Homework 4 Write-up

Denis A. St-Onge
APC-524 – Software Engineering for Scientific Computing

November 18, 2016

1 Overview

In this assignment we solve the heat diffusion equation

$$\frac{\partial T}{\partial t} = \kappa \nabla^2 T \tag{1}$$

with κ = constant on a two-dimensional domain of size $0x \leq \pi$ and $0 < y \leq \pi$. We use a finite different scheme with various implementations of parallel architectures (serial code, OpenMP and MPI).

1.1 Serial code

The serial code performs adequately for $N = 128$ with a runtime of around 2.5 seconds. However, at $N = 512$ it takes 885 seconds (roughly fifteen minutes). This is roughly consistent with a scaling of N^4 (N^2 with the box size, and N^2 with the timestep). Final temperature heat maps are plotted in Fig. 2.

1.2 OpenMP

OpenMP is implemented by using a single `#pragma omp parallel` call encapsulating the entire time loop. Four specific for loops throughout the program are then encapsulated with `#pragma omp for`, enabling the parallelization of crucial loops, which are always chosen to be the outer loop. A barrier must be used prior to calculating the boundary conditions to ensure all processors are finished the time integration.

Quantitatively there is no difference in the temperature heat maps or the final volume averaged temperature between the serial and OpenMP-enabled runs. This is to be expected, as parallelization

N	T
128	0.4968300
256	0.4969596
512	0.4970229

Table 1: Finally temperature for various values of N. Values are equal across all implementations and processor number for a given N.

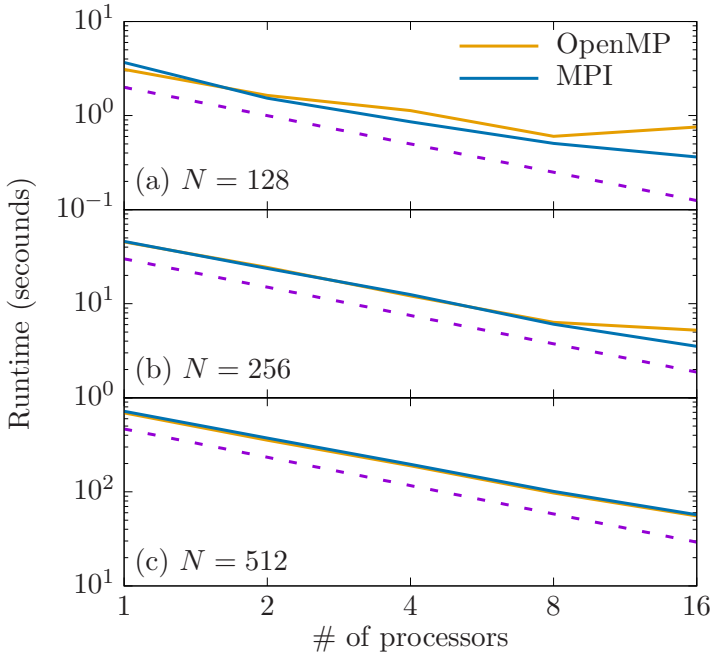

Figure 1: Runtimes plotted again number of processors for various values of N using both OpenMP and MPI. Dashed purple line denotes (# processors)$^{-1}$ scaling.

should not greatly affect the numerics nor the physics of the equations being modeled. Heat maps of the temperature are plotted in Fig. 2. For our problems, the runtime scales very closely to the optimal $N^{-1}_{\text{processors}}$ scaling, demonstrating strong-scaling of the parallelization. The sole exception of this is at $N = 128$ with 16 processors. Here, one could imagine the overhead of processor communication dwarfing the calculations needed on a single subgrid.

1.3 MPI

MPI is implemented in a slightly different, though no more difficult, manner than the OpenMP implementation. Here, subgrids representing partitioned portions of the full grid are initialized on each processor. The equations of motion are simply evolved on these grids. The only parallel aspects of the algorithm appear in the computation of the boundary conditions. Here, information on the physical edge of the grid as passed along to neighbouring processors and dumped into the appropriate ghost zones. This is done using a non-blocking send and a blocking received. This combination guarantees the avoidance of dead-locks while ensuring all processors have successfully processed the time integration.

Once again, the quantitative aspects of the simulation are unchanged. This indicates proper implementation. In addition, MPI exhibits strong scaling for all problems. Heat maps of the temperature are again plotted in Fig. 2.

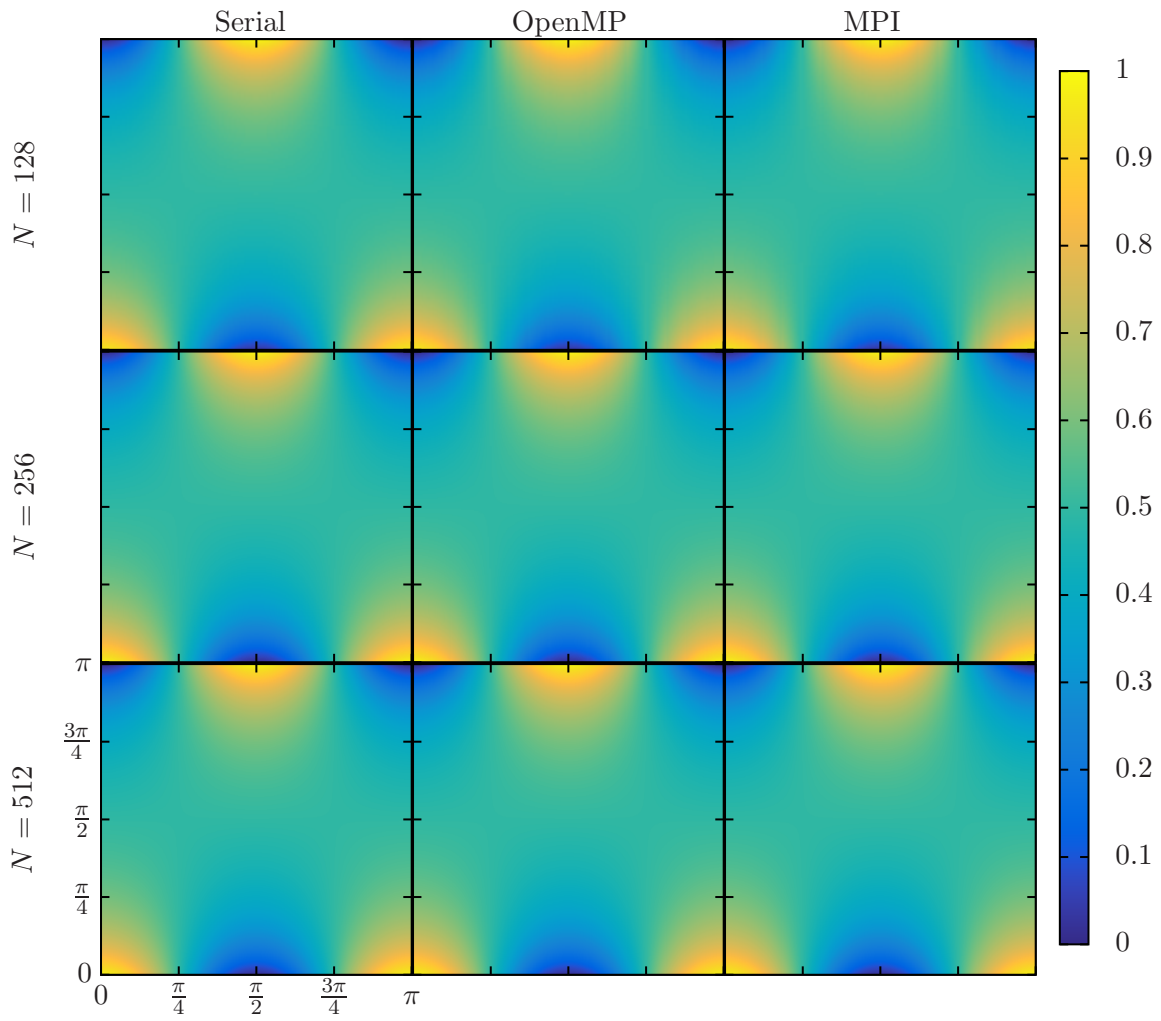

Figure 2: Heat map of the temperature for various runs of different resolutions using different parallel architectures.

2 OpenMP vs. MPI

From these results, one may question whether MPI truly offers any benefit over OpenMP. One must understand however that these simulations were performed on a single node. If one would desire to utilize more core than the node provides (28 cores per node on Perseus, the cluster used here), OpenMP would cease to be an option. In that case, MPI would be absolutely necessary. In addition, the additional coding effort required to implement MPI over OpenMP in this case is only marginal (I actually found the MPI coding here to be far easier, taking me only a few tens of minutes). Perhaps the most important deciding factor is whether or not the programmer is up to it.