Master Informatique Fondamentale - M1 Compilation

Où va la Compilation?

Paul Feautrier

ENS de Lyon
Paul.Feautrier@ens-lyon.fr
perso.ens-lyon.fr/paul.feautrier

21 avril 2009

L'Environnement / I

Des applications de plus en plus gourmandes.

- Application de l'Intelligence artificielle :
  - Reconnaissance de la parole, de l'écriture, analyse de scénes
  - "Sémantique"
- Grande variabilité des applications (téléphonie, télévision, jeux)
- Grille

L'Environnement / II

Des performances unitaires qui plafonnent.

- Pour des raisons de dissipation, la fréquence d'horloge est bloquée vers 3Ghz
- La densité continue à croire, mais moins vite, et il est toujours possible d'augmenter la surface (wafer-scale, puces 3D)
- Le parallélisme implicite a atteint ses limites
- Donc, parallélisme explicite : multi-cœurs, systèmes et réseaux sur puce.

Référence : M. Hall, D. Padua, K. Pingali : *Compiler Research, the Next 50 Years*, CACM, 52-2, Feb 2009, pp 60–67.

Outils de programmation parallèle

Les outils existants (threads POSIX ou Java, OpenMP, MPI) sont de très bas niveau et difficiles à utiliser.
De plus, chacun est adapté à une architecture, et si les programmes sont portables, les performances ne le sont pas !

- Parallélisation automatique : élargir le champ d'application
- Parallélisation manuelle : prendre en compte les concepts et les habitudes des utilisateurs (parallélisation "sémantique").
  Exemple : les réseaux de processeurs pour les applications de type "calcul au fil de l'eau" (*streaming*)
- Parallélisation mixte : partage des compétences entre le paralléliseur et le programmeur (grain fin / gros grain)
- Parallélisation dynamique (à l'exécution)

Les langages métiers (*Domain Specific Language*)

Au lieu d'utiliser un langage généraliste, développer un langage dédié à un certain type d'application et écrire son compilateur. Exemple, les langages d'interrogation des bases de données (SQL), les langages pour le calcul formel, les langages de description du matériel.

- Il est possible d'incorporer dans le langage la "sémantique" du domaine d'application, et d'obtenir des programmes plus compacts, plus sûrs et plus efficace (Exemple : comparer un calcul algébrique écrit en Java à sa version Maple).
- Difficulté : il faut écrire le compilateur. Deux approches :
  - Etendre un langage existant (Exemple : C++ $\rightarrow$ SystemC)
  - Inventer un *Compilateur de Compilateur* ou méta-compilateur

Fiabilité / Sécurite

Les compilateurs peuvent-ils participer à la quète du logiciel fiable ?

- Prouver que le compilateur est sans erreur. Permet de vérifier le logiciel à haut niveau et non en langage machine
- Incorporer des outils de vérification (analyse statique, preuve)
- Incorporer des outils d'authentification (signatures cryptographique, voir le compilateur Java)
- Détection des points d'attaque.

La Consommation Electrique

Critique pour les systèmes embarqués (batteries), pour les systèmes à récupération d'énergie, et même pour les équipements de bureau. Est-il possible de construire un compilateur orienté "économies d'énergie" ?

- De nombreuses optimisations classiques engendrent des économies d'énergie (Exemple : optimiser la localité)
- En existe-t-il d'autres ?
  - Ralentir le processeur en fonction de l'activité
  - Ne pas alimenter les blocs qui ne servent pas (exemple : l'opérateur flottant)
- Interaction avec le système d'exploitation.

Un Compilateur de Recherche

- Il est difficile de justifier une nouveauté en compilation sans présenter des mesures.
- Si les mesures sont prises sur un compilateur rudimentaire, elles ont peu de chances de battre `gcc` ou le compilateur d'IBM !
- Il existe des compilateurs de recherche : SUIF, Open64, LLVM, Rose, etc. Ils ne sont pas encore assez développés.
- Un espoir : il existe une version instrumentée de `gcc`. `http ://ctuning.org`