IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Programmation threads Posix

UE OSAP

Printemps 2024

1 Introduction

En 1674, Gottfried Wilhelm Leibniz propose une estimation de π par la série alternée :[1][2][3]

$$\pi = 4\sum_{n=0}^{\infty}\frac{(-1)^n}{2n+1} = 4\left(\frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + ... + (-1)^n\frac{1}{2n+1} + ...\right)$$

2 Implémentation itérative

```c
#include <stdio.h>
#include <stdlib.h>

#define N_MAX    100000000

long double sum = 0.0;

void simple_sum() {
  unsigned long long n;
  long double sign = 1.0;

  for (n = 0; n < N_MAX; n++, sign = -sign)
    sum += sign / (2.0 * n + 1.0);
}


int main () {
  simple_sum();

  printf("pi = %.20Lf\n", 4.0 * sum);

  exit(EXIT_SUCCESS);
}
```

3 Implémentation multi-threadée

Cette méthode d'estimation est basée sur l'addition, une opération commutative. Elle se parallélise donc très bien : peu importe que l'on calcule pour $n = 100$ avant $n = 50$ ou même pendant.

L'idée est donc d'avoir un *pool* de threads, c'est à dire un nombre TC (*threads count*) de threads qui se répartissent les $n_{\max}$ calculs (aussi équitablement que possible). Chacun de ces threads ajoutera son résultat à la variable globale `sum`, qu'il conviendra de protéger[4]...

1. Proposez votre implémentation multi-threadée du calcul de π
2. Comparez les performances par rapport à la version itérative, grâce à la commande `time` (lisez le man)

1. https://fr.wikipedia.org/wiki/Approximation_de_%CF%80
2. https://fr.wikipedia.org/wiki/Formule_de_Leibniz#S%C3%A9rie_altern%C3%A9e
3. http://pi314.net/fr/leibniz.php
4. Dans notre exemple, on se contente de faire une addition sur notre variable partagée, une opération tellement élémentaire que l'on ne constatera peut-être pas de problème pratique si l'on oublie de la protéger. Mais ce n'est pas une raison pour ne pas le faire, on ne sait jamais !