Un langage pour la programmation fonctionnelle

Armelle Merlin

L'un des enjeux principaux de l'informatique est de faire des calculs de grande taille et d'être capable de prévoir le temps que prendront ces calculs. Pour augmenter la rapidité des calculs, de nombreuses pistes sont explorées. L'une d'entre elle consiste à programmer sur des ordinateurs massivement parallèles. Pour assurer une programmation simple et portable, nous avons étendu le langage fonctionnel CAML à l'aide d'opérations de communication entre processus sur le modèle BSP (Valiant [5]). Ce modèle permet d'évaluer les temps de calcul de façon réaliste et portable. Il prend en compte la taille des données, le nombre de processeurs utilisés ainsi que la latence et la bande passante du réseau qui les relie. Pour rendre ce modèle de performance effectif pour le programmeur, notre extension de CAML doit être formellement modélisée à l'aide d'une machine abstraite ou machine à pile qui décrit pécisément les calculs parallèles engendrés par un programme. Nous présenterons ici les différents paradigmes de la programmation parallèle, le modèle BSP, notre extension de CAML appelée BSMLlib, les machines abstraites et les résultats obtenus du point de vue de la prévisibilité des performances.

Les ordinateurs parallèles sont des machines qui comportent une architecture parallèle, constituée de plusieurs processeurs identiques concourant tous au traitement d'une seule application. Le terme de massivement parallèle est couramment utilisé lorsque le système à architecture parallèle comporte de quelques dizaines de processeursà plusieurs dizaines de milliers de processeurs, ou plus encore. Ce systèmes sont souvent scalables, c'est-à-dire que leur puissance est extensible, dans une certaine plage, à peu près proportionnellement au nombre de leurs processeurs. Plusieurs architectures types caractérisent ce domaine :

- Les architectures M.I.M.D. ou encore à mémoire distribuée. Elles concernent les calculateurs où chaque processeur dispose d'une mémoire de données et de programme indépendante (Multiple Instructions Multiple Data), les échanges interprocesseurs s'effectuant par passage de message.
- Les architectures SIMD (Single Instruction Multiple Data). Elles proposent une mémoire de programmation centralisée pour tous les pro-

cesseurs qui exécutent donc de manière synchrone le même programme sur des données différentes.

Les éléments constitutifs d'un système massivement parallèle sont : les processeurs, le réseau qui relie ces processeurs, la mémoire plus ou moins répartie et la partie logiciel (langages et compilateurs pour la réalisation d'algorithmes de traitements en parallèle des tâches, systèmes d'exploitation...). Ces systèmes font intervenir des techniques de communication entre les différents éléments du système, de cohérence mémoire, de parallélisation et d'allocation des tâches, et de systèmes d'exploitation répartis. La machine PRAM (Parallel Random Access Memory) modélise, en le simplifiant, le fonctionnement d'un architecture MIMD à mémoire partagée.

Le modèle *Bulk-Synchronous Parallelism* (BSP) est un modèle de programmation parallèle introduit par Valiant [5] pour offrir un niveau d'abstraction comparable aux modèles PRAM tout en permettant des performances prévisibles et portables sur une large variété d'architectures. Un ordinateur BSP contient un ensemble de paires processeur-mémoire, un réseau de communication permettant l'échange de messages inter-processeur et une unité de synchronisation globale qui exécute des demandes collectives de barrières de synchronisation. Ses performances sont caractérisées par trois paramètres : le nombre p de paires de processeur-mémoire, le temps l nécessaire à une barrière de synchronisation et le temps g nécessaire à une 1-relation (phase de communication où chaque processeur envoie ou reçoit au plus un mot). Pour n'importe quel h le réseau peut réaliser une h-relation, c'est-à-dire une phase de communication où chaque processeur envoie ou reçoit au plus h mots, en temps gh. Un programme BSP est exécuté comme une séquence de *super-étapes*, chacune étant au plus divisée en trois phases successives et logiquement disjointes. Pendant la première phase, chaque processeur utilise ses données locales pour du calcul séquentiel et pour demander des transferts de données vers ou depuis d'autres noeuds. Pendant la seconde phase, le réseau effectue les transferts de données demandées. Pendant la troisème phase, une barrière de synchronisation se produit, rendant disponibles pour la super-étape suivante les données tranférées. Le temps d'exécution d'une super-étape s est ainsi la somme du maximum des temps de calculs locaux, du temps de communication des données et du temps de synchronisation globale :

$$Time(s) = \max_{i:processeur} w_i^{(s)} + \max_{i:processeur} h_i^{(s)} * g + l$$

où $w_i^{(s)}$ = temps de calcul local du processeur i durant la super-étape s et $h_i^{(s)} = \max\{h_{i+}^{(s)}, h_{i-}^{(s)}\}$ où $h_{i+}^{(s)}$ (resp. $h_{i-}^{(s)}$) est le nombre de mots transmis (resp. reçus) par le processeur i durant la super-étape s. Le temps d'exécution $\sum_s Time(s)$ d'un programme BSP composé de S super-étapes est la somme des trois termes : $W + H * g + S * l$ où $W = \sum_s \max_i wi^{(s)}$ et $H = \sum_s \max_i h_i^{(s)}$.

Le langage CAML, proche du λ-calcul, a été étendu [3] à l'aide d'opérations BSP. Cette extension, le BS-λ, présenté succintement ici, est confluent.

Soient $\dot{\mathcal{V}}$ l'ensemble des variables *locales* et $\bar{\mathcal{V}}$ l'ensemble des variables *globales*. Le point $\dot{}$ symbolise un processeur précis du réseau, la barre $\bar{}$ symbolise le réseau tout entier. Les termes *locaux* e sont des λ-termes représentant des valeurs ou des programmes stockés dans la mémoire locale d'un processeur. L'ensemble $\dot{\mathcal{T}}$ des termes locaux est donné par la grammaire suivante :

$$e \quad ::= \quad \dot{x} \mid e\, e \mid \lambda \dot{x}.\ e \mid c$$

où $\dot{x}$ dénote une variable locale arbitraire. Les termes *globaux* représentent des fonctions d'un ensemble fixé de processeurs vers des valeurs. Le terme $\pi\, e$ représente un champ de données dont les valeurs sont données par la fonction e. L'ensemble $\bar{\mathcal{T}}$ des termes globaux est donné par la grammaire suivante :

$$\begin{array}{rcl} E & ::= & \bar{x} \mid E\, E \mid E\, e \mid \lambda \bar{x}.\ E \mid \lambda \dot{x}.\ E \\ & \mid & \pi\ e \mid E \# E \mid E ? E \mid E \xrightarrow{e} E, E \end{array}$$

La dénotation de $\pi\, e$ a, au processeur n_i, la valeur de $e\, n_i$. Les formes $E_1 \# E_2$ et $E_1 ? E_2$ sont appelées application parallèle (*apply-par*) et *get* respectivement. Apply-par représente l'application point à point d'un champ de fonctions à un champ de valeurs (phase de calcul pur d'une super-étape BSP). Get représente la phase de communication d'une super-étape BSP : un échange collectif de données avec une barrière de synchronisation. dernière forme de terme global définit la conditionnelle globale synchrone. La signification de $E_1 \xrightarrow{e} E_2, E_3$ est celle de E_2 (resp. E_3) si le champ de données dénoté par E_1 a la valeur T (resp. F) au processeur de nom dénoté par e.

Dans la sémantique équationnelle l'égalité des termes globaux est définie par des règles basées sur la syntaxe et des règles de contexte qui déterminent l'applicabilité des premières.

Le BSML est un langage purement fonctionnel de données parallèles conçu pour programmer des algorithmes BSP. La BSMLlib est l'implantation du formalisme BSλ.

Une machine abstraite est le lien entre le langage compilé et son exécution. Le langage est d'abord traduit sous forme d'une suite de commandes (c'est-à-dire compilé). La machine exécute les commandes à l'aide de transitions. Pour obtenir un langage dont l'exécution est mieux adaptée au parallélisme, il parrait intéressant d'étendre une machine abstraite spécifique. La SECD, [2], est une machine entièrement documentée. Son extension pour le langage BSλ simplement typé [4] a été implanté et testé. Cela a permis de vérifier que les constantes peuvent être calculées *a priori*. La BSP-CAM, extension de la CAM, [1], permet d'avoir une gestion des environnements plus précise. Dans les deux cas, il existe une machine abstraite en chaque processeur qui exécute ses transitions de manière synchrone uniquement pour les opérations BSP.

Le but du projet est une réalisation complète d'une version BSP du langage Caml. Les différentes étapes réalisées à ce jour ont permis de vérifier la bonne structure du langage et ont mis en évidence les difficultés à surmonter pour une implantation parallèle complète et efficace.

Références

[1] G. Cousineau, P. L. Curien, and M. Mauny. The categorical abstract machine. In J.-P. Jouannaud, editor, *Functional Programming Languages and Computer Architecture*, pages 50–64. Springer-Verlag, Berlin, DE, 1985. Lecture Notes in Computer Science 201Proceedings of. Conference at Nancy.

[2] Peter J. Landin. The mechanical evaluation of expressions. *The Computer Journal*, 6(4):308–320, January 1964.

[3] F. Loulergue. *Conception de langages fonctionnels pour la programmation massivement parallèle.* Thèse de Doctorat d'Université, Université d'Orléans, janvier 2000.

[4] A. Merlin. Bsλ simplement typé: Typage et sémantique naturelle. Rapport de DEA, Université d'Orléans, Septembre 2000.

[5] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 22(8):103–111, August 1990.

Armelle Merlin
Laboratoire d'Informatique Fondamentale d'Orléans
BP 6759
45067 Orléans Cedex 2
France
`merlin@lifo.univ-orleans.fr`