ETH zürich

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

High Performance Computing for Science and Engineering I

Fall semester 2019

Set 4 - Blocking MPI Routines and Processor Pipelines

Issued: November 8, 2019
Hand in (optional): November 22, 2019 08:00am

Question 1: Diffusion in 2D (50 points)

Mass diffusion in a medium can be described by the following equation

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

where $\rho(x,y,t)$ represents the concentration of a substance at position $\boldsymbol{r}$ and time t, and D represents a constant diffusion coefficient. Lets define the domain Ω in two dimensions as $x, y \in [-1, 1]$. We will use the boundary condition:

$$\rho(x,y,t) = 0 \quad \forall\, t \geq 0 \,\text{and}\, (x,y) \notin \Omega \tag{2}$$

and an initial distribution:

$$\rho(x,y,0) = \begin{cases} 1 & |x,y| < 1/2 \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

For this exercise, use only **blocking** MPI routines.

a) In the provided skeleton code, the 2D diffusion equation is computationally solved on a uniform grid, using central finite differences in space and forward Euler in time. Implement the MPI parallelization of the 2D diffusion equation by filling in all parts of the code marked by `TODO`, in functions `advance` and `main`. Decompose the domain using tiling decomposition scheme (described in the lecture notes). (i.e. distribute the rows evenly to the MPI processes).

b) Compute an approximation to the integral of $\rho(x,y,t)$ over the entire domain in the function `compute_diagnostics` by implementing the missing MPI parts marked by `TODO`, and plot the result as a function of time using $D = 1$, $L = 2$ and $N = 100$.

c) Compute a histogram of $\rho(x,y,t)$ over the entire domain in the function `compute_histogram` by implementing the missing MPI parts marked by `TODO`, and plot the resulting histogram at simulation time $t = 0.5$ using $D = 1$, $L = 2$ and $N = 100$.

Question 2: MPI Bug Hunt (20 points)

Find the bug(s) in the following MPI code snippets and find a way to fix the problem!

a)

```
const int N = 10000;
double* result = new double[N];
// do a very computationally expensive calculation
// ...

// write the result to a file
std::ofstream file("result.txt");

for(int i = 0; i < N; ++i){
    file << result[i] << std::endl;
}

delete[] result;
```

b)

```
// only 2 ranks: 0, 1
double important_value;

// obtain the important value
// ...

// exchange the value
if(rank == 0)
MPI_Send(&important_value, 1, MPI_DOUBLE, 1, 123, MPI_COMM_WORLD);
else
MPI_Send(&important_value, 1, MPI_DOUBLE, 0, 123, MPI_COMM_WORLD);

MPI_Recv(
&important_value, 1, MPI_INT, MPI_ANY_SOURCE,
MPI_ANY_TAG, MPI_COMM_WORLD, MPI_STATUS_IGNORE
);

// do other work
```

c) What is the output of the following program when run with 1 rank? What if there are 2 ranks? Will the program complete for any number of ranks?

```
MPI_Init(&argc, &argv);

int rank, size;
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);

int bval;
if (0 == rank)
{
    bval = rank;
    MPI_Bcast(&bval, 1, MPI_INT, 0, MPI_COMM_WORLD);
}
else
{
    MPI_Status stat;
    MPI_Recv(&bval, 1, MPI_INT, 0, rank, MPI_COMM_WORLD, &stat);
}

cout << "[" << rank << "] " << bval << endl;

MPI_Finalize();
return 0;
```

Question 3: Processor Pipelining (30 points)

a) In the lecture we have looked at the pipeline block diagram of the Intel Coffee Lake CPU, which is identical to the pipeline in the Intel Skylake CPUs. You can find this diagram on page 78 in the lecture slides about CPU pipelining or directly at this link

https://en.wikichip.org/w/images/7/7e/skylake_block_diagram.svg

i) Use this diagram to determine the number of data loads and data stores you can perform in the same cycle.

ii) With the help of the lecture slides, explain why the architecture designers have chosen the particular ratio of data load and data store instructions.

b) Suppose we have the following functional units to process instructions with latencies:

IF	(Instruction Fetch)	2 ns
ID	(Instruction Decode)	2 ns
EX	(Execute Instruction)	3 ns
MEM	(Physical Memory Access)	6 ns
WB	(Write Back Result)	2 ns

i) If we use these units to build a non-pipelined processor, how long does it take to execute a single instruction?

ii) If we use these units to build the 5-stage pipelined processor discussed in the lecture, what is the shortest possible cycle time, i.e. the time for executing a single stage?

iii) How long does it take to execute at least N instructions in a *full* pipeline? Assume the pipeline is empty at the beginning.

iv) What is the speedup of this pipelined processor over the non-pipelined implementation? Assume a large number of instructions that do not cause pipeline stalls.