ puis à son second niveau fait un \bigvee de tous ces résultats.

Sous menu

Quelques problèmes parallélisables

Addition

Multiplication

Multiplication de matrices booléennes

Clôture transitive

Inversion matricielle et déterminant

Clôture transitive

- Soit $A = (a_{i,j})$ une matrice $n \times n$ à coefficients dans $\{0, 1\}$.
- A peut se voir comme la matrice d'adjacence d'un graphe.
- l'entrée i, j de la matrice A^k vaut 1 si et seulement s'il existe un chemin de longueur k entre i et j dans le graphe codé par A .

Proposition

La clôture transitive de A , définie par $A^ = \bigvee_{k=0}^{+\infty} A^k$ se calcule par un circuit AC^1 , et donc NC^2 .*

Preuve

- On vérifie facilement que $A^* = (I \cup A)^{2^{\lceil \log(n) \rceil}}$.
- On peut donc calculer A^* en partant de $B = A \cup I$, et en élevant au carré B $\lceil \log(n) \rceil$ fois.
- Puisque la multiplication de matrices est dans AC^0 , la clôture transitive est dans AC^1 .

Corollaire

On peut donc déterminer s'il existe un chemin dans un graphe donné par sa matrice d'adjacence par un circuit AC¹, et donc NC².

Sous menu

Quelques problèmes parallélisables

Addition

Multiplication

Multiplication de matrices booléennes

Clôture transitive

Inversion matricielle et déterminant

Théorème

L'inversion matricielle, et le calcul du déterminant d'une matrice carrée peut se calculer par un circuit NC^2 .

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Sous menu

Quelques résultats

Relations avec les autres classes

Résultats de séparation

Théorème

On a

$$\text{NC}^1 \subset \text{LOGSPACE}.$$

Plus généralement,

Théorème

Soit $i \geq 1$. On a

$$\text{NC}^i \subset \text{SPACE}(\log^i(n)).$$

Preuve

- Sur une entrée w , on peut produire le circuit $C_{|w|}$ correspondant à cette entrée en espace logarithmique,
- et utiliser ensuite une recherche en profondeur récursive partant de la sortie du circuit pour déterminer $C_{|w|}(w)$.
- Cela nécessite seulement de mémoriser le chemin entre la sortie et le sommet que l'on est en train d'explorer, et de mémoriser les résultats intermédiaires que l'on a déjà pu obtenir le long de ce chemin.
- Puisque le circuit est de profondeur logarithmique (resp. $\log^i(n)$) un espace logarithmique (resp. $\log^i(n)$) est suffisant.

Corollaire

On a

$$\text{NC} \subset \text{P}.$$

Théorème

On a

$$\text{NLOGSPACE} \subset \text{AC}^1.$$

Preuve

- Soit L un langage de NLOGSPACE.
- Déterminer si un mot w est accepté revient à déterminer s'il existe un chemin dans un graphe G_w entre $X[w]$ et X^* .
- On a vu que cela se ramenait au calcul de la clôture transitive de la matrice d'adjacence du graphe G_w , et que cela se réalisait par un circuit AC^1 .

Résumé

Théorème

$$\text{NC}^1 \subset \text{LOGSPACE} \subset \text{NLOGSPACE} \subset \text{AC}^1$$

Théorème

$$\text{NC} = \bigcup_i \text{NC}^i = \bigcup_i \text{AC}^i$$

Théorème

$$\text{NC} \subset \text{P}.$$

Sous menu

Quelques résultats

Relations avec les autres classes

Résultats de séparation

Théorème

Le problème PARITY, où l'on se donne $a_1, \dots, a_n, p \in \{0, 1\}$ et l'on veut déterminer si $p = \sum_i a_i$ modulo 2, est dans NC^1 mais n'est pas dans AC^0 .

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Théorème

Si $L \leq_L L'$, et que le problème L' est dans NC alors $L \in \text{NC}$

- (on peut s'y attendre, mais cependant, il faut prouver formellement que la réduction en espace logarithmique préserve la complexité parallèle).

Preuve

- Soit f la réduction en espace logarithmique de L vers L' .
- Il existe un algorithme A' qui décide le langage constitué des couples $\langle w, \langle i \rangle \rangle$, où i est un entier inférieur à $|f(w)|$, tel que le i ème bit de $f(w)$ vaut 1.
- Maintenant, en résolvant le problème REACH sur le graphe G_w correspondant à cet algorithme A' , on peut calculer le i ème bit de $f(w)$.
- Si on résout tous ces problèmes par des circuits NC^2 en parallèle, on peut donc calculer tous les bits de $f(w)$.
- Une fois que l'on a $f(w)$, on peut utiliser le circuit pour L' pour décider si $w \in L$.

On déduit de cette preuve :

Corollaire

Si $L \leq_L L'$, et que le problème L' est dans NC^i , pour $i \geq 2$, alors $L \in NC^i$.

Sous menu

Problèmes intrinsèquement non-parallélisables
Problèmes non-parallélisables

Problèmes non-parallélisables

- La question $NC = P$? est une question ouverte
 - ▶ on sait cependant que $NC \neq PSPACE$, puisque $NC \subset LOGSPACE$.
- Si $NC \neq P$ (comme cela est généralement conjecturé), les problèmes de P les plus difficiles à paralléliser doivent contenir les problèmes P -complets :
- on dit qu'un problème est *non-parallélisable* s'il est P -difficile.

Exemples

- CIRCUITVALUE, SATVALUE ainsi que MONOTONECIRCUITVALUE sont dans ce cas.
- MONOTONECIRCUITVALUE peut aussi se reformuler ainsi : “Étant donné une séquence de n équations booléennes du type $e_1 = 0$, $e_1 = 1$, et $e_k = e_i \wedge e_j$ ou $e_k = e_i \vee e_k$, pour $1 \leq j < k \leq n$, calculer la valeur de e_n ”.
- Étant donné un graphe orienté, deux sommets distingués s et t , et une fonction c qui associe à chaque arc un entier positif ou nul, le problème de déterminer si le flot maximum de s vers t est un entier impair est P-complet.

Au menu

Retour sur les machines RAM

Le modèle PRAM

Modèles basés sur les circuits

Quelques problèmes parallélisables

Quelques résultats

Problèmes intrinsèquement non-parallélisables

Thèse du parallélisme

Circuits non-polynomiaux

- Avec la notion d'uniformité considérée jusque là, on impose que la fonction qui à 1^n associe la description du circuit C_n soit calculable en espace logarithmique en n , ce qui implique en temps polynomial en n .
 - ▶ On est donc incapable de produire un circuit avec un nombre exponentiel de portes, puisqu'en temps polynomial on ne peut écrire qu'un nombre polynomial de symboles.

Un notion d'uniformité plus adaptée

- $(C_n)_{n \in \mathbb{N}}$ une famille de circuits booléens , où le circuit C_n possède exactement n entrées et 1 sortie.
- La famille est dite BC-uniforme si la fonction qui à $\mathbf{1}^n$ associe la description $\langle C_n \rangle$ du circuit C_n est calculable en espace logarithmique **en la taille du circuit** C_n .

Remarque

- Pour les circuits de taille polynomiale, et donc pour tout ce qui précède, les notions de L -uniformité, et BC-uniformité coïncident.
- On aurait pu (du ?) utiliser cette notion partout.

Théorème (Thèse du parallélisme)

Le temps parallèle polynomial (sans contrainte sur le nombre de processeurs, c'est-à-dire possiblement un nombre exponentiel de processeurs) correspond à l'espace polynomial.

En d'autres termes :

$$\begin{aligned} \text{PSPACE} &= \text{CIRCUIT}(2^{n^{\mathcal{O}(1)}}, n^{\mathcal{O}(1)}) \\ &= \text{UCIRCUIT}(2^{n^{\mathcal{O}(1)}}, n^{\mathcal{O}(1)}) \\ &= \text{EREW}(2^{n^{\mathcal{O}(1)}}, n^{\mathcal{O}(1)}) \\ &= \text{CREW}(2^{n^{\mathcal{O}(1)}}, n^{\mathcal{O}(1)}) \\ &= \text{CRCW}(2^{n^{\mathcal{O}(1)}}, n^{\mathcal{O}(1)}) \end{aligned}$$

où la notion d'uniformité utilisée dans chaque membre droit est la BC-uniformité (= celle qui leur donne un sens).