L3 MI / PROG
TP#1

2021-W37

Exercice 1 : produit scalaire, compilation

Ce court exercice a pour but de constater l'impact des optimisations de compilation sur un programme très simple, ainsi que de fournir un entraînement à l'utilisation d'un `Makefile`.

Rappels :

- Pour utiliser les types `uint32_t`, `uint64_t`, il faut inclure le fichier `stdint.h` via `#include <stdint.h>`.
- Pour afficher les mêmes types via `printf` sur une machine 64 bits, utilisez les formats d'affichage `%u` et `%lu` ou `%llu` respectivement.
- Pour mesure le temps d'exécution d'un programme `prog`, dans un shell UNIX, faire `time prog`.

Q.1 : Implémentez les fonctions de calcul de produit scalaire (dans $\mathbb{R}$) suivantes :

- `double psd(double x[], double y[], int dim);`
- `float psf(float x[], float y[], int dim);`
- `uint64_t psu64(uint64_t x[], uint64_t y[], int dim);`
- `uint32_t psu32(uint32_t x[], uint32_t y[], int dim);`

dans un fichier `ps.c`, et déclarez les dans un fichier `ps.h` correspondant.

Q.2 : Implémentez les fonctions de test suivantes :

- `void test_double(int dim, int repet)`
- `void test_float(int dim, int repet)`
- `void test_u64(int dim, int repet)`
- `void test_u32(int dim, int repet)`

qui déclarent deux vecteurs `x` et `y` de type approprié et de taille `dim` (par ex. via la construction `type x[dim], y[dim];`), les initialisent avec tous leurs coefficients à 1, calculent `repet` fois leur produit scalaire en dimension `dim`, accumulent (c-à-d somment) le résultat (dans une variable du type approprié, initialisée à zéro), et l'affichent sur la sortie standard.

Ces fonctions devront être écrites dans un fichier `psm.c`, qui comportera aussi votre fonction `main`.

Q.3 : Si vous êtes familier avec `make`, écrivez un `Makefile` vous permettant de compiler votre programme et de spécifier aisément le compilateur et les options de compilation à utiliser.

Q.4 : Lancez chacune de vos fonctions de tests avec les arguments 1000 et 10000000 avec au moins deux compilateurs (par ex. clang et gcc) et les options suivantes :

- `-O0`

— `-O2`
— `-O2 -march=native`

Comment pouvez-vous expliquer : 1) les temps de calcul obtenus ; 2) les différentes valeurs des résultats ?

Q.5 : Modifiez vos fonctions de la question 1 en déroulant la boucle, par exemple sur 4 ou 8 instructions. Pour simplifier les choses, vous pouver supposer que la dimension de l'entrée est divisible par 8. Faites cependant attention à ne pas avoir de dépendances du flot des données entre chaque instruction de votre boucle déroulée. Faites à nouveau les tests de la question précédente. Que constatez-vous ?