Two Parallel Algorithms for a Mass Transfer Simulation of Magnetic Nanoparticles

Kanok Hournkumnuard[1], **Chantana Phongpensri** [2], **Bunpot Dolwittayakul**[3], **Sergei Gorlatch**[4], and **Torsten Hoefler**[5], Non-members

ABSTRACT

We present a comparative study on the performance of two parallel algorithms for simulating mass transfer of weakly magnetic nanoparticles during the process of High Gradient Magnetic Separation (HGMS). The dynamics of mass transfer is investigated statistically in term of particle volume concentration and is described by the continuity equation which is solved numerically using the finite-difference. For parallelization, the concentration data are divided into equal parts that are distributed to a group of parallel processes. Parallel computations are performed by using two communication schemes of MPI, the pair-wise blocking and non-blocking operations. We compare the performance of both schemes in terms of parallel speedup, efficiency, and communication overhead. The results show that parallel simulation using the non-blocking communication has better performance than the blocking communication and also shows the better scalability with increasing number of processes.

Keywords: Parallel Computing, MPI, Mass Transfer Simulation, High Gradient Magnetic Separation

1. INTRODUCTION

The process of mass transfer plays an important role in many engineering and scientific applications, such that thermal diffusion, convection by fluid media, and influences of various forces (electrostatic, magnetostatic, gravitational, surface force etc. The governing equations of the mass transfer process are often non-linear partial differential equations of second or higher order which are difficult to be solved analytically, hence numerical methods are used. Finite-difference method is a standard method for solving partial differential equations. The more discrete points are used, the accurate are the results, but also the longer is the runtime necessary to accomplish the computation.

Parallel computing is used to improve the accuracy of the results and reduce the computation time. There are two major approaches of parallel programming. The first one is to use the compiler generating the parallel code. In this way, the parallel code is generated automatically. The compiler attempts to find the parallelism and derives the parallel code for the parts that are parallelizable. Another approach is an explicit way to specify parallelism by a user program. In this approach, supported libraries and compiler directives are needed. Such approaches can be done using message passing libraries (MPI) (www.mpi-forum.org), openMP (www.openmp.org), Berkley UPC (upc.lbl.gov), CUDA (www.nvidia.com) and many others.

MPI is one of the common approaches that is based on process communication concepts. It is evolved from PVM (Parallel Virtual Machine) and becomes a standard which is supported by many hardware vendors. Many previous works on parallel simulations are using MPI such as CLUSTEREASY [1] and NIRVANA [2]. The CLUSTEREASY is the version of LATTICEEASY which is the C++ program for doing lattice simulations of the evolution of interacting scalar fields in an expanding universe. It is the lattice simulation running on the cluster version. NIRVANA code is a general-purpose C code for astrophysical research which numerically integrates the 2D/3D equations of time-dependent, non-relativistic, compressible magnetohydrodynamics on Cartesian/cylindrical/spherical grids. Both of them are fundamental tools in physic simulations but have not focused on the nanoparticle capturing process.

In MPI, when a processor wants to send a message to another processor. The sender needs to call send function while the receiver needs to call *receive* function. There are two kinds of *send/receive* functions: blocking and non-blocking. The blocking one means the sender needs to wait until the receiver has already taken the data from the buffer and it can proceed to do other jobs. On the contrary, the non-blocking call implies that the sender can proceed after it finishes the *send* function without worrying about the receiver. However, the sender needs to perform the testing call at some point to ensure that the receiver gets the message. The time between the call

Manuscript received on August 1, 2009 ; revised on January 31, 2010.

[1,2,3] The authors are with Department of Computing, Faculty of Science, Silpakorn University, Thailand, E-mail: kanok_h@hotmail.com, ctana@su.ac.th and

[4] The author is with Universität Münster, Institut für Informatik, Münster, Germany, E-mail: gorlatch@math.uni-muenster.de

[5] The author is with 4Open System Lab, Indiana University, Bloomington, IN 47405, USA, E-mail: htors@cs.indiana.edu

and the testing can be used to perform useful computations. This is called overlapping communication with the computation.

In this work, we study the parallel simulation of mass transfer of weakly magnetic nanopartilces subjected to High Gradient Magnetic Separation (HGMS). Two communication schemes are studied – with blocking and non-blocking communication – and their performance is compared.

The paper is organized as follows. Firstly, the characters of the problem and the numerical method used are described. Secondly, procedures of sequential simulation and its parallelization are described. We implement two communications schemes using MPI and report th e results of experiments on a parallel cluster.

2. HIGH GRADIENT MAGNETIC SEPARATION

The scheme of the High Gradient Magnetic Separation (HGMS[3]) is shown in Fig. 1. The suspension of weakly-magnetic nanoparticles and a micron-size ferromagnetic capture (collector) are placed in a non-magnetic canister. A strongly uniform magnetic field is then applied, perpendicular to the collector s axis. All particles in the region close to the collector are subject to a magnetic force [4]:

$$\overrightarrow{F_m} = \frac{1}{2}\mu_0 \chi V_p \overrightarrow{\nabla}(H^2) \qquad (1)$$

where $\mu_0, \chi = \chi_p - \chi_f, V_p$, and H are the magnetic permeability of free space, the difference between the magnetic susceptibility of the particle and the fluid, the volume of an individual magnetic particle, and the magnitude of local magnetic field at the position of the particle, respectively. An efficient HGMS process, which means a strong magnetic force, requires the high strength and gradient of the magnetic field.

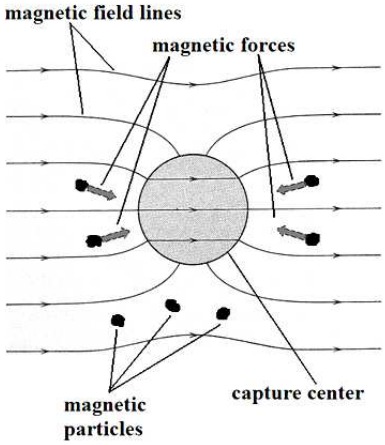

Fig.1: *Scheme of HGMS.*

In this work, we model the collector as a long ferromagnetic cylindrical wire. Due to the symmetry of the problem, dynamics of mass transfer can be studied in normalized polar coordinates r_a, θ as shown in Fig. 2, where the radial distances, r, is measured in the unit of wire s radius a. The mass transfer is studied statistically in term of particle volume concentration, denoted by c, which is a function of space and time and is defined as the fraction of particle volume contained in an infinitesimal volume element of the system (fluid with suspended particles) at any point.

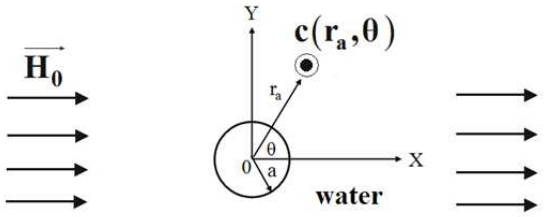

Fig.2: *Normalized polar coordinates.*

Dynamics of the concentration, which is a function of positions in the fluid and time, satisfies the continuity equation. For ordinary volume elements in the fluid which are not adjacent to any impervious surfaces, the continuity equation can be expressed as in [5]:

$$\frac{\partial c}{\partial \tau} = \frac{\partial^2 c}{\partial r_a^2} + \frac{1}{r_a}\frac{\partial c}{\partial r_a} + \frac{1}{r_a^2}\frac{\partial^2 c}{\partial \theta^2} - \frac{G_r c}{r_a} - G_r \frac{\partial c}{\partial r_a} - c\frac{\partial G_r}{\partial r_a} - \frac{G_\theta}{r_a}\frac{\partial c}{\partial \theta} - \frac{c}{r_a}\frac{\partial G_\theta}{\partial \theta}, \qquad (2)$$

where functions G_r, G_θ and factor G_0 were defined in [5] and $\tau = Dt/a^2$ is the normalized time, D and u are translational diffusion and translational mobility of the particle in the fluid. The surface of the wire and the surface of the static build-up of particles are considered as impervious surfaces where the radial flux of particles is equal to zero. Figure 3 depicts the special volume elements, labelled by the letter I, that are adjacent to the impervious surface.

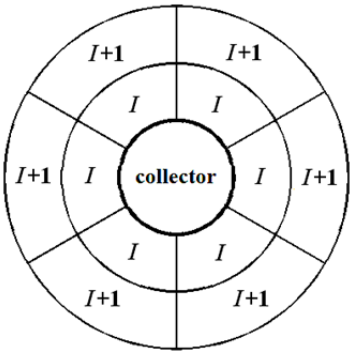

Fig.3: *Special volume elements.*

The normalized continuity equation for special volume elements can be expressed as shown in [5]:

$$\left(\frac{\partial c}{\partial \tau}\right)_I = \frac{1}{(r_a)_I^2}\left(\frac{\partial^2 c}{\partial \theta^2}\right)_I - \left(\frac{G_\theta}{r_a}\frac{\partial c}{\partial \theta}\right)_I - \left(\frac{c}{r_a}\frac{\partial G_\theta}{\partial \theta}\right)_I + \frac{(G_r c)_{I+1}}{\delta r_a} - \frac{1}{\delta r_a}\left(\frac{\partial c}{\partial r_a}\right)_{I+1}, \quad (3)$$

In the Equation subscript I and $I+1$ refer the special volume and δr_a its immediate adjacent volume element. is the distance between these two volume elements.

In particular, Equation (2) is used for ordinary volume elements whereas Equation (3) is used for special volume elements that are adjacent to the impervious surfaces such as the wire surface or the surface of the saturation buildup of the particle. It is assigned that the particle flux in the radial direction cannot pass through the impervious surface, $\partial c/\partial ra = 0$. Then, Equations (2) and (3) are solved numerically to obtain the image of the concentration distribution around the collector at any given normalized time.

3. SIMULATION SETUP

3.1 Discretization of Continuity Equation, and Computing Domain Granularity

For simulation, a uniform grid is constructed in an annular region enclosing the collector as in Figure 4a. The outer boundary is located at $r_{a,L} = 10, c^n_{i,j}$ denotes the value of concentration in an infinitesimal control volume located at the point $(r_{a,i}, \theta_j)$ at the instant of normalized time τ_n. Grid points adjacent to the impervious surface are called "special points" and denoted by $c^n_{s,j}$. Total concentration data are stored in a two-dimensional array as shown in Figure 4b. Two equal-size arrays are used for storing data at the current step τ_n, and at the next step of time τ_{n+1}. The concentration $c^n_{i,j}$ is the data stored at the i^{th} column and j^{th}row. In Figure 4b, index i_{max} corresponds to the outer boundary (r_{aL}) and the index j_{max} corresponds to the maximum angle $\theta_{max} = 360^\circ - \Delta\theta$where $\Delta\theta$ is the discrete step of θ in the finite-difference method. Initially, at $\tau = 0$, initial condition is assigned as $c^0_{i,j} = C_0$, ($C_0 = 0.0010$ in this work) for all i and j . In practice, inter-particle forces and interactions limit the concentration to a finite value and the saturation or static build-up occurs. Saturation build-up occurs approximately at $C_{sat} \approx 0.10$ [5]. A grid point with a concentration $c^n_{i,j} \geq C_{sat}$ is assumed to be the saturation point. The points of saturation are considered to be static build-up and are excluded from the process of computation by holding the concentration at C_{sat}. The concentration on the outer boundary, where the influence of the magnetic force is relative small, is held equal to the initial value C_0.

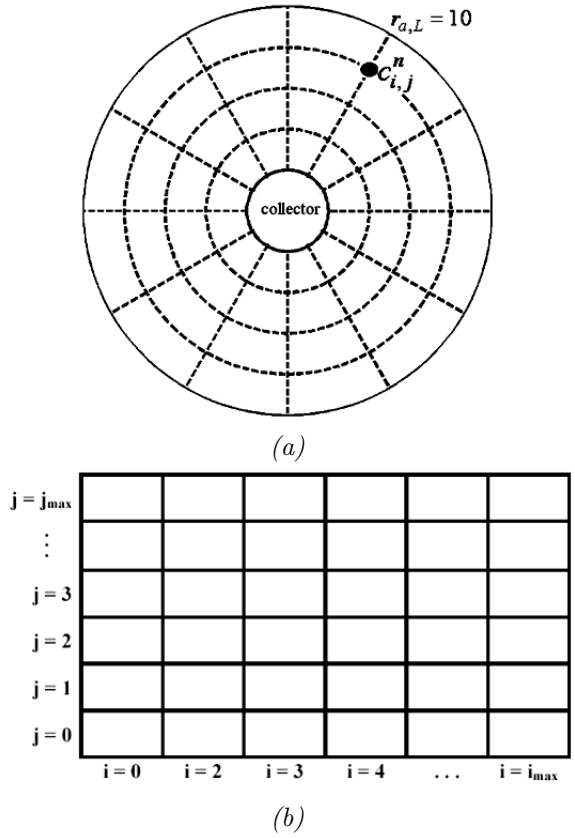

Fig.4: *(a) Grid construction, and (b) Corresponding concentration array.*

At first, Equations (2) and (3) are discretized via the finite-difference approximation. Finally, the value of concentration at a new step of normalized time, $c^{n+1}_{i,j}$, at a non-special point can be computed, by using the concentration of the present grid point and four surrounding grid points at the previous step of normalized time as follows:

$$\begin{aligned} c^{n+1}_{i,j} = & \left[1 - \frac{2(\Delta\tau)}{(\Delta r_a)^2} - \frac{2}{(r_a)_i^2}\left(\frac{\Delta\tau}{(\Delta\theta)^2}\right) - \left(\frac{(G_r)_{i,j}}{(r_a)_i} + \left(\frac{\partial G_r}{\partial r_a}\right)_{i,j} + \frac{1}{(r_a)_i}\left(\frac{\partial G_\theta}{\partial \theta}\right)_{i,j}(\Delta\tau)\right)\right] c^n_{i,j} \\ & + \left[\frac{\Delta\tau}{(\Delta r_a)^2} + \left(\frac{1}{2(r_a)_i} - \frac{(G_r)_{i,j}}{2}\right)\left(\frac{\Delta\tau}{\Delta r_a}\right)\right] c^n_{i+1,j} \\ & + \left[\frac{\Delta\tau}{(\Delta r_a)^2} + \left(\frac{1}{2(r_a)_i} - \frac{(G_r)_{i,j}}{2}\right)\left(\frac{\Delta\tau}{\Delta r_a}\right)\right] c^n_{i+1,j} \\ & + \left[\frac{1}{(r_a)^2} - \left(\frac{\Delta\tau}{(\Delta\theta)^2}\right) - \frac{(G_\theta)_{i,j}}{2(r_a)_i}\left(\frac{\Delta\tau}{\Delta\theta}\right)\right] c^n_{i,j+1} \\ & + \left[\frac{1}{(r_a)^2} - \left(\frac{\Delta\tau}{(\Delta\theta)^2}\right) - \frac{(G_\theta)_{i,j}}{2(r_a)_i}\left(\frac{\Delta\tau}{\Delta\theta}\right)\right] c^n_{i,j+1} \quad (4) \end{aligned}$$

Following similar steps, the difference equation used for special grid points can be obtained as

$$
\begin{aligned}
c_{I,j}^{n+1} = & \left[1 - \frac{2(\Delta\tau)}{(r_a)_I^2(\Delta\theta)^2} - \frac{\Delta\tau}{(r_a)_I}\left(\frac{\partial G_\theta}{\partial\theta}\right)\right] c_{I,j}^n \\
& + \left[\frac{(\Delta\tau)}{(r_a)_I^2(\Delta\theta)^2} - \frac{(G_\theta)_{I,j}(\Delta\tau)}{2(r_a)_I(\Delta\theta)}\right] c_{I,j+1}^n \\
& + \left[\frac{(\Delta\tau)}{(r_a)_I^2(\Delta\theta)^2} - \frac{(G_\theta)_{I,j}(\Delta\tau)}{2(r_a)_I(\Delta\theta)}\right] c_{I,j+1}^n \\
& - \left[\frac{(\Delta\tau)}{(\Delta r_a)^2} + \frac{(G_r)_{I+1,j(\Delta\tau)}}{\Delta r_a}\right] c_{I+1,j}^n \\
& + \left[\frac{(\Delta\tau)}{(\Delta_{r_a})^2}\right] c_{I+2,j}^n . \qquad (5)
\end{aligned}
$$

Figures 5(a) and 5(b) show the patterns of data dependence for typical grid points and special grid points, respectively, implied by Equations (4) and (5).

3.2 SEQUENTIAL SIMULATION

Let *max_round* denotes the total round of iteration. Parameters of simulation are the magnetic field strength H_0, magnetization of the collector M, absolute temperature T, particle s radius b_p, initial concentration C_0, saturate concentration C_{sat}, grid steps Δr_a,$\Delta\theta$ and $\Delta\tau$, and the size of concentration array.

The sequential simulation proceeds in five steps, as shown in Fig. 6, as follows:

Step 1: Setup simulation parameters.

Step 2: Compute various constants used in the simulation such as factor G_0.

Step 3: Assign initial value to concentration arrays.

Step 4: Iterative computing : while (*round* $\leq$ *max_round)*

4.1 For each row, do:

- Specify the column index "s" of special points.
- Perform the iterative computing, by using (4), starting from column $i_{max} - 1$ down to column $s + 1$ as shown in Fig. 6.
- Perform computation by using (5) in column s.

4.2 Copy data in new concentration array into old concentration array.

Step 5: Save simulation results in output files.

4. APPROACHES TO PARALLELIZATION

For parallelization we divide the concentration data into equal parts column-wise, such that each part, with the number of columns equal to *column_each_rank* as shown in the Figure 7, is assigned to one process. As usual when using MPI, we organize computations on a group of processes, each with unique identification number (rank). According to the data dependencies shown in Figures 5(a) and 5(b), the computation in the first and the last column of each subarray held by each process requires the data in the subarrays held by the two corresponding adjacent processes. On the other hand, each process provides the data in the first/ last column of its subarrays for the two neighboring processes. The

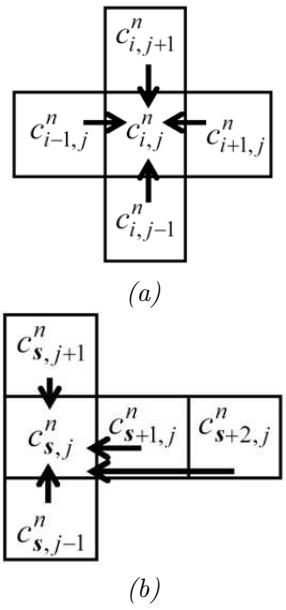

Fig.5: *(a) Data dependency for typical grid points, and (b) Data dependency for special grid points.*

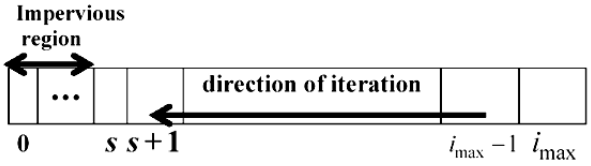

Fig.6: *Computing in a row.*

column-wise data partitioning implies the radial-wise decomposition of annular region enclosing the collector into many separate annular domains as shown in Figure 8.

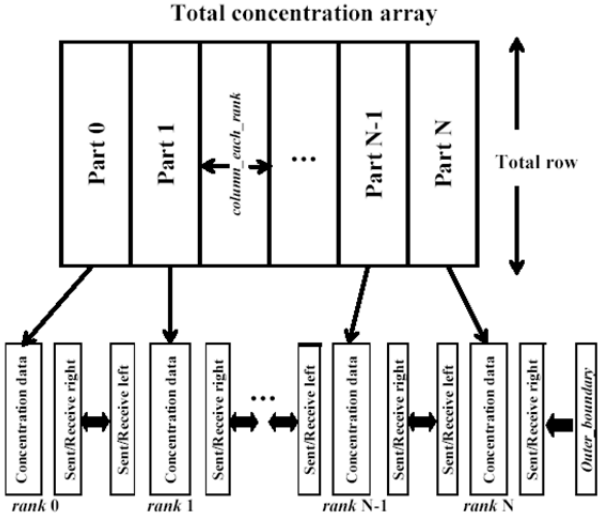

Fig.7: *Configuration of data distribution and pattern of linear data exchanges between processes.*

According to the pattern of data dependency in Fig. 5, each process must send its concentration data in the first and last columns to its two adjacent processes. However, data exchanges between first rank and highest rank processes are not required. Consequently, the processes communicate to its neighbors in the straight line pattern. Data exchange between subarrays is also performed via four one-dimensional arrays. These arrays have the same number of rows as the big array. These buffer arrays are called *Sent_left, Receive_left, Sent_right* and *Receive_right*. The *Sent_left* and *Receive_left* arrays are used to exchange data with the adjacent lower-rank process while the *Sent_right* and *Receive_right* arrays are used to exchange data with the adjacent higher-rank process. The first rank process uses only *Sent_right* and *Receive_right* arrays. An additional one-dimensional array called *Outer_boundary* which contains initial concentration C_0 is used for computing in the last column of subarray occupied by the highest rank process. Major scheme of parallel computation for column-wise partitioning can be described as follows. Let the subarray occupied by each process have the number of columns equal to *column_each_rank* which is obtained by the total number of columns in the big array divided by the number of processes. Then we have i_max = *colum_each_rank* - 1.

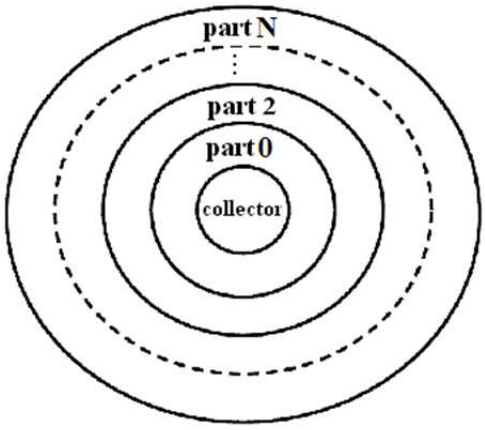

Fig.8: *Radial-wise annular domain decomposition.*

4.1 PARALLEL SIMULATION PROCEDURE

The steps of parallel simulation are as follows. Let N be the maximum rank in the process group.
Steps 1-3: The same as in the sequential process and adding parameter of *column_each_rank*.
Step 4: Iterative computation

4.1 Determine the associated minimum normalized radius from the relation

$$r_{a,min} = 1.0 + (rank \times column\_each\_rank \times \Delta r_a). \quad (6)$$

While (round $\leq$ max_round)

4.2 Check the rank of the process

4.2.1 If rank = 0 then follows step 4.1 of sequential computing.

4.2.2 If rank > 0 then, for each row of the associated subarray,

- Use Equation (4), starting from the column $i_{max} - 1$ down to column $i = 1$.

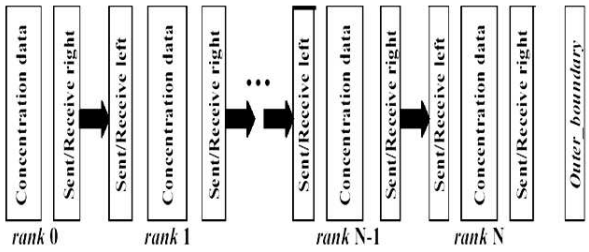

Fig.9: *Linear data exchanges before computing column i = 0.*

4.3 Exchange data with neighbour processes before computing column $i = 0$.

- Rank $0 \leq$ p $<$ N copies its data in column $i = i_{max}$ into its *Sent_right* array and sends the data to *Receive_left* array of the process rank p+1.
- Rank $1 \leq$ p $<$ N receives data from *Sent_right* of the process rank p-1 into its *Receive_left* .

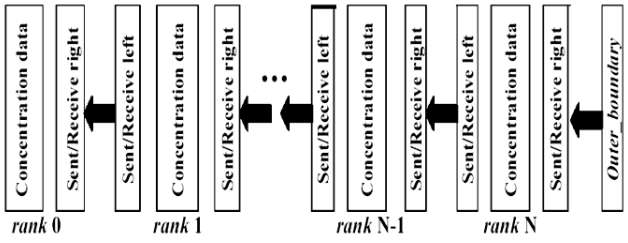

Fig.10: *Data exchange before computing column $i = i_{max}$.*

4.4 Compute the new concentration in the column $i = 0$.

4.5 Exchange data with neighbour processes before computing column $i = i_{max}$

- Rank $1 \leq$ p $\leq$ N copies its data in column $i = 0$ into its *Sent_leftt* array and sends the data to *Receive_right* array of rank p-1.
- Rank $0 \leq$ p $\leq$ N-1 receives data from *Sent_left* rank p+1 into its *Receive_right*
- Rank N receives data from *Outer_boundary* into its *Receive_right*.

4.6 Compute the new concentration in column $i = i_{max}$.

4.8 Copy data in the new concentration array into the old concentration array.

Step 5: Save simulation results in the output files.

Figures 9 and 10 show the pattern of data exchange between adjacent processes (Steps 4.3, 4.5) before computing in the column $i = 0$ and $i = i_{max}$, respectively.

4.2 NON-BLOCKING COMMUNICATION IMPROVEMENT

The first scheme of pair-wise communication is the *MPI_Send* and *MPI_Recev* which is the blocking communication. After each process finishes the computation in the columns of index $1 \leq i \leq i_{max} - 1$, it will start when the required column arrives. After the communication is finished, the computation can proceed further. The first linear chain of communications start in the pattern $P_0, P_1, ..P_N$ and in the second linear chain starts as $P_N, P_{N-1}, ..P_0$. Consequently, process 0 starts the communication first and finishes the communication last in overall. It is the process with the most waiting time as well. Thus, the blocking communication leads to a certain communication overhead and limits the performance of the parallel simulation.

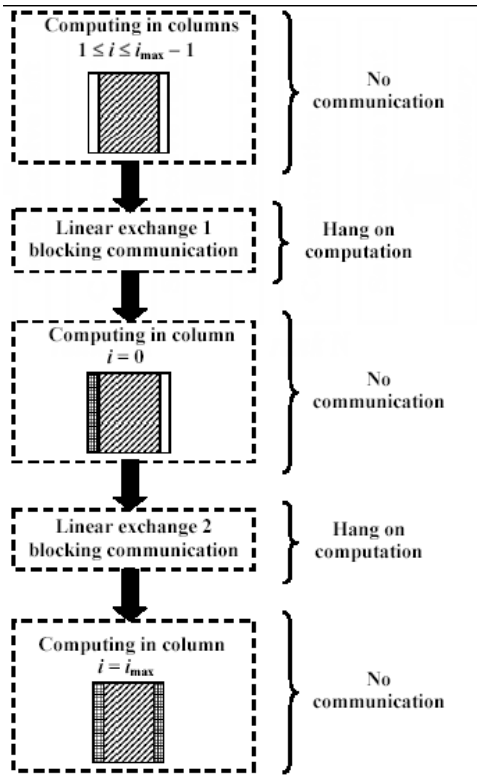

Fig.11: *Computation and communication steps in blocking communication.*

In the non-blocking scheme, we use *MPI_Isend*, and *MPI_Irecv* instead. We move the computation of column $1 \leq i \leq i_{max} - 1$ to hide the latency of the communications. Figure 11 shows computation and communication steps in Section 4.1. In Figure 12, we move the communications to the first step. Then, while we compute the column $1 \leq i \leq i_{max} - 1$ the communication is performed. When we need column $i = 0$, the *MPI_Wait()* is called to check whether the data is ready. Similarly, when we need column i_{max}, the *MPI_Wait()* is called to check whether the data is ready.

5. EXPERIMENTAL RESULTS AND DISCUSSION

5.1 CONCENTRATION DISTRIBUTION

We perform our experiments on a 32 nodes, totally 64 cores Linux cluster, with a Gigabit Ethernet interconnection at Louisiana Technology University, USA. In the cluster, each core is Intel Xeon 2.8GHz with 512 MB RAM. The cluster runs LAM-MPI 7.1 and on Gigabit Ethernet network.

We simulate mass transfer of paramagnetic Mn2P2O7 particles of radius bp = 12 nm which are dispersed in a static water. The effective magnetic susceptibility of the system (water + $Mn_2P_2O_7$ particle) is $\chi = +4.73 \times 10^{-3}$ [5]. The ferromagnetic cylindrical collector is considered to be homogeneously saturated magnetized by a uniform magnetic field $H_0 = 1 \times 10^6$ A/m which is perpendicular to the collector's axis, with saturation magnetization equal to $M_s = 1.6 \times 10^6$ A/m and the absolute temperature 300 K. The values of factors G_0 = -16.62 and K_w = 0.80. The value of initial concentration at every grid point is equal to $C_0 = 0.0010$ and the saturation concentration is equal to $C_{sat} = 0.10$. Grid steps are $\Delta r_a = 0.010$, $\Delta\theta = 0.10$ and $\Delta\tau = 0.0000010$. Hence, there are 3,600 rows and 901 columns in the whole computational domain as shown in Figure 7. The outer boundary of the annular domain $r_{a,L}$ is 10 and the boundary condition is assigned, such that the values of particle volume concentration at all grid points on the outer boundary are held fixed at the initial concentration C_0 . Figure 13 shows the family of concentration contours around the collector.

In Figure 13, we see the buildup of $Mn_2P_2O_7$ particles on the ferromagnetic cylindrical collector. Regions around the collector can be specified into three zones. The first zone is the saturation region, denoted , where the concentration at all points is equal to the saturation $C_{sat} = 0.10$. The second zone is the accumulation region, denoted by , where the value of concentration is larger than the initial value but less than the saturation value, i.e. $C_0 < c < C_{sat}$. Particles are accumulated dynamically in the this region. The radial magnetic force is active in both saturation and accumulation regions. The third zone is the depletion region, denoted by , where the value of concentration is less than the initial value:$0 < c < C_0$.

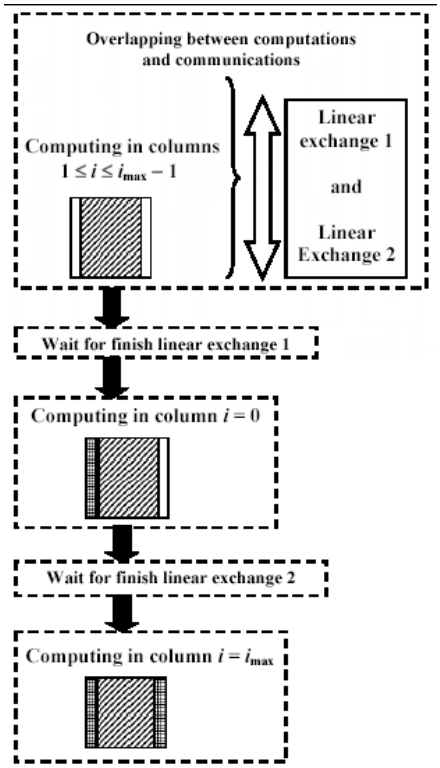

Fig.12: *Computation and communication steps in non- blocking communication.*

The radial magnetic force is repulsive in the depletion regions. In Figure 17, we see that, in paramagnetic mode, the buildup of $Mn_2P_2O_7$ particles on the collector occurs in the direction parallel to the direction of uniform external magnetic field $\overrightarrow{H_0}$. Particles are depleted in the direction that is perpendicular to the direction of $\overrightarrow{H_0}$ because they are carried to other regions by repulsive magnetic force.

Table 1: *Speedup and efficiency for blocking communication approach.*

No. of processes	$t_{avg}(s)$	S_p	E_p
1	22271	1.000	1.000
4	7229	3.081	0.770
6	4888	4.556	0.759
8	3619	6.154	0.769
10	3060	7.278	0.728
12	2607	8.544	0.712
16	1942	11.468	0.717
20	1635	13.623	0.681

5.2 SPEEDUP AND EFFICIENCY

A standard method to evaluate a parallel algorithm is to investigate its speedup and efficiency. The speedup can be computed as follows [6]:

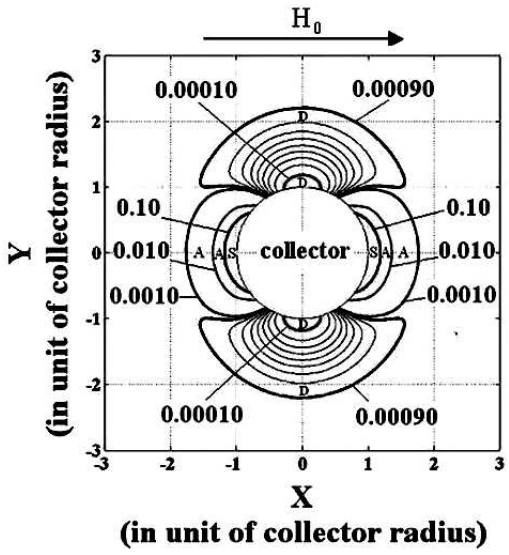

Fig.13: *Concentration contours around the collector.*

Table 2: *Speedup and efficiency for non-blocking communication approach.*

No. of processes	$t_{avg}(s)$	S_p	E_p
1	22271	1.000	1.000
4	7073	3.149	0.787
6	4720	4.719	0.786
8	3425	6.502	0.813
10	2825	7.885	0.788
12	2351	9.474	0.789
16	1744	12.774	0.798
20	1428	15.594	0.780

$$S_p = \frac{t_1}{t_p}, \tag{7}$$

where t_1 is average runtime of the sequential computation and t_p is the average runtime of parallel computation for the number of processes p. The efficiency of the parallel processes can be computed as follows:

$$E_p = \frac{S_p}{p}. \tag{8}$$

Speedup and efficiency of parallel simulation using blocking and non-blocking communication are shown in Table 1 and Table 2, respectively.

Figure 14 shows the comparison of the speedup of our two parallel algorithms using blocking and non-blocking communication respectively. Figure 19 shows the comparison with respect to efficiency. As the number of processes increases, the speedup increases consistently and monotonically. As expected, the speedup and efficiency of the non-blocking algorithm is significantly better than in the blocking case.

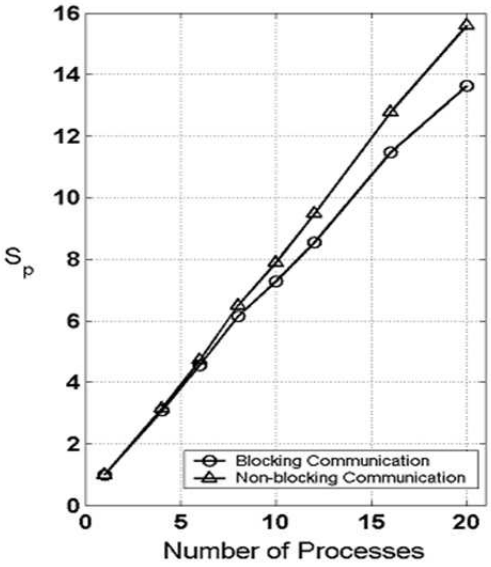

Fig.14: *Comparison of speedup.*

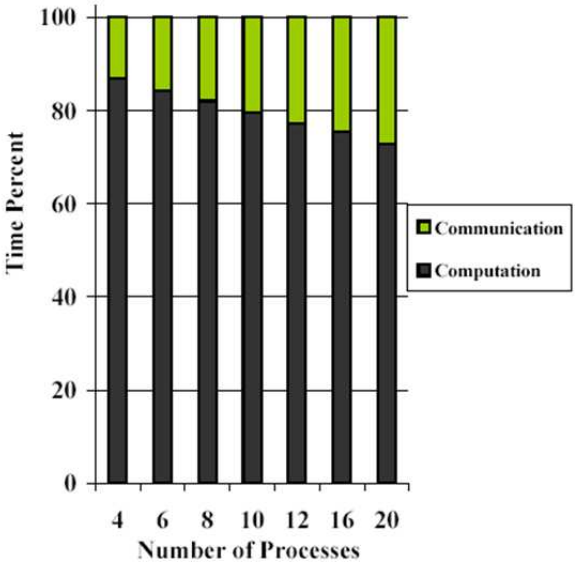

Fig.16: *Time percent for blocking communication.*

5.3 COMMUNICATION OVERHEAD

The communication overhead is studied by measuring the average total communication time of all MPI calls for each process per iteration. We then compute the percentage of computation and communication time per iteration based on total simulation time for both blocking and non-blocking communication as shown in Figure 16 and 17, respectively.

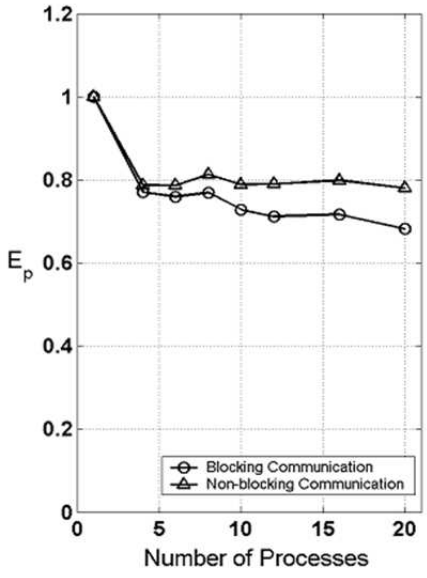

Fig.15: *Comparison of efficiency.*

We observe that the blocking communication incurs more overhead than the non-blocking one. In the blocking case, the communication is about 10-25% for each iteration while in the non-blocking case, it is about 10%. When there are more processes, a smaller number of rows of data is sent. However, the communication performs using two linear chains. Both chains get longer when the number of processes increases. Process 0 is the bottleneck since it starts the first chain in step 4.3 and it ends the last chain in step 4.5. When there are more processes, the computation domain gets smaller in step 4.4. Thus, the waiting time for process 0 increases with the number of processes. Then, this increases the proportion of the communication overhead per iteration of computation.

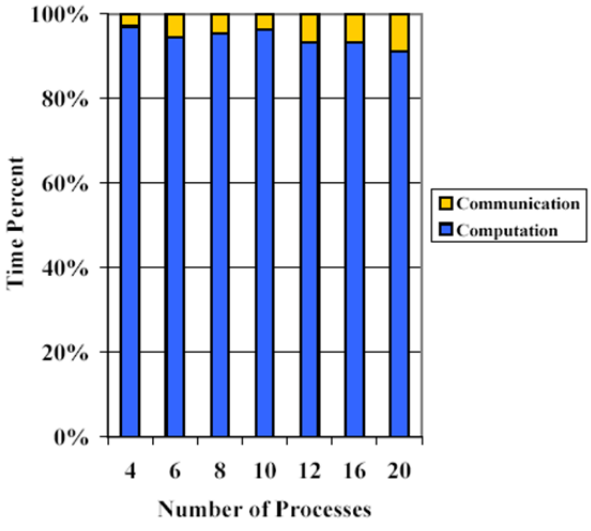

Fig.17: *Time percent for non-blocking communication.*

Figure 18 shows the comparison of total communication time of the blocking and non-blocking communication case. We observe that the non-blocking approach incurs much smaller overhead than the blocking one. When there are more processes, the com-

munication overhead is reduced because fewer data elements are sent, though the linear chain gets longer which has a small overall effect. Note that the effect of size of linear chain is visible more for each iteration in Figure 17. In the non-blocking scheme, the communication overhead is almost constant, i.e. the communications is hidden in these cases

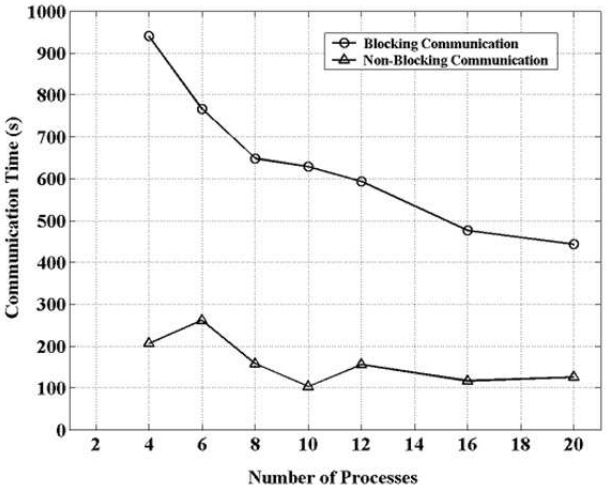

Fig.18: *Comparison of communication time.*

6. CONCLUSION

We developed, implemented and experimentally compared two parallel algorithms for High Gradient Magnetic Separation (HGMS) of nanoparticles. In both schemes, we distribute the domain of computation equally column-wise. The first algorithm uses blocking communication based on MPI_Send/Recv linear chain. The second algorithm uses non-blocking MPI_Isend/Irecv and computation steps hide communication latency. The results show that in on our cluster both algorithms provide a good speedup rate. However, the non-blocking algorithm performs better because the overhead incurred by the non-blocking MPI calls is lower and can be hidden totally in the overlapped computation.

7. ACKNOWLEDGEMENT

We would like to thank Assoc. Prof. Dr. Box Leangsuksun for providing a cluster for our experiments. Also, we thank Silpakorn University Research and Development Institute for the partial support.

References

[1] G. Felder, "CLUSTEREASY: A program for lattice simulations of scalar fields in an expanding universe on parallel computing clusters," *Computer Physics. Communications*, Vol. 179, pp. 604 - 606, 2008. Also available at (http://www.science.smith.edu/departments/Physics/fstaff/gfelder/latticeeasy/).

[2] U. Ziegler, "The NIRVANA code: Parallel computational MHD with adaptive mesh refinement," *Computer Physics Communications*, Vol. 179, pp. 227 - 244, 2008.

[3] F. Matthias, H. Uwe, R. Gerhard, "Magnetic filters on duty for cleaner metallic surface," *Geo-und Wassertechnologie*, Vol. 3, pp. 66 - 75, 2002.

[4] B. I. Bleaney, B. Bleaney, *Electricity and Magnetism*, Claredon Press, Oxford, 1965. Chapter 6.

[5] L. P. Davies, R. Gerber, "2-D simulation of ultrafine particle capture by a single-wire magnetic collector," *IEEE Transactions on Magnetics*, Vol. 26, No. 5, pp. 1867 - 1869, 1990.

[6] Micheal Quin, *Parallel Programming in C with MPI and OpenMp*, McGraw Hill, Singapore, 2003, Chapter. 7.

Kanok Hournkumnuard received the bachelor degree of electrical engineering (control engineering) in 1998 from King Mongkut s Institute of Technology Ladkrabang, Bangkok, Thailand. In 2004, he received the master degree of science (physics) from Chulalongkorn University, supported by Thai government scholarship. He has studied for Ph. D. of science (physics) since 2007 until now. Currently, he is a staff of department of physics, faculty of science, Silpakorn University, Nakhon Pathom, Thailand. His research interests include magnetic separation, computational physics, environmental physics, and physical waste water treatments.

Chantana Phongpensri received the B.S. degree of computer science(2^{nd} Class Honor) in 1991 from Thammasat University, Bangkok, Thailand. She then received the M.Sc. degree of computer science from Northeastern University, Boston, MA in 1994, and the Ph.D. degree in Computer Science and Engineering from the University of Notre Dame, Notre Dame, IN, USA in 1999, both on a Thai government scholarship. Currently, she is an associate professor at department of computing, faculty of science, Silpakorn University, Nakhon Pathom, Thailand. Her research interests include architecture synthesis for VLSI design, fuzzy system prototyping, real-time and embedded systems, parallel processing, wireless application development, and visual system design.

Bunpot Dolwittayakul

Sergei Gorlatch obtained his Master degree in Computer Science from the State University of Kiev in 1979 and a PhD degree in Computer Science from the National Institute of Cybernetics, Kiev, in 1984. In 1991-1992 he was a Humboldt research fellow at the Technical University of Munich/Germany. From 1992 to 1999 he worked as a research assistant and then as Assistant Professor at the University of Passau/Germany, where he obtained his "Habilitation" (qualification for professorship) in 1998. From 2000 to 2003, he was Associate Professor at the Technical University of Berlin. Since October 2003, Sergei Gorlatch has been Full Professor of Computer Science at the University of Muenster/Germany. He has more than 100 publications in renowned international journals and conferences, and has led several research and development projects in the field of parallel, distributed and Grid computing.

Torsten Hoefler