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

HW 3 - Advanced MPI and Parallel Tasking

Issued: March 29, 2021
Due Date: April 19, 2021, 10:00am

2-Week Milestone: Solve tasks 1 and 2

Task 1: 2D Wave Equation (35 Points)

In this exercise we will employ a distributed solver for the wave equation. The communication between processes is done with the help of MPI's vector datatypes. This is much less error prone than writing a code where processes send and receive individual messages of fundamental datatypes (i.e. MPI_DOUBLE, MPI_FLOAT).

The wave equation in 2D is given by

$$\frac{\partial^2 u}{\partial t^2} - c^2 \left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \right) = 0. \tag{1}$$

Equation (1) can easily be discretized using centered finite differences space and forward differences in time (Euler integrator). By applying these discretization techniques, we can reformulate eq. (1) into an update function with a 5-point stencil

$$u_{i,j}^{n+1} = 2u_{i,j}^n + c^2 \frac{\delta t}{h^2} (u_{i+1,j}^n + u_{i-1,j}^n + u_{i,j+1}^n + u_{i,j-1}^n - 4u_{i,j}^n), \tag{2}$$

where $u_{i,j}^n = u(n\,\delta t, (i+\frac{1}{2})h, (j+\frac{1}{2})h)$ and $h = 1/N_{tot}$.

The set of initial and (periodic) boundary conditions we will use to solve eq. (1) are as follows

$$\begin{aligned} u(0,x,y) &= f(x,y), \\ \frac{\partial u}{\partial t}(0,x,y) &= 0, \\ f(x,y) &= 0.3\ e^{-50r}, \\ u(t,0,y) &= u(t,1,y), \\ u(t,x,0) &= u(t,x,1) \quad \forall x,y \in [0,1]. \end{aligned} \tag{3}$$

This models a circular wave centered in the middle of the domain $[0,1] \times [0,1]$ as can be seen in fig. 1. With $r = \sqrt{(x-0.5)^2 + (y-0.5)^2}$ being the displacement from the center of the domain.

The skeleton code for this problem can be found in `wave.cpp`. It contains the necessary information to set up the grid and communicator in the constructor. For this problem it is

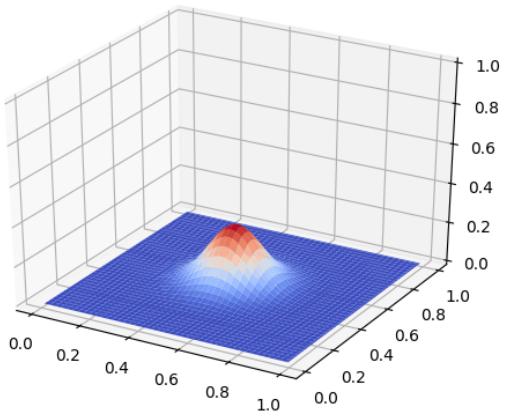

Figure 1: Initial displacement at $t = 0$

suitable to use a Cartesian MPI topology to be able to easily obtain information from the neighboring ranks. Each rank holds a local $(N + 2) \times (N + 2)$ array, with a padding of 1 halo cell on each. At each timestep the ranks then communicate the data on the boundary of the inner $N \times N$ grid to their respective neighbors.

a) **(5 Points)** Set up a Cartesian MPI topology, given the ranks in `MPI_COMM_WOLRD` and then compute the neighbor ranks. There are two sections in the code (in "`/* ... */`" style comments) which you have to uncomment when finished with this subquestion (for finding the relative position from the origin [0,0] and for saving the grid).

b) **(15 Points)** In the routine `WaveEquation::run()` create the necessary custom MPI datatypes to send and receive the boundaries between the processes. For this task, use the convenient `MPI_Type_create_subarray` function. In total there should be 8 custom datatypes (one `send` and one `recv` type for each face). Do not forget to free them when you do not need them anymore!

c) **(15 Points)** In the `while`-loop, use the previously created types to exchange the boundary values.

Have a look at the `README` file for instructions on how to compile and run the program. We also provide a plotting script to create an animation of the time evolution of the solution (might be helpful for debugging purposes).

Task 2: Particle Simulation with custom MPI Datatypes (25 Points)

A distributed N-body solver employs particles with positions (x_i, y_i, z_i) and masses $m_i = m$ for $i = 0, 1, \ldots, N-1$. Since the masses stay constant during the simulation, the MPI ranks need to exchange only the updated positions. Use the skeleton code to implement MPI datatypes in order to send only the positions of the particles in the given data structure (SoA or AoS).

a) **(10 Points)** Complete the Makefile and the first part of the skeleton: Create an MPI datatype that accounts only for the positions of the particles (given an AoS data structure). Then use it to send all positions from rank 0 to rank 1 with a single `send/recv` pair.

b) **(15 Points)** Complete the second part of the skeleton. This part uses a different data structure to represent the particle positions (SoA). Create an MPI datatype for sending or receiving the positions of particles in this data structure. Now send the positions (stored in the AoS datastructure) from rank 0 with the datatype defined in the previous subquestion and receive them with the newly created datatype on rank 1, storing them in the SoA datastructure (again with a single `send/recv` pair).

Task 3: Parallel Tasking Theory (24 Points)

You work at a newly instituted supercomputing center and each morning you receive a queue of 8 jobs that need to be executed. Every job can be executed **independently** and cannot be parallelized. The computational cost of each job is given in the following in terms of a reference computational load C, but you do not know this load a-priori to design an optimal parallel tasking strategy.

Job order:	1	2	3	4	5	6	7	8
Comp. load:	8	8	1	1	1	1	2	1

We assume for simplicity that the load can be directly translated to a fixed reference computational time, e.g. that a job of load $C = 7$ can be executed by a single rank in $T = 7$ time units. In the following, time measurements need to be provided in terms of these time units. You are provided with a small compute node with **4 cores**, each capable of running a single job at a time, and you are asked to analyze parallel tasking algorithms.

a) **(4 Points)** In the classical divide-and-conquer method the tasks are divided to the available processors. You cannot affect the ordering of the jobs. This leads to the following partition:

Job order:	1	2	3	4	5	6	7	8
Comp. load:	8	8	1	1	1	1	2	1
Exec. by rank:	1	1	2	2	3	3	4	4

Compute:

1. the total run-time T_{total}
2. the average rank run-time T_{avg}
3. the load imbalance ratio I
4. the average rank idle time W_{avg}.

b) **(10 Points)** Now you are asked to apply the producer-consumer strategy in the aforementioned example. Note that you cannot affect the ordering of the jobs assigned to the ranks. In this example, the computational cost of communicating a job or a signal, e.g. indicating that the rank is idle/ready, is $C_{com} = 1$. Complete the following producer-consumer schedule with one of the signals S_i for $i \in \{1, \ldots, 4\}$, or J_j for $j \in \{1, \ldots, 8\}$, where S_i means that the rank is currently sending a message/job to rank i, and J_j means that the the rank is currently executing job j. The master rank is rank 1. Leave the square **empty** if a rank is idle at a given time step.

Rank 1:	S_2										S_2	S_3					
Rank 2:		J_1	J_1	J_1	J_1	J_1	J_1	J_1	J_1	S_1		J_6	S_1					
Rank 3:			J_2	J_2	J_2													
Rank 4:																		
Time:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

1. What is the total run-time T_{total} in this case?
2. What is the average rank run-time T_{avg}?
3. What is the load imbalance ratio I?
4. What is the average rank idle time W_{avg} (when a rank is communicating it is **not** considered idle)?
5. What is the speed-up compared to the divide-and-conquer strategy (if any)?
6. By changing the communication protocol, we manage to reduce the cost of communication C_{com}. How does the load imbalance change **qualitatively**? Would that be always the case? (Check the limit $C_{com} \to 0$).

c) **(10 Points)** In the **divide-and-conquer** strategy, the load imbalance of the jobs may lead to various unwanted scenarios resulting in high execution times. Recall that the compute node you posses has **4 cores**. Assume that you have three simulations whose computational loads are $C = 1, 2$ and 8. For scientific validation purposes you have to execute the simulation with $C = 1$ five times, the simulation with $C = 2$ one time and the simulation with $C = 8$ two times.

1. The worst case scenario (in terms of run-time) occurs when by chance, both computationally heavy jobs $C = 8$ (running the $C = 8$ simulation two times) are assigned to the same rank. Can you describe when the **second-worst case** scenario occurs in this case? What is an **example** of an ordering of the computational loads that leads to the **second-worst case** scenario?
2. What is the probability of **any** second-worst case scenario ordering occurring (Assume that an external client randomly sets the ordering, so that every **ordering of the computational work loads** is equiprobable)[1]?

Guidelines for reports submissions:

- Submit **a single zip file** with your solution (code in a compressed format (Task 1 and 2), and a pdf file (Task 3)) via Moodle until April 19, 2021, 10:00am. The total size of the zip file must be **less than 20 MB**.

[1] The ordering $[8, 8, 1, 1, 1, 1, 2, 1]$ has the same probability of occurring as $[1, 1, 1, 1, 1, 2, 8, 8]$