EFFICIENT PARALLEL EVALUATION OF STRAIGHT-LINE CODE AND ARITHMETIC CIRCUITS by Gary L. Miller, Vijaya Ramachandran, AND Erich Kaltofen

Anaël Grandjean

6 Décembre 2012

Table des matières

1 Introduction

Cet article traite du problème du calcul rapide de programmes staight-line. En particulier de méthodes de parallélisation de ce calcul. Pour cela les auteurs proposent un algorithme qui permet de résoudre les circuits arithmétiques, modèle équivalent aux programmes straight-line. On a alors le résultat suivant :

Théorème 1.1. *Un circuit de degré d et de taille n, défini sur un semi-anneau commutatif R, peut être résolu en temps $O(log(n)log(nd))$ avec $M(n)$ processeurs.*

Ce théorème nécessite les définitions suivantes :

Définition 1.2. $M(n)$ est le nombre de processeurs nécessaires pour effectuer la multiplication de deux matrices $n \times n$ en temps $O(log(n))$ sur le semi-anneau R.

On peut remarquer que $n^2 = o(M(n))$ et que $M(n) = O(n^{2.49})$.

Définition 1.3. Le degré d'un nœud d'un circuit arithmétique est définit par induction :
- Le degré d'une feuille est 1.
- Le degré d'un nœud additif est le maximum des degrés de ses fils.
- Le degré d'un nœud multiplicatif est la somme des degrés de ses fils.

Le degré d'un circuit est le maximum des degrés de ses nœuds.

2 Les procédures

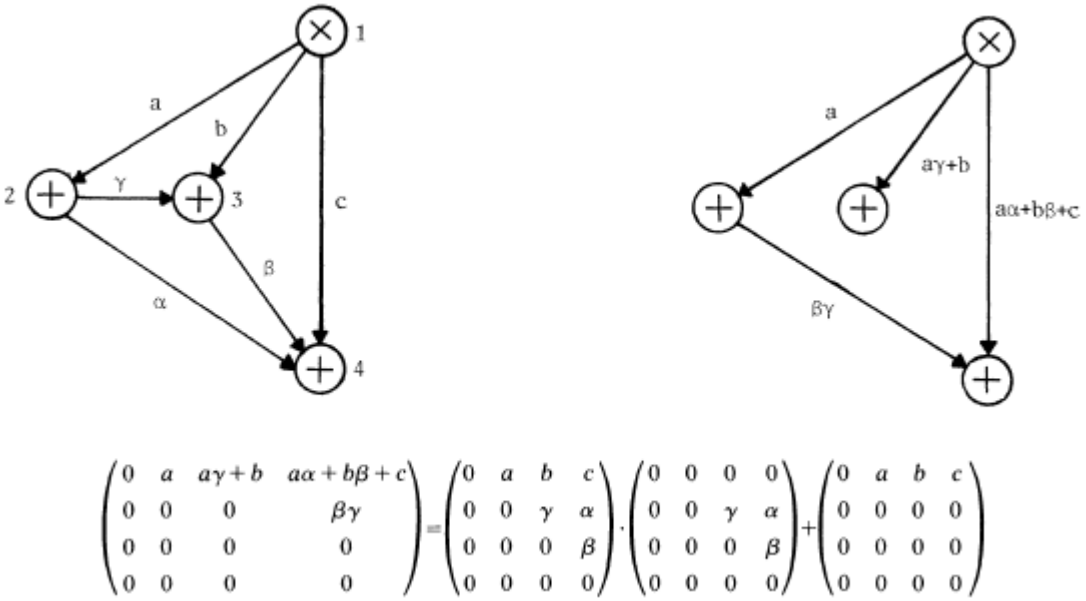
L'algorithme proposé dans cet article se base sur trois procédures qui transforment les circuits arithmétiques.

2.1 Matrix Multiply (MM)

Pour définir cette première procédure il est nécessaire d'avoir une vision matricielle d'un circuit arithmétique. En effet un circuit est caractérisé par une matrice d'adjacence. Cette matrice peut être prise triangulaire supérieure quitte à renommer les nœuds. On définit alors les matrices suivantes a partir de la matrice d'adjacence U d'un circuit arithmétique :
- $U(+,+) = U_{ij}$ si v_i et v_j sont des nœuds additifs, 0 sinon
- $U(X,+) = U_{ij}$ si v_j est un nœud d'addition, 0 sinon
- $U(X,X) = U_{ij}$ si v_j ou v_i n'est pas un nœud additif, 0 sinon

On peut noter qu'avec les définitions précédentes, $U(+,+) + U(X,X) = U$.
Enfin, l'opération de Matrix Multiply (MM) est la suivante :
$MM(U) = U(X,+) \times U(+,+) + U(X,X)$
Cette opération "court-circuite" les arêtes de $U(+,+)$ en les combinant avec les arêtes de $U(X,+)$ lorsque cela est possible. La figure 1 illustre cette procédure par un exemple.

La procédure MM peut être effectuée en temps $O(log(n))$ avec $M(n)$ processeurs, par définition de $M(n)$.



$$\begin{pmatrix} 0 & a & a\gamma+b & a\alpha+b\beta+c \\ 0 & 0 & 0 & \beta\gamma \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} = \begin{pmatrix} 0 & a & b & c \\ 0 & 0 & \gamma & \alpha \\ 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \end{pmatrix} \cdot \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & \gamma & \alpha \\ 0 & 0 & 0 & \beta \\ 0 & 0 & 0 & 0 \end{pmatrix} + \begin{pmatrix} 0 & a & b & c \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

FIGURE 1 – exemple d'application de MM

2.2 $Eval_+$

Cette procédure est plus nettement plus simple que la précédente, $Eval_+$ se contente d'évaluer les nœuds additifs dont les fils sont des feuilles puis de supprimer ces feuilles. Pour effectuer cette procédure en temps $O(log(n))$ il suffit d'utiliser $O(n^2)$ processeurs. Cette étape de calcul est donc moins couteuse que l'étape précédente.

2.3 $Eval_x$

Cette procédure se décompose en deux étapes. C'est la deuxième étape qui fait la particularité de l'algorithme proposé dans cet article.
La première étape consiste en l'évaluation des nœuds multiplicatifs dont les fils sont des feuilles, puis l'élimination de ces feuilles. Comme pour $Eval_+$ cette étape se calcule en temps $O(log(n))$ avec $O(n^2)$ processeurs.
La deuxième étape va "court-circuiter" les nœuds multiplicatifs dont un seul fils est une feuille. On rappelle ici que les nœuds multiplicatifs, contrairement aux nœuds additifs sont limités à deux fils. En effet une arête partant d'un nœud multiplicatif dont un fils est une feuille peut se traduire par une arête partant du fils qui n'est pas une feuille, le poids de l'arête représentant le passage par ce nœud multiplicatif. La figure 2 illustre les effets de la procédure $Eval_x$. Cette deuxième étape nécessite autant de calcul que l'étape précédente, à une constante près.

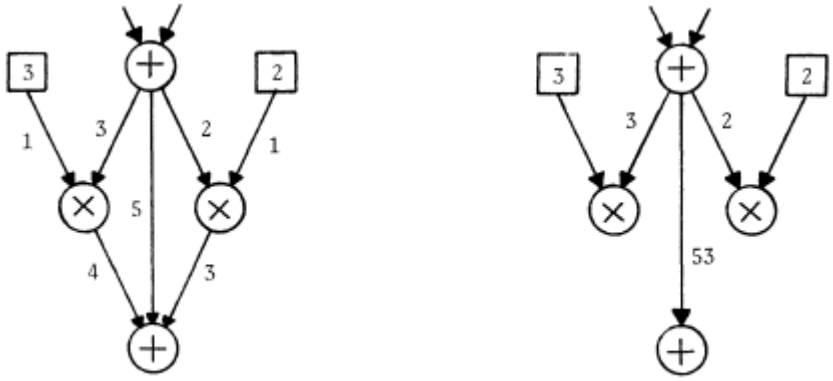


FIGURE 2 – exemple d'application de $Eval_+$

3 L'algorithme

Une fois ces procédures présentées, l'algorithme est assez simple. Il suffit d'appliquer successivement et récursivement MM, $Eval_+$ et $Eval_x$ au circuit. L'application successive de ces trois procédures est appelé $Phase$ et est illustré dans la figure 3.

3.1 Idée de l'algorithme

Le principal changement par rapport aux algorithmes existants pour évaluer des polynômes réside dans la deuxième partie de la procédure $Eval_x$. En effet cette étape permet en quelque sorte de factoriser, et ainsi d'avancer le calcul plus vite au cœur du circuit, permettant ainsi une meilleure parallélisation du calcul. Pour prouver l'efficacité de cet algorithme, les auteurs prouvent d'abord la justesse de leurs procédures. Chaque procédure renvoie un circuit de même valeur que le circuit initial. Cette preuve se fait par induction structurelle sur le circuit U et sera détaillée dans la partie suivante. Pour borner la complexité temporelle de l'algorithme l'article se base sur une nouvelle définition de la hauteur d'un circuit, qui permet de borner efficacement le nombre d'appels aux procédures définies précédemment.

Définition 3.1. La hauteur d'un circuit arithmétique est définie par induction :
- Une feuille a pour hauteur 1
- Un noeud multiplicatif a pour hauteur la somme des hauteurs de ses fils
- Un noeud additif v a pour hauteur le maximum entre $a + 1/2$ et m, où a est le maximum des hauteurs des fils de v qui sont des nœuds additifs, et m le maximum des hauteurs de ses autres fils.

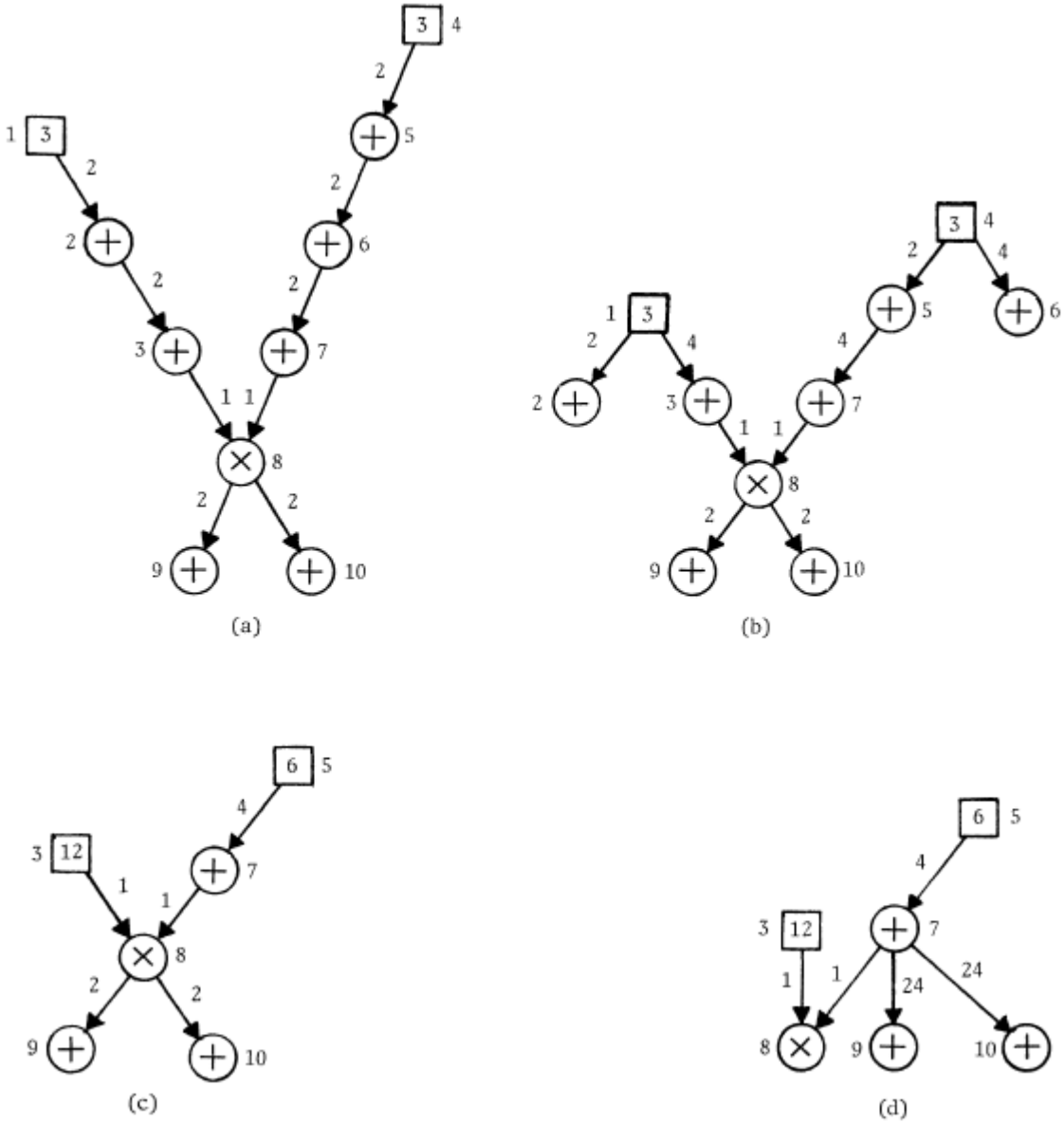


FIGURE 3 – exemple d'application de MM puis $Eval_+$ puis $Eval_x$

3.2 Résultats préliminaires

Lemme 3.2. *Les procédures MM, $Eval_+$ et $Eval_x$ appliquées à un circuit arithmétique renvoient des circuits arithmétiques de même valeur.*

Démonstration. La démonstration de ce théorème se fait par induction sur la taille de U. Elle ne figure pas dans l'article mais utilise les propriétés de commutativité et d'associativité du semi anneau R. Par ailleurs cette preuve est assez simple bien que fastidieuse. □

Théorème 3.3. *Soit U un circuit arithmétique de taille n, de degré d et contenant e arête du type $(+,+)$. Alors la hauteur de U est au plus $\frac{1}{2}ed + d$*

Démonstration. Ce théorème se prouve lui aussi par induction sur la taille de U. Cette induction est suivie par une divergence de cas selon la nature de la racine de U. La hauteur est effectivement liée fortement liée au degré puisque la seule différence entre ces deux notions est la constante $\frac{1}{2}$ qui arrive lors d'arêtes $(+,+)$. □

Ces résultats préliminaires permettent de simplifier les preuves des théorèmes suivants qui portent directement sur les performances de l'algorithme proposé dans cet article.

3.3 Propriétés de l'algorithme

Lemme 3.4. *Soit U un circuit arithmétique. Soit U' son image par la transformation $Phase$. Soit v' un nœud de U' qui n'est ni une feuille ni un nœud d'output, soit v son antécédent dans U. Alors sa hauteur est au plus la moitié de la hauteur de v.*

Idée de preuve :
Ce lemme se prouve encore une fois par induction sur la taille de U. Ce résultat est dû à l'efficacité des procédure MM et $Eval_x$ vis-à-vis de toutes les feuilles. En effet ces procédures permettent d'avancer le calcul au niveau des nœuds dont seul un fils est une feuille. Ainsi chaque branche voit sa hauteur diminuer, ce qui a pour effet une diminution drastique de la hauteur pour chaque nœud du circuit.

Théorème 3.5. *Soit U un circuit arithmétique de hauteur h, alors après au plus $\lfloor log_2(h) \rfloor + 1$ applications de la procédure $Phase$, tous les nœuds de U sont évalués.*

Ce théorème découle immédiatement du lemme précédent. Pour obtenir la borne exacte il faut examiner plus en profondeur l'algorithme et en faire une analyse pointue. Cette analyse n'est pas présentée explicitement dans l'article.

Théorème 3.6. *Soit U un circuit arithmétique de taille n et de degré d, alors la valeur de U peut être calculée parallèlement en temps $O(log(n)log(nd))$ avec $O(M(n))$ processeurs.*

On a vu dans la partie 2 que le temps nécessaire à l'application de chaque procédure est borné par $O(log(n))$ avec $O(M(n))$ processeurs. Le théorème précédent donne une borne du nombre d'utilisation de la procédure *Phase* en fonction de la hauteur du circuit. Le théorème 3.3 relie la hauteur au degré et à la taille du circuit. En appliquant ces théorèmes on obtient résultat principal.

4 Conclusion

Cet algorithme se base sur une parallélisation massive de la réduction de circuits arithmétique. En effet l'idée principale de l'article est de réussir à calculer le plus possible avec des nœuds dont les fils ne sont pas des feuilles, et ainsi accélérer les calculs suivants. Contrairement l'algorithme proposé par Valiant, Skyum, Berkowitz et Rackoff, cet algorithme parvient a effectuer le précalcul sur les nœuds profonds en même temps que le calcul sur les nœuds plus proches des feuilles. La limite principal de ce résultat est la restriction aux anneaux commutatifs. En effet la procédure $Eval_x$ n'est pas correcte dans un anneau non-commutatif, comme par exemple l'espace des matrices carrées. L'autre limite que présente cet algorithme est la restriction aux circuits sans divisions.