Rapport de projet: Pi
CAI Chaolei
17812776

Le but du projet est de calculer pi en multi-thread, avec la plus de précision possible.
Il y a surtout 2 obstacle majeur à régler pour le projet, le premier étant la bonne formule mathématique facile à implémenter et la deuxième étant la structure de donnée pour stocker Pi. Ces 2 obstacles sont résolus grâce à la formule de Bailey-Borwein-Plouffe ainsi que la bibliothèque GMP, (plus précisément gmpxx.h pour c++).

Dans sa forme originelle, la formule BBP est donnée par

$$\pi = \sum_{k=0}^{\infty} \frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right).$$

La formule est très facile à implémenter, il s'agit juste une somme allant de 0 à l'infinie avec une seule variable «k». Il peut être partagé par différents threads sans aucun conflit, il suffit que chaque thread calcule de son côté les rangs qui lui sont attribué, puis d'additionner tous les résultats obtenue afin d'obtenir notre approximation de Pi désiré.
mpf_class BBP(int k) est ma fonction qui calcule la somme au dessus pour un rang k donée. gmpxx permet d'implémenter ces calculs, mpf_class gère pour nous la structure de donnée de stockage. Les opérateurs tells +-*/ sont implémenté pour cette classe, mais on peut aussi faire les calculs via les fonctions du type *mpf_add/div*.

void loop_BBP(int step, int begin, int kmax, mpf_class ret[]) est la fonction principale qu'exécuteras chaque thread, c'est juste une petit boucle for qui appel BBP et qui somme les résultats dans un tableau de *mpf_class*.
Chaque thread a droit à sa case dans le tableau ret, chaque thread avance toujours du nombre de pas qu'il y a de thread.
Par exemple s'il y a 4 thread, le premier calcule le rang 0, et avance de 4 à chaque fois, il arrive à 4. Le deuxième calcule le rang 1 et passe au rang 5, et ainsi de suite. Il n'y a pas de mutex comme il n'y en a pas besoin. Les données sont indépendante entre elles.

La structure du main est linéaire, en premier partie je recupère les arguments données en paramètre puis j'indique à la bibliothèque GMP la précision que j'ai besoin via *mpf_set_default_prec(precision * 4);* la précision est multiplié par 4 car la fonction règle la précision en fonction du nombre de bits donnée et non en base de 10.
Puis dans la suite, j'ai juste à lancer tous les threads, attendre qu'ils terminent et d'additionner leurs résultats pour obtenir mon approximation de Pi. Enfin je me suis servit de la fonction *gmp_fprintf* pour écrire dans le fichier « resultat.txt » le résultat.

Voici donc les performances sur mon ordinateur portable i5-7300hq (4coeurs 4 threads), 8Go Ram, Archlinux.

Nb de thread/precision (seconde)	1 000	10 000	100 000	1 000 000
1	0,002	0,133	19,484	2190,31
2	0,002	0,066	10,117	1185,053
3	0,000879	0,053	7,707 9,192	906,299
4	0,001	0,053	8,653	881,726

Le temps est donné en seconde, convertion en minutes ci-dessous.
900s = 15 minutes
1020s = 17 minutes
1200s = 20 minutes
1800s = 30 minutes
2160s = 36 minutes

Au dessus de 3 threads, il n’y a pas de différence significative.