High performance computing exercise 1

B04902103 蔡昀達

1. Here is an OpenMP example of Matrix Multiply.

   → Download success !

```
b04902103@linux1 [~] cat omp_mm.c | head -n 30
/******************************************************************
* FILE: omp_mm.c
* DESCRIPTION:
*   OpenMp Example - Matrix Multiply - C Version
*   Demonstrates a matrix multiply using OpenMP. Threads share row i
*   according to a predefined chunk size.
* AUTHOR: Blaise Barney
* LAST REVISED: 06/28/05
******************************************************************
#include <omp.h>
#include <stdio.h>
#include <stdlib.h>

#define NRA 62                  /* number of rows in matrix A */
#define NCA 15                  /* number of columns in matrix A */
#define NCB 7                   /* number of columns in matrix B */
```

2. Compile and run it on a server in the CSIE Workstation Lab…

   Make sure it runs correctly.

   → Compile success !

```
int main (int argc, char *argv[])
{
int     tid, nthreads, i, j, k, chunk;
double  a[NRA][NCA],            /* matrix A to be
        b[NCA][NCB],            /* matrix B to be
        c[NRA][NCB];            /* result matrix C

chunk = 10;                     /* set loop iterat

/*** Spawn a parallel region explicitly scoping a
#pragma omp parallel shared(a,b,c,nthreads,chunk)
  {
  tid = omp_get_thread_num();
b04902103@linux1 [~] gcc omp_mm.c -fopenmp -O2
b04902103@linux1 [~]
```

   → Execute success !

```
#pragma omp parallel shared(a,b,c,nthreads,chunk
  {
  tid = omp_get_thread_num();
b04902103@linux1 [~] gcc omp_mm.c -fopenmp -O2
b04902103@linux1 [~] vim omp_mm.c
b04902103@linux1 [~] time ./a.out

real    0m0.027s
user    0m0.398s
sys     0m0.014s
b04902103@linux1 [~]
```

3. Comment out the printf statements. Run it with 1, 2, 4, 8,… threads and report the execution time

Threads	**2**	**4**	**8**	**16**	**32**	**64**	**128**
Real time	0.002	0.002	0.002	0.015	0.004	0.004	0.008

4. Double the values of NRA, NCA, and NCB to observe the execution time.
5. Repeat Step 4 until a problem happens to the system. Report your observations.

(Number of threads = 16)

(NRA, NCA, NCB)	**(62, 15, 7)**	**(124, 30, 14)**	**(248, 60, 28)**	**(496, 120, 56)**	**(992, 240, 112)**	**(1984, 480, 224)**
Real time	0.005	0.007	0.008	0.011	0.038	Segmentation fault

→ The execution time grows until segmentation fault

6. (10% Bonus) Try to optimize the code for cache when the matrices are large. Report your results.

Since matrix b is symetric, we can change b[k][j] to b[j][k] to utilize cache and improve performance.

The below test runs the matrix multiplication for 100 times.

```
  for (i=0; i<NRA; i++)
    {
    //printf("Thread=%d did row=%d\n",tid,i);
    for(j=0; j<NCB; j++)
      for (k=0; k<NCA; k++)
        c[i][j] += a[i][k] * b[k][j];
    }
  }   /*** End of parallel region ***/
```

```
b04902103@linux1 [~] time ./a.out

real    0m0.652s
user    0m12.538s
sys     0m0.064s
```

```
  for (i=0; i<NRA; i++)
    {
    //printf("Thread=%d did row=%d\n",tid,i);
    for(j=0; j<NCB; j++)
      for (k=0; k<NCA; k++)
        c[i][j] += a[i][k] * b[j][k];
    }
  }   /*** End of parallel region ***/
```

```
b04902103@linux1 [~] time ./a.out

real    0m0.549s
user    0m10.890s
sys     0m0.044s
```

Original : 0.652

Optimized : 0.549

→ Improved ~~~~ !!!!