APC 524 / AST 506 / MAE 506

Software engineering for scientific computing

Assignment 4: Parallel programming with OpenMP and MPI

Assigned: November 11, 2014

Due: 11:55pm November 18, 2014

Introduction

Consider solving the heat diffusion equation

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

with $\kappa = constant$ on a two-dimensional domain of size $0 \le x, y \le \pi$. Let the boundary conditions be

$$\begin{aligned} T(x,0) &= \cos^2 x \\ T(x,\pi) &= \sin^2 x \\ T(0,y) &= T(\pi,y) \quad \text{(periodic in } x\text{)} \end{aligned} \tag{2}$$

In this assignment, you will write three implementations to solve this problem using finite differences.

1. Write a C, C++ or Fortran program that solves this problem starting from the initial condition $T = 0$ everywhere. Run the solution on three grids until time $t = 0.5\pi^2/\kappa$. Use grid sizes of 128^2, 256^2, and 512^2. Plot contours or an image of the final temperature, and report the final, volume averaged temperature.
2. Now extend the program you wrote in part (1) to run with OpenMP. Repeat the calculation using 1, 2, 4, and 8 threads, and report the speed-up you get in each case.
3. Now extend the program you wrote in part (1) to run with MPI. Repeat the calculation using 1, 2, 4, 8, and 16 processes, and again report the speed up. Plot contours of the final temperature and report the volume-averaged value (which will require an MPI_Allreduce call), and describe how you handled I/O from more than one processor.
4. Discuss the advantages and disadvantages of parallelizing this problem with OpenMP versus MPI.

Finite difference method

This equation can be solved by centered finite differences in space and the forward Euler method in time,

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \Delta t \kappa \left(\frac{T_{i-1,j}^{n} + T_{i+1,j}^{n} + T_{i,j-1}^{n} + T_{i,j+1}^{n} - 4T_{i,j}^{n}}{\Delta x^2} \right) \tag{3}$$

where $\Delta x = \Delta y$ is the grid spacing, n denotes the time step, and $\Delta t < \frac{\Delta x^2}{4\kappa}$ for numerical stability.

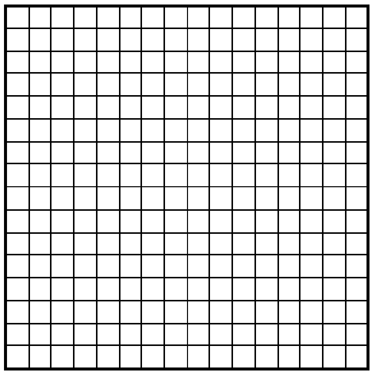
Implementations

- **Serial:** For the serial version, write a program `heat_serial` that runs with command line options `./heat_serial <nx>` for a solution with grid size nx^2.
- **OpenMP:** From the serial version, write a parallel version `heat_omp` that runs with command line options `./heat_omp <nx> <nthreads>`. The OpenMP version should be parallelized using the `!$OMP PARALLEL DO` (Fortran) or `#pragma omp parallel for` (C/C++) directive on the appropriate loops.
- **MPI:** From the serial version, write a parallel version `heat_mpi` that runs with `mpiexec ./heat_mpi <nx>`. Parallelize this using *domain decomposition.*

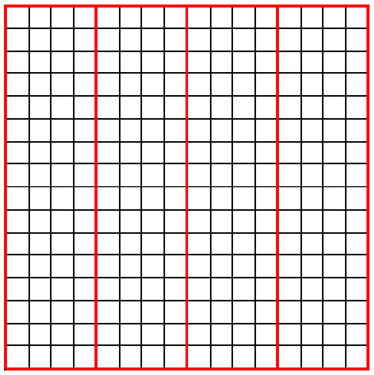
Domain decomposition

Consider a 16×16 grid in figure 1a that we want to run on 4 processors, where the solution variable is in the center of the cells. The easiest way to decompose this is into 4 identical slices of size 16×4 as in figure 1b.

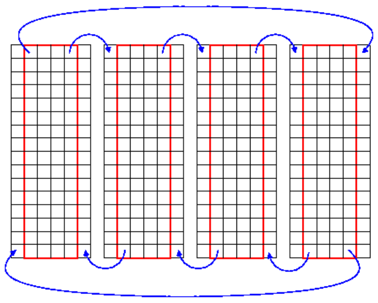
Note now instead of solving a single domain of size 16^2, each MPI process solves for 16×4 elements, with storage for 16×6 elements to account for the *halo* or *ghost* cells indicated in figure 1c. At each time step, you will need to send and receive the appropriate columns of data between processes to populate the ghost cells.



(a) Initial 16^2 domain.



(b) Slice decomposition.



(c) Data dependency.

Figure 1: Domain decomposition

Running parallel jobs

Locally (or on nobel)

While Princeton's clusters are very powerful, they are also very busy. While you are developing and debugging your code, it is better to run OpenMP and MPI locally. You can install on your own computer:

- a modern version of gcc to be able to use OpenMP, and
- OpenMPI (or another MPI library) to be able to use MPI.

Alternatively, you can develop your code on nobel where everything is already installed. Just use `module load openmpi` to load the MPI environment.

And since you need rapid feedback, limit yourself to the smallest grid size so your code runs (and maybe crashes) faster. When you are confident about your code, it's time to run it on the cluster.

On the Adroit cluster

Information on this cluster is available here:
`http://www.princeton.edu/researchcomputing/computational-hardware/adroit/`
You'll need to register for an account if you don't already have one.

Jobs on the Princeton research computing clusters are handled by submitting scripts to the Slurm scheduler. For running the OpenMP and MPI versions on 8 cores, a submission script `heat.run.8` would look like this:

```
#!/bin/bash
# Parallel job using 8 processors:
#SBATCH -N 1                  # number of nodes
#SBATCH --ntasks-per-node=8   # number of processors per node
#SBATCH -t 0:15:00            # run for 15 minutes max
#SBATCH --mail-type=begin     # send email when process begins...
#SBATCH --mail-type=end       # ...and when it ends...
#SBATCH --mail-type=fail      # ...or when it fails.
#SBATCH --mail-user=<yourNetID>@princeton.edu   # Don't forget to define your email!

# Load openmpi environment
module load openmpi

# Make sure you are in the correct directory
cd ~/apc524_hw4/

for nx in 128 256 512
do
    ./heat_omp $nx 8 > heat_omp.$nx.8.out
    srun ./heat_mpi $nx > heat_mpi.$nx.8.out
done
```

The first three `#SBATCH` lines tell the scheduler to reserve 8 processors-per-node on one node, for a maximum of 15 minutes. The standard outputs (and errors) of the programs are being sent to the `slurm-<JobID>.out` files, where the program outputs timing information. You should have a different Slurm script for runs with a fixed number of processors; if you reserve 8 processors for runs that only use 1, it will take longer to be scheduled, and prevent other users from accessing the remaining 7 processors.

Adroit has 8 cores per node, so the OpenMP variant can only run on up to 8. For the MPI version using 16 processors, modify the nodes line to `#SBATCH -N 2` to use 8 cores on each of 2 nodes. Jobs are submitted by the command `sbatch heat.run.8`, and can be checked with `qstat -u $USER`.

Submission

Your submission must include the following:

- `hw4_summary.pdf`: A document containing the relevant plots, and brief discussion of advantages and disadvantages of your OpenMP and MPI implementations.

- Source code for serial, OpenMP, and MPI implementations.
- `Makefile`: Makefile to build the above sources.
- `heat.run.1, heat.run.2, heat.run.4, heat.run.8, heat.run.16`: Slurm job submission files to run all jobs.

As before, submit these files as a bundled Git repository, `hw4.bundle`. To create a suitable bundle, execute the following from within your repository:

```
git bundle create hw3.bundle master
```

More information on Git can be found at `http://git-scm.com/documentation` and in the slides from lecture.

When you are finished, submit the assignment using the CS Dropbox system at `https://dropbox.cs.princeton.edu/APC524_F2014/Parallel`