Algorithmique et architectures parallèles
Partiel du 28 octobre 2008:
Réseaux de tri et P-RAM

(Durée: 2 heures)

Florent Bouchez Veronika Sonigo Anne Benoit

1 Réseaux de permutation

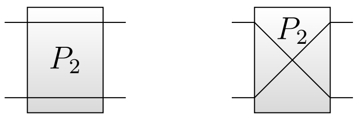


Fig. 1 – Bascule : relie les entrées aux sorties en les permutant ou non.

Note : les réseaux que l'on manipulera ne sont pas *primitifs.*

Un *réseau de permutations* à n entrées et n sorties possède des bascules lui permettant de relier ses entrées à ses sorties selon n'importe laquelle des $n!$ permutations possibles. La figure 1 montre le réseau de permutation P_2 à 2 entrées et 2 sorties, ne comportant qu'une bascule qui peut alimenter les sorties soit avec les entrées telles qu'elles se présentent, soit en les croisant.

Question 1. *Dessinez un réseau de permutation P_3 à 3 entrées et 3 sorties. Dans quel état doivent être vos bascules pour transformer l'entrée $\langle 1, 2, 3\rangle$ en $\langle 2, 3, 1\rangle$?*

Question 2. *Montrer que si l'on remplace chaque comparateur dans un réseau de tri par la bascule de la figure 1, le nouveau réseau est un réseau de permutation. Autrement dit, pour une permutation π quelconque, il existe une façon de configurer les bascules dans le réseau de sorte que chaque entrée i soit reliée à la sortie $\pi(i)$.*

La figure 2 montre la construction récursive d'un réseau de permutation P_8 à 8 entrées et 8 sorties, utilisant deux copies de P_4 et 8 bascules. Les bascules ont été configurées pour réaliser la permutation $\pi = \langle \pi(1), \pi(2), \ldots, \pi(8)\rangle = \langle 4, 7, 3, 5, 1, 6, 8, 2\rangle$, qui nécessite (récursivement) que le P_4 supérieur sache réaliser $\langle 4, 2, 3, 1\rangle$ et que le P_4 inférieur sache réaliser $\langle 2, 3, 1, 4\rangle$.

Question 3. *Montrer comment réaliser la permutation $\langle 5, 3, 4, 6, 1, 8, 2, 7\rangle$ sur P_8 en dessinant les configurations des bascules et les permutations effectuées par les deux P_4.*

Soit n une puissance exacte de 2. On définit P_n récursivement en fonction de deux $P_{n/2}$ de la même façon que pour P_8.

Question 4. *Décrire un algorithme en $O(n)$ (sur une machine ordinaire) capable de configurer les n bascules reliées aux entrées et aux sorties de P_n, et qui puisse déterminer les permutations à réaliser par chaque $P_{n/2}$, de manière à effectuer une permutation quelconque à n éléments. Prouvez la validité de votre algorithme.*

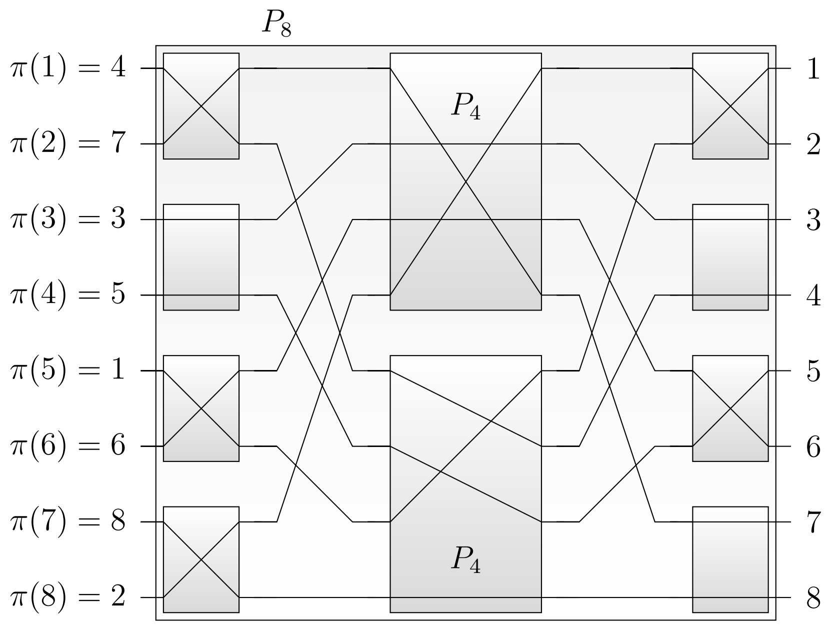


FIG. 2 – Construction récursive de P_8.

Question 5. *Quelles sont la profondeur et la taille de P_n ? Combien de temps faut-il sur une machine ordinaire pour calculer toutes les configurations de bascule, y compris celles situées à l'intérieur des $P_{n/2}$?*

2 Évaluation des expressions arithmétiques sur une PRAM CREW

2.1 Idées générales

Cette deuxième partie du partiel concerne l'évaluation d'une expression arithmétique $(+, -, \times, /)$ de longueur n en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs sur une PRAM CREW. L'algorithme proposé est donc à la fois optimal en temps (regarder l'évaluation de $\sum_{i=1}^{n} x_i$) et en travail (puisque le travail global est de l'ordre de n). La première partie consiste à construire un arbre d'évaluation de l'expression arithmétique en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs et la seconde partie porte sur l'évaluation de cet arbre en temps $O(\log n)$ avec $O(n)$ processeurs (l'algorithme optimal étant trop long pour ce partiel) en utilisant des techniques de type saut de pointeurs. Contrairement à ce qu'on pourrait imaginer, le principe n'est pas de construire un arbre très "dense" pour utiliser le schéma de réduction appliqué à une somme par exemple. L'idée est de construire un arbre qui se prête bien à une évaluation "pseudo-formelle".

En effet, dans le cas général, il n'est pas facile de construire un arbre d'évaluation équilibré pour une expression arithmétique. Regardons par exemple l'évaluation d'un polynôme par le schéma de Horner :

$$a_1 + x \times (a_2 + x \times (a_3 + x \times (a_4 + x)))$$

Les arbres naturels correspondant au schéma de Horner et à la forme développée sont présentés dans la figure 3. L'évaluation avec la forme développée conduit à un arbre mieux équilibré mais elle introduit des calculs redondants et ne permet donc pas une évaluation beaucoup plus rapide.

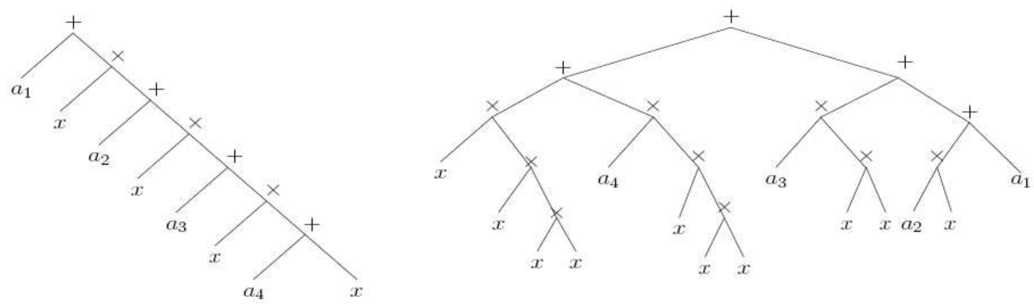


FIG. 3 – Arbres d'évaluation du polynôme.

2.2 Construction de l'arbre d'évaluation

Définition 1. *Une expression correctement parenthésée est une expression arithmétique contenant autant de parenthèses ouvrantes que de parenthèses fermantes et telle qu'il y ait plus de parenthèses ouvrantes que de parenthèses fermantes dans tout préfixe.*

Définition 2. *Une expression simple est soit une variable entourée de parenthèses, soit une expression arithmétique de la forme*

$$(E_1 op_1 E_2 \dots op_{k-1} E_k)$$

où tous les opérateurs op_i ont la même priorité et où les E_i sont elles mêmes des expressions simples.

Définition 3 (La fonction match)**.** *Pour une expression correctement parenthésée, on définit la fonction match associant à une parenthèse ouvrante la parenthèse fermante qui lui correspond.*

Question 6. *Transformer l'expression bien parenthésée suivante en une expression simple :*
$(1 + 2 * 3) - 4/(5 * 6 - 7)$

Question 7. *Écrire un algorithme qui transforme une expression correctement parenthésée entourée de parenthèses en une expression simple en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs.*

On considère maintenant une chaîne constituée uniquement de parenthèses (de longueur $n = 2^m$ et m divise 2^m) obtenue par la suppression dans une expression simple de tous les autres caractères.

Définition 4. *On définit l'opérateur de réduction $\mathcal{R}$ sur une paire de parenthèses par*

$$\mathcal{R}(((\,) = (($$
$$\mathcal{R}()\,))) =))$$
$$\mathcal{R}(()) =$$
$$\mathcal{R}()\,() =)($$

On dira qu'une séquence de parenthèses a été réduite (est sous forme irréductible) si l'application de l'opérateur de réduction sur une paire quelconque de deux parenthèses successives est sans effet.

Question 8. *Quelle est la forme d'une séquence réduite de parenthèses dans le cas général ? Quelle est la forme réduite de la concaténation de deux séquences irréductibles de parenthèses ?*

Question 9. *Donner un algorithme parallèle pour construire en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs un arbre dont chaque nœud représente la séquence irréductible des feuilles de son sous-arbre (on pourra supposer que la longueur de la séquence de parenthèses initiale est 2^k et que k divise 2^k).*

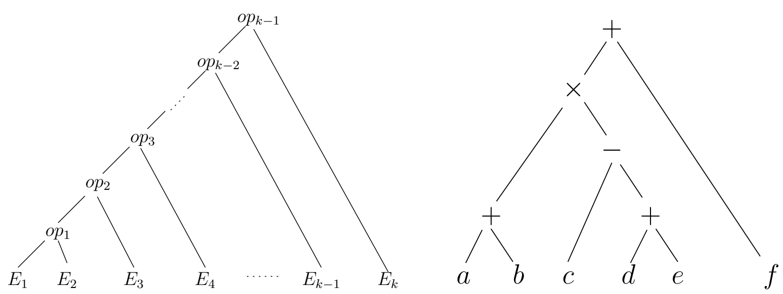
Supposons que l'on dispose d'un algorithme **match** qui calcule en temps $O(n \log n)$ avec $O(n \log n)$ processeurs la fonction match pour toutes les parenthèses ouvrantes. Plus précisement, l'appel **match(i)** renvoi la position de la parenthèse fermante correspondant à la parenthèse ouvrante en position i dans une expression simple. Par exemple, pour l'expression ci-dessous,

$$\textbf{Ex1}: \qquad ((((a)+(b))\times((c)-((d)+(e))))+(f))$$

match(3) renvoie la parenthèse juste avant l'opérateur $\times$ (position 11), et **match(8)** retourne la parenthèse fermante de l'expression b (position 10).

On est maintenant en mesure de proposer un algorithme pour construire l'arbre d'évaluation d'une expression simple. La figure 4(a) montre l'arbre d'évaluation correspondant à l'expression simple

$$\Big((E_1)op_1(E_2)\ldots op_{k-1}(E_k)\Big)$$



(a) Arbre d'évaluation correspondant à une expression simple.

(b) Arbre d'évaluation correspondant à l'expression simple de l'exemple (Ex1)

FIG. 4 –

Question 10. *Proposer un algorithme qui construit l'arbre d'évaluation d'une expression simple en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs.*

Nous avons donc montré qu'il est possible de construire l'arbre d'évaluation d'une expression quelconque correctement parenthésée de longueur n en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs. Dans la partie suivante, nous allons commencer à montrer comment évaluer l'expression en temps $O(\log n)$ avec $O(n)$ processeurs. Il est possible de réaliser cette évaluation en temps $O(\log n)$ avec $O(\frac{n}{\log n})$ processeurs mais cela sort du cadre de ce partiel.

2.3 Évaluation de l'expression

La figure 5 présente brièvement le schéma d'évaluation que nous allons utiliser. L'idée est d'associer à chaque nœud (i.e., chaque opérateur) de l'arbre une fonction et d'utiliser une forme de saut de pointeur (afin d'obtenir le temps en $O(\log n)$) pour évaluer l'expression.

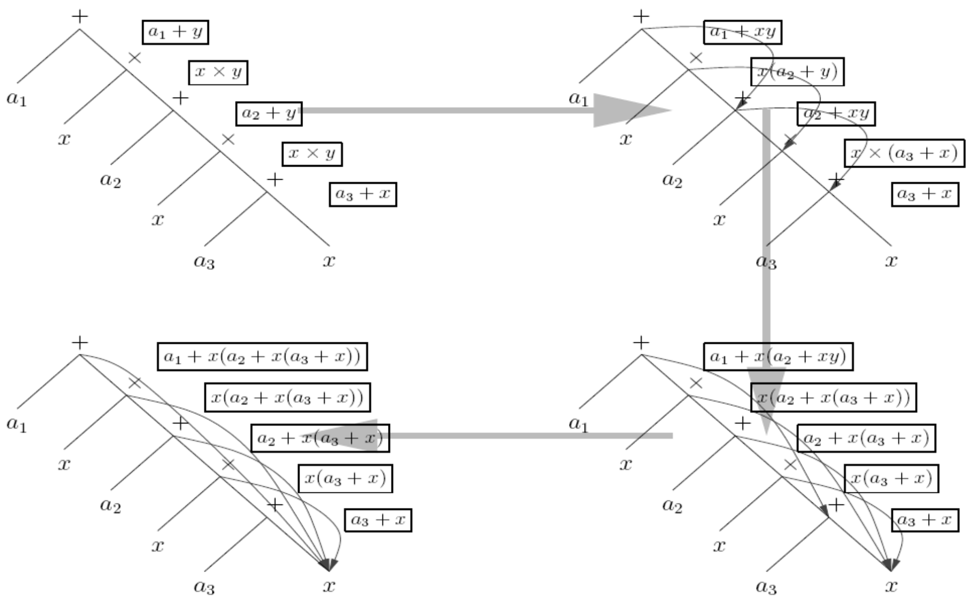


FIG. 5 – Arbre d'évaluation symbolique du polynôme.

Le principe de l'évaluation est le suivant. À chaque nœud v de l'arbre d'évaluation, on va associer un nœud $cond(v)$. Au départ, pour tout nœud v on pose $cond(v) = v$. Un nœud est dit **marqué** si on sait évaluer la valeur de l'expression du sous-arbre enraciné en ce nœud. Dès qu'un des fils d'un nœud est marqué, on dit que le nœud est **actif**. À tout moment, si un nœud est actif, la connaissance de la valeur de $cond(v)$ doit permettre de calculer la valeur de v. Pour cela, on associe à chaque nœud actif une fonction de la forme $f_v(x) = \frac{ax+b}{cx+d}$, où x représente la valeur de $cond(v)$. Pour obtenir une évaluation rapide de l'expression, l'idée est de faire du **saut** de pointeur sur $cond(v)$, ce qui nécessite de composer les fonctions associées au nœud.

Question 11. *Évaluer l'arbre de la figure 4(b) suivant ce principe.*

Question 12.

- *Proposer des fonctions Activer, Sauter et Marquer.*
- *Montrer qu'il est possible de calculer la composée de deux fonctions associées à deux nœuds en temps constant et qu'il est possible d'évaluer la fonction en temps constant dès que $cond(v)$ est marqué.*