ETH zürich

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

Spring semester 2022

Set 3 - Advanced MPI High Throughput Computation

Issued: March 28, 2022
Hand in (optional): April 04, 2022 12:00am

Question 1: 2D Wave Equation (35 + 5 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

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

eq:pde can easily be discretized using centered finite differences space and central differences of the second derivative in time. By applying these discretization techniques, we can reformulate eq:pde into an update function with a 5-point stencil

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

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 (1) are as follows

$$\begin{aligned} u(0,x,y) &= f(x,y), \\ \frac{\partial u}{\partial t}(0,x,y) &= 0, \\ f(x,y) &= 1 - \sin(\pi r) \cdot \exp(-r), \\ u(t,0,y) &= u(t,1,y), \\ u(t,x,0) &= u(t,x,1) \quad \forall x,y \in [0,1]. \end{aligned} \tag{4}$$

This models a circular wave centered in the middle of the domain $[0,1] \times [0,1]$ as can be seen in fig:i$_c.With$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 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.

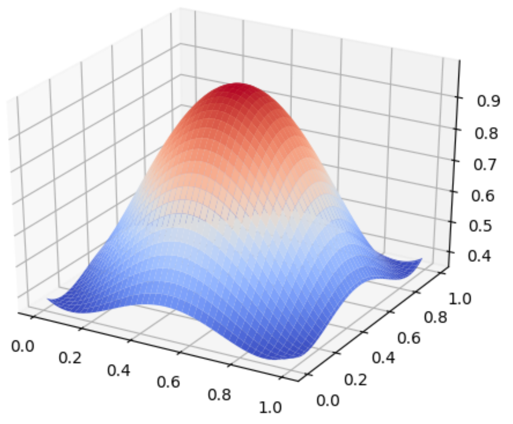

Figure 1: Initial displacement at $t = 0$

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.

   Hint: Verifying that the energy functional defined in equation (5) remains roughly constant is an indication that your implementation is correct.

d) (**+5 points**) This exercise is optional and is only intended to show how one can potentially derive energy functionals for the specially interested. This task can only provide bonus points; you don't have to solve this exercise to get the top grade

   In order to test our numerical solver, it's often very convenient if we are able to verify that some quantity intrinsic to the PDE remains constant through time. There generally exists such quantities for hyperbolic boundary value problems such as the wave equation. We call these quantities for the PDE's energy functional and the PDE in equation (1) has the energy functional as given in equation (5).

$$E(t) := \frac{1}{2}\int_{\Omega} \left(\frac{\partial u(\mathbf{x},t)}{\partial t}\right)^2 + c^2 \left\|\nabla_{\mathbf{x}} u(\mathbf{x},t)\right\|^2 d\mathbf{x} \tag{5}$$

   Using equation (1) and (5) prove that $E'(t) = 0$, meaning that this energy functional remains constant through time.

Hint: Use Green's first formula:
$\int_\Omega \nabla_{\mathbf{x}} u(\mathbf{x}) \cdot \nabla_{\mathbf{x}} v(\mathbf{x}) d\mathbf{x} = \int_{\partial\Omega} u(\mathbf{x}) \cdot \nabla_{\mathbf{x}} v(\mathbf{x}) \cdot \mathbf{n}(\mathbf{x}) d\mathbf{x} - \int_\Omega u(\mathbf{x}) \cdot \Delta_{\mathbf{x}} v(\mathbf{x}) d\mathbf{x}$

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).

Question 2: Parallel Tasking Theory (30 points)

This question will provide an insight into more complex versions of `MPI_Datatype`. We will be sending individual messages between two ranks as well as general structures of messages. In order to do this, we will have to define our own versions of `MPI_Datatype`. The messages will be of the following shape:

```
struct Message {
  double x1, x2;
  int z1, z2;
};
```

a) (5 points)
Complete the function `init_base_type` and initialize `base_type` to be a custom `MPI_Datatype` for sending a struct of type `Message`.

b) (10 points)
Now we want to start sending structures of messages. This can for instance be a structure of type `std::vector<Message>` or for example a `std::list<Message>`. We want to be able to handle any STL-container with an iterator. This means that your code should work for any data container where we can write the following:

```
for(const auto& x :  some_container)  {
  do_some_stuff(x);
};
```

Complete the function `init_aos_type` for a general container.

c) (5 points)
From our extensive work with vectors, we suspect that the container `std::vector` is special and we want to handle this case seperately. Complete the function `init_aos_type` for a `std::vector`. In this question you are prohibited from using `MPI_Type_create_struct`.

d) (10 points)
Sometimes we want to send one structure and receive it in a structure with a compete different layout. This is for instance the case when we want to send an array of structures and receive it in a structure of arrays. This is a common use case in for instance particle simulations. Here, we want to generalize this concept and send a container of messages and receive it in a structure of containers as can be seen bellow.

```
struct MessageArray {
  Container<double> x1, x2;
  Container<int> z1, z2;
};
```

Here, `Container<...>` is some generic STL-container with an iterator. This means that we can iterate over the elements in `m.x1` with the following code.

```
for(auto x1_curr = m.x1.begin(); x1_curr != m.x1.end(); ++x1_curr) {
  do_some_stuff_with_the_element(*x1_curr)
};
```

Complete the function `init_soa_type` for a general container.

Question 3: Parallel Tasking Theory (28 points)

This exercise will provide a theoretical insight into how Korali's load balance is affected by running experiments simultaneously instead of sequentially. In order to analyse the situation, we make the following assumptions:

1. We have N ranks and we run E experiments. Each experiment has exactly T generations and each generation consist of N samples; one for each rank.
2. The duration for computing one sample is uniformly distributed. We denote the runtime for the sample ran on rank k in the t-th generation of experiment e with the random variable $X_k^{E \cdot t+e} \sim$ Unif$([0,1])$. Hence, for any $l \in \mathbb{N}$ we have that $\mathbb{E}[X_k^l] = \frac{1}{2}$.

a) (10 points)

In this sub-question we are interested in a single generation of a sequentially ran experiment. Denote I^l as the imbalance ration in the t-th generation of experiment e where $l = E \cdot t + e$. Moreover assume that N is large enough such that the law of large numbers allows us to make the following approximation for the load imbalance ratio:

$$I^{te} = \frac{\max\limits_{k=1}^{N} X_k^{te} - \frac{1}{N}\sum\limits_{k=1}^{N} X_k^{te}}{\max\limits_{k=1}^{N} X_k^{te}} \approx \frac{\max\limits_{k=1}^{N} X_k^{te} - \frac{1}{2}}{\max\limits_{k=1}^{N} X_k^{te}} \tag{6}$$

For $\alpha \in [0,1]$, compute $\mathbb{P}[I^{te} > \alpha]$ for any finite N as well as the limit $\lim\limits_{N\to\infty} \mathbb{P}[I^{te} > \alpha]$. Give an interpretation of this result.

b) (5 points)

We would now like to investigate what happens when we run multiple experiments simultaneously. In order to treat this formally, we denote the runtime of computing the first l samples on rank k with the random variable $Y_k^l := \sum\limits_{i=1}^{l} X_k^i$. The important insight is that we can always utilize all ranks fully if we can assure that all ranks have finished computing their sample for the t-th generation in experiment e before any rank wants to start computing the sample in the $t+1$-th generation in the same experiment, or more formally we want to ensure the condition in inequality 7.

$$\forall k\, \forall e\, \forall t \ : \ Y_k^{E \cdot t+e} < Y_k^{E\cdot(t+1)+e} \tag{7}$$

To directly find a lower bound on the probability of inequality (7) holding can be challenging. We therefore want to work the more manageable inequality (8).

$$\forall k\, \forall l \ : \ |Y_k^l - \mathbb{E}[Y_k^l]| < \frac{E}{4} \tag{8}$$

Show that inequality (8) implies inequality (7).

c) (**5 points**)

In order to find an upper bound of inequality (8) not holding, we use the Chernoff-bound and get the following upper bound:

$$\mathbb{P}\left[|Y_k^l - \mathbb{E}[Y_k^l]| \geq \frac{E}{4}\right] \leq 2 \cdot \exp\left(-\frac{E^2}{8t}\right) \tag{9}$$

Using inequality (9) show that the probability that any Y_k^l deviates more than $\frac{E}{4}$ from its mean is upper-bounded by inequality (10).

$$\mathbb{P}\left[\vee_{k=1}^{N} \vee_{t=1}^{T \cdot E} \left(|Y_k^l - \mathbb{E}[Y_k^l]| \geq \frac{E}{4}\right)\right] \leq 2 \cdot N \cdot T \cdot E \cdot \exp\left(-\frac{E}{8T}\right) \tag{10}$$

Hint: Use the union bound: $\mathbb{P}[\cup_i A_i] \leq \sum_i \mathbb{P}[A_i]$

d) (**8 points**)

Using the results from b) and c), conclude that for any number of ranks N if T and E are large enough, then for any $\epsilon, \delta > 0$ we can guarantee that the load imbalance ratio is smaller than ϵ with probability larger than $1 - \delta$. Is this always practically possible or does there exist any limitations to this idealized scenario?

Hint: We are not looking for a formal proof - sketching out the main ideas is enough.