ETH zürich

High Performance Computing for Science and Engineering I

P. Koumoutsakos
ETH Zentrum, CLT E 13
CH-8092 Zürich

Fall semester 2019

Set 2 - OpenMP: Brownian Motion and Bug Hunting

Issued: October 11, 2019
Hand in (optional): October 25, 2019 08:00am

Question 1: Brownian motion (40 points)

The Brownian motion of N particles in a one-dimensional space

$$x_i(t) \in \mathbb{R}, \quad i = 1 \dots N$$

is described with random walks. The algorithm performs M time steps until $t = t_{\text{max}}$. One step is given by

$$x_i(t + \Delta t) = x_i(t) + \xi_i^{(t)} \sqrt{\Delta t}, \quad i = 1 \dots N,$$

where $\xi_i^{(t)}$ are independent random variables from the standard normal distribution $\mathcal{N}(0, 1)$ and $\Delta t = t_{\text{max}}/M$. The initial positions $x_i(0)$ are sampled from a uniform distribution over $[-\frac{1}{2}, \frac{1}{2}]$.

a) Given a serial implementation of the algorithm provided in the skeleton code, write a parallel version using OpenMP. Proceed in steps:

- Compile the serial version using **make main** and run it with **./main 100000 100** (corresponding to $N = 100000$ and $M = 100$) to produce files **hist_0.dat** and **hist_1.dat**. Use **make plot** to create **hist.pdf** with the histograms of the initial and final configurations of particles. Later you can simply type **make** that by default combines all the above stages.
- Add compiler flags and headers necessary for OpenMP following **TODO 1**. After implementing function **GetWtime()**, check that the program reports non-zero timings for time stepping (**walk:**) and histogram computation (**hist:**).
- Parallelize the time stepping (**TODO 2**) by splitting the particles among multiple threads. Make sure you do not introduce race conditions and each thread uses a separate random generator with a unique seed value. For storing the thread-local data, you may need to use arrays indexed by the thread-id or rely on data-sharing attributes of OpenMP.
- Parallelize the histogram computation (**TODO 3**).

b) Report strong scaling (speedup) on Euler up to 24 cores separately for time stepping (**walk:**) and histogram computation (**hist:**). The values of N and M should be sufficiently large such that the speedup does not depend on them.

c) Answer the following questions:

- Is the amount of computational work equal among all threads (for large N and M)?

- Do you observe perfect scaling of your code? Explain why.
- Run your program with $N = 1000$ and $M = 1$ on 1 thread and 500 threads. Then change the initial seed for the generator and run the program again. Plot the histograms in all four cases and include in your report. Does changing the number of threads have stronger effect on the final histogram than changing the initial seed? Explain why.

Question 2: OpenMP bug hunting (10 points)

a) Identify and explain any *bugs* in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
1  // assume there are no OpenMP directives inside these two functions
2  void do_work(const float a, const float sum);
3  double new_value(int i);
4
5  void time_loop()
6  {
7    float t = 0;
8    float sum = 0;
9
10   #pragma omp parallel
11   {
12
13     for (int step=0; step<100; step++)
14     {
15       #pragma omp parallel for nowait
16       for (int i=1; i<n; i++) {
17         b[i-1] = (a[i]+a[i-1])/2.;
18         c[i-1] += a[i];
19       }
20
21       #pragma omp for
22       for (int i=0; i<m; i++)
23         z[i] = sqrt(b[i]+c[i]);
24
25       #pragma omp for reduction(+:sum)
26       for (int i=0; i<m; i++)
27         sum = sum + z[i];
28
29       #pragma omp critical
30       {
31         do_work(t, sum);
32       }
33
34       #pragma omp single
35       {
36         t = new_value(step);
37       }
38     }
39   }
40 }
```

b) Identify and explain any *improvements* that can be made in the following OpenMP code. Propose a solution. Assume all headers are included correctly.

```
1  void work(int i, int j);
2
3  void nesting(int n)
4  {
5    int i, j;
6    #pragma omp parallel
7    {
8      #pragma omp for
9      for (i=0; i<n; i++) {
10       #pragma omp parallel
11       {
12         #pragma omp for
13         for (j=0; j<n; j++) {
14           work(i, j);
15         }
16       }
17     }
18   }
19 }
```