ETH zürich

M. Troyer, P. Hadjidoukas
ETH Zentrum, ML H 44
CH-8092 Zürich

High Performance Computing for Science and Engineering I

Fall semester 2016

Set 4 - Diffusion and OpenMP

Issued: October 21, 2016

Question 1: Diffusion in 2D

Heat flow in a medium can be described by the diffusion equation

$$\frac{\partial \rho(x,y,t)}{\partial t} = D\nabla^2 \rho(x,y,t), \tag{1}$$

where $\rho(x,y,t)$ is a measure for the amount of heat at position (x,y) and time t and the diffusion coefficient D is constant.

Lets define the domain Ω in two dimensions as $\{x,y\} \in [-1,1]^2$. Equation 1 then becomes

$$\frac{\partial \rho(x,y,t)}{\partial t} = D\left(\frac{\partial^2 \rho(x,y,t)}{\partial x^2} + \frac{\partial^2 \rho(x,y,t)}{\partial y^2}\right). \tag{2}$$

Equation 2 can be discretized with a central finite difference scheme in space and explicit Euler in time to yield:

$$\frac{\rho_{r,s}^{(n+1)} - \rho_{r,s}^{(n)}}{\delta t} = D\left(\frac{\rho_{r-1,s}^{(n)} - 2\rho_{r,s}^{(n)} + \rho_{r+1,s}^{(n)}}{\delta x^2} + \frac{\rho_{r,s-1}^{(n)} - 2\rho_{r,s}^{(n)} + \rho_{r,s-1}^{(n)}}{\delta y^2}\right) \tag{3}$$

where $\rho_{r,s}^{(n)} = \rho(-1 + r\delta x, -1 + s\delta y, n\delta t)$ and $\delta x = \frac{2}{N-1}$, $\delta y = \frac{2}{M-1}$ for a domain discretized with $N \times M$ gridpoints.

We will use open boundary conditions

$$\rho(x,y,t) = 0 \quad \forall\, t \geq 0 \,\text{and}\, (x,y) \notin \Omega \tag{4}$$

and an initial density distribution

$$\rho(x,y,0) = \begin{cases} 1 & |x,y| < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

a) Analyze the stability of the scheme given in Equation 3 using the von Neumann stability analysis.

To analyze the stability of the method, we substitute in Equation 3 the assumption made in the von Neumann stability analysis about the round-off error,

$$\epsilon_{r,s}^{(n)} = \zeta^n e^{i(k_x x_r + k_y y_s)} = \zeta^n e^{ik_x x_r} e^{ik_y y_s}, \tag{6}$$

By requiring that the error satisfies the time stepping scheme we obtain:

$$\zeta^{n+1}e^{ik_x x_r}e^{ik_y y_s} = \zeta^n e^{ik_x x_r}e^{ik_y y_s} + \zeta^n \delta t D \frac{e^{ik_x x_{r-1}}e^{ik_y y_s} - 2e^{ik_x x_r}e^{ik_y y_s} + e^{ik_x x_{r+1}}e^{ik_y y_s}}{\delta x^2} + \zeta^n \delta t D \frac{e^{ik_x x_r}e^{ik_y y_{s-1}} - 2e^{ik_x x_r}e^{ik_y y_s} + e^{ik_x x_r}e^{ik_y y_{s+1}}}{\delta y^2}.$$

Note that in this case ρ is used to indicate the heat and thus we use ζ for the amplitude term in the assumption.

For stability we require

$$|\zeta| \leq 1 \tag{7}$$

otherwise the solution will grow and diverge.

Taking the above equation and substituting $x_{r\pm1} = x_r \pm \delta x$ and $y_{s\pm1} = y_s \pm \delta y$ and dividing the whole by $\zeta^n e^{ik_x x_r}e^{ik_y y_s}$ we get:

$$\zeta = 1 + D\delta t \left(\frac{e^{-ik_x \delta x} - 2 + e^{ik_x \delta x}}{\delta x^2} + \frac{e^{-ik_y \delta y} - 2 + e^{ik_y \delta y}}{\delta y^2} \right). \tag{8}$$

Using the trigonometric properties $\frac{e^{i\theta}+e^{-i\theta}}{2} = \cos\theta$ first and $\frac{1-\cos\theta}{2} = \sin^2\frac{\theta}{2}$ we get

$$\zeta = 1 + \frac{D\delta t}{\delta x^2}\left((2\cos(k_x\delta x) - 2) + \frac{D\delta t}{\delta y^2}\left(2\cos(k_y\delta y) - 2\right)\right. \tag{9}$$

$$= 1 - 4\frac{D\delta t}{\delta x^2}\sin^2\left(\frac{k_x\delta x}{2}\right) - 4\frac{D\delta t}{\delta y^2}\sin^2\left(\frac{k_y\delta y}{2}\right), \tag{10}$$

which together with Equation 7 and the worst case scenario for the trigonometric functions ($\sin^2(x) \in [0,1]$) leads to the stability condition

$$-2 \leq -4\frac{D\delta t}{\delta x^2} - 4\frac{D\delta t}{\delta y^2} \leq 0. \tag{11}$$

The time step should therefore satisfy

$$\delta t \leq \frac{1}{2D}\frac{\delta x^2 \delta y^2}{(\delta x^2 + \delta y^2)}. \tag{12}$$

For equal spacing in x and y direction $\delta x = \delta y$ this simplifies to

$$\delta t \leq \frac{\delta x^2}{4D}. \tag{13}$$

b) Parallelize your code from Exercise 1a using OpenMP and make both strong and weak scaling plots up to 24 cores. Define the problem size as the number of grid points in the discretization of Ω. Try different problem sizes for the scaling plots, do they affect the scaling? Try using both the gcc OpenMP and the Intel OpenMP libraries. Can you see a difference?

To parallelize our code with OpenMP, we use two variants as in the last exercise. Namely we write a variant in which the threads are "spawned and rejoined" at every time iteration and another version creating the threads only once at the beginning of the simulation and then relying on a barrier to keep synchronization. In the former case we add a `#pragma omp parallel for` in front of the main loop in space in the method `advance()`:

```
value_type advance()
{
    // Open boundaries
    // central differences in space, forward Euler in time
#pragma omp parallel for
    for(size_type i = 0; i < N_; ++i) {
        for(size_type j = 0; j < N_; ++j) {
            rho_tmp[i*N_ + j] = rho_[i*N_ + j] +
            fac_
            *
            (
             (j == N_-1 ? 0. : rho_[i*N_ + (j+1)])
             +
             (j == 0    ? 0. : rho_[i*N_ + (j-1)])
             +
             (i == N_-1 ? 0. : rho_[(i+1)*N_ + j])
             +
             (i == 0    ? 0. : rho_[(i-1)*N_ + j])
             -
             4.*rho_[i*N_ + j]
             );
        }
    }
```

In the same manner we parallelize the loop that sets the initial conditions. Note that this is needed on Non-Uniform Memory Architecture (NUMA) nodes such as Euler's[1], where the access time to a NUMA node from another is different than the access time within the same NUMA node. Parallelizing the data initialization as mentioned above, is a simple — but quite effective — way to make the code "NUMA-aware": each thread creates the data it will work on in its own physical memory so that we reduce the need for threads to fetch data that others have. The principle of initializing the data owned by the thread is called "first-touch" policy.

Additionally, on Euler, to get good and stable performance using all 24 cores of a node we set the environment variable `OMP_PROC_BIND` so that each thread is mapped onto a single core.

In the version of the code using explicit barriers, we modify the `advance` method to take in the edges of a segment corresponding to a single thread and we add a method `t()` to get the current time from the diffusion class

```
#pragma omp parallel
{
    auto id = omp_get_thread_num();
    auto nthreads = omp_get_num_threads();

    while (time < tmax) {
```

[1]The 24 core Euler nodes have 2 NUMA nodes.

```
7           system.advance( id * N / nthreads , (id + 1) * N /
               nthreads );
8           #pragma omp single
9           {
10              time = system.t();
11          }
12      }
13  }
```

Figure 1 shows the strong scaling obtained with both variants of the parallelized code for two different problem sizes. In all cases, we ran the code with and without setting `OMP_PROC_BIND` with Intel icpc 15.0 and gcc 4.9 and the `-O3` optimization flag. The results shown are averaged over 10 samples of 10000 time steps each. Since the precise set of optimizations performed by each compiler on both serial as well as OpenMP versions of the code is not the same (inspite of specifying only the `-O3` flag in both cases), we choose to take as reference points the best single-threaded OpenMP times for each compiler and compare scaling of the different codes with respect to the best single-threaded version of its own compiler.

We observe how icpc is strongly influenced in a positive way by the environment variable when using 24 threads, whereas gcc is much less affected. However, this setting can have a negative impact on the overall performance of the code for smaller system sizes, specially for the icpc version.

From the plots we can also see the effect of problem size on scaling: larger problem sizes tend to scale much better than smaller ones. This is a common behavior and is explained by the amount of work that each thread has to perform: for smaller problem sizes, the workload is also small and overheads tend to dominate as well as possible load imbalances. This suggests that for a given problem size, we might not want to just use all the computing resources we have at our disposal as we would just waste them.

The plots show how not only the code has an impact on the performance but the compiler, the knowledge of the hardware and the problem size as well.

In the case of the largest problem we studied, we obtained the best speedup of $\sim$19x when using 24 threads and pinning each thread to a physical core. It is interesting to note that, unlike in the threaded version of the code from exercise 2 where we used c++11 threads, Open MP reduces the overhead of thread handling by keeping the threads alive even in the "spawn+join" version. This is nicely reflected in the nearly imperceptible difference between both versions of the code for the larger problem size.

A factor that has an important influence on these results is the amount of memory bandwidth available, as an increase of the number of threads corresponds to a linear increase in computational performance but not in memory bandwidth. We will later see how memory bandwidth is a constraint that is more important than computational capacity in many scientific applications.

In the weak scaling plot (Figure 2) we can observe a similar behavior as for the strong scaling, ie. as we increase the number of cores the scaling deviates from the ideal case with the strongest deviations being observed when using gcc. In the case of icpc binding the threads to individual cores appears to have a relatively weak effect and slightly more noticeable in the case of gcc. We observe that with the Intel compiler we get consistently better results

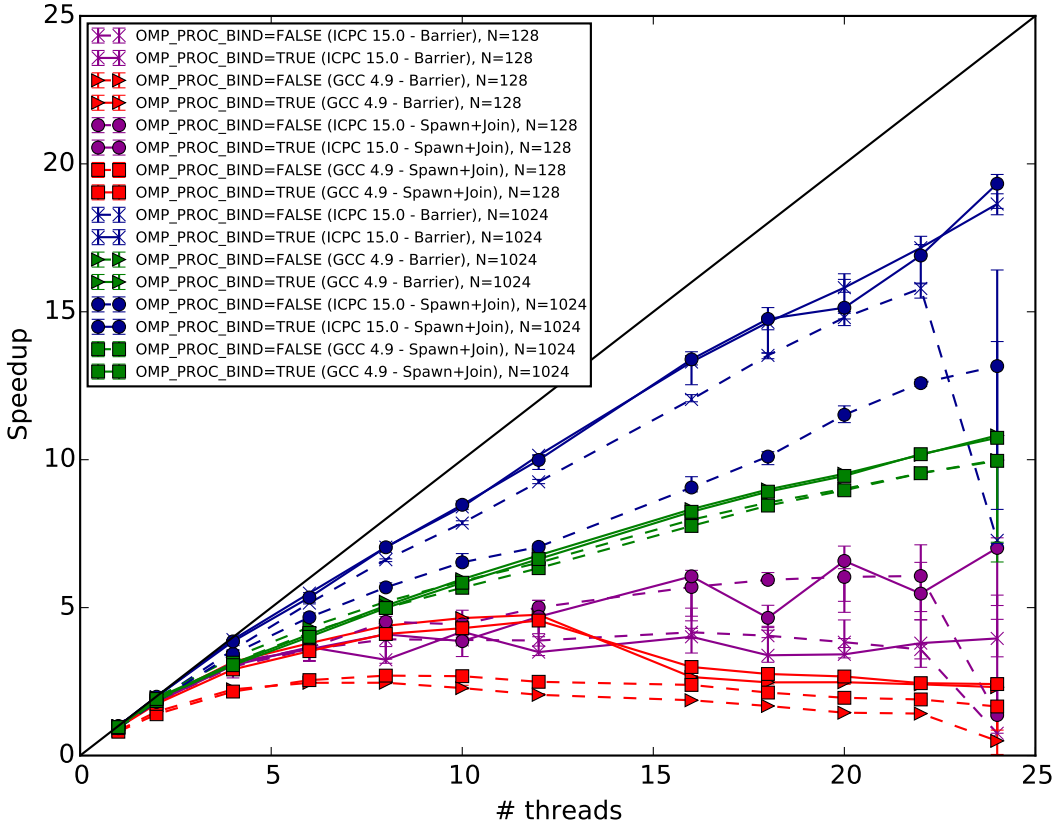


Figure 1: Strong scaling measured by averaging 10 samples of 10000 time steps for problem sizes N=128 and N=1024. Reference timings correspond to the best single-threaded version of the code for each compiler.

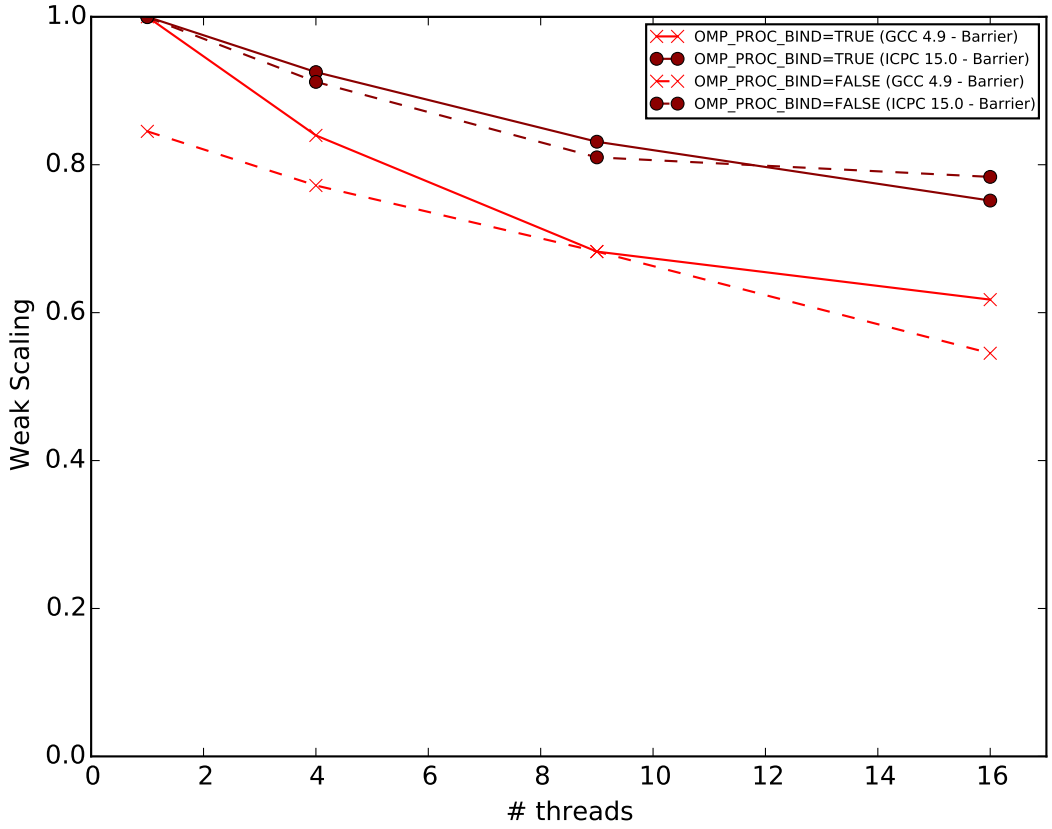


Figure 2: Weak scaling plot.

and for the largest number of threads ploted a weak scaling of ∼0.8 is attained.

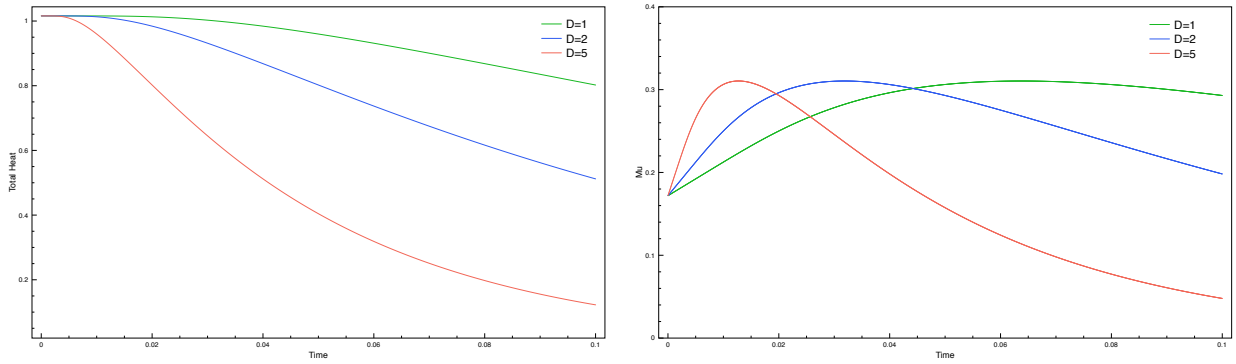
Parallel code: `diffusion2d_openmp.cpp`

The total amount of heat in the system $N(t)$ and the second moment of the cloud $\mu^2(t)$ at a time t can be computed as

$$N(t) = \int_\Omega dx\, dy\, \rho(x, y, t), \qquad \mu^2(t) = \int_\Omega dx\, dy\, \rho(x, y, t)(x^2 + y^2). \tag{14}$$

c) Make a plot of the time evolution of $N(t)$ and $\mu^2(t)$ for $D = 1, 2, 5$ and $t \in [0, 10]$. What is the effect of the boundary? What is the effect of D?

Figure 3a shows that after a certain amount of time, the system starts losing heat, depending on D. This is caused by the boundary condition that absorb the "heat" that reaches the boundary and thus removes it from the system. As heat diffuses away from the initial position the second moment μ^2, which measures the spread of the cloud, increases and reaches a maximum when a significant amount of heat has escaped from the system (see Figure 3b).



(a) Total heat (b) $\mu^2(t)$

Figure 3: Total heat $N(t)$ and second moment $\mu^2(t)$. Domain size $\Omega = 128 \times 128$