High Performance Computing

Calcul de π

Plessia Stanislas

Février 2018

Table des matières

1 Analyse

1.1 Définition du problème

On veut calculer une valeur approximative de π. Pour cela, nous allons utiliser le fait que $arctan(1) = \frac{\pi}{4}$

Pour calculer cette valeur, nous allons effetuer le calcul suivant :

$$\pi = 4(arctan(1) - arctan(0)) = 4\int_0^1 \frac{d(arctan(x))}{dx}dx = 4\int_0^1 \frac{1}{1+x^2}dx$$

Ainsi en calculant une intégrale il est possible d'avoir une valeur approchée de π. Le calcul d'intégrale est un classique algorithmique, nous allons utiliser la méthode des trapèzes :

On subdivise le domaine de l'intégrale $[a, b]$ en n segments $[a_i, a_{i+1}]_{i\in[\![0,n-1]\!]}$ avec $a_0 = a$, $a_n = b$, soit :

$$a_i = a + i \cdot \frac{b-a}{n} \text{ et } \int_a^b f(x)dx = \sum_{i=0}^{n-1}\left(\int_{a_i}^{a_{i+1}} f(x)dx\right)$$

Chaque petite intégrale est ensuite approchée par l'aire du trapèze de sommet $(a_i, a_{i+1}, f(a_{i+1}), f(a_i))$ soit :

$$\int_{a_i}^{a_{i+1}} f(x)dx \approx (a_{i+1} - a_i)) \cdot \frac{f(a_{i+1}) + f(a_i)}{2} = \frac{b-a}{n} \cdot \frac{f(a_{i+1}) + f(a_i)}{2}$$

Donc :

$$\begin{aligned}\int_a^b f(x)dx &\approx \sum_{i=0}^{n-1}(a_{i+1} - a_i)) \cdot \frac{f(a_{i+1}) + f(a_i)}{2} \\ &= \frac{b-a}{n} \cdot \sum_{i=0}^{n-1}\frac{f(a_{i+1}) + f(a_i)}{2} \\ &= \frac{b-a}{n} \cdot \left(\sum_{i=1}^{n-1} f(a_i) + \frac{f(a) + f(b)}{2}\right)\end{aligned}$$

D'où le code :

```
double trapeze(double a, double b, int n){
  double h = (b - a)/n;
  double integral = (f(a) + f(b))/2;
  for( int k = 1; k < n; k++){
    integral += f(a + k * h);
  }
  integral *= h;
  printf("%lf, %lf, %lf, %d\n", integral, a, b, n);
  return integral;
}
```

2 Implémentation

2.1 Parallélisation

Nous devons à présent paralléliser ce programme. La boucle ne semble présenter aucune dépendance et peut donc être facilement parallélisée.

Les paramètres du problèmes sont alors :

— Le nombre de subdivision du domaine intégral n

— Le nombre de processeur qui vont faire le calcul p

L'objectif est alors de découper l'intégrale en p sous intégrales qui seront chacune calculée dans un processeur. Le résultat de chaque sous intégrale seraenvoyée au processeur `root` qui ajoutera les résultats pour afficher l'initégrale globale.

```
int main(int argc, char* argv[]){
  double a = 0, b = 1;
  int n = 24000; // default subdivision value
  double integral;
  if ( argc > 1){
    n = atoi(argv[1]);
  }
  double pas = (b-a)/n;

  int rank, size;
  MPI_Init(&argc, &argv);           // starts MPI
  MPI_Comm_rank(MPI_COMM_WORLD, &rank); // get current process id
  MPI_Comm_size(MPI_COMM_WORLD, &size); // get number of processes

  switch(rank){
    case 0:
      integral = trapeze(a, a + pas * n / size, n / size );
      double int_from_sub_proc;

      for( int k = 1; k < size; k++){
        printf("Receving data from proc %d\n", k);
        MPI_Recv(&int_from_sub_proc, 1, MPI_DOUBLE, k, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        integral += int_from_sub_proc;
      }
      printf("Integral value : %lf, with error %.3e\n", integral, M_PI - integral);
    break;
    default:
      printf("Calculating process %d\n", rank);
      double loca = a + rank * pas * n / size;
      double locb = loca + pas * n / size;

      double inte = trapeze(loca, locb, n / size );
      MPI_Send(&inte, 1, MPI_DOUBLE, 0, 0, MPI_COMM_WORLD);
    break;
  }

  MPI_Finalize();
}
```

3 Résultats

3.1 Vitesse d'exécution

Essayons d'estimer le temps gagné en passant le calcul sur plusieurs processus.

Les valeurs de n testées sont : $24 \cdot 10^k, k \in [\![0, 6]\!]$

Pour $n < 10^8$, la différence n'est pas sensible et le processeur n'a pas le temps de monter à 100% d'utilisation. De façon étonnante, Lecomportement n'est pas linéaire par rapport au nombre de processeurs, et semble perturbant pour certaines valeurs de p.

J'ai fais l'experience sur un processeur `Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz` qui dispose de 4 coeurs physiques et de 8 coeurs logiques.

n	log10(n)	P=1	P=2	P=3	P=4	P=5	P=6	P=7	P=8
24	1,380	0,381	0,366	0,390	0,372	0,391	0,397	0,408	0,424
240	2,380	0,047	0,062	0,051	0,060	0,458	0,464	0,451	0,441
2400	3,380	0,498	0,434	0,454	0,416	0,057	0,060	0,067	0,068
24000	4,380	0,045	0,070	0,062	0,055	0,441	0,438	0,446	0,434
240000	5,380	0,033	0,441	0,419	0,441	0,068	0,066	0,063	0,069
2400000	6,380	0,438	0,079	0,063	0,061	0,444	0,440	0,446	0,444
24000000	7,380	0,241	0,549	0,514	0,495	0,100	0,093	0,109	0,104
240000000	8,380	2,231	1,358	1,076	0,908	0,813	0,768	0,679	0,727

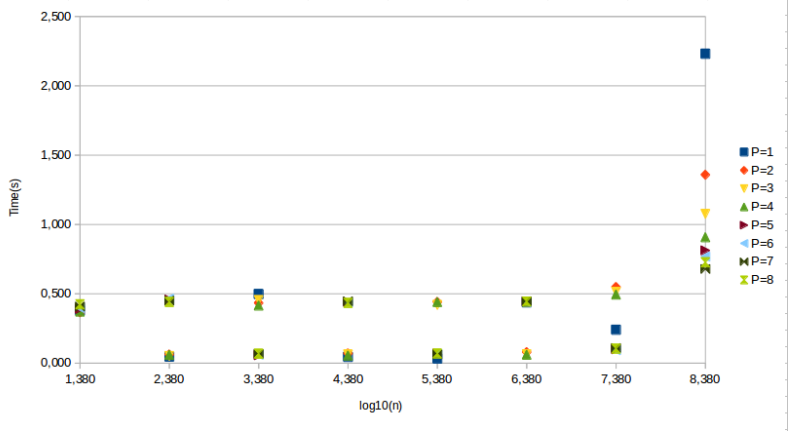

FIGURE 1 – Performance du programme en fonction de n et de p

3.2 Précision

Pour comparer les précisions, j'utilise la constante π de **math.h**. On prend soin de garder n divisible par k pour que les processus aient la même charge de travail et qu'ils aient chacun les même erreurs.

Le nombre de processus affecte très peu la précision. Je suppose que l'on voie des différences dûes aux processus pour $n = 240000$, puis l'on atteint les limites de précision des `double`. On remarque que à chaque pas (10 fois plus de trapèzes), la précision est multipliée par 100, ce qui est cohérent avec la majoration de l'erreur de cet algorithme :

$$\left| \int_a^b f(x)\mathrm{d}x - T_n \right| \leq \frac{M(b-a)^3}{12n^2}$$

avec T_n l'approximation avec n trapèzes, M constante dépendant de f supposée de classe $\mathcal{C}^2$ sur $[a, b]$.