Vers du prototypage rapide pour le calcul haute-performance

Matthieu Aussal, CMAP

Le prototypage rapide et le calcul haute-performance ne sont pas connus pour faire bon ménage. En effet, ces deux approches du développement logiciel défendent des paradigmes de programmation que beaucoup de principes opposent. Et pourtant...

Commençons par définir ces deux notions. Le prototypage rapide peut être vu comme la conception de maquettes logicielles, répondant le plus souvent a un besoin ponctuel issu de la recherche ou de l'industrie. Ces maquettes sont généralement écrites dans des langages dits de haut-niveau, pour minimiser à la fois les temps de développement et de maintenance (Matlab, Scilab, Python, Julia, etc). Le but est bien souvent de répondre à une question scientifique dans un temps raisonnable, sans attentes particulières de performances ou de génie logiciel. D'autre part, le calcul haute performance a pour objectif de réaliser des outils rapides et puissants, si possible compatibles avec des architectures parallèles (processeurs multi-cœurs, clusters, cartes graphiques, etc.). De nombreux de logiciels qui remplissent ces objectifs sont communément développés dans des langages de bas-niveau (C, C++, Fortran, etc.), et utilisent des protocoles d'échanges comme le MPI, l'OpenMP ou encore le Cuda. Le but est de concevoir des outils robustes, capables de traiter des cas non triviaux dans un temps minimal.

Ainsi, la chaîne de valorisation d'un logiciel suit naturellement le processus suivant :

1. Conception d'un prototype (langage haut-niveau, interprété),
2. Industrialisation de cette maquette (langage bas niveau, compilé),
3. Utilisation du logiciel final (librairie ou interface graphique).

Cependant, les langages de haut-niveau ont atteint ces dernières années une maturité suffisante pour envisager d'éviter la seconde étape, qui est coûteuse à la fois en temps et en ressources. En effet, il est désormais possible de concevoir des prototypes capables de suivre les principes du calcul haute-performance, avec des résultats plutôt prometteurs.

A titre d'exemple, je présenterais les performances d'un nouvel algorithme de la classe des méthodes multipolaires rapides (*FMM* [1, 2, 3]). Nommé *Fast and Furious Method*, j'ai conçu cet algorithme pour s'adapter à la philosophie des langages de haut-niveau, afin de réaliser en Matlab un prototype compatible haute-performance. La FFM réalise donc des convolutions rapides par des noyaux de green, pour tout points $(\mathbf{x}_i)_{i\in[1,N]}$ et $(\mathbf{y}_j)_{j\in[1,N]}$ dans $\mathbb{R}^3$, telles que :

$$u(\mathbf{x}_i) = \sum_{j=1}^{N} G(\mathbf{x}_i, \mathbf{y}_j) f(\mathbf{y}_j),$$

Avec cet algorithme et le prototype Matlab associé, la barre du milliard de points en $\mathbf{x}$ et $\mathbf{y}$, soit l'équivalent de 10^{18} interactions, vient d'être franchie.

Références

[1] Alouges, F., & Aussal, M. (2015). The sparse cardinal sine decomposition and its application for fast numerical convolution. *Numerical Algorithms*, 70(2), 427-448.

[2] Greengard, L., & Rokhlin, V. (1997). A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta numerica*, 6, 229-269.

[3] Hackbusch, W. (1999). A sparse matrix arithmetic based on cal h-matrices. part i: Introduction to $\mathcal{H}$-matrices. *Computing*, 62(2), 89-108.

Matthieu Aussal, CMAP - Ecole polytechnique, route de Saclay, 91128 Palaiseau cedex, France
`matthieu.aussal@polytechnique.edu`