Penerbit Akademia Baru

CFD Letters

Journal homepage: www.akademiabaru.com/cfdl.html
ISSN: 2180-1363

On the Parallelization of Moving Particle Level Set (MPLS) Method for Multiphase Flow Simulation using OpenMP

Open Access

Abdalla Mohamed Elfatih Ibrahim[1], Khai Ching Ng[2,*], Yee Luon Ng[1]

[1] Center of Fluid Dynamics, Universiti Tenaga Nasional (UNITEN), Jalan IKRAM-UNITEN, 43000 Kajang, Selangor, Malaysia
[2] School of Engineering, Taylor's University, Taylor's Lakeside Campus, No. 1, Jalan Taylor's, 47500 Subang Jaya, Selangor Darul Ehsan, Malaysia

ARTICLE INFO

Article history:
Received 24 September 2018
Received in revised form 17 October 2018
Accepted 2 November 2018
Available online 11 January 2019

ABSTRACT

As the field of Computational Fluid Dynamics (CFD) continues to grow, some advanced simulation methods such as mesh-less (or particle) methods have been devised to solve fluid flow problems involving complex dynamics. The popular Moving Particle Semi-implicit (MPS) method is one of the particle methods commonly used to solve fluid flow problems without relying on the pre-existing mesh structure. Recently, based on the MPS method, two new methods, i.e. the Moving Particle Pressure Mesh (MPPM) and the Moving Particle Level-Set (MPLS) methods have been developed in our research team to simulate single- and multi-phase flow problems and meanwhile to obtain a smoother pressure field. In the current work, the Open Multi Processing (OpenMP) library was used to parallelize the MPLS code on a shared memory machine and the effects of parallelism on the computation performances of MPLS were studied. The test case used to benchmark the computation performance was the multi-phase problem, i.e. the problem involving Rayleigh-Taylor Instability (RTI). The machine used in this work has 6 physical cores with 12 logical threads. The maximum speedup was 3.71x, which was comparable to those achieved by similar particle methods such as Direct Simulation Monte Carlo, Discrete Element Method, etc. Strong and weak scaling studies were conducted and the memory access (cache hit rates) under different scheduling patterns was investigated. It was found that the speedup and core performance of the level-set function implemented in the MPLS code was relatively high.

Keywords:
CFD, MPS, MPPM, particle method, OpenMP, Shared-memory machine

1. Introduction

Applications involving multiphase flow are abundant, e.g. refrigeration, flow boiling, haze modelling, etc. As multiphase flow applications are being explored, numerical method such as the Moving Particle Level-Set (MPLS) method has been developed to precisely simulate those applications [1]. Similarly, in order to simulate large-scale problems, the need for increased computational power becomes evident. This can be achieved through various parallelization techniques such as those employing Message Passing Interface (MPI), Open Multi-Processing (OpenMP) or Graphics Processing Unit (GPU) parallelization. Generally, the numerical methods

* *Corresponding author.*
E-mail address: ngkhaiching2000@yahoo.com (Khai Ching Ng)

designed for fluid flow simulation can be classified as Eulerian, Lagrangian or a hybrid of both. There have been many works performed to parallelize these methods on shared-memory, distributed-memory and hybrid machines using various types of parallelization techniques. Eulerian methods are grid-based (grids can be adaptive or rigid), which are regarded as the classical methods in computational fluid dynamics. Grid-based methods have been proven to be adequate for simulating single-phase flow. Nevertheless, when these methods are adopted for simulating multi-phase problems, there are difficulties in detecting the rapid changes of fluid interface [2]. The level-set method [3] was used to capture the interface of different fluids. This method was then combined with the Volume of Fluid (VOF) method in order to improve mass conservation [4]. Nonetheless, Eulerian methods are less effective when they are used to compute multiphase flow with large deformations. The Eulerian ROCFIRE code was parallelized on a 64-core shared-memory machine using OpenMP and they have reported a speedup of 47x [5]. The domino approach was used to parallelize the Moving Computational Domain method (MCD) using OpenMP [6]. The authors reported a speedup of 1.34x and 4.81x out of 6-core and 20-core shared-memory machines, respectively. Their work, along with that of [7], have shed some lights on the importance of data locality and cache size effects on the computational performance. Asao *et al.* [6] reported that the problem of low speedup was most likely caused by the huge communication cost between the processor cores and the data storage locations. Usually, the matrix solver for the Pressure Poisson Equation (PPE) consumes most of the computational effort (hence time). The Bi-Conjugate Gradient Stabilized (BiCGSTAB) is a pressure solver used in countless methods including the current work. The solver was reported to be responsible for up to 70% of computation time [8, 9]. OpenMP was also implemented to accelerate the dam-break flooding simulation on a 16 core machine and the authors have reported speedup ranging from 7.62x to 8.64x on various grid densities [10].

Recently, in order to address the limitation of grid-based methods in simulating flows involving complex dynamics (such as those involving multiple phases), Lagrangian methods (particle methods) such as the Smoothed Particle Hydrodynamics (SPH) method [11] and the Moving Particle Semi-implicit (MPS) method [12] have been developed. The parallelization of SPH has been reported in several works. For example, Goozee and Jacobs [13] reported a speedup of 3.11x on a shared-memory machine with 8 cores while Wróblewski and Boryczko [14] reported a relative efficiency of 42% and 79% for OpenMP and MPI respectively on the same simulation domain. They also found that the relative efficiency increased as the number of neighboring particles increased. Other fairly successful parallelization attempts on hyper-threaded shared-memory machines have been carried out [15, 16]. However, both groups have reported that the speedup rates decreased when the number of threads exceeded the number of physical cores. The MPS method was firstly parallelized by Sueyoshi and Naito [17]. However, the obtained speedup was low, which might be due to the fact that the solver was not ready for parallelization. Then, Iribe *et al*. [18] improved the speedup of their OpenMP MPS code from 1.90x to 3.37x on a 4-core shared memory machine. This was done by simply changing the indexing method of particle registration to become proximity-dependant. When the same code was executed on a distributed-memory machine, the reported speedup of the matrix solver for PPE was very low (7.19x from 64 cores). The GPU parallelization of MPS have been carried out by Hori *et al*. [19] and Ovaysi and Piri [20]. Other numerical methods such as Discrete Element Method (DEM), Lumped Particle Modelling Framework and Local Radial Basis Function Collocation Method have been parallelized on shared-memory machines respectively by [21-23].

As compared to grid-based methods, the extension of Lagrangian methods to simulating fluid structure interaction problem is relatively straightforward [24]. Nonetheless, upon reviewing the existing particle methods, Hwang [25] have argued that those methods are inaccurate when computing incompressible flow due to the inaccurate pressure gradient calculation and the non-zero

velocity divergence. Thus, the Moving Particle Pressure Mesh (MPPM) method, which can be regarded as an Eulerian-Lagrangian (E-L) method, was developed to address these issues by storing the pressure variable (appeared in the PPE) in the Eulerian mesh (grid). Despite the significant improvement that MPPM offers over MPS (see [26, 27]), the computation of multi-phase flow involving fluids of high density ratio using MPPM was unstable. Thus, the MPLS method [1] was proposed to address this issue. Ng *et al*. [1] coupled the MPPM solver with the conservative Level-Set (LS) method where the LS function was used to determine the smoothed density of moving particles while the divergence-free velocity field was determined through the MPPM solver. There have been many works detailing on the parallelization of Eulerian-Lagrangian (E-L) method. For example, the Direct Simulation Monte Carlo (DSMC) method (coupled with grid-based technique for particle-indexing) was parallelized by Gao and Schwartzentruber [7] using OpenMP. They have reported near-linear speedups for free-stream flow simulations with low particle count (3.74x with 4 cores). However, very low speedup was obtained as more particles were employed on an 8-thread shared-memory machine. On distributed-memory machines, Darmana *et al*. [28] reported a 20x speedup with 32 cores while Kafui *et al*. [29] reported a speedup of 37x using 64 cores with the particle-based portion exhibited a speedup of only 6x as opposed to 44x for the grid-based portions of the same code. Similarly, Yakubov *et al*. [30] applied hybrid parallelism on an E-L method and observed similar speedup behaviors for the grid- and particle-based portions of their code. Their overall reported speedup was 8.8x using 128 cores with the Lagrangian part achieving a speedup of 2.3x at best.

In this work, the new MPLS method was parallelized using OpenMP and optimized for a shared-memory machine using strong and weak scaling analyses. The optimum scheduling method for distributed-memory machines is determined through cache analysis.

2. Numerical Methods

2.1 Moving Particle Pressure Mesh (MPPM) Method

The continuity equation of incompressible fluid flow can be written as

$$\nabla \cdot \vec{u} = 0, \tag{1}$$

Eq. 1 is solved together with the momentum conservation equations in x- and y-directions

$$\begin{aligned} \rho \frac{Du}{Dt} &= -\frac{\partial P}{\partial x} + \mu \nabla^2(u) + S_x, \\ \rho \frac{Dv}{Dt} &= -\frac{\partial P}{\partial y} + \mu \nabla^2(v) + S_y, \end{aligned} \tag{2}$$

where P is fluid pressure, $\vec{u} = \langle u, v \rangle$ is the velocity vector, S_α is the source term in the α direction, ρ is the fluid density and μ is the dynamic viscosity. Both μ and ρ are space-dependent in a multi-phase flow problem.

The fractional time step method was used to solve Eq. 1 and 2, following the time-splitting technique developed by Ataie-Ashtiani and Farhadi [31]. The velocity of each particle at different time levels can be computed by integrating Eq. 2 using the 1st order explicit scheme

$$\vec{u}_k^{n+1} = \vec{u}_k^{*} - \frac{\Delta t}{{\rho_k}^n} \nabla P_k^{n+1}, \tag{3}$$

with $\vec{u}_k^{n+1}$ is the particle velocity at the new time level and $\vec{u}_k^*$ is the velocity vector at the intermediate time level. For stability, the time step size was calculated in the following manner

$$\Delta t = min\left(\Delta t_D, C\frac{h}{max(|\vec{u}_p^n|)}\right), \tag{4}$$

where Δt_D is the diffusive time constraint defined as $\Delta t_D \leq \frac{\rho_k^n}{Z_k}$. Here,

$$Z_k = \frac{2d}{\sum_{j\neq k} w'_{kj}} \sum_{j\neq k} \frac{\mu_{kj} w'_{kj}}{|\vec{r}_j^n - \vec{r}_k^n|^2}, \tag{5}$$

where d is the flow dimensionality and w'_{kj} is the weight. The maximum value of the fluid particle speed, $|\vec{u}_p^n|$can be found as the solution progresses. The Courant number C was set to 0.1 [32].

Based on Figure 1, Eq. 1 can be discretised as

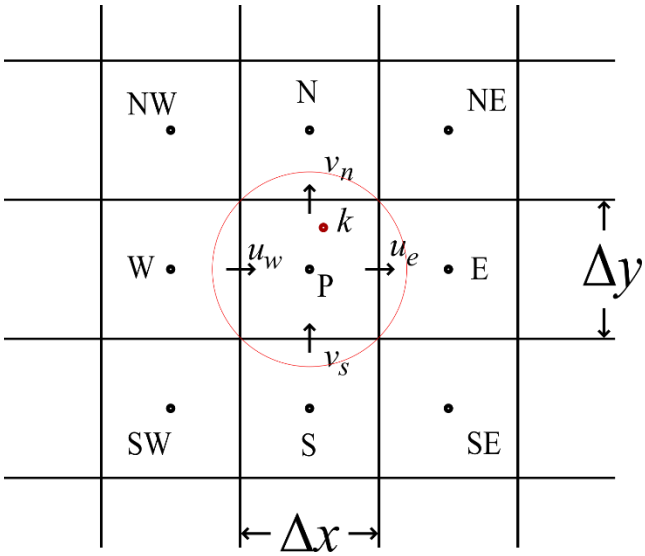

Fig. 1. Discretized pressure field with particle P and neighboring particle k

$$(u_E^{n+1} - u_W^{n+1})\Delta y + (v_N^{n+1} - v_S^{n+1})\Delta x = 0, \tag{6}$$

The splitting velocity concept [33] was implemented to produce the corrected velocity equations which were then substituted into Eq. 6 to yield

$$2\left(\frac{\Delta x}{\Delta y} + \frac{\Delta y}{\Delta x}\right) P_p^{n+1} = \frac{\Delta y}{\Delta x} P_e^{n+1} + \frac{\Delta y}{\Delta x} P_n^{n+1} + \frac{\Delta x}{\Delta y} P_w^{n+1} + \frac{\Delta x}{\Delta y} P_s^{n+1} - \frac{\rho}{\Delta t}[(u_e^* - u_w^*)\Delta y + (v_n^* - v_s^*)\Delta x], \tag{7}$$

Assuming the mesh spacing is fixed $(h = \Delta x = \Delta y)$, Eq. 7 becomes

$$\frac{h}{\Delta t}[u_e^* - u_w^* + v_n^* - v_s^*] = \frac{1}{\rho_e^n}P_E^{n+1} + \frac{1}{\rho_w^n}P_W^{n+1} + \frac{1}{\rho_n^n}P_N^{n+1} + \frac{1}{\rho_s^n}P_S^{n+1} - \left(\frac{1}{\rho_e^n} + \frac{1}{\rho_w^n} + \frac{1}{\rho_n^n} + \frac{1}{\rho_s^n}\right)P_p^{n+1}, \quad (8)$$

Finally, the pressure gradient acting on a particle *k* can be computed using the bilinear interpolation (see Figure 2)

$$\frac{\partial P_k^{n+1}}{\partial x} = \frac{P_{NE}^{n+1} - P_N^{n+1}}{h}\varepsilon_k + \frac{P_E^{n+1} - P_P^{n+1}}{h}(1 - \varepsilon_k), \quad (9)$$

$$\frac{\partial P_k^{n+1}}{\partial y} = \frac{P_{NE}^{n+1} - P_N^{n+1}}{h}\vartheta_k + \frac{P_E^{n+1} - P_P^{n+1}}{h}(1 - \vartheta_k), \quad (10)$$

where $\vartheta_k = (x_k - x_p)/h$ and $\varepsilon_k = (y_k - y_p)/h$. The complete MPPM algorithm is outlined in Figure 3 and more details about the MPPM method can be found in [25]. Nonetheless, the direct application of MPPM in simulating multi-phase flow involving large density ratio was unsuccessful, prompting the development of the MPLS method.

2.2 Moving Particle Level-Set (MPLS) Method

This method was developed to complement MPPM by adding the ability to accurately simulate immiscible multi-phase flow problems [1]. Ng *et al*. [1] have argued that the commonly used density interpolation methods tend to produce wiggly solutions caused by particle intrusion. The same phenomenon was witnessed by Shakibaeinia and Jin [34] even in a flow problem involving fluids of low density ratio. The proposed solution by Ng *et al*. [1] involved the improvements of the density interpolation and the interfacial recognition schemes. This was done by solving the level-set differential equation on the pressure grid in MPPM in order to capture the fluid interface. Upon finding the velocities from the MPPM method, the level-set function Ψ can be solved using

$$\frac{\partial \Psi}{\partial t} = -\nabla \cdot (\vec{u}\Psi) = L(\Psi), \quad (11)$$

Eq. 11 was integrated using a 3rd order TVD-RK scheme

$$\begin{aligned} \Psi^1 &= \Psi^n + \Delta t L(\Psi^n), \\ \Psi^2 &= \Psi^1 + \frac{\Delta t}{4}[-3L(\Psi^n) + L(\Psi^1)], \\ \Psi^{n+1} &= \Psi^2 + \frac{\Delta t}{12}[-L(\Psi^n) + 8L(\Psi^2) - L(\Psi^1)], \end{aligned} \quad (12)$$

By using this approach, the interface thickness may vary and the artificial compression technique should be used to compress the interface via solving the additional PDE

$$\Psi_\tau + \nabla \cdot \left[\Psi(1 - \Psi)\underline{n}\right] = \lambda \nabla(\nabla \Psi), \quad (13)$$

here, the thickness can be controlled via the parameter λ. Lastly, the level-set value of each particle can be found using the bi-linear interpolation

$$\Psi_k = \Psi_N(1-\varepsilon_k)\vartheta_k + \Psi_P(1-\varepsilon_k)(1-\vartheta_k) + \Psi_E\varepsilon_k(1-\vartheta_k) + \Psi_{NE}\varepsilon_k\vartheta_k, \quad (14)$$

where $\varepsilon_k = (x_k - x_p)/h$ and $\vartheta_k = (y_k - y_p)/h$. Particles employed in this work can be added or removed from the flow domain whenever necessary.

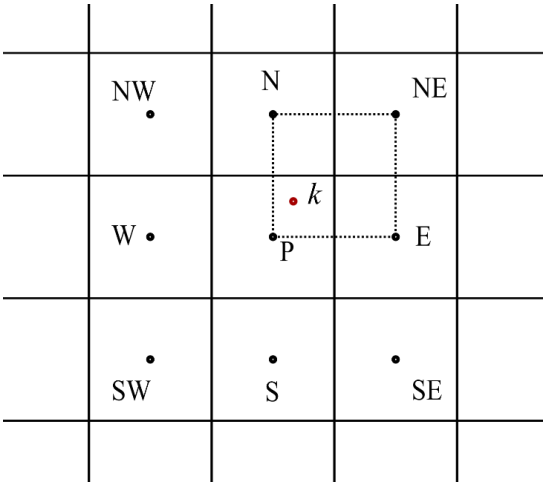

Fig. 2. Evaluating the pressure gradient on particle k using bi-linear interpolation

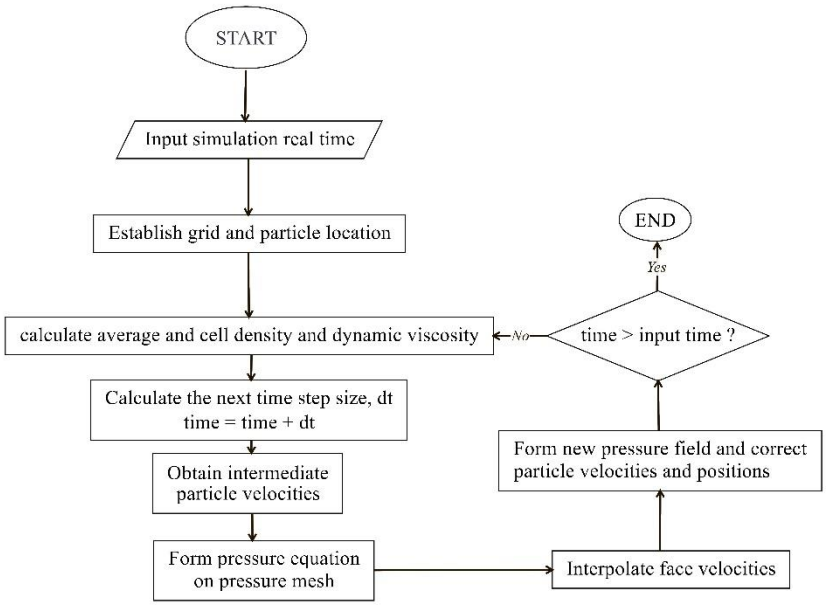

Fig. 3. MPPM process flowchart

3. Rayleigh-Taylor Instability (RTI) Problem

This MPLS code was parallelized using OpenMP in the current work. Originally, the MPLS code was written using FORTRAN77 (F77). However, considering the future plan of GPU parallelization, the code was converted to C++. Strong and weak scaling analyses were carried out. The selected RTI problem was obtained from Ng *et al*. [1]. The RTI problem is one where a heavy fluid is placed atop a lighter fluid [35-37]. The position of the initial fluid interface y_{int} is

$$y_{int} = 1.0 - 0.15(2\pi x), \quad (15)$$

The densities of the light and heavy fluids are 1.0 and 1.8 kg/m^3, respectively. The domain height was set to 2.0m and the domain width was fixed at 1.0m. The simulated period was 5 seconds. The kinematic viscosities of both fluids were similar and the gravitational acceleration was prescribed as 1.0 m/s^2. The Reynolds number was 420. Boundaries were treated as no-slip walls. In what follows, the code used for this simulation was denoted as MPLS-RTI.

4. Parallel Implementation

4.1 Equipment Specification

The MPLS code was parallelized on a shared-memory machine equipped with the Intel® Xeon® E5-2630-v2 processor that has 6 cores (12 threads) with base frequency of 2.60 GHz. The CPU cache is L3 cache size of 15 MB.

4.2 Language Conversion

Eliminating the weak points of F77 is the focus of the conversion. The language uses go-to style statements often lead to unreadable code (Spaghetti code: unnecessarily complex code). Thus, all do loops were converted to for loops in C++ whenever applicable. Additionally, the actual go-to statements were reconstructed and converted to while loops in C++ whenever permissible. Figure 4 presents a simple code segment as an example of the conversion approach. The conversion has effectively eliminated a total of 261 go-to statements in the MPLS-RTI code. Another essential scope of the conversion is variable declaration. Repetitive Common block variables in F77 were declared once as global scope variables in C++, hence yielding a more readable code.

```
while(np != 0)
{
        if(Id[np] == IdFluid)
        {
        presGrad2(np, dpdx, dpdy);
        //rest of statements ... ;
        }
        np= npnext[np];
}
```

```
8650 continue
     if(np.eq.0)  go to 8800
     if(Id(np).ne.IdFluid) go to 8700
     call presGrad2(np,dpdx,dpdy)
     cc rest of statements ...
8700 continue
     np=npnext(np)
        go to 8650
8800 continue
```

Fig. 4. Go-to statements to while loops conversion. Left: C++; RIGHT: F77

4.3 First Touch Parallelization

In an effort to study how susceptible the code is to parallelization, incremental parallelization was the method of choice in this work [5]. OpenMP directives and constructs were introduced into the code whenever possible and their effects were taken into consideration before proceeding. The following steps were taken to initialize the parallelization process

I. All involved variables, especially those that would change in value after each iteration, are identified.
II. Applicable loops are parallelized by introducing *#pragma omp parallel {#pragma omp for (…)}*

III. Variables that are accessed by different threads in parallel regions are defined as private variables to prevent data racing.
IV. The reduction construct is used to address accumulating variables in parallel loops along with the respective operators.
V. When needed, the *critical{…}* construct is used to force serial execution within parallel regions. However, this is performed once in each model.
VI. Lastly, the number of threads involved in each parallel region is controlled using the function: *omp_set_num_threads* (*NUM_THREADS*), where *NUM_THREADS* is previously defined during compilation.

These steps ensure that the parallel output is identical to that of the serial one. However, functions where particle ID reassigning takes place are not applicable for parallelization due to the fact that data racing bound to happen if they are parallelized. Upon applying the aforementioned steps, computation time was recorded and it was found that after the number of threads used in a simulation exceeded a certain number, the computation time would grow thereafter. Hence, more fine tuning efforts are required.

4.4 Parallelization Tuning

Upon recording the computation times for all functions, it was noticed that certain functions seem to perform worse as the number of threads exceeded a certain number. This was named a parallel threshold. Each code has some functions with different thresholds. Thus, the number of threads involved in those regions was fixed to the optimum amount. Work sharing clauses were found to have the most impact on improving the parallel performance. Wei *et al*. [22] reported that OpenMP works best when utilizing the dynamic clause and Hoeflinger *et al*. [5] reported that the default static clause improved data-locality. However, it was found that the most suitable work-sharing clause is the guided clause. Parallel sections were tested. No improvement was found and hence they were ignored along with locks. Fine tuning test results are reported in the next section.

5. Results and Discussion

5.1 Language Conversion Effects

In order to validate the C++ code, its output must be compared to that originated by the original F77 code. Both languages have a value range of 1e-308 to 1e+308 with 16 decimal digits. Interestingly enough, since each language has its own way of calculating and storing values of variables, the results were slightly different despite the codes were identical. This is believed to be caused by the peculiar way both languages handle decimal numbers beyond the 8^{th} decimal points as shown in Figure 5. Despite the value of Sigma is declared as 0.075, this number appears differently in both languages with the C++ value coming closer to the actual value (-3.0e-18 deviation compared to +2.98e-9 in F77). The effects of these miniscule differences become more apparent in CFD calculations where millions of operations take place and the propagating effects become more visible as the simulation progresses. This is presented in Figure 6 where particle locations generated by each language are plotted against each other. It is noticeable that as time progresses, more particles have different placements when comparing Figure 6(a) and Figure 6(b). The propagating effect becomes more apparent as the grid density increases, which is notable when comparing Figure 6(a) and Figure 6(c).

FORTRAN77

rho1	1.00000000000000	REAL(8)
rho2	2.00000000000000	REAL(8)
sigma	7.500000298023224D-002	REAL(8)

```
      dsten=0.0 ! set to 0.0 if no
      sigma=0.075 !surface tension
cc
```

C++

rho1	1.0000000000000000	double
rho2	2.0000000000000000	double
sigma	0.074999999999999997	double

```
long double     dsten=0.0;  // set
long double     sigma=0.075;
```

Fig. 5. The effect of decimal precision on variable declaration

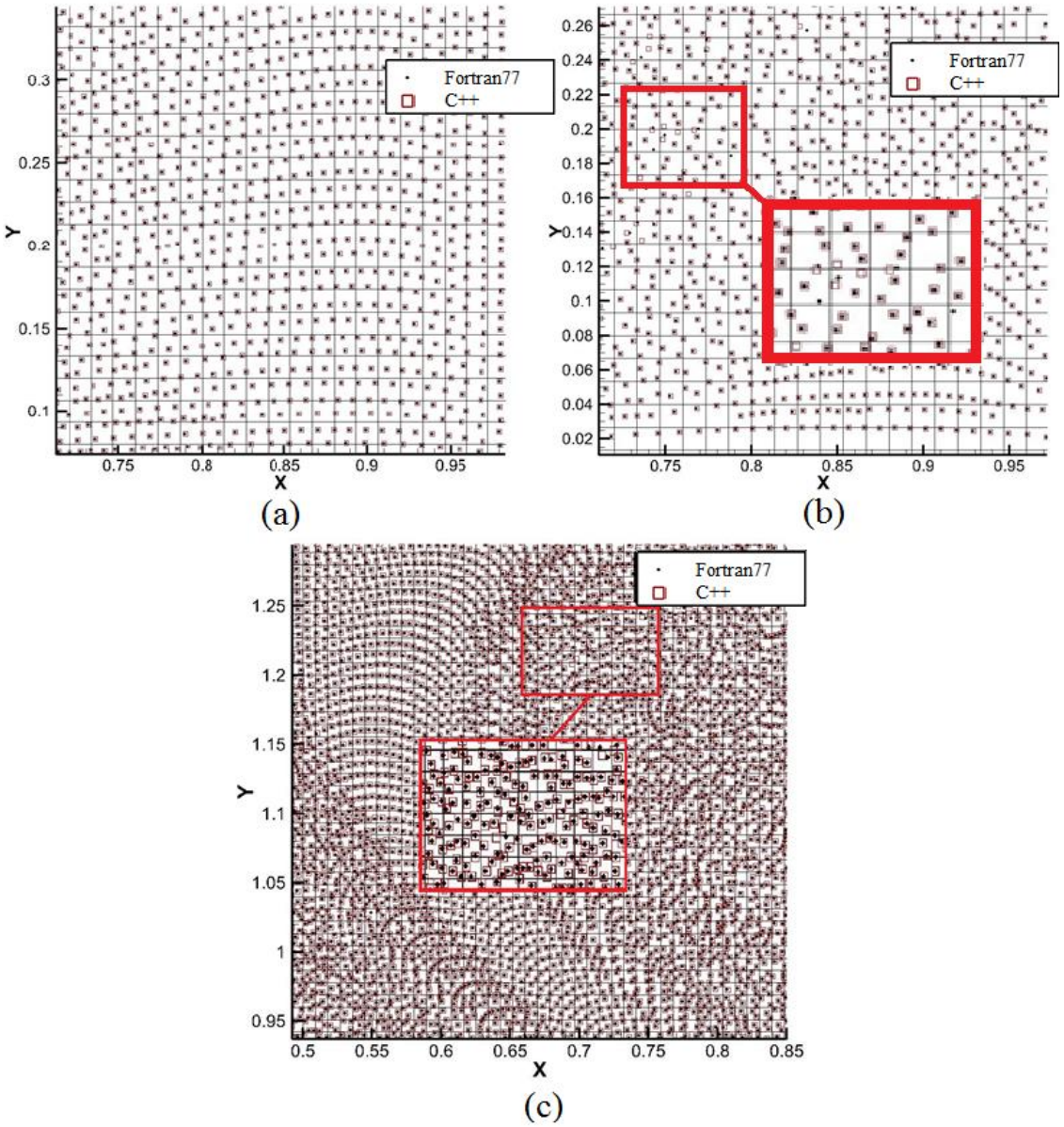

Fig. 6. Comparison of particle locations (a) Grid (75x150) at t= 3s (b) Grid (75x150) at t= 5s (c) Grid (112x225) at t= 3s

5.2 Parallelization Validation

In order to validate the results generated by the parallel C++ code, its output is compared to that generated by the serial C++ code. By observing the particle locations (Figure 7) the parallel output was indeed identical to the serial one. Figure 8 compares the pressure plots which were found to be in complete agreement. Similarly, the plots of level-set, x- and y-velocity (not shown here) were identical as well.

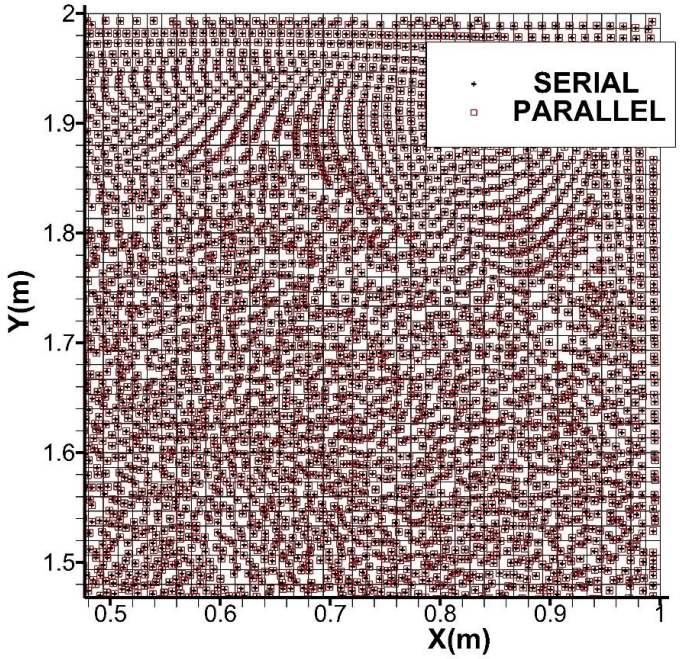

Fig. 7. Comparison of particle locations simulated from the serial and the parallel MPLS-RTI code (75x150 grid, t = 5s)

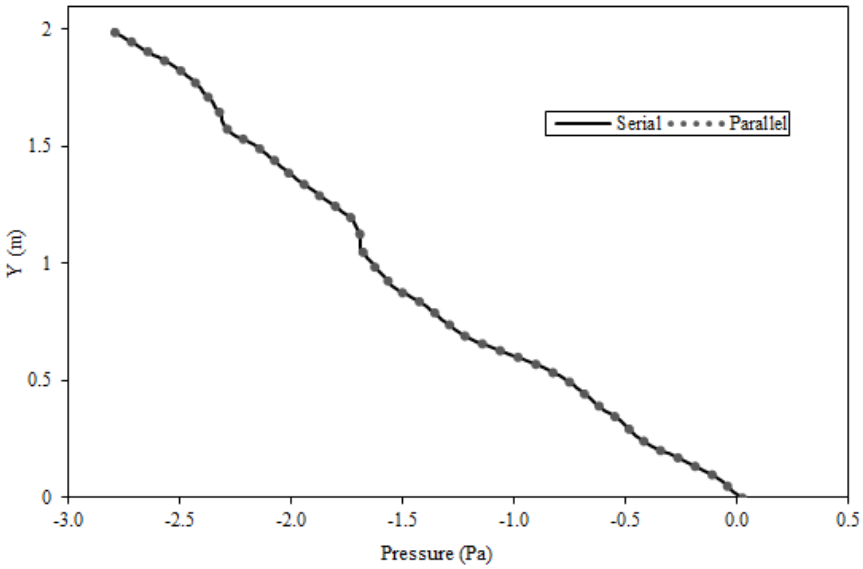

Fig. 8. Comparison of pressures values predicted from serial and parallel MPLS-RTI code at x = 0.5 m (75x150 grid, t = 5s)

5.3 Strong Scaling Analysis

The MPLS code was parallelized on a shared-memory machine equipped with the Intel® Xeon® E5-2630-v2 processor that has 6 cores (12 threads) with base frequency of 2.60 GHz. The CPU cache is L3 cache size of 15 MB.

There are several approaches that can be used to evaluate the parallel performance. Amongst those, parallel speedup is the main concern of strong scaling in this work. Derived from Amdahl's Law [38], the following equation was used to find the speedup S_N

$$S_N = \frac{T_S}{T_N}, \tag{16}$$

where T_S is the computation time when the code is run in serial and T_N is parallel computation time when N threads are used. From Eq. 16, the maximum speedup can be found through

$$S_{Nmax} = \frac{1}{F + (1 - F)/N}, \tag{17}$$

with F being the percentage of code that runs in serial. Conversely, Eq. 16 can determine the parallelizability of a code by using the obtained maximum speed up. For this analysis, a medium sized grid was selected and the number of threads applied throughout the codes was varied from 1 to 12 threads (6 physical cores). Both speedup and computation time were then reported.

The RTI problem was simulated for 5s on a 75x150 pressure grid. The simulation progress can be observed through the level-set contours presented in Figure 9. The computation time of each individual function is reported in Figure 10 in order to identify the most time-consuming function in the MPLS-RTI code. The tasks performed by each function are listed in Table 1. As seen, the time-consuming functions are *calcUVstar* and *calcpprebicgstab* in the serial code (46.5% and 26.3% respectively). The former is concerned with the intermediate velocity calculation while the latter is the pressure solver. Fortunately, since *calcUVstar* has no parallel threshold limitations, it is entirely parallelized and the resulting speedup is high (7.52x). *Calcpprebicgstab*, on the other hand, has a low parallel threshold of 4 threads with this grid density, resulting in a low speedup value (1.63x). The highest speedup value of 8.44x was achieved by the face velocity calculation function *calcUVface* which solves Eq. 6 (Eulerian function). Figure 11 shows good scaling for the RTI problem with a maximum speedup of 3.71x using 6 cores (12 threads) which is comparable, if not better than, other E-L methods (Gao and Schwartzentruber [7] achieved 1.54x speedup using 8 threads; Yakubov *et al.* [30] reported 8.8x speedup using 128 cores). It is important to note that the level-set function *solvelevelset* (Eq. 11) reports a speedup of 6.1x while being computationally expensive at first. The speedup increase rate drops after the number of threads involved exceeds that of physical cores, which is expected as reported in open literature [15, 16]. This is due to the overhead communication cost incurred by the addition of logical cores. Nonetheless, by applying Eq. 17, we find that only 12.3% of the code is executed in serial.

5.4 Weak Scaling

In weak scaling, the main focus is to analyse the single-core performance. This is done by varying the thread count along with the problem size in a manner that ensures consistent load distribution between threads. Normally, the rate of increase of threads/problem-size is to the power of 2.

However, since the machine used in this work has 12 threads, the grid variations were designed based on 2, 6 and 12 threads with consistent cells/thread values. The variation settings are listed in Table 2. In this work the parameter selected was grid density. Since parallel thresholds exist for each function, the reporting of individual function scaling would better reflect the parallelization effect.

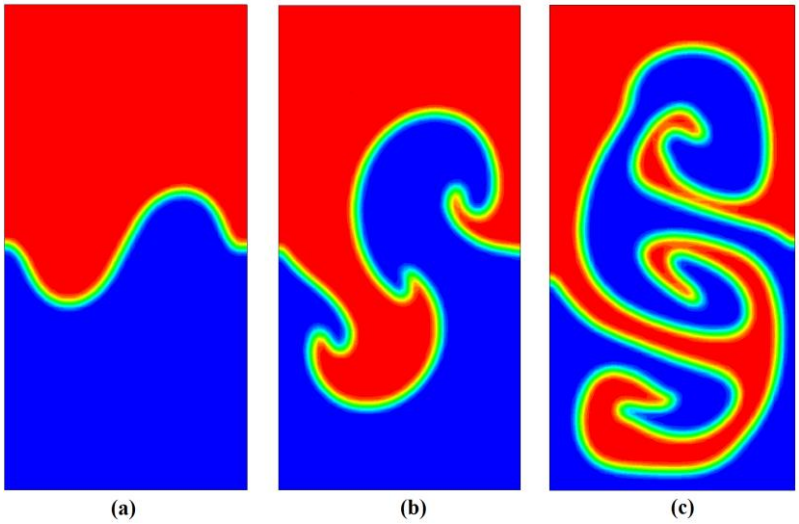

Fig. 9. The level-set contours of RTI problem at (a) t = 1s, (b) t = 3s and (c) t = 5s, (48x96 grid)

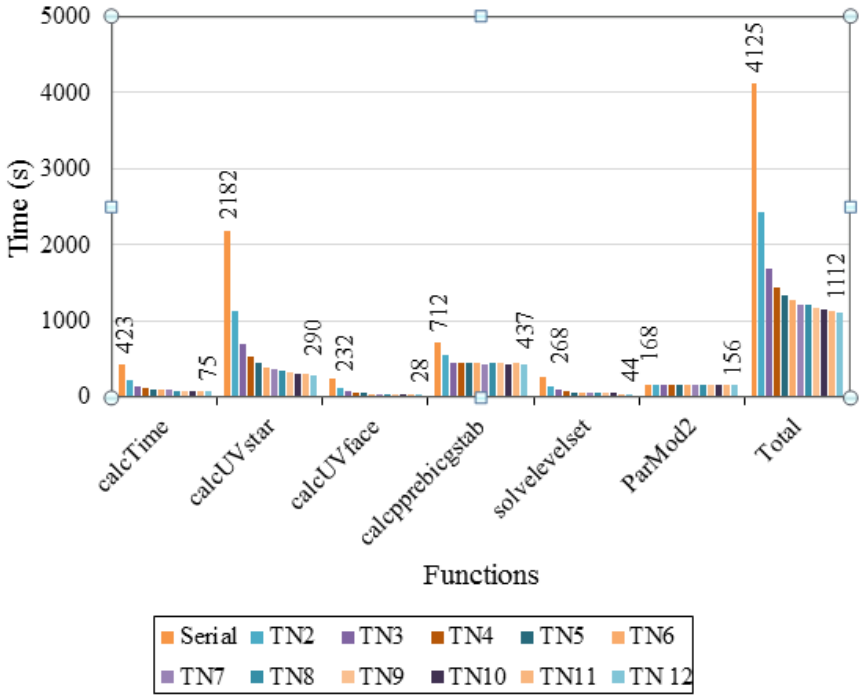

Fig. 10. Computation times for the RTI case function. TN indicates the number of threads used (75x150 grid)

The weak scaling study for RTI presented in Figure 12 shows that with the exception of functions *calcpprebicgstab* and *calcUVstar*, all functions scale well with a slight increase in computation time, which could be attributed to the increase in communication load as the number of particles increases. It is important to note that the pressure solver is limited to 4 threads which in a way invalidate the weak scaling approach for

this function. Interestingly enough, *calcUVstar* experiences an increase in computation time despite of the fact that it scales very well in the strong scaling study. This can be attributed to the exponential increase in the number of particles that takes place as the grid is refined, which in turn causes the communication cost to increase greatly as observed. Keeping in mind that the number of particles also increases as simulation time increases due to particle insertion protocols. Intriguingly, the level-set functions scale exceptionally well, seeing that they are considered as time-consuming functions in serial computation.

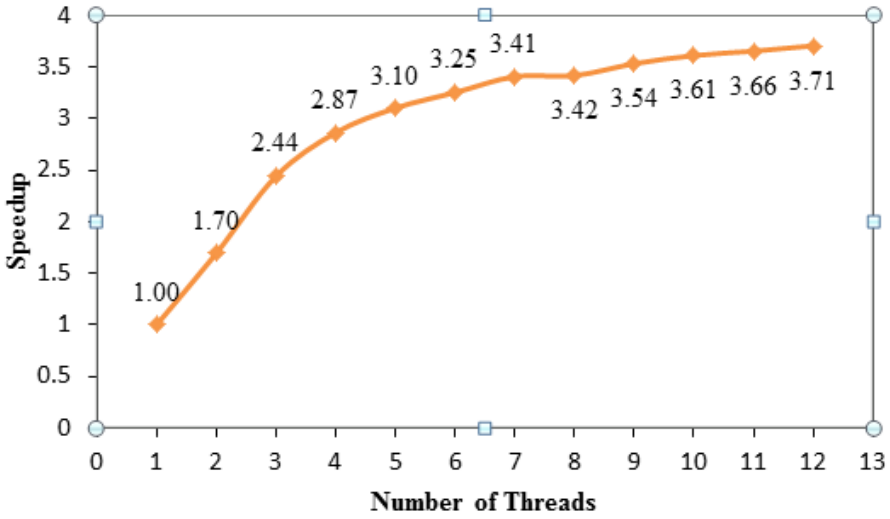

Fig. 11. Strong scaling result of the RTI case (75x150 grid)

Table 1
Tasks performed by code functions

Function	Task
calcrhomiuavg	Calculates the smoothed density and dynamic viscosity
calcrhoavg_cell	Calculates the cell density.
calcTime	Calculates the time step size, dt
calcUVstar	Calculates the intermediate particle velocity
calcXYstar	Calculates the intermediate particle position
ParUpdate	Updates the information of particles at new position
calcUVface	Calculates the intermediate face velocity
coefPresBicgstab	Calculates the coefficients in PPE.
calcpprebicgstab	Solves the PPE using the Preconditioned BiCGSTAB method.
ParMod2	Adds/removes particles.
solvelevelset	Solves level-set equations.
correctlvloolson2	Solves the artificial compression PDE.

Table 2
Weak scaling variation settings

Setting	Threads	Grid Size	Cells/Thread
1	2 threads	43x86	1849
2	6 threads	75x150	1875
3	12 threads	105x210	1837.5

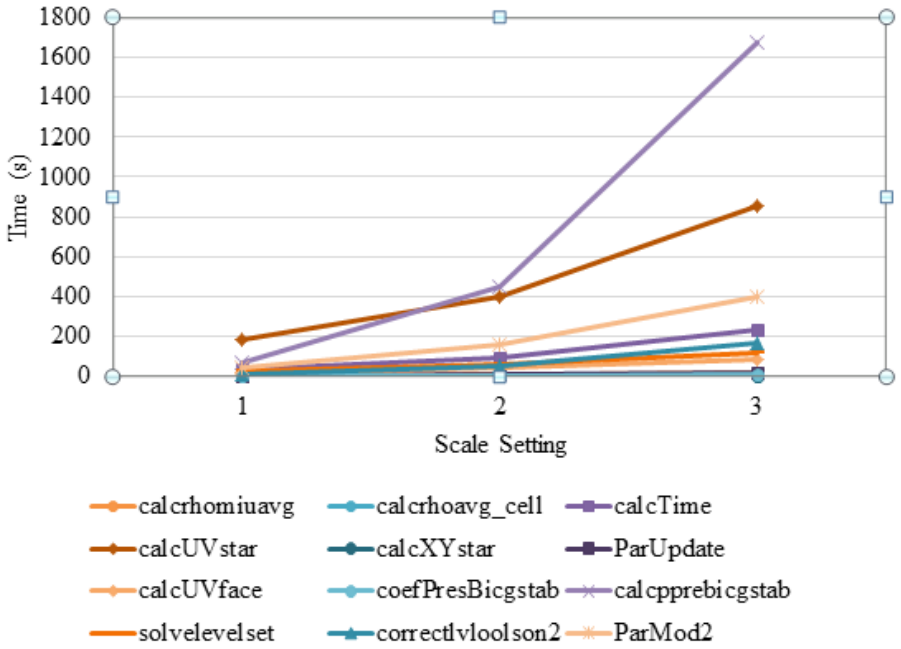

Fig. 12. Weak scaling results for the RTI problem

5.5 Cache Analysis

There are several levels of memory where data is stored, starting from registers which are extremely small and fast (100 bytes size and latency of <1 ns) all the way to disk memory which is much slower in comparison (terabytes in size, 5~20 ms latency). The second level is cache memory. Data can be stored temporarily in any of its three levels (L1, L2 and L3) which conform to the same speed-size hierarchy. L3 cache has been found to govern the performance in Lagrangian methods as reported by Kosec *et al*. [23]. Different approaches have been used to store data in cache with varying effects as reported by Asao *et al*. [6]. In this section, we would like to determine the best OpenMP scheduling method for cache accessibility in the Euler-Lagrangian method such as MPLS. Two scheduling approaches were tested: guided and dynamic scheduling methods which were then compared to the scheduling-free counterpart. Even though the guided scheduling was found to be most suitable for shared-memory machines, this analysis gives an insight on the most suitable method for distributed-memory machines. Hit counters for Last-Level-cache hits were recorded using the Visual Studio profiling tool [39]. Hit rate (HR) for any cache level can be defined as

$$\mathrm{HR} = \frac{\text{number of successful hits}}{\text{number of all calls}} \tag{18}$$

HR was calculated for both inclusive and exclusive performances. Exclusive performance of a function accounts for execution within that function excluding the time spent in functions called from that function. Inclusive performance is the execution performance within that function including the time spent in function called from that function. In other words, inclusive HR provides an overview of the code performance while exclusive HR gives an insight on individual function performance. As such, the grid density was varied and the performance was recorded as seen in Figure 13.

For a coarse grid, it can be seen that the change of scheduling has an unnoticeable effect on HR. As the grid density increases to an intermediate level, it is visible that when functions are individually examined, their HR values are high for both scheduling types (exclusive HR). In general, the scheduling-free code reported the highest HR followed by the dynamic scheduling. Lastly, in the case

of fine grid, all scheduling types performed well for individual functions, while dynamic scheduling proved to be the better scheduling approach in terms of cache HR. This analysis, coupled with the strong scaling analysis, has indicated that guided scheduling is the best option for shared memory device while dynamic scheduling is the choice for distributed memory device meant for the parallelization of Euler-Lagrangian method such as MPLS.

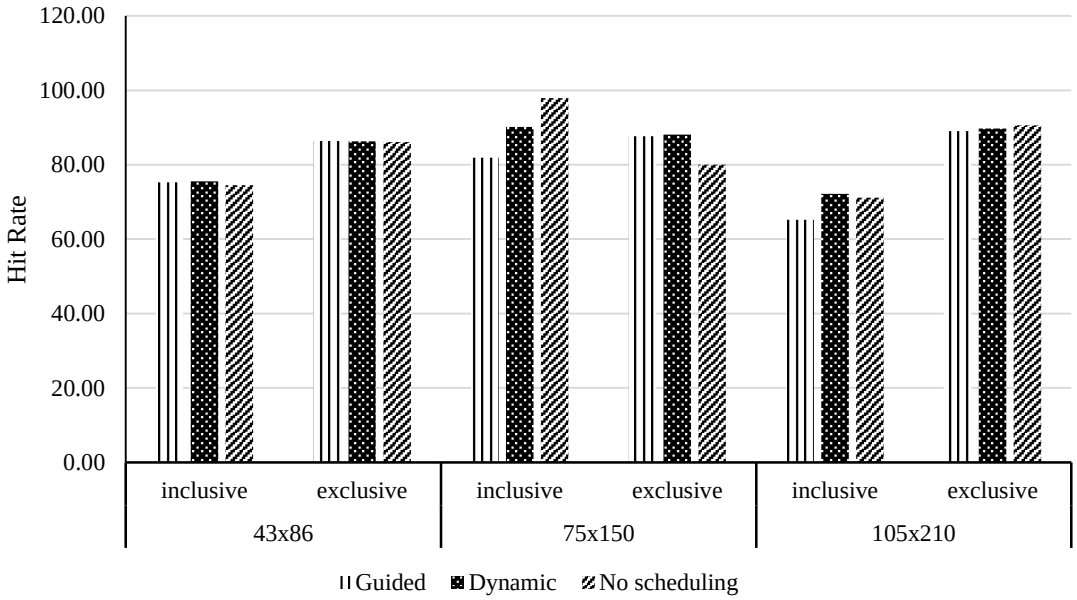

Fig. 13. Last-level-cache hit rate for various grid sizes

6. Conclusions

In the current work, the MPLS method has been converted from Fortran77 to C++ and their effects on the flow results have been presented. The C++ MPLS code has been parallelized using incremental parallelism. Strong and weak scaling studies were performed. The speedup and the computation time for each function were reported. For the RTI problem investigated in the current work, the code has shown good strong scaling capability with a speedup of 3.71x from 6 cores. This speedup value is comparable to those of other E-L methods. Also, most functions have scaled well in the weak scaling study. The level-set functions have been found to scale very well in both strong and weak scaling, thus validating the parallelizability of the MPLS method. The analysis of cache hit rates has highlighted that dynamic scheduling is more suitable when considering distributed memory parallelization. In contrast, guided scheduling is good for shared-memory parallelization. Attempts to incorporate GPU parallelism and to resolve the issue of parallel thresholds in pressure solver are underway.

Acknowledgement
The financial supports provided by the Ministry of Education Malaysia (Project no: FRGS/2/2013/TK01/UNITEN/02/1), Ministry of Science, Technology and Innovation (MOSTI) Malaysia (Project no: 06-02- 03-SF0258) and the Internal UNITEN research grant (Project no: J510050755) are greatly acknowledged and appreciated.

References

[1] Ng, K. C., Y. H. Hwang, Tony WH Sheu, and C. H. Yu. "Moving Particle Level-Set (MPLS) method for incompressible multiphase flow computation." *Computer Physics Communications* 196 (2015): 317-334.
[2] Duan, Guangtao, and Bin Chen. "Comparison of parallel solvers for Moving Particle Semi-Implicit method." *Engineering Computations* 32, no. 3 (2015): 834-862.
[3] Sussman, Mark, Peter Smereka, and Stanley Osher. "A level set approach for computing solutions to incompressible two-phase flow." *Journal of Computational physics* 114, no. 1 (1994): 146-159.
[4] Cubos-Ramírez, J. M., J. Ramírez-Cruz, M. Salinas-Vázquez, W. Vicente-Rodríguez, E. Martinez-Espinosa, and C. Lagarza-Cortes. "Efficient two-phase mass-conserving level set method for simulation of incompressible turbulent free surface flows with large density ratio." *Computers & Fluids* 136 (2016): 212-227.
[5] Hoeflinger, Jay, Prasad Alavilli, Thomas Jackson, and Bob Kuhn. "Producing scalable performance with OpenMP: Experiments with two CFD applications." *Parallel Computing* 27, no. 4 (2001): 391-413.
[6] Asao, Shinichi, Kenichi Matsuno, and Masashi Yamakawa. "Parallel computations of incompressible flow around falling spheres in a long pipe using moving computational domain method." *Computers & Fluids* 88 (2013): 850-856.
[7] Gao, Da, and Thomas E. Schwartzentruber. "Optimizations and OpenMP implementation for the direct simulation Monte Carlo method." *Computers & Fluids* 42, no. 1 (2011): 73-81.
[8] Wang, H., F. Liu, L. Xia, B. K. Li, and S. Crozier. "An efficient BiCGstab solved impedance method for induced field evaluation with a hyperthermia applicator." *In Engineering in Medicine and Biology Society*, 2008. EMBS 2008. 30th *Annual International Conference of the IEEE*, pp. 5636-5639. IEEE, 2008.
[9] Amritkar, Amit, Danesh Tafti, Rui Liu, Rick Kufrin, and Barbara Chapman. "OpenMP parallelism for fluid and fluid-particulate systems." *Parallel Computing* 38, no. 9 (2012): 501-517.
[10] Zhang, Shanghong, Zhongxi Xia, Rui Yuan, and Xiaoming Jiang. "Parallel computation of a dam-break flow model using OpenMP on a multi-core computer." *Journal of hydrology* 512 (2014): 126-133.
[11] Gingold, Robert A., and Joseph J. Monaghan. "Smoothed particle hydrodynamics: theory and application to non-spherical stars." *Monthly notices of the royal astronomical society* 181, no. 3 (1977): 375-389.
[12] Koshizuka, Seiichi, and Yoshiaki Oka. "Moving-particle semi-implicit method for fragmentation of incompressible fluid." *Nuclear science and engineering* 123, no. 3 (1996): 421-434.
[13] Goozee, Richard J., and Peter A. Jacobs. "Distributed and shared memory parallelism with a smoothed particle hydrodynamics code." *ANZIAM Journal* 44 (2003): 202-228.
[14] Wróblewski, Pawel, and Krzysztof Boryczko. "Parallel simulation of a fluid flow by means of the SPH method: OpenMP VS. MPI comparison." *Computing and Informatics* 28, no. 1 (2012): 139-150.
[15] Domínguez, José M., Alejandro JC Crespo, Daniel Valdez-Balderas, Benedict D. Rogers, and Moncho Gómez-Gesteira. "New multi-GPU implementation for smoothed particle hydrodynamics on heterogeneous clusters." *Computer Physics Communications* 184, no. 8 (2013): 1848-1860.
[16] Li, X., Xu, F., Gao, X., & Yang, Y. (2014). "Shared Memory OpenMP Parallelization of SPH Program and Its Application to Solid Fluid Interaction," In *European Conference on Computational Fluid Dynamics*, (Barcelona, Spain, ECFD VI 2014), 20-25.
[17] Sueyoshi, M. "3 dimensional simulation of nonlinear fluid problem by particle method over one million particles parallel computing on PC cluster." *J. Kansai Soc. N. A* 241 (2004): 133-142.
[18] Iribe, Tsunakiyo, Toshimitsu Fujisawa, and Seiichi Koshizuka. "Reduction of communication in parallel computing of particle method for flow simulation of seaside areas." *Coastal Engineering Journal* 52, no. 04 (2010): 287-304.
[19] Hori, Chiemi, Hitoshi Gotoh, Hiroyuki Ikari, and Abbas Khayyer. "GPU-acceleration for moving particle semi-implicit method." *Computers & Fluids* 51, no. 1 (2011): 174-183.
[20] Ovaysi, Saeed, and Mohammad Piri. "Multi-GPU acceleration of direct pore-scale modeling of fluid flow in natural porous media." *Computer Physics Communications* 183, no. 9 (2012): 1890-1898.
[21] Tsuji, Takuya, Keizo Yabumoto, and Toshitsugu Tanaka. "Spontaneous structures in three-dimensional bubbling gas-fluidized bed by parallel DEM–CFD coupling simulation." *Powder Technology* 184, no. 2 (2008): 132-140.
[22] Wei, Wenjie, Omar al-Khayat, and Xing Cai. "An OpenMP-enabled parallel simulator for particle transport in fluid flows." *Procedia Computer Science* 4 (2011): 1475-1484.
[23] Kosec, Gregor, Matjaz Depolli, Aleksandra Rashkovska, and Roman Trobec. "Super linear speedup in a local parallel meshless solution of thermo-fluid problems." *Computers & Structures* 133 (2014): 30-38.
[24] Chikazawa, Yoshitaka, Seiichi Koshizuka, and Yoshiaki Oka. "A particle method for elastic and visco-plastic structures and fluid-structure interactions." *Computational Mechanics* 27, no. 2 (2001): 97-106.
[25] Hwang, Yao-Hsin. "A moving particle method with embedded pressure mesh (MPPM) for incompressible flow calculations." *Numerical Heat Transfer, Part B: Fundamentals* 60, no. 5 (2011): 370-398.

[26] Hwang, Yao-Hsin. "Assessment of diffusion operators in a novel moving particle method." *Numerical Heat Transfer, Part B: Fundamentals* 61, no. 5 (2012): 329-368.
[27] Ng, K. C., Y. H. Hwang, and Tony WH Sheu. "On the accuracy assessment of Laplacian models in MPS." *Computer Physics Communications* 185, no. 10 (2014): 2412-2426.
[28] Darmana, Dadan, Niels G. Deen, and J. A. M. Kuipers. "Parallelization of an Euler–Lagrange model using mixed domain decomposition and a mirror domain technique: Application to dispersed gas–liquid two-phase flow." *Journal of Computational Physics* 220, no. 1 (2006): 216-248.
[29] Kafui, D. K., S. Johnson, C. Thornton, and J. P. K. Seville. "Parallelization of a Lagrangian–Eulerian DEM/CFD code for application to fluidized beds." *Powder Technology* 207, no. 1-3 (2011): 270-278.
[30] Yakubov, Sergey, Bahaddin Cankurt, Moustafa Abdel-Maksoud, and Thomas Rung. "Hybrid MPI/OpenMP parallelization of an Euler–Lagrange approach to cavitation modelling." *Computers & Fluids* 80 (2013): 365-371.
[31] Ataie-Ashtiani, B., and Leila Farhadi. "A stable moving-particle semi-implicit method for free surface flows." *Fluid dynamics research* 38, no. 4 (2006): 241.
[32] Tsuruta, Naoki, Abbas Khayyer, and Hitoshi Gotoh. "A short note on dynamic stabilization of moving particle semi-implicit method." *Computers & Fluids* 82 (2013): 158-164.
[33] Hwang, Yao-Hsin. "Arbitrary domain velocity analyses for the incompressible Navier-Stokes equations." *Journal of Computational Physics* 110, no. 1 (1994): 134-149.
[34] Shakibaeinia, Ahmad, and Yee-Chung Jin. "MPS mesh-free particle method for multiphase flows." *Computer Methods in Applied Mechanics and Engineering* 229 (2012): 13-26.
[35] Cummins, Sharen J., and Murray Rudman. "An SPH projection method." *Journal of computational physics* 152, no. 2 (1999): 584-607.
[36] Hu, X. Y., and Nikolaus A. Adams. "An incompressible multi-phase SPH method." *Journal of computational physics* 227, no. 1 (2007): 264-278.
[37] Grenier, Nicolas, Matteo Antuono, Andrea Colagrossi, David Le Touzé, and B. Alessandrini. "An Hamiltonian interface SPH formulation for multi-fluid and free surface flows." *Journal of Computational Physics* 228, no. 22 (2009): 8380-8393.
[38] Amdahl, Gene M. "Validity of the single processor approach to achieving large scale computing capabilities." In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pp. 483-485. ACM, 1967.
[39] "Profiling in Visual Studio," Microsoft Visual Studio Docs, accessed 15 September, 2017, https://docs.microsoft.com/visualstudio/profiling/