Défis en compilation, horizon 2025

Florian Brandner, Albert Cohen, Alain Darte, Paul Feautrier
Grigori Fursin, Laure Gonnord, Sid Touati

13 octobre 2014

1 Contexte

La communauté compilation, de part l'émergence des processeurs multi-coeurs qui équipent tous les systèmes informatiques, des smartphones jusqu'aux accélérateurs de calcul (GPUs, FPGAs), ainsi que les super-calculateurs des centres de calcul a vu émerger de nouveaux problèmes tant matériels que logiciels. Le parallélisme est maintenant à la portée de tous, mais sans qu'il soit accessible à tous de part sa complexité.

2 Problématique

Maîtriser la difficulté de programmation de ces architectures tout en offrant une certaine portabilité des codes et des performances, et offrir une meilleure interaction entre les utilisateurs et les compilateurs (collaboration dans les deux sens), nécessite des recherches sur les langages, notamment parallèles, en compilation (analyse et optimisation de codes), et en systèmes d'exploitation (OS). Ces thèmes de recherche nécessitent des travaux parfois théoriques, mais surtout pratiques (développements d'outils, analyses d'applications, analyses de performances et validation expérimentale, en lien avec les domaines applicatifs).

3 Challenges identifiés

Voici trois challenges que nous relevons, parmi ceux de la communauté Compilation/Calcul Haute Performance :

1. **Les langages parallèles** : le défi est bien résumé dans la *road-map* d'HIPEAC, paragraphe 1.4.3 [1].

   Avec la mise sur le marché de plateformes de plus en plus hétérogènes, le gain en performance des applications se fait maintenant en utilisant des processeurs dédiés à des tâches spécifiques. La programmation de tels systèmes logiciels/matériels devient beaucoup plus complexe, et les langages parallèles existants (MPI, openMP) semblent peu adaptés à cette tâche. Pour diminuer le coût du développement et du déploiement de logiciels sur des plateformes spécifiques toujours changeantes, nous avons besoin d'approches et de langages qui permettent l'expression du parallélisme potentiel des applications et la compilation vers une plateforme parallèle spécifique.

   De plus, la complexité et diversité de ces plateformes justifient le développement de langages ou approches de haut niveau (comme X10, Chapel, OpenAcc, CAF, OpenStream, etc.) et d'optimisations au niveau source, en amont des langages natifs cible,

1. `http://www.hipeac.net/system/files/hipeac_roadmap1_0.pdf`

dialectes souvent proches du langage C. Ceci offrira plus de portabilité de performances mais aussi, pour l'utilisateur, une meilleure compréhension des transformations effectuées par le compilateur. L'amélioration de l'interface entre le programmeur et le compilateur est un également un point fondamental à améliorer. L'interaction langage/compilateur/OS/environnement d'exécution (*runtime*) est également un point clé à améliorer, la portabilité de la performance passant par l'adaptation à la plateforme via cette interaction.

2. **La compilation optimisante** pour les performances en temps et en mémoire. Même source, paragraphe 1.4.1. La consommation d'énergie des systèmes (embarqués surtout) est maintenant en grande partie due au mouvement et au stockage des données. Pour s'adapter à cette problématique, les données et leur mouvement doivent être exposées au programmeur, et les compilateurs doivent prendre en compte la trace mémoire et les mouvement des données par exemple entre deux calculs qui s'effectuent sur des unités de calcul différentes. Des approches existent déjà en compilation, notamment en ce qui concerne la compilation vers FPGA, mais ce n'est que le début vers une prise en compte plus générique des contraintes de mémoire. Dans un sujet connexe, la recherche de solutions dont le pire cas peut être garanti (WCET) est rendu encore plus difficile par la présence de parallélisme. À l'heure actuelle, la compilation à performances prédictibles reste un problème largement ouvert.
3. **La validation théorique des analyses et optimisations** que nous réalisons au sein des compilateurs. Les analyses de programmes (séquentiels ou parallèles), les transformations de code, la génération de code pour des architectures précises, doivent être définies et prouvées précisément. Si accompagner les algorithmes de leur preuves reste essentiel, des techniques comme la *preuve assistée* et la *translation validation* permettent maintenant de valider des compilateurs entiers (Compcert). Le challenge ici réside dans l'application plus générale des méthodes formelles, que ce soit pour définir précisement des algorithmes ou pour valider des optimisations dans le cadre parallèle en particulier.
4. **La mesure et la reproductivité des résultats**, notamment dans le domaine de l'optimisation du logiciel. Contrairement aux autres sciences expérimentales (sciences naturelles), la branche expérimentale de l'informatique souffre d'un manque de principe scientifique : la reproductibilité et la vérification des résultats expérimentaux en informatique ne sont pas entrés dans nos habitudes. Dans le domaine de la compilation optimisante en particulier, lorsque des chiffres de performances sont publiés, il est très rare que ces chiffres puissent être vérifiés ou observés par une partie tiers. Définir des méthodes expérimentales de validation statistique de résultats et des modèles présupposés (modèles de coût, modèles de programmation, abstractions ...) reste un défi majeur pour notre communauté, même si certaines conférences de notre domaine commencent à proposer des évaluations expérimentales [2].

Ces problématiques orientées compilation/langages sont aussi liées aux problématiques générales de Génie Logiciel que sont la complexité et l'hétérogénéité des logiciels. La spécificité ici est que la même complexité est à prendre en compte aussi du côté matériel (plateformes hétérogènes, accélérateurs matériels, ...) où se rajoutent en plus des contraintes énergétiques (consommation) et technologiques.

2. `http://evaluate.inf.usi.ch/artifacts`