1. Introduction

In the previous exercises, you should have obtained basic knowledge about the architecture, the development process, and the syntax of the parallel computing algorithms. Now it is time for you to try to parallelize some code yourself – using the OpenMP library. There are three tasks to solve.

a) In the first task, you must calculate the sum of the first 10 million integers with a for loop (with – of course – no fancy shenanigans with infinite sums!) and compare the running times of examples with or without the parallelization. See Section 2.

b) In the second task, you should parallelize the algorithm which calculates the value of π by means of the following integral:

\[ \pi = \int_0^1 \frac{4}{1+x^2} \, dx. \]

See Section 3.

c) In the third task, you have to implement the matrix multiplication procedure:

\[ C = A \cdot B. \]

Some of the code has already been provided – the initialization of matrices and the end print of results. What is missing is the core part – multiplication itself. See Section 4.

2. The sum of integers – the code

The following code computes the sum of first 10 million integers. Parallelize it and be careful about the data sharing!

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

int main (int argc, char *argv[]) {
    int N = 10000000;
    long sum = 0;
    double start_time, run_time;

    start_time = omp_get_wtime();

    for (int i=0; i < N; i++)
        sum += i;

    run_time = omp_get_wtime() - start_time;
    printf("\n Sum = %ld in %f seconds.\n", sum, run_time);
}
```
3. The value of π – the code

The following code is sequential. Use `#pragma omp ...` or other `omp_...` function calls wherever and whichever you see fit to obtain better running times. Plot the speed up in dependency from the number of processes.

```c
/* pi.cpp */
#include <stdio.h>
#include <omp.h>

static long num_steps = 100000000;
double step;

int main (int argc, char *argv[])
{
    int i;
    double x, pi, sum = 0.0;
    double start_time, run_time;

    step = 1.0/(double) num_steps;

    start_time = omp_get_wtime();
    for (i=1; i <= num_steps; i++)
    {
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
    }
    pi = step * sum;
    run_time = omp_get_wtime() - start_time;
    printf("\n Result: PI with %ld steps is %lf in %lf seconds\n",num_steps,pi,run_time);
}
```

4. Matrix multiplication – the code

The following code is missing the most important part – the multiplication itself. Its place in code is marked. Use `#pragma omp for ...` or other commands / function calls wherever and whichever you see fit to obtain correct results. If you get a `__gxx_personality_v0` error, compile your code with `g++` compiler.

```c
/* matrix_multiply.cpp */
#include <stdio.h>
#include <stdlib.h>
#include <omp.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 */

int main()
{
    int tid, nthreads, i, j, k, chunk;
    double a[NRA][NCA], /* matrix A to be multiplied */
           b[NCA][NCB], /* matrix B to be multiplied */
c[NRA][NCB];          /* result matrix C */
chunk = 10;           /* set loop iteration chunk size */

/*** Spawn a parallel region explicitly scoping all variables ***/
#pragma omp parallel shared(a,b,c,nthreads,chunk) private(tid,i,j,k)
{
  tid = omp_get_thread_num();
  if (tid == 0)              /* the master thread */
  {
    nthreads = omp_get_num_threads();
    printf("Starting matrix multiple with %d threads\n",nthreads);
    printf("Initializing matrices...\n");
  }
  /*** Initialize matrices ***/
#pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCB; j++)
      a[i][j] = i+j;
#pragma omp for schedule (static, chunk)
  for (i=0; i<NCA; i++)
    for (j=0; j<NCB; j++)
      b[i][j] = i*j;
#pragma omp for schedule (static, chunk)
  for (i=0; i<NRA; i++)
    for (j=0; j<NCB; j++)
      c[i][j] = 0;
  /***
  Try and also display who does which iterations ***/
}   /*** End of parallel region ***/

/*** Print results ***/
/***/
printf("";n");
printf("Result Matrix:\n");
for (i=0; i<NRA; i++)
{
  for (j=0; j<NCB; j++)
    printf("%6.2f ", c[i][j]);
    printf("\n");
}
printf("";n");
printf ("Done.";n");

5. Contact

Should any difficulties arise, please do not hesitate and contact me per email. But remember –
Google is your friend (http://www.giyf.com)!

Tutor: Stefan Markic   stefan.markic@tum.de