Revisiting MMPDE algorithm with Python: A possible direction into parallel algorithm of MMPDE

Henokh Lugo Hariyanto and Arif Wicaksono Septyanto
Institut Teknologi Kalimantan

Introduction

- With increasing in a trend of muticore CPUs, demands to develop parallel software is high [3]
- Solving PDE with initial value that contains jump is very problematic with uniform mesh. In here we will demonstrate how to solve it using an adaptive mesh.
- Adaptive mesh categorized into three types: r-adaptive mesh, h-adaptive mesh, and p-adaptive mesh. The moving mesh PDE (MMPDE) algorithm is categorized as r-adaptive mesh.
- One of the most recent work to parallelize MMPDE [2] is only shown for the mesh generation, without the consideration to couple it for solving PDE

Problem definition

MMPDE algorithm is tested to the following a toy problem Burger's equation with an initial values that contains jump

$$\begin{cases} u_t = \varepsilon u_{xx} - \left(\dfrac{u^2}{2}\right)_x, \quad x \in (0,1), t > 0 \\ u(0,t) = 1; \quad u(1,t) = 0 \\ u(x,0) = \dfrac{0.1\exp\{\frac{-x+0.5}{20\varepsilon}\} + 0.5\exp\{\frac{-x+0.5}{4\varepsilon}\} + \exp\{\frac{-x+0.375}{2\varepsilon}\}}{\exp\{\frac{-x+0.5}{20\varepsilon}\} + \exp\{\frac{-x+0.5}{4\varepsilon}\} + \exp\{\frac{-x+0.375}{2\varepsilon}\}} \end{cases} \tag{1}$$

The above equation has an exact solutions

$$u(x,t) = \frac{0.1\exp\{\frac{-x+0.5-4.95t}{20\varepsilon}\} + 0.5\exp\{\frac{-x+0.5-0.75t}{4\varepsilon}\} + \exp\{\frac{-x+0.375}{2\varepsilon}\}}{\exp\{\frac{-x+0.5-4.95t}{20\varepsilon}\} + \exp\{\frac{-x+0.5-0.75t}{4\varepsilon}\} + \exp\{\frac{-x+0.375}{2\varepsilon}\}}$$

Methods

The idea of MMPDE is to find the transformation function from computational coordinates ξ to physical coordinates x, such that it satisfied the gradient flow of the following functional.

$$I[\xi] := \frac{1}{2}\int_a^b \frac{1}{\rho(x,t)}\left(\frac{\mathrm{d}\xi}{\mathrm{d}x}\right)^2 \mathrm{d}x$$

where $\rho(x,t)$ is the mesh density defined in a mesh $\mathcal{I}_h := x_1 = a < x_2 < \ldots < x_N = b$ for a specific time t as

$$\int_{x_1}^{x_2} \rho(x)\,\mathrm{d}x = \ldots = \int_{x_{N-1}}^{x_N} \rho(x)\,\mathrm{d}x$$

We can write explicitly the gradient flow of $I[\xi]$

$$\frac{\partial \xi}{\partial t} = -\frac{P}{\tau}\frac{\delta I}{\delta \xi}$$

substitute $\delta I/\delta\xi$ from the steady state of $I[\xi]$, we have

$$\frac{\partial \xi}{\partial t} = -\frac{P}{\tau}\frac{\partial}{\partial x}\left(\frac{1}{\rho}\frac{\partial \xi}{\partial x}\right) \tag{2}$$

where P is a positive-definite differential operator which can be chosen with considerate flexbility, and $\tau > 0$ is a user specified parameter for adjusting the response time of mesh movement to change in $\rho(x,t)$.

In Fig. 1, we showed the program flow how to implement MMPDE. This flowchart is an adaptation of [1] with more detail.

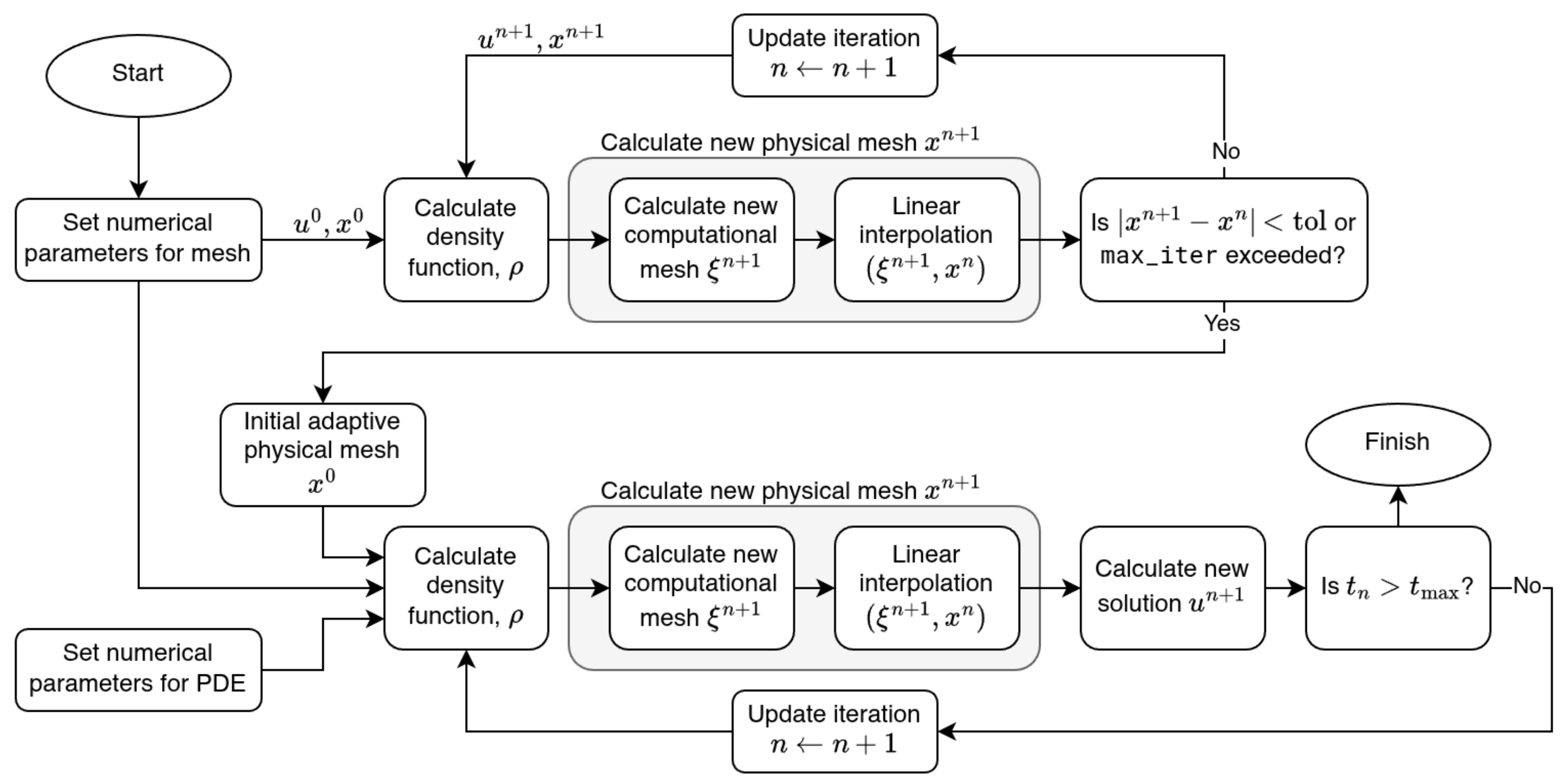
Start
Set numerical parameters for mesh
u^0, x^0
u^{n+1}, x^{n+1}
Update iteration $n \leftarrow n+1$
Calculate density function, ρ
Calculate new physical mesh x^{n+1}
Calculate new computational mesh ξ^{n+1}
Linear interpolation (ξ^{n+1}, x^n)
Is $|x^{n+1} - x^n| < \mathrm{tol}$ or max_iter exceeded?
No
Yes
Initial adaptive physical mesh x^0
Set numerical parameters for PDE
Calculate density function, ρ
Calculate new physical mesh x^{n+1}
Calculate new computational mesh ξ^{n+1}
Linear interpolation (ξ^{n+1}, x^n)
Calculate new solution u^{n+1}
Is $t_n > t_{\max}$?
No
Finish
Update iteration $n \leftarrow n+1$

Figure 1. A program flow to solve a PDE using MMPDE. It is divided into two parts: finding the best initial mesh, and performed calculation to the PDE solution.

Preliminary results

We tested our implementation in Python for both uniform mesh and adaptive mesh. All programs are written in Python v3.11. Some libraries that we use are NumPy v1.25, SciPy v1.11, and mpi4py v4.1. We running all programs in the machine with specification Intel® Core™ i7-9750H CPU @ 2.60GHz × 12.

In Fig. (2), we showed the initial mesh after we solved the steady state solution of Eq. (2). The discrete value of $\rho(x_i)$ is calculated with the formula $\langle\rho\rangle_{\mathcal{I}_j} := \left(\int_{I_j}\rho(x)\,\mathrm{d}x\right)/|I_j|$ where $I_j := x_j - x_{j-1}$. The mesh points are accumulated to the area near the jump to increase the resolution of the solutions.

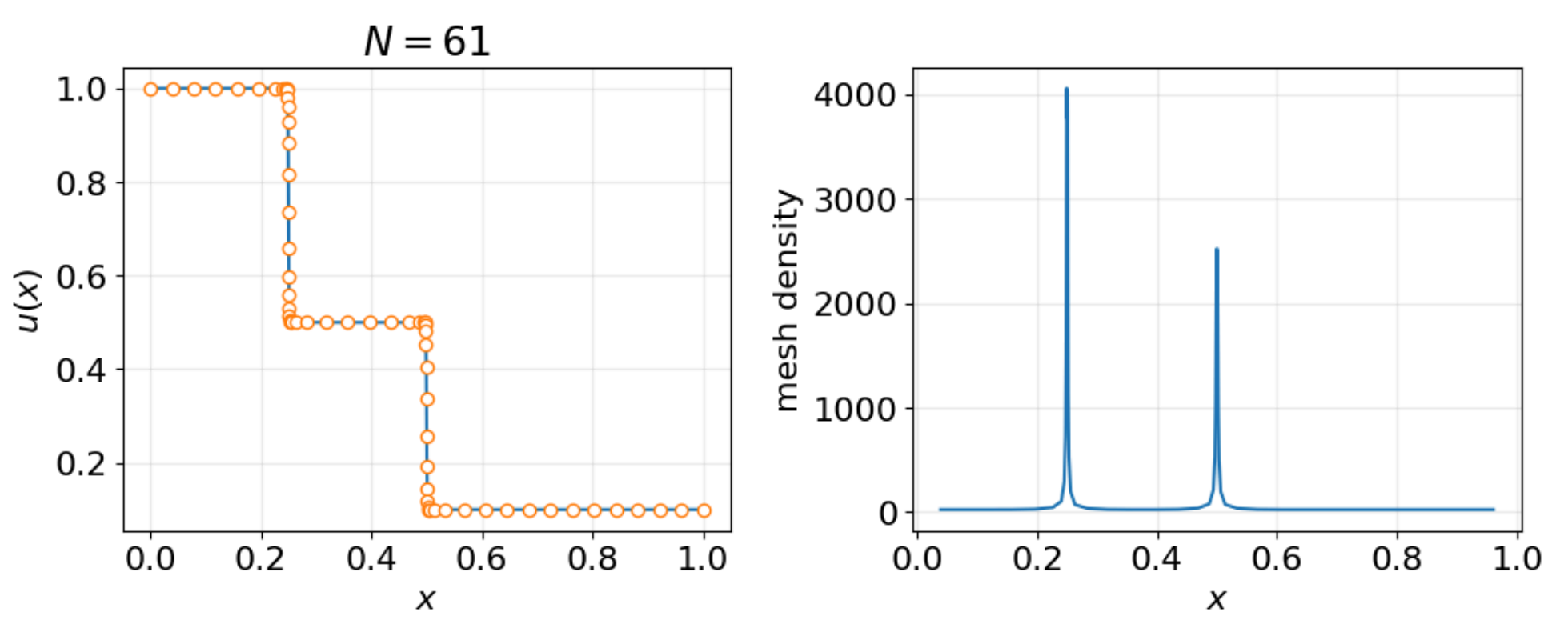

Figure 2. (left) the distribution of mesh points in the domain. (right) discrete value of $\rho(x_i)$

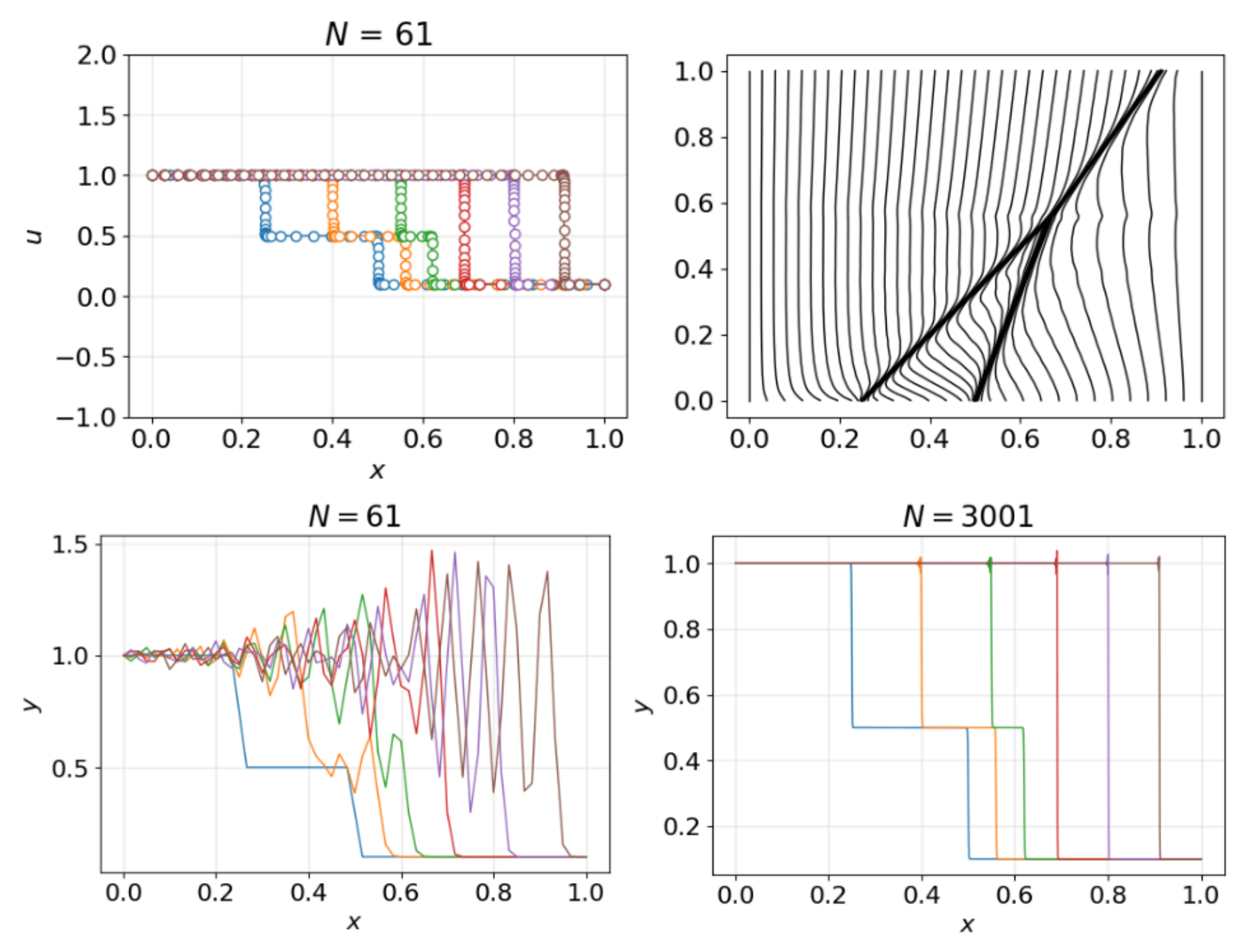

Figure 3. (top-left) the solution of Eq. (1) using MMPDE; (top-right) the evolution of mesh $\mathcal{I}_h$; (bottom-left) the solution of Eq. (1) using uniform mesh with coarse mesh; (bottom-right) the solution of Eq. (2) using uniform mesh with fine mesh;

In Fig. (3) we clearly see that with the same amount of number of points, adaptive mesh is outperformed uniform mesh in its quality of solution.

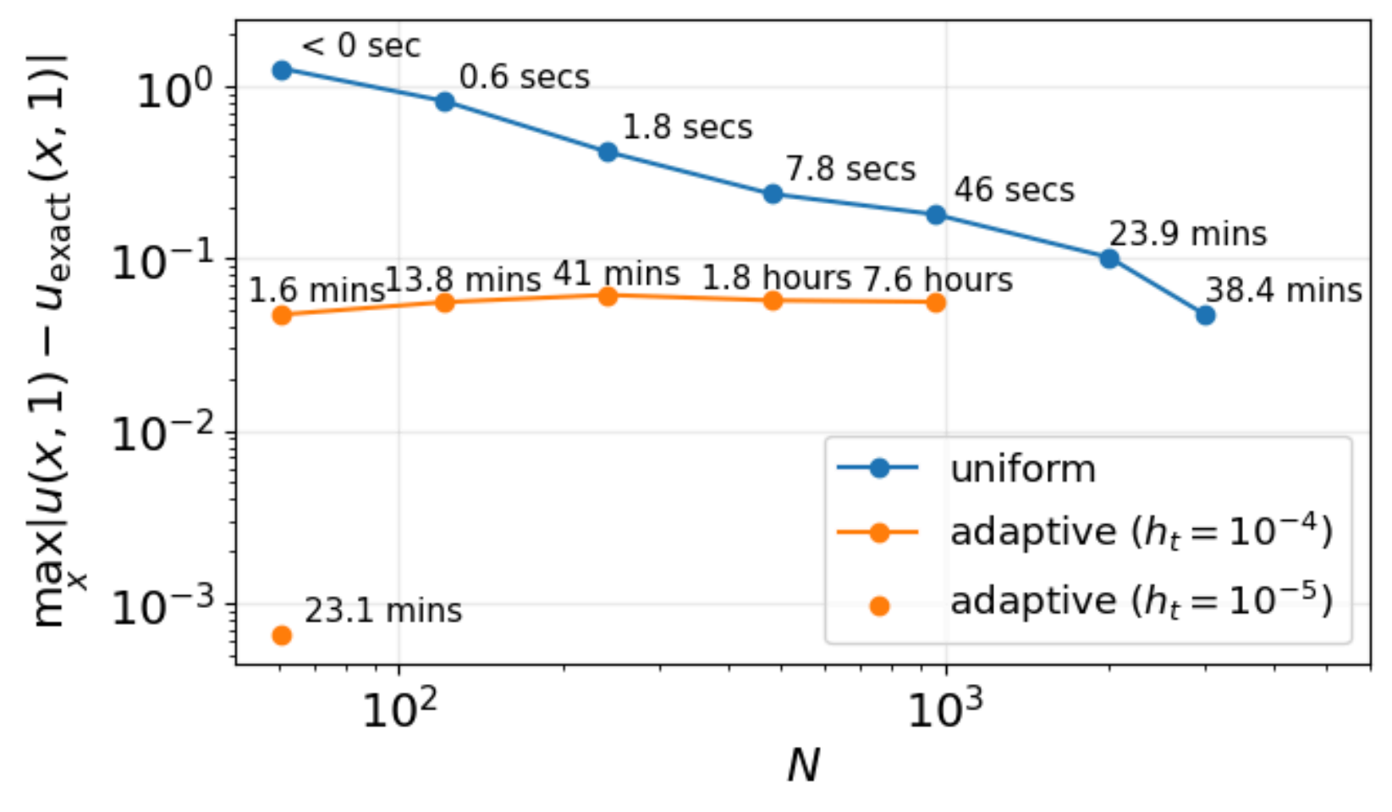

Figure 4. All L^1-norm error for uniform mesh and adaptive mesh for different time steps ($h_t = 10^{-4}$ and $h_t = 10^{-5}$)

Comparsion to its L^1-norm error, it is really expensive to achieve the same amount of error that has been attained by the adaptive mesh. In Fig. (4), we notice that it only takes 1.6 minutes using adaptive mesh with $h_t = 10^{-4}$ to get the similar error with using uniform mesh (38.4 minutes). There is a speedup around 24 times.

On-going work

After we have tested our Python implementation gained a speed up and work according to the plan, we moved to the next step to speed up more using mpi4py library. This plan for parallel implementation is intra-node communicatio means that we utilize the cores in the same chip, not in the different machine or node.

To design the paralle algorithm of MMPDE, we identify the primitive task (e.g., the calculation of new physical mesh in mesh calculation and PDE calculation, see Fig. (1)), identifying data communication pattern among the discretization mesh, and looking for ways to agglromerate those task. This procedure has been confirmed to be scalable the vibrating string equation [5].

Some technical details how to implement our parallel version of MMPDE heavily follows the structure in [4]. But instead of using multiprocessing module, we use mpi4py.

Acknowledgment

This research is funded by Indonesian Directorate General of Education, Research and Technology through Young Lecturer Scheme with grant number 7713/IT10.II/PPM.04/2023.

References

[1] W. Huang and R.D. Russel. 2010. *Adaptive moving mesh methods.* Vol. 174. Springer Science & Business Media.
[2] C. Tannahill and J. Wan. 2023. MM-ADMM: Implicit integration of MMPDEs in parallel. *Computer & Mathematics with Applications* 141 (2023), 67-79. doi:10.1016/j.camwa.2023.03.019
[3] P. Pacheco and M. Malensek. 2021. *An Introduction to Parallel Programming.* Morgan Kuafmann.
[4] J. Palach. 2014. *Parallel Programming with Python.* Packt Publishing.
[5] M.J. Quinn. 2004. *Parallel Programming in C with MPI and OpenMP.* McGraw-Hill Higher Education.