Algorithmique parallèle
Bibliothèque OpenMP

Formation d'ingénieurs de l'Institut Galilée MACS-2

Philippe d'Anfray, Xavier Juvigny
Philippe.d-Anfray@cea.fr, xavier.juvigny@onera.fr

29 janvier 2013

Produit de matrices parallélisé

Nous considérons deux matrices carrées `A` et `B` de dimension `n X n` dont les éléments sont de type `double`. Nous calculons le produit `C = A x B`.

1. Écrire le produit de matrice "séquentiel" en `C` ; plusieurs stratégies sont possibles, par lignes, par colonnes, par blocs ;
2. Écrire le produit de matrice "parallèle", parallélisez le code avec `OpenMP`, en conservant et en justifiant la stratégie qui vous semble la plus efficace ;
3. Effectuez des mesures et comparaisons de performances (sur des matrices de taille raisonnables au moins 1000x1000) pour les versions séquentielles et parallèles.

Exemples en OpenMP

Le code se compile avec `gcc -fopenmp ... essai.c`

Qui suis-je parmi combien ?

Pas forcément le plus utile ici mais cetres parlant !

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
  {
  int nthreads, tid;
  #pragma omp parallel private(nthreads, tid)
    {
    tid = omp_get_thread_num(); /*numero thread*/
    printf("Bonjour de thread = %d\n", tid);
    if (tid==0)
      {
      nthreads = omp_get_num_threads(); /*combien ?*/
      printf("Il y a  %d threads\n", nthreads);
      }
    }
  /* fin de la section parallele */
  exit(0);
}
```

Boucle optimisée

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define  N       100
int main (int argc, char *argv[])
  {
  int nthreads, tid, i, bloc;
  double a[N], b[N], c[N];
  for (i=0; i <N; i++)
    a[i]=b[i]=i*1.0;
  bloc=T_BLOC;
  #pragma omp parallel shared(a,b,c,nthreads),
private(i,tid)
    {
    tid = omp_get_thread_num();
    if (tid==0)
      {
      nthreads = omp_get_num_threads();
      printf("Nombre de threads = %d\n", nthreads);
      }
    printf("Demarrage thread %d \n",tid);
    #pragma omp for
    for (i=0;i<N;i++)
      {
      c[i] = a[i] + b[i];
      printf(" Thread %d: c[%d]= %lf\n",tid,i,c[i]);
      }
    }
  /* fin de la section parallele */
  exit(0);
}
```

Boucle optimisée améliorée

Gestion plus fine des tâches à répartir :

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#define T_BLOC     10
#define N          100
int main (int argc, char *argv[])
  {
  int nthreads, tid, i, bloc;
  double a[N], b[N], c[N];
  for (i=0;i<N;i++)
    a[i]=b[i]=i*1.0;
  bloc=T_BLOC;
  #pragma omp parallel shared(a,b,c,nthreads,bloc),
private(i,tid)
    {
    tid=omp_get_thread_num();
    if (tid==0)
      {
      nthreads=omp_get_num_threads();
      printf("Nombre de threads = %d\n", nthreads);
      }
    printf("Demarrage thread %d \n",tid);
    #pragma omp for schedule(dynamic,bloc)
    for (i=0; i<N; i++)
      {
      c[i]=a[i]+b[i];
      printf(" Thread %d: c[%d]= %lf\n",tid,i,c[i]);
      }
    }
  /* fin de la section parallele */
  exit(0);
}
```

Réduction

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
  {
  int   i, n=100;
  double a[100], b[100], s=0.0, sl;
  for (i=0; i<n; i++)
    a[i]=b[i]=i*1.0;
  #pragma omp parallel shared (n,a,b,s),private(i,sl)
    sl=0.0;
    #pragma omp for
    for (i=0; i<n; i++)
      sl=sl+(a[i]*b[i]);
    #pragma omp critical
      s=s+sl;
    #pragma omp end critical
  #pragma omp end parallel
  /* fin de la section parallele */
  printf("␣Somme␣=␣%lf\n",s);
  exit(0);
}
```

Réduction simplifiée

Utilisation d'une directive spéciale

```
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
int main (int argc, char *argv[])
  {
  int    i, n=100;
  double a[100], b[100], s=0.0;
  for (i=0; i<n; i++)
    a[i]=b[i]=i*1.0;
    #pragma omp parallel for reduction(+:s)
      for (i=0; i<n; i++)
        s =s +(a[i]*b[i]);
  /* fin de la section parallele */
  printf(" Somme = %lf \n",s );
  exit(0);
}
```