Analysis OpenMP performance of AMD and Intel architecture for breaking waves simulation using MPS

M N A Alamsyah\textsuperscript{1,*}, A Utomo\textsuperscript{2} and P H Gunawan\textsuperscript{3}

School of Computing, Telkom University, Jalan Telekomunikasi No. 1 Terusan Buah Batu, Bandung 40257, Indonesia.

E-mail: \textsuperscript{1}alamsyah@student.telkomuniversity.ac.id; \textsuperscript{2}ardhitoutm@student.telkomuniversity.ac.id; \textsuperscript{3}phgunawan@telkomuniversity.ac.id.

*Corresponding author

Abstract. Simulation of breaking waves by using Navier-Stokes equation via moving particle semi-implicit method (MPS) over close domain is given. The results show the parallel computing on multicore architecture using OpenMP platform can reduce the computational time almost half of the serial time. Here, the comparison using two computer architectures (AMD and Intel) are performed. The results using Intel architecture is shown better than AMD architecture in CPU time. However, in efficiency, the computer with AMD architecture gives slightly higher than the Intel. For the simulation by 1512 number of particles, the CPU time using Intel and AMD are 12662.47 and 28282.30 respectively. Moreover, the efficiency using similar number of particles, AMD obtains 50.09 \% and Intel up to 49.42 \%.

1. Introduction

Making a mathematical model for solving a real life problem is needed, since it will be unpractical and time consuming to be solved by direct experiment. For instance in wave problems, obtaining solutions with a simulation is more viable than doing an experiment. This is due to experiments are depended by many unnecessary factors, such that it is hard to focus on the wave’s movement only.

In every atmospheric models, Navier-Stokes equation is used because it is a non-linear differential equation which can describe the flowing of fluid substances such as a gas or water. One of the example in a branch of atmospheric model is the wave problem. This problem can be approximated by using either grid or particle methods. Generally, both of those methods have their own characteristics. In the grid method, a model is constructed based on the value of macroscopic variables of its domain. An example of grid method is known as a robust method which is called a Finite Volume Method (FVM), see in [1] for more detail. Unlike grid method, the particle method provides the solutions in a form of particles, which gives more detail and comprehensive result. This is because of the deformation of the topology can be analyzed on each of particles. Several particle methods are available in some references, for instance, a Finite Pointset Method (FPM)[2], Smoothed Particle Hydrodynamics (SPH)[3], and Moving Particle Semi-Implicit (MPS) [4].
Based on the given solution, particle method is separated into two types, probabilistic and deterministic model. The probabilistic model works by seeing each particle statistics in a big picture, which means the accurate average value is gained from seeing the overall particles alteration. Whereas, deterministic particle model calculation depends on each particle in every loop, which decreases the computation load and memory usage.

Particle method is a numerical method of Lagrangian model, which is commonly used to approximate atmospheric models [5]. Here, Lagrangian model works by determining the movement of particles by calculating wind effect, buoyancy force, and turbulence effect [6]. This model indeed in contrast with the Eulerian model, which works with a grid system that monitors atmospheric properties such as temperature, pressure, and chemical concentration. Moreover, Eulerian model may ensued errors in spatial discretization that can be avoided by using the Lagrangian method [6].

The calculation time of particles movement takes a long time simulation, since the number of iteration is immense and also the use of a large amount of particles. Therefore in this paper, the parallel computing will be implemented to accelerate the computational time. Here, multicore architecture which is OpenMP platform will be used to parallelize code in the implementation of MPS method.

2. Moving Particle Semi-Implicit Method

In this section, the MPS method will be elaborated through the governing equation, particle dynamics and kernel function.

2.1. Governing equation

Navier-Stokes equations are the main foundation in every atmospheric model. These equations, which are used in this paper, express the conservation of two forms: mass and momentum [7]. These equations are written as,

\[ \partial_t \rho + \nabla \cdot (\rho u) = 0, \quad (1) \]
\[ \partial_t u + u(\nabla \cdot u) = -\frac{1}{\rho} \nabla p + F + \mu \nabla^2 u. \quad (2) \]

where \( \rho \) is the density function, and \( u \) is the velocity function, \( p \) is the pressure function, \( F \) is the force, and \( \mu \) is the viscosity of the fluid. Equations (1) and (2) are called a mass and momentum equation respectively. Here, the solution of Equations (1) and (2) will be approached using Lagrangian model. In the next section, the particle dynamic based on Equations (1) and (2) will be given.

2.2. Particle dynamics

The fluid used in this simulation is an incompressible fluid, which means the density of the fluid is unlikely to change over time, in other word \( \partial_t \rho = 0 \). Then it leaves Equation (1) becomes \( \nabla \cdot (\rho u) = 0 \). Hence, Equation (2) can be rewritten as

\[ \frac{Du}{Dt} = -\frac{1}{\rho} \nabla p + F. \quad (3) \]

According to Equation (3), there are two sources of particles acceleration, vector gradient of pressure and external force. In MPS method, the solving process of Equation (3) is separated into two steps. The steps can be seen as the predictor-corrector steps.
Particles velocity caused by an external force is calculated first explicitly with Equation (4) which is given as

$$u_i^* = u_i^n + \Delta t F,$$

(4)

where \(u_i^*\) is the temporal velocity of particle-\(i\), \(u_i^n\) is the velocity of particle-\(i\) at time step \(n\), and \(\Delta t\) is the time step interval [8]. Then, the vector gradient of pressure is taken into account after the pressure is solved implicitly as described in Equation (5):

$$u_i^{n+1} = u_i^* - \frac{1}{\rho} \nabla p,$$

(5)

where \(u_i^{n+1}\) is the velocity of particle-\(i\) at time step \(n + 1\).

In particle term, the vector gradient of pressure in Equations (2),(3), and (5) is a vector gradient between particle-\(i\) and particle-\(j\) of scalar function \(p\) and is defined as

$$\nabla p_i = \frac{p_j - p_i}{|r_j - r_i|^2} (r_j - r_i).$$

(6)

However, the interaction between particles can occur with any combination of two particles, since a particle has several neighboring particles. Therefore, the accumulated vector gradient of pressure in particle-\(i\) is formulated as

$$\nabla p_i = \frac{d}{n_0} \sum_{j \neq i} \frac{p_j - p_i}{|r_j - r_i|^2} (r_j - r_i) \cdot w(|r_j - r_i|),$$

(7)

where \(d\) is the number of spatial dimensions, \(n_0\) is the particle number density, and \(w\) is the kernel function which weights each neighboring particle influence and depends on the distance between particle-\(i\) and particle-\(j\) [4]. This kernel function can be found in Subsection 2.3 for more detail.

2.3. Kernel function

Kernel function is a function controlled by a condition either a particle is ranged by the effect of its surrounding or not. The condition is mainly affected by an area bounded by \(r_e \in \mathbb{R}^+\) around a particle. If some particles are inside an area of \(r_e\) of a particle, the existence of those particles will affect that particle. Thus, the formulation of kernel function is given as

$$w(r) = \begin{cases} 
\frac{r_e}{r} - 1, & 0 \leq r < r_e \\
0, & r \geq r_e 
\end{cases}$$

(8)

From Equation (8), it is explained that a particle can not be placed at the same spot as another particle, because the gap of particle-\(i\) and -\(j\), \(r = |r_j - r_i|\), of both particles is going to be 0 and the weighting function to be infinity.

3. Parallel Algorithm and Implementation

3.1. OpenMP and shared memory parallelism

Parallel computing is a technique in the computational field which some tasks in a program are processed concurrently by many processors such that the execution time does not explode despite the huge computation load. In general, there are three models of parallelism commonly used based on the architecture of CPUs and memories:

(i) Shared memory model,

(ii) Distributed memory model,
(iii) Hybrid model.

Each of models has some Application Program Interfaces (APIs) which can be implemented and are still developed. The shared memory model has POSIX Threads and OpenMP as its API. Meanwhile, there is Message Passing Interface (MPI) as a standard API for distributed memory model. As for hybrid model, it uses a combination of an API of each model. For more details see [9]. In this paper, the shared memory parallelism with OpenMP as the API will be used since its simplicity [10, 11, 12, 13, 14, 15].

Shared memory parallelism is implemented in this paper because of its memory addressing and accessing concept is easy and user-friendly from the programmer’s perspective. Moreover, data sharing between processes is fast and uniform due to the proximity of memory to CPUs [9]. In the other side, OpenMP is used as the API because of the facts that it can be implemented on many different platforms (multi-platform). Moreover, it is compiler directive based, thus it easier to develop a parallel code from serial directly with a full control over parallelization to programmers [16].

3.2. Algorithm

The algorithm of MPS in this paper is classified into nine main processes. Six of them can be parallelized entirely, one of them partially, and the rest cannot be parallelized at all. The general overview of processes and its order is shown at Figure 1. The description and the explanation of each process are described as follows,

- **Initial Position**: generating particles by initializing their coordinates, velocity, and pressure value.
- **Particle Density**: calculating particle concentration of each particle with the formula in Equation (9):
  \[
  n_i = \sum_{j \neq i} w(|r_j - r_i|),
  \]
  where \( n_i \) is the number density of particle-\( i \) and \( w \) is the kernel function which is described in Equation (8) [4, 8]. This process is done at the beginning of the simulation and every time iteration.
- **Determine Boundary**: determining particles which become the free surface of the fluid with condition \( n_i < n_0 \beta \), where \( n_0 \) is the initial number density and \( \beta \) is the boundary parameter in range \((0, 1)\) since particles at free surface have less neighboring particles[4][8]. This process also is done at the beginning of the simulation and every time iteration.
- **Set dt**: calculating the optimal time step interval of every iteration, so that the stability of the numerical simulation is conserved, yet the number of total iteration needed does not blow up. The interval is bounded by particles velocity and fluid viscosity.
- **Explicit Step**: calculating the velocity of each particle caused by source terms and updates its position.
- **Collision Step**: calculating the momentum of colliding particles then updates their velocity and position.
- **Poisson Solver**: generating the linear equations system of pressure of particles.
- **LU Decomposition**: is a sub-process of Poisson Solver to solve the equations.
- **Implicit Step**: calculating particles velocity caused by pressure gradient then updates their velocity and position.
- **Print Particle Data**: writing particles position and pressure into text files as the data output.
Figure 1. Flowchart of MPS implementation in serial and parallel part.

Note that each processor executes the parallel part in Figure 1 with the number of particles is defined in maximum \[ \lceil \text{Total particles}/p \rceil \] where \( p \) denotes the number of processors. Moreover, each processors in parallel part is executed in same time.

4. Numerical Results and Parallel Performances

4.1. Breaking waves problem

Wave Breaking is a phenomenon of fluid dynamics commonly seen at ocean and coastal area. This problem is simulated often due to its great impact to abrasion, man-made structures, and even the ocean current itself [17]. Here, the problem is about to see the profile of water surface at a breaking wave. The program simulates a wave tank in which breaking waves are happened. The configuration of the wave tank initial condition is shown in Figure 2. In order to generate waves, a wave paddle is given in the left boundary, which oscillates in \( x \) direction with its velocity depends on time \( f(t) = 0.2\pi \sin(4\pi t) \). The wave tank also has a topography with slope \( m = \frac{1}{2} \) in the right side. Here, the simulation is observed for 5 seconds and the wave paddle has oscillated 10 times since the frequency is 2 Hz based from \( f(t) \).

Several simulation for different numbers of particles (489, 646, 833, 1037, 1263, and 1512...
particles) are given in this paper. Those numbers of particles are depend on the particle resolution. The particle resolution of 489 particles simulation is 50, which means there are 50 particles in a line along a unit length. Then, particle resolutions for particle numbers 646, 833, 1037, 1263, and 1512 are 60, 70, 80, 90, 100 respectively. Results of some simulations are shown in Figure 3.

Figure 2. The wave tank setup used for breaking wave simulation.

Figure 3. Breaking wave simulations using MPS method at $t = 5$ seconds with different numbers of particles: a) 489 particles, b) 1037 particles, and c) 1512 particles.
From Figure 3, the resolution of the results is depend on the number of particles. Using number of particles 1512 produces higher resolution than using 489 particles. However, at the same time, the water profile is shown similar for each simulation. In other side, using a large amount of particles, the computational time will increasing even to get the small final time of simulation. The analysis of parallel performance to handle the simulation time will be explained in the next subsection.

4.2. Computation details

In this experiment, the simulations are run on two computers with different CPU architecture, Intel and AMD. Details of the specification is shown at Table 1. Although both of them have different number of processors, these simulations only use 4 processors for the sake of relevance of this comparison analysis.

| Model Name         | Intel(R) Core(TM) i5-2500 | AMD Opteron 6376 |
|--------------------|---------------------------|------------------|
| Number of processor(s) | 4                        | 16               |
| Clock frequency (MHz)    | 1600                     | 2600             |
| RAM size (GB)           | 8                        | 8                |
| Cache size (KB)         | 6144                     | 512              |
| Operating System       | Ubuntu 12.04.3 LTS       | Ubuntu 14.04.3 LTS |

In parallel performance analysis, two benchmark tools are used which are speedup and efficiency. Speedup shows how much the parallel implementation affects the execution time. Speedup number is obtained by dividing the serial execution time by the parallel execution time. Meanwhile, efficiency shows how well the parallel implementation utilizes the resource. Efficiency number is obtained by dividing the speedup number by the total number of processors used, in this case is 4 processors. Details of speedup and efficiency of these simulations is shown in Table 2.

It can be noticed that according to the execution time in serial and parallel, the execution time of Intel processor is far better than the AMD one. The Intel processor only needs half time the AMD processor needed to run the simulation. However, from Table 2, the speedup and efficiency of AMD processor has a slightly higher than the Intel processor in general.

Note that the execution time using Intel is observed higher than AMD, since from Table 1, cache size of Intel is given larger than AMD. The size of cache in Intel is proportional to the size of the total particles in simulation, thus high speed of memory interaction is obtained.

5. Conclusion

The parallel implementation of MPS for breaking wave simulation has been elaborated. In the term of algorithm, the efficiency of parallel implementation of MPS stays 50.66% in average, since not all processes can be parallelized, even some processes that can be entirely parallelized still have a large chunk of serial code in each processor. Moreover, MPS method has a linear equations system which needs to be solved and the solver algorithm used in this paper is LU Decomposition which has an immense computation load. Then, in the term of CPU architecture, AMD gives slightly better result in efficiency of parallelization with 50.09% compared with Intel which has 49.42% for 1512 particles. However, Intel gives far better result in CPU time of parallel code with 12662.47 seconds of execution time, less than half of the execution time needed by AMD with 28282.30 seconds.
Table 2. Details of speedup and efficiency of simulations for each number of particles.

| Processor | Number of Particles | Execution Time (s) | Speedup | Efficiency (%) |
|-----------|---------------------|--------------------|---------|----------------|
|           |                     | Serial             | Parallel|                |
| Intel     | 489                 | 209.62             | 102.04  | 2.05           | 51.36          |
|           | 646                 | 616.30             | 310.88  | 1.98           | 49.56          |
|           | 833                 | 1770.36            | 880.20  | 2.01           | 50.28          |
|           | 1037                | 4545.67            | 2321.00 | 1.95           | 48.96          |
|           | 1263                | 11037.56           | 5765.20 | 1.91           | 47.86          |
|           | 1512                | 25031.81           | 12662.47| 1.98           | 49.42          |
| AMD       | 489                 | 577.90             | 257.47  | 2.24           | 56.11          |
|           | 646                 | 1726.40            | 788.23  | 2.19           | 54.76          |
|           | 833                 | 4794.59            | 2314.30 | 2.07           | 51.79          |
|           | 1037                | 11201.86           | 5839.13 | 1.92           | 47.96          |
|           | 1263                | 25392.28           | 12763.27| 1.99           | 49.74          |
|           | 1512                | 56671.11           | 28282.30| 2.00           | 50.09          |

References
[1] Eymard R, Gallouët T and Herbin R 2000 Handbook of numerical analysis 7 713–1018
[2] Onate E, Idelsohn S, Zienkiewicz O and Taylor R 1996 International journal for numerical methods in engineering 39 3839–3866
[3] Liu G R and Liu M B 2003 Smoothed particle hydrodynamics: a meshfree particle method (World Scientific)
[4] Koshizuka S and Oka Y 1996 Nuclear science and engineering 123 421–434
[5] GOTOH H and SAKAI T 1999 Coastal Engineering Journal 41 303–326
[6] Lagzi I, Meszaros R, Gelybo G and Leeosy A 2013 Atmospheric Chemistry (hgpu.org)
[7] Temam R 1984 Navier-stokes equations vol 2 (North-Holland Amsterdam)
[8] Koshizuka S, Nobe A and Oka Y 1998 International Journal for Numerical Methods in Fluids 26 751–769
[9] Barney B 2012 Available on https://computing.llnl.gov/tutorials/parallel comp
[10] Gunawan P H 2016 Information and Communication Technology (ICoICT), 2016 4th International Conference on (IEEE) pp 1–5
[11] Mulyani, Putri N D and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th International Conference on (IEEE) pp 1–5
[12] Pahevi M R and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th International Conference on (IEEE) pp 1–4
[13] Juliati S and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th International Conference on (IEEE) pp 1–5
[14] Alameyah M N A, Simanjuntak C A, Bagustara B A R H, Pradana W A and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th International Conference on (IEEE) pp 1–6
[15] Iryanto and Gunawan P H 2017 Information and Communication Technology (ICoICT), 2017 5th International Conference on (IEEE) pp 1–5
[16] Barney B 2008 Website: https://computing.llnl.gov/tutorials/openMP
[17] Cokelet E 1977 Nature 267 769–774