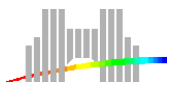
Parallélisation automatique
L'avenir de la parallélisation automatique

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr

16 décembre 2008



Ce que l'on trouve sur le Web

... en $n + 1$ exemplaires :

```
Second International Workshop on Multicore Software Engineering

Aims and Scope
==============

With the emergence of multicore computers, software engineers face
the challenge of parallelizing performance-critical applications of
all sorts. Compared to sequential applications, our repertoire of
tools and methods for cost-effectively developing reliable, fault
tolerant, robust parallel applications is spotty. The purpose of this
workshop is to bring together researchers and practitioners with
diverse backgrounds in order to advance the state of the art in
software engineering for multi/manycore parallel applications.

The workshop is aimed at making parallelism available to a wide
range of applications using systematic software engineering
methodology.

    * Parallel patterns
    * Frameworks and libraries for multicore software
    * Parallel software architectures
    * Programming languages/models for multicore software
    * Compilers for parallelism
    * Testing and debugging parallel applications
    * Parallel algorithms and data structures
    * Software reengineering for parallelism
    * Autotuning
    * Operating system support, scheduling
    * Development environments for multicore software
    * Experience reports from research or industrial projects
```

Pourquoi ?

Le matériel a des problèmes.

- La fréquence d'horloge n'augmente plus
- La puissance consommée devient prohibitive
- Le parallélisme caché atteint ses limites : parallélisation automatique, par le hard, d'un bloc de base, en ne tenant compte que des accès aux registres pour calculer les dépendances
  - taille du bloc de base (5 instructions)
  - nombre de registres
- La densité augmente encore, mais au ralenti
- Il est possible (mais difficile) d'augmenter la taille du chip

Comment ?

- Il faut bien utiliser tous ces transistors
- Le parallélisme fait baisser la consommation électrique
- Mémoire partagée : multicore / multithread, limité par la bande passante mémoire
- Mémoire distribuée : cluster / grappe / grille /cloud, limité par la bande passante du réseau

Et le logiciel ?

Il n'y a pas, pour les systèmes parallèles, de modèle unique de programmation :

- Threads
- Passage de message
- Parallélisme de données
- Parallélisme de flot
- Diviser pour régner
- etc. etc.

Threads

(POSIX, Java)

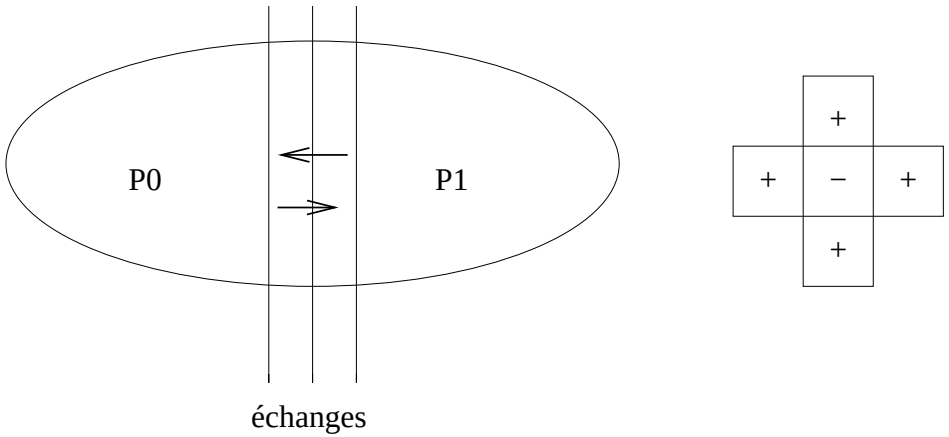
- Un thread = une fonction qui retourne *avant* d'avoir terminé son travail
- L'appelé et l'appelant fonctionnent en parallèlisme (réel ou simulé)
- Tous les threads d'une application partagent la même mémoire (différence avec les processus Unix), d'où possibilité de conflits = violation d'une dépendance
- Nécessité d'instructions de synchronisation (sections critiques, moniteurs, sémaphores) donc possibilité d'étreintes mortelles
- Problème de la réentrance des bibliothèques
- Difficulté de la mise au point : les bugs sont difficiles à reproduire : non déterminisme

Passage de message

- Sur chaque processeurs, un programme indépendant, qui échange des messages avec les autres processeurs
- primitives `send` / `receive`, qui servent à la fois au transfert de données et à la synchronisation
- Les couches de bas niveau du système (sockets) prennent en charge les aspects de bas niveau (émission, réception, mise en ordre, traitement des erreurs)
- Mais le programmeur doit prendre en charge l'organisation des échanges (protocole)
- Difficile à programmer, sauf si on peut s'appuyer sur une analogie physique

Un exemple : la décomposition de domaine

De nombreux problèmes (simulation physique, météo, traitement d'images) s'effectuent sur une grille de points.



- Le calcul élémentaire implique un point et ses voisins sur la grille
- On découpe en sous-domaine ; chaque processeur traite un sous-domaine
- Les calculs sont indépendants, sauf au voisinage de la frontière
- A chaque itération, les processeurs échangent les données de leur zone frontière

Quelques solutions

- Laisser faire les éditeurs de logiciels
- Inventer des langages de programmation paralléle
- Parallélisation automatique statique
- Parallélisation dynamique spéculative

Laisser faire les éditeurs

De nombreux logiciels sont déja parallélisés, ou en cours de parallélisation :

- Systèmes de Gestion de Bases de données
- Jeux (partie graphique, car les cartes graphiques – GPU – sont de puissants superordinateurs parallèles)
- Serveurs Web
- Google, Amazon, eBay, etc
- Bibliothèques de calcul numérique (algèbre linéaire dense et creuse, FFT, traitement du signal)

Mais cela ne résoud pas les problèmes des utilisateurs occasionels et des sociétés de service.

Langages parallèles

Chaque langage correspond à un type d'architecture et à un type d'application :

- Langages vectoriels (APL, HPF, Fortran 95)
- Langages à flot de données (StreamIt)
- Langages fonctionnels parallèles (dérivés de Haskell et Ocaml), patrons de parallélisme
- Extensions parallèles de langages classiques (Open MP)

Le problème : la portabilité !

Parallélisation dynamique

Sur l'exemple d'une boucle et des dépendances PP :

```
for(i=0; i<n; i++)
  A[f(i)] = ...;
```

- On prépare un tableau de bits `A_w` de même taille que A et initialement à zéro
- on exécute la boucle en parallèle ...
- ... mais on ajoute les instructions :

```
if(A_w[f(i)] == 1)
    arréter tout
else A_w[f(i)] = 1;
```

- Si on détecte une dépendance, on recommence en séquentiel ; on doit préserver l'état antérieur de A (utilisation du cache)
- On traite de la même façon les autres types de dépendances, mais c'est un peu plus compliqué.

Les problèmes de la parallélisation automatique

- Il reste quelques problèmes à résoudre dans le cadre polyédrique
  - Prendre en compte la localité
  - Prendre en compte les contraintes de ressource
  - Paralléliser les réductions
- Le modèle polyédrique est fragile
- Il faudrait étendre les méthodes d'analyse de programme (qui sont également polyèdrique)
- Est-il possible de combiner parallélisation automatique et manuelle ?
- Existe-t-il d'autres modèles pour la parallélisation ?

Contraintes de ressources

La version la plus simple :
Trouver un ordonnancement dont les fronts
$(\mathcal{F}(t) = \{u \in E \mid \theta(u) = t\})$ sont de taille bornée.

- Sur un gros ordinateur, la taille du front n'a pas d'importance : c'est le système ou la bibliothèque d'exécution qui répartit le travail entre les processeurs disponibles
- Sur un processeur embarqué (ou en matériel) il n'y a pas de bibliothèque d'exécution. Le problème doit être traité à la compilation
- En général, il faut ralentir l'ordonnancement. Quelques suggestions :
  - Introduire des écritures fictives
  - Introduire des dépendances fictives
  - Calculer un placement qui respecte les dépendances et le tuiler
- Le problème est encore largement ouvert.

Fragilité des parallléliseurs automatiques

Ne savent pas en général traiter :

- Les indices compliqués (tableaux par bloc, FFT, histogrammes)
- Contrôle dynamique (tests, boucles `while`)
- Fonctions et procédures (récursives ou non)
- Structures de données dynamiques

Traiter n'importe quel programme

Un compilateur ordinaire compile n'importe quel programme *correct*.
Il faudrait qu'un paralléliseur en fasse autant, même s'il ne trouve pas de parallélisme.

- Il existe toujours un ordonnancement qui reproduit l'exécution séquentielle. On peut l'utiliser quand on ne sait pas faire mieux
- Autre solution : déclarer qu'il y a dépendance quand on ne peut pas prouver le contraire. Mais ça risque de faire échouer l'ordonnanceur
- Le problème est le traitement du contrôle.

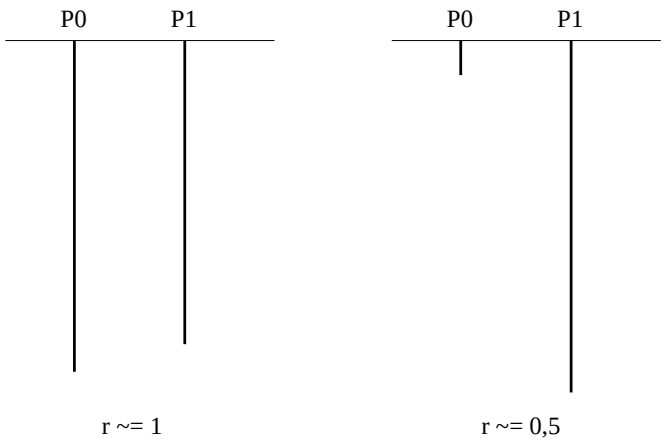
Le problème du contrôle

On peut toujours traiter les instructions de contrôle de façon conservative :

- Tests : faire comme si les deux branches étaient prises
- Boucle `while` et boucle `for` complexe : remplacer par une boucle infinie
- Fonction : calculer l'effet de la fonction sur ses arguments (aggrégation)

Problème : équilibrage de charge.

Equilibrage de charge

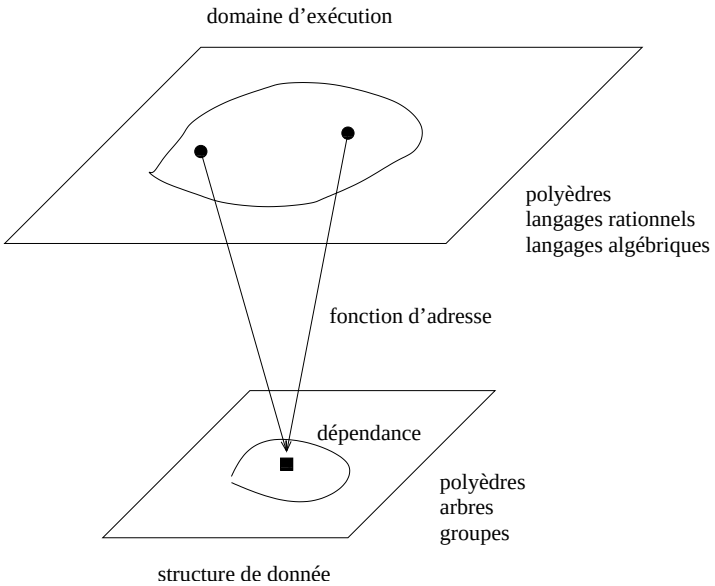


- Le parallélisme ne sert à rien si la charge des processeurs n'est pas équilibrée
- Facile si toutes les opérations ont approximativement la même durée
- Difficile dans le cas contraire. Recours presque obligatoire aux techniques dynamiques (Guided Self-Scheduling)

Améliorer les méthodes d'analyse

- Analyses exactes au dela du linéaire
  - Techniques empruntées au calcul algébrique, à l'analyse et à l'intelligence artificielle
  - Deux défis : le traitement des matrices par bloc (polynomes)
  - La FFT (exponentielles)
- Analyses approximatives, emprunts à l'interprétation abstraite
- Exemple : recherche de contraintes linéaires implicites par l'algorithme de Cousot-Halbwach

Le modèle polyédrique est-il seul au monde ?



Identifier :

- L'espace de contrôle D et l'espace des données (ou des adresses) A
- Caractériser l'ordre d'exécution séquentiel $<_{\text{seq}} \subseteq D \times D$
- Calculer les fonctions d'adressage $f : D \rightarrow A$ (ou les relations d'adressage)
- Une dépendance :

$$\exists u, v \in D : u <_{\text{seq}} v, f(u) = f(v)$$

Combiner parallélisation automatique et parallélisation manuelle

- Une tentative : les pragmas (commentaires actifs)
  - Demander au programmeur de signaler les boucles parallèles ou les sections de code parallèles (Open MP)
  - Mais les programmeurs ne savent pas ...
- Distinguer suivant le grain de parallélisme :
  - Gros grain : deux compilations simultanées : parallélisation manuelle
  - Grain fin : addition de deux vecteur : parallélisation automatique
- Mettre de la sémantique :
  - Distinguer (manuellement) des *types de parallélisme*
  - Appliquer des algorithmes de compilation appropriés