Énumération parallèle d'un monoïde généralisant le tri par bulle

Florent Hivert

Janvier 2016

Contexte

De nombreux algorithmes peuvent mieux se comprendre si on les modélise comme une successions d'opérations élémentaires. Par exemple, le tri par bulle est une succession d'opérations de tri élémentaire du type comparaison/échange. On voit ainsi chaque opération élémentaire comme une fonction de l'ensemble des états possibles de la structure de tableau dans lui même. L'algorithme de tri devient juste une certaine composition de ces fonctions élémentaires. L'une des variantes les plus simples d'un tri par bulle sur 5 valeurs peut par exemple être codée par la suite de fonctions

$$s_0s_1s_2s_3s_4\ s_0s_1s_2s_3\ s_0s_1s_2\ s_0s_1\ s_0$$

où s_i désigne l'opération élémentaire de comparer puis échanger si besoin les valeurs en position i et $i+1$. Il est alors tentant de chercher à comprendre l'opération effectuée par une autre succession comme $s_2s_0s_4s_1s_3$ ou de chercher à décrire toutes les opérations possibles que l'on peut obtenir par une composition d'opérations élémentaires. On dit que l'on décrit le *sous monoïde du monoïde des fonctions engendré par les opérations* s_i. Le cas précis du tri par bulle est bien compris, il s'agit d'un monoïde connu en algèbre sous le nom de monoïde de Hecke. Il contient exactement $n!$.

Problématique

Le travail proposé consiste en une exploration informatique d'un monoïde où l'on rajoute une étape intermédiaire dans l'opération de comparaison/échange. La difficulté est que le calcul devient très vite extrêmement gros tant en mémoire nécessaire qu'en temps de calcul. Par exemple, pour les tableaux à 6 éléments, chaque fonction prend en mémoire de l'ordre de 37 Kio et on a déjà 56883 fonctions. Pour l'étape suivante, chaque fonction

va prendre 320 Kio et on s'attend à avoir plusieurs centaines de millions de fonctions. Il faut donc utiliser des méthodes sophistiquées pour avoir une chance de voir le résultat.

Objectifs du stage

Le travail proposé va donc suivre le plan suivant :

— implantation en C++ des opérations élémentaires ;
— génération exhaustive des éléments du monoïde engendré ;
— optimisation de l'espace mémoire nécessaire en utilisant des caches sur le disque dur ;
— parallélisation multi-coeurs de l'algorithme (utilisation possible de l'extension CilkPlus du langage C++) ;
— s'il reste du temps : modularisation du code pour qu'il soit réutilisable pour d'autres problèmes similaires ;
— autre extension possible s'il reste du temps : distribution sur un cluster de machine.

Pour tout ce développement, on pourra s'inspirer d'un prototype déjà écrit en Python sur le système de calcul SageMath.