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

Fall semester 2020

Set 6 - MPI IO, Hybrid MPI + OpenMP, ADI scheme, and PSE method

Issued: December 11, 2020
Hand in (optional): December 24, 2020 08:00am

Grading: To get full points, solve one question out of three. Other questions will not be graded.

Question 1: Diffusion: Parallel I/O (20 Points)

In this question you will implement MPI I/O functions to store the result of a working MPI application. The skeleton code is the working *Diffusion* solver from the previous homework.

The code contains the function `write_mpi_sequential()`, which saves the field from the whole domain into a single file. The function performs an `MPI_Gather` operation that sends the content of all ranks to the *root* rank, which in turns write all the data to a file. This is a sequential operation that cannot be efficiently scaled.

a) Use MPI's I/O functions to store the concentration field in parallel. The goal is to have all ranks collaborate to generate a single output file. Complete the code inside the function `write_mpi_parallel()` to save the concentration field into the file `dump_parallel.dat`. Hints:
   - Skip the rows that contain only ghost cells. For simplicity, keep the columns that store the boundary ghost cells. The datafile will hence contain $N(N+2)$ elements.
   - Visualize the datafile using `plot.py`.
   - Verify your solution using `run_test.sh`.
   - On Euler, load the following modules

```
module load new open_mpi/2.1.1 python/3.7.1
```

b) Perform a scaling analysis to compare the performance taken by the naive approach and your parallel implementation. Fix the number of MPI ranks to the number of cores in your system of choice (the more, the better) and vary the size of the grid between 1024 and 6144 with steps of size 512.

c) Plot your results showing the grid sizes in the x-axis and the time taken for both sequential and parallel I/O on the y-axis. Write a paragraph explaining your findings.

Question 2: Cannon: Hybrid MPI+OpenMP (40 Points)

Cannon's Algorithm[1] is one of the first proposals for computing dense square matrices multiplication on distributed systems. The algorithm presents a peculiar communication pattern that is well suited for MPI.

a) Implement Cannon's algorithm using the provided skeleton code by filling the TODO sections. The verification algorithm will report whether your solution is correct. The pseudocode and overview of Cannon's algorithm can be found here: http://people.eecs.berkeley.edu/~demmel/cs267/lecture11/lecture11.html#link_5 (look at the implementation on a 2D mesh). You can use **make test** to test your implementation. Note: for simplicity, the input matrices are initialized already skewed so you may omit the initial skewing procedure in the algorithm. On Euler, load the following modules

```
module load new open_mpi/2.1.1 openblas python/3.7.1
```

b) Cannon's algorithm decomposes a single matrix-matrix multiply into a series of the same operation on smaller grids, each on a specific rank. Assuming your code is running on a large distributed system, each MPI rank may have access to multiple CPU cores that you could use to further improve the performance of your code.

In this subquestion, replace the original call to BLAS' `dgemm` by a call to `ompCannon()`, a modified version of Cannon's algorithm that takes into account shared memory. In your solution, you will have OpenMP threads collaborate in the calculation of local submatrix updates. Since all threads work on the same shared memory space, they do not need to explicitly exchange submatrices, but simply exchange/swap pointers between their A and B sub-submatrices. To perform multiplication of these sub-submatrices, each thread will call `dgemm`.

Fill in the *ompCannon* function to create an OpenMP parallel version of Cannon's algorithm. Your code should run as a distributed application with MPI and use shared memory for the local updates. You can use **make test** to test your code. Hint: be mindful the initial skew required in the algorithm.

Note: On Euler, hybrid jobs are not fully supported (https://scicomp.ethz.ch/wiki/Hybrid_jobs). Your program may run but not show any speedup with OpenMP. If you want to achieve speedup, which is optional for this exercise, you can run the job as follows. Load the modules and set `OMP_NUM_THREADS=16` before submitting the job. After the allocation is granted, you can set `OMP_NUM_THREADS` to a smaller number. Here we request two nodes with 36 cores for an interactive job.

```
module load new open_mpi/2.1.1 openblas python/3.7.1
export OMP_NUM_THREADS=16
bsub -n 72 -R fullnode -Is bash
```

Once your interactive job is running, you need to unset the environment variable which controls the affinity of threads and pass flags to `mpirun` to allow the MPI processes use more cores.

```
unset LSB_AFFINITY_HOSTFILE
OMP_NUM_THREADS=16 mpirun --map-by node:PE=16 -n 4 ./cannonHybrid 512
```

[1] Lynn Elliot Cannon. 1969. A cellular computer to implement the kalman filter algorithm. https://dl.acm.org/doi/book/10.5555/905686

Question 3: Diffusion: ADI scheme (50 points)

Consider the two-dimensional diffusion equation

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

where ϕ is the concentration of a substance at position $(x,\ y)$ and at time t, and D is the diffusion constant. The diffusion process happens in the domain $[0,1]^2$ and in time $t \in [0,T]$. The concentration is zero on the boundaries of the domain. The initial concentration is

$$\phi(x,y,0) = \big(\sin 4\pi x\ \sin 2\pi y\big)^2.$$

a) (5 points) Discretize the equation using the Alternating Direction Implicit (ADI) scheme. Write down the linear system of one implicit substep in the matrix form. Is the matrix diagonally dominant?

b) (35 points) Implement the ADI scheme with Thomas algorithm for the corresponding implicit systems. To get started, the program `diffusionADI.cpp` implements the fully explicit scheme and the script `plot.py` visualizes the solution. On Euler, load the following modules

```
module load new python/3.7.1
```

c) (5 points) The explicit and implicit substeps can be easily parallelized with OpenMP by splitting the outer loop among multiple threads. Discuss if the same approach is applicable with MPI and comment on the additional complexity.

d) (5 points) The skeleton code computes the maximum of the field with the function `getMax()` and writes it to a file as a time series. Use the script `vary_dt_stat.sh` to run the program for various time steps Δt and plot the maximum of the field over time. Do you observe convergence of the solution as the time step gets smaller? For every Δt report the corresponding number $R = \frac{D\Delta t}{\Delta x^2}$ and discuss if the fully explicit scheme would be stable, i.e. if the stability condition $R < 1/4$ is satisfied.