Travaux Dirigés n°7

Cours de Parallélisme

—Maîtrise Informatique—

Ce TD porte sur l'étude d'une méthode récente de calcul de π, méthode due à David Bailey, Peter Borwein et Simon Plouffe. L'algorithme a l'avantage de pouvoir être distribué puisqu'il permet de calculer une hexadécimale de π sans calculer les autres.

Jusqu'à présent, on croyait que calculer la i-ième décimale d'un nombre transcendent comme π était aussi difficile que de calculer π lui-même. En fait, il n'en est rien comme viennent de le montrer David Bailey, Peter Borwein et Simon Plouffe. En effet, ils ont pu calculer le 10 milliardième chiffre du développement hexadécimal de π en utilisant la formule :

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

Cette formule présente cet immense avantage de permettre de calculer la i-ième hexadécimale de π sans être obligé de calculer toutes les précédentes.

▸ **Exercice 1.** (algorithme de Bailey, Borwein et Plouffe) *Selon Bailey, Borwein et Plouffe, partant de cette formule, on peut déduire un algorithme ne nécessitant pas d'arithmétique multi-précision, n'utilisant que peu de mémoire et dont les performances seraient linéaires par rapport à l'ordre de l'hexadécimale désirée, pour calculer le développement hexadécimal de π. Puisque nous pouvons calculer une hexadécimale de π sans calculer les autres, nous pouvons distribuer le calcul de n hexadécimales sur plusieurs machines, chaque machine calculant par exemple un intervalle de digits hexadécimaux.*

Nous avons développé un programme réalisant effectivement la distribution du calcul de π sur un réseau de machines hétérogènes regroupées en une machine parallèle virtuelle à l'aide de `PVM`*. Certains sources se trouvent dans le répertoire* `/tmp/PVM/pi`*. Plus précisément, on y trouve le fichier* `pi_master.c` *qui administre le calcul et le fichier* `pi_slave` *qui calcule un intervalle de 13 hexadécimales. Ces deux fichiers sont accompagnés d'un fichier* `pi.h` *qui définit quelques constantes intéressant notre calcul et du* `makefile` *permettant de compiler puis d'installer les exécutables sur la machine virtuelle. Le fichier* `pi_slave.c` *n'est pas fourni mais le protocole de calcul correspond à l'extrait de programme suivant :*

```
void main(void)
{ int mytid, pos, master, i, j=0, k, decitranch, result_pos;
  double y, s1, s2, s3, s4, ss, pi_decimal;
  char pi_hexa[2*PART_SIZE];
  int buffer_id, bytes, msgtag, tasktid;
  char deci_hexa[16]=
    {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
```

```
  mytid=pvm_mytid();                                        /* enrolling PVM */
  master=pvm_parent();                                      /* parent's tid */

  decitranch=(int)(PART_SIZE/PARTS_DIVISION)+1;

  while(1)                                                  /* infinite loop */
  { buffer_id=pvm_recv(master, -1);                         /* blocking recv */
    pvm_bufinfo(buffer_id, &bytes, &msgtag, &tasktid);     /* information */

    switch(msgtag)
    { case TAG_POSITION :                                   /* new position */
                                                            /* to compute */
                        pvm_upkint(&pos, 1, 1);
                        result_pos=pos;
                        for(k=0;k<PARTS_DIVISION;++k) /* computing */
                        { s1=s2=s3=s4=0;              /* the series */
                          Serie(pos);
                          ss=4*s1-2*s2-s3-s4;
                          pi_decimal=ss-(int)ss+1.;
                          y=fabs(pi_decimal);
                          for(i=j;i<j+decitranch;i++) /* decoding */
                          { y=16*(y-(floor(y)));
                            pi_hexa[i]=deci_hexa[(int)floor(y)];
                          }
                          j+=decitranch;
                          pos+=decitranch;
                        }
                        j=0;
                        pvm_initsend(PvmDataDefault);

                        ...                           /* computing the */
                        ...                           /* execution time */

                        pvm_pkdouble(&user_sys_time, 1, 1);
                        pvm_pkint(&result_pos, 1, 1);
                        pvm_pkbyte(&pi_hexa[0], PART_SIZE, 1);
                        pvm_send(master, TAG_OK);     /* sending time */
                        break;
      case TAG_KILL     : pvm_exit();
                        exit(0);
    }
  }
}
```

1. *Utiliser le programme* `pi_master` *à partir de la console* `PVM` *pour calculer les 5000 premières hexadécimales de π en séquentiel.*

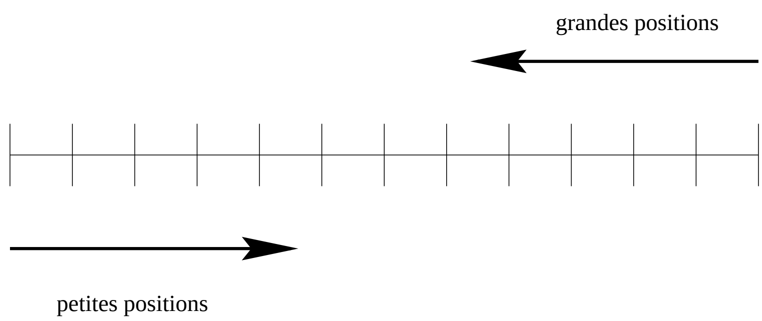
2. *Utiliser le programme* `pi_master` *à partir de* `XPVM` *pour calculer les 5000 premières hexadécimales de π en parallèle avec une, deux, trois, etc. tâches* `pi_slave` *lancées sur la machine parallèle virtuelle.*

3. *Vérifier à l'aide du fichier généré* `new_times` *que le temps de calcul d'une tranche d'hexadécimales à partir de la position p est bien en $\mathcal{O}(p)$. Pour cela, on utilisera Maple (commande* `xmaple`*) en faisant un* `read new_times;` *puis on tracera la courbe exprimant la position de la tranche d'hexadécimales en fonction du temps mis à la calculer.*

4. *Estimer le temps de calcul de la tranche d'hexadécimales commençant à la position 10^6 en séquentiel.*

5. *En déduire une estimation du temps de calcul des 10^6 premières hexadécimales de π sur la machine virtuelle.*

6. *Le programme calcule les tranches d'hexadécimales en partant de la position la plus grande jusqu'à la position la plus petites, pour une raison pratique. Les tâches* `pi_slave` *calculant ces tranches sont lancées sur les différentes machines constituant la machine parallèle sans aucun contrôle au niveau du programme* `pi_master`*. En fait, il serait judicieux de distribuer les calculs des premières tranches sur des machines peu puissantes, les machines plus efficaces se chargeant de calculer les tranches correspondant aux positions les plus grandes. Quel paramètre du fichier de configuration de la machine virtuelle permettrait de gérer cette notion de "puissance des machines" ?*



7. *Modifier le programme* `pi_master.c` *afin qu'il intègre cette idée de distribution des calculs des tranches en fonction de la puissance des machines. En particulier, on positionnera le paramètre de la question précédente afin d'avoir autant de machines dites performantes que de machines moins efficaces. On créera alors par exemple une seule tâche* `pi_slave` *par machine et on fera en sorte que les tranches les plus longues à calculer le soient sur les machines puissantes.*

8. *Faire varier le pourcentage de processeurs efficaces constituant la machine parallèle afin de mettre en valeur son influence sur le temps de calcul total.*