OhHelp: A Scalable Domain-Decomposing Dynamic Load Balancing for Particle-in-Cell Simulations

Hiroshi Nakashima
ACCMS, Kyoto University
Yoshida Hon-machi, Sakyo Kyoto, 606-8501, Japan
h.nakashima@mediakyoto-u.ac.jp

Yohei Miyake*
RISH, Kyoto University
Gokasho, Uji Kyoto, 611-0011, Japan
y-miyake@mediakyoto-u.ac.jp

Hideyuki Usui†
RISH, Kyoto University
Gokasho, Uji Kyoto, 611-0011, Japan
h-usui@port.kobe-u.ac.jp

Yoshiharu Omura
RISH, Kyoto University
Gokasho, Uji Kyoto, 611-0011, Japan
omura@rish.kyoto-u.ac.jp

ABSTRACT
This paper proposes a new method for Particle-in-Cell (PIC) simulations which aims at achieving both good load balancing and scalability so as to be efficiently executed on distributed memory systems. This method, named OhHelp, simply and equally partitions the space domain where charged particles reside and assigns each partitioned subdomain to each computation node for scalable simulation with respect to the size of the domain. Load balancing and thus the scalability in terms of the number of particles are accomplished by making each node help another heavily loaded node which deputes a part of particles in its subdomain and replicated subdomain field data to its helpers. The OhHelp load balancer monitors particle movements through subdomain boundaries to check if the helpand-helpers configuration keeps good load balancing and, when unacceptable imbalance is found, dynamically reconfigures it to regain perfect balancing. The efficiency and scalability of OhHelp are confirmed through our experiment with a production-level full-3D plasma simulator and with uniform and non-uniform particle distributions. As a result, we found 256-core parallel simulations, including an extremely imbalanced setting to pack all the particles in a small region, exert 159–190 speedup compared to sequential execution.

*Currently with ACCMS, Kyoto U.
†Currently with Grad. School of Engineering, Kobe U.

Categories and Subject Descriptors
D.1.3 [Programming Techniques]: Concurrent Programming—Parallel Programming; J.2 [Computer Applications]: Physical Sciences and Engineering

General Terms
Algorithms

Keywords
particle-in-cell simulation, load balancing, scalable parallelization

1. INTRODUCTION
Particle-in-Cell (PIC) simulations have played an indispensable role in theoretical and practical research of high-energy physics, space plasma physics, cloud modeling, combustion engineering, and so on, since early 1980’s (e.g., [7]). In typical PIC simulations, a huge number of charged particles interact with electromagnetic fields mapped onto a large number of grid points, governed by Maxwell’s equations and the Lorentz force law. These hugeness and largeness of the simulation essentially require to parallelize the computation not only for efficient execution but also for feasible implementation on distributed memory systems which are the majority of modern supercomputers. That is, the simulation has to be decomposed almost equally so that good load balancing is achieved and, more importantly, each decomposed subproblem is accommodated by a local memory of limited capacity. This almost-equal decomposition is a necessary condition to make the simulation scalable so that we fully utilize larger scale systems with nearly stable efficiency by enlarging the problem size proportionally to the system size.

Many researches on the parallel PIC simulation have been conducted for more than two decades[6] as we can see early proposals in literatures[14, 15, 16, 17, 30]. These proposals and successors can be projected onto a spectrum whose
two extremes are particle decomposition and domain decomposition. In a particle-decomposed simulation\[13, 17, 18, 29\], the set of simulated particles is statically divided into equal-size subsets each of which assigned to each computation node, while all the nodes share the whole of space domain data associated to grid points. Sharing the domain data is easy and natural for the implementations on shared memory machines and is fairly simple even for distributed memory machines because we just need to distribute the copies of them. Since the computational load for particles is perfectly balanced and is usually dominant over field solving, this method works well if the space domain is small. However, particle decomposition is not scalable with respect to the domain size because each node at least has the whole grid point data for reasonably efficient particle/field interaction. Moreover, as the domain size and machine scale becomes larger, all-reduce type communications to calculate the current and/or charge density of each grid point should become a significant factor and, furthermore, field solving or exchanging its result cannot be kept negligible.

On the other hand, a domain-decomposed simulation\[1, 9, 11, 16, 27\] is scalable with respect to the domain size because the space domain is equally partitioned to assign each subdomain to each computation node which is responsible of the particles in its subdomain. However, if particles are not uniformly distributed in the domain, severe load imbalance should be found to degrade parallel performance significantly. In the extreme case where all the particles gather in a subdomain, the simulation itself will collapses due to the memory shortage of the node for the subdomain. Thus simple domain decomposition cannot be scalable in terms of the number of particles.

To mitigate the load imbalance, the domain decomposition method can be modified to move the subdomain boundaries dynamically so that particle amounts in subdomains are balanced. The boundary shifting is relatively easy if the domain partitioning is one-dimensional\[12, 14, 23\] while more sophisticated methods such as ORB\[10\] and Octree\[2\] have been adopted\[5, 19, 24, 32\]. However, solving the load balancing problem with these dynamic repartitioning methods is insufficient for scalability because a subdomain can be almost as large as the whole domain if particles concentrate in a small region. Note that we cannot cap the subdomain size to avoid the problem, or a subdomain will have unacceptably many particles.

In this paper, therefore, we propose a new domain-decomposed PIC simulation method named OhHelp\[1\], which is scalable in terms of the number of particles as well as the domain size. Its problem decomposition and load balancing mechanisms are outlined as follows.

1. The space domain is equally partitioned to assign each subdomain to each node as its primary subdomain.

2. If one or more subdomains have too many particles, i.e., more than average plus a certain tolerance, every but one node is responsible of another subdomain which has particles more than average as its secondary subdomain.

Y OhHelp stands for “One-handed Help” because a node gives its one hand to help another node responsible for a subdomain with high particle density.

Since a node has to have at most two subdomains, OhHelp is scalable with respect to the domain size. As for the number of particles, OhHelp keeps its excess over the per-node average less than the tolerance by dynamically rearranging the secondary subdomain assignment and thus also achieves good scalability.

The rest of the paper is configured as follows: we first show OhHelp’s load balancing mechanism in Section 2; a PIC simulator implementation with OhHelp and its performance are discussed in Section 3 and 4 respectively; after revisiting the scalability issue referring to related work in Section 5, we conclude the paper in Section 6.

2. DOMAIN-DECOMPOSED LOAD BALANCING BY OhHelp

2.1 Overview

As shown in Figure 1, OhHelp simply partitions the simulated space domain into equal-size subdomains and assigns each subdomain to each computation node as its primary subdomain. In the figure, non-italic black numbers are the identifiers of nodes and also those of primary subdomains assigned to them. Each node is responsible for its primary subdomain, and also all the particles in it if the numbers of those primary particles in subdomains are balanced well, or more specifically, if the number of particles $P_d$ in a subdomain $d$ satisfies the following inequality for all $d$,

$$P_d \leq (P/N)(1 + \alpha) \equiv P_{\text{dom}}$$

where $P$ is the total number of particles, $N$ is the number of nodes, and $\alpha$ is the tolerance factor greater than 0. We refer to the simulation phases in this fortunate situation as those in primary mode.

Otherwise, i.e., if the inequality (1) is not satisfied for some subdomain $d$ as shown in Figure 1, the simulation is performed in secondary mode. In this mode, every node, except for one node (12 in the figure), is responsible for a secondary subdomain having particles more than the average, in addition to its primary one. For example, the subdomain 22 has helper nodes 02, 30 and 33 shown in italic and blue letters in Figure 1. The particles in a densely populated subdomain are also distributed to its helper nodes as their secondary particles so that each node $n$ has $Q_n$ particles in total, which reside in the primary or secondary subdomain.

Figure 1: Space Domain Partitioning

3. A part of particles in the secondary subdomain of a node are assigned to the node so that no nodes have too many particles.

4. OhHelp stands for “One-handed Help” because a node gives its one hand to help another node responsible for a subdomain with high particle density.
of \( n \), satisfying the following inequality for balancing similar to (1) for all \( n \).

\[
 Q_n \leq (P/N)(1 + \alpha) = P_{\text{hom}}
\]  
(2)

Note that since all but one nodes have secondary subdomains, a node whose primary subdomain is densely populated, e.g., node 22, is not only helped by other nodes but also helps another node 20, as the balancing algorithm discussed in Section 2.2 orders. It could seem irrational that a node with densely populated primary subdomain has to help other node, but this is the key for the balancing algorithm to achieve perfect balancing. For example, suppose a node with densely populated primary subdomain has to help other node, but this is the key for the balancing algorithm discussed in Section 2.2 orders. It could seem irrational that a node whose primary subdomain is densely populated to help another node 20, as the balancing algorithm discussed in Section 2.2 orders. It could seem irrational that a node whose primary subdomain is densely populated to help another node 20, as the balancing algorithm discussed in Section 2.2 orders.

Also note that the load in secondary mode is balanced not only in the number of particles but also in the size of responsible subdomains, although the latter load is twice as heavy as that in the primary mode. This is another justification for making a node with densely populated primary subdomain help another node.

The examination whether the load is balanced well and the mode switching possibly with load rebalancing are performed as follows every simulation time step in which particles can move crossing subdomain boundaries\(^3\).

1. If the inequality (1) is satisfied for all subdomains, the mode stays in or turns to primary. In the case of staying, only the particles crossing subdomain boundaries are transferred between nodes by neighboring communications. Otherwise, in addition to boundary crossing ones, particles that have been secondary are transferred to nodes responsible of them as primary particles.

2. If the current mode is secondary and the inequality (1) is not satisfied but (2) is satisifiable keeping the secondary subdomain assignment, the mode stays in secondary without global rebalancing. Particles may be transferred among the helpers and their helpnodes\(^4\) for the local load balancing in addition to the transfer of the particles crossing boundaries. The statisifiability check for (2) and the local balancing are discussed in Section 2.3.

3. Otherwise, the secondary subdomain assignments are performed (or modified) so that \( Q_n \) is equal to \( P/N \) for all \( n \) to accomplish perfect balancing\(^4\). The subdomain assignment algorithm is discussed in Section 2.2.

\(^3\)We could reduce the frequency of these operations by overlapping adjacent subdomains a little bit more heavily and by exploiting the fact that the velocity of a particle is limited to some upper bound, e.g., light speed, but our current implementation does not adopt this optimization.

\(^4\)We know English does not has such a word but dare to neologize to mean "the node helped by other nodes."

\(^5\)If \( P \) is a multiple of \( N \). Otherwise, \( Q_n \) is \( [P/N] \) or \( [P/N] \), but we assume \( P \) is a multiple of \( N \) hereafter for the sake of explanation simplicity.

2.2 Secondary Subdomain Assignment

When it is detected that the inequality (1) or (2) is unsatisfiable in primary or secondary mode respectively, secondary subdomains are assigned to nodes, by modifying the original assignment if the mode has already been in secondary, to accomplish perfect balancing. The fundamental assignment algorithm is quite simple as follows.

(b1) Split the set of nodes into two disjoint subsets \( L = \{ n \mid P_n < P/N \} \) and \( G = \{ n \mid P_n \geq P/N \} \). Let the tentative value of \( Q_n \) be \( P_n \) for all \( n \).

(b2) Repeat the following steps (b3) and (b4) until \( L \) becomes empty.

(b3) Remove a pair of elements \( l \) from \( L \) and \( g \) from \( G \). Assign the subdomain \( g \) to the node \( l \) as its secondary subdomain and also assign \( P_{g,l} = (P/N) - Q_l \) particles in the subdomain \( g \) to the node \( l \) so that \( Q_l \leftarrow Q_l + P_{g,l} = P/N \). Now \( Q_g \) becomes \( Q_g - P_{g,l} \).

(b4) If \( Q_g < P/N \), add \( g \) to \( L \). Otherwise add \( g \) back to \( G \).

(b5) If \( G \) has two or more elements, pick an arbitrary element \( r \) from \( G \) and assign the subdomain \( r \) to other nodes in \( G \) without particle assignment. Otherwise, i.e., \( G \) has one element only, let \( r \) be this node.

It is obvious the algorithm stops making every node \( n \) except for \( r \) have a secondary subdomain and \( Q_n = P/N \) for all \( n \). As mentioned in Section 2.1, the key for perfect balancing is the step (b4) where we add \( g \) with \( P_{g,l} \geq P/N \) but \( Q_g < P/N \) to \( L \) so that it helps other node when it has deputed so many particles to its helpers that \( Q_g \) becomes less than \( P/N \) tentatively.

The real algorithm is a little bit more complicated with a heuristics for the choice of \( l \) and \( g \) in the step (b3). For example, the balancing result shown in Figure 2 is obtained by choosing \( g \) in heaviest-first manner to supply its particles to \( l \) being lightest in \( L \), so as to make the number of helpers for a helpnode small. The figure corresponds to the particle distribution shown in Figure 1, and illustrates the number of particles in each subdomain (a) and that assigned to each...
node (b) by the bar whose color and numbers above and below it represent the subdomain and the node.

This heuristics is applied to initial balancing and primary to secondary transition, but should be harmful when we have been already in secondary mode because unnecessarily large number of particles can be moved between nodes. That is, this heuristics may randomly exchange helpers of helpands and thus may force secondary particle transfers between old and new helpers. Therefore, if we have been in secondary mode, we choose the pair of $l$ and $g$ so that the helpand-helper relations are kept as long as the perfect balancing is accomplished.

More specifically, the choice of $l$ and $g$ is performed as follows.

(i) The node $l$ is chosen so that $Q_l$ is the smallest.

(ii) If the mode is secondary and $l$ has been helping a node $n$ in $G$, let $g$ be $n$.

(iii) Otherwise, the node $g$ is chosen so that $Q_g$ is the greatest.

Note that the heuristics above only costs $O(N \log N)$ time to fulfill the aim to reduce the number of helpers of a helpand and of transferred particles at a certain level, although it may not give the optimum result which should cost $O(N^2)$ or longer time. Also note that other criteria, such as the physical communication bandwidth between a helpand and its helpers, may not give the optimum result which should cost $O(N^2)$ time. In fact, our fairly simple heuristics works quite as well as discussed in Section 4.2.

2.3 Checking and Keeping Local Balancing

In the secondary mode, the particle movements crossing subdomain boundaries could break the satisfiability of the inequality (2) if we stuck to the secondary subdomain assignment. To examine the satisfiability and to keep the local balancing among a helpand-helper family, we form a tree $T$ whose vertices are the computation nodes and edges represent helpand-helper relationship. That is, the root of the tree is the node $r$ defined in the step (b5) of the previous section, and the parent of a non-root node is its helpand. The tree corresponding to the balancing result in Figure 2(b) is show in Figure 3.

The examination of the satisfiability of (2) is performed by traversing the tree $T$ in a bottom-up (leaf-to-root) manner as follows.

(e1) Let a set of nodes $S$ be that of leafs of the tree $T$. Let $P_n^0$ be $P_n$ for all $n \in S$. If there is an element $n \in S$ such that $P_n = P_n^0 > P_{lim}$, the examination fails.

(e2) Repeat the following steps (e3) and (e4) until $S$ becomes $\{r\}$.

(e3) Find a node $n$ such that the set of its helpers $H(n)$ is a subset of $S$, and remove $H(n)$ from $S$.

(e4) Add $n$ to $S$ and let $P_n^0$ be as follows.

$$P_n^0 = \max(0, P_n - \sum_{m \in H(n)} (P_{lim} - P_m^0))$$

If $P_n^0 > P_{lim}$, the examination fails.

Since a leaf node does not have helpers, the failure in the step (e1) obviously means that the inequality (2) cannot be satisfied. As for the failure in (e4), since $\sum_{m \in H(n)} (P_{lim} - P_m^0)$ means the maximum particle amount which n’s helpers can accommodate as their secondary particles and thus $P_n^0$ is the minimum number of particles in $n$ which the node n has to be responsible, $P_n^0 > P_{lim}$ leads us that the inequality (2) is unsatisfiable. Therefore, the algorithm is complete. On the other hand, when the algorithm stops at (e2) with $P_n^0 \leq P_{lim}$ for all $n$, it is assured that, for all $n$, $P_n$ particles can be distributed among $n$ and its helpers keeping $Q_m^0 \leq P_{lim}$ for all $m \in F(n)$ where $F(n)$ is defined as $\{n\} \cup H(n)$. That is, even if $n$ has to accommodate $P_{lim} - P_m^0$ particles for its helpand, $P_n - P_n^0$ particles can be accommodated by its helpers because they are at most $P_m^0 - P_m^0$. Therefore, the algorithm is sound.

If the examination passes, a part of particles in a subdomain $n$ are redistributed to the members of the family $F(n)$, i.e., the node $n$ and its helpers in $H(n)$. The target of the redistribution is the following, where $Q_k^0$ is the number of particles in the subdomain $n$ and currently accommodated by the node $k$.

- Particles currently in a node $m \notin F(n)$, which has just crossed a boundary and moved into the subdomain $n$ from other subdomain.
- Particles overflowed from a node $m \in F(n)$. More specifically, particles are overflowed from $m$ in either of the following cases.

  - $m \neq n$ and $Q_m^0 + P_m^0 > P_{lim}$ and thus $Q_m^0 + P_m^0 - P_{lim}$ particles are overflowed to satisfy the minimum requirement defined by $P_m^0$.

  - $m = n$ and $Q_n^0 + R_n > P_{lim}$, where $R_n$ is the number of particles assigned to $n$ as the result of the redistribution for the family to which $n$ belongs as a helper. The number of overflowed particles is $Q_n^0 + R_n - P_{lim}$.

Note that the criteria above are to minimize the amount of particle transfer rather than to minimize the load deviation among the nodes. Let $D_n$ be the total number of redistributed particles defined above or, more specifically, be as follows.

$$D_n = \sum_{k \notin F(n)} Q_k^0 + \sum_{m \in H(n)} \max(0, Q_m^0 + P_m^0 - P_{lim}) + \max(0, Q_n^0 + R_n - P_{lim})$$
The local balancing in a helpand-helper family is partly achieved by the following algorithm traversing the tree $T$ in a top-down manner.

(d1) Let a set of node $S$ be \{r\}, and $R_r$ be 0. 

(d2) Repeat the following steps (d3) to (d6) until $S$ becomes empty.

(d3) Remove a node $n$ from $S$. If $n$ is the leaf node, let $Q_n$ be $P_n + R_n$ and skip the following steps (d4) to (d6). Otherwise, add the helpers of $n$, i.e., $H(n)$, to $S$.

(d4) If the following inequality is satisfied:

$$P_n + R_n + \sum_{m \in H(n)} \max(P_m^0, Q_m^0) \leq P_{im} \cdot |F(n)|$$

we need not to push down primary particles of any node $m$ to its own helpers. If this holds, let $B_m$ (\(m \in H(n)\)) = $\min(P_{im}, Q_m^0 + \max(P_m^0, Q_m^0))$ to represent the baseline number of particles above which we place particles to be redistributed as shown in Figure 4(a). Otherwise, let the baseline $B_m$ be $\min(P_{im}, Q_m^0 + P_m^0)$ to allow us to push down $Q_m^0 - P_m^0$ particles as shown in Figure 4(b). In both cases, let $B_n$, the baseline of $n$, be $\min(P_{im}, Q_m^0 + R_n)$.

(d5) Find the minimum subset $F'(n)$ of $F(n)$ such that the followings are satisfied.

$$\forall m' \in F'(n), \forall m \in F(n) \cap F'(n): B_{m'} \leq B_m$$

$$\forall m \in F(n) \cap F'(n): D_n + \sum_{m' \in F'(n)} B_{m'} \leq B_m \cdot |F'(n)|$$

(d6) Let $R_m$ for all $m \in H(n)$ and $Q_n$ be the followings.

$$R'_m = \begin{cases} (D_n + \sum_{m' \in F'(n)} B_{m'})/|F'(n)| - B_m & m \in F'(n) \\ 0 & m \notin F'(n) \end{cases}$$

$$R_m = R'_m + Q_n = B_n + R_n$$

The step (d5) is to find the leftmost three bars (nodes) in Figure 4(a) and (b) for the local load balancing among these lightly loaded nodes by distributing $R'_m$ given in the step (d6).

---

**Figure 4: Particle Redistribution in a Family**

(a) without pushing down (b) with pushing down

---

**Figure 5: 3D PIC Simulator with OhHelp**

3. IMPLEMENTATION OF 3D PIC SIMULATOR WITH OhHelp

To investigate the efficiency and scalability of OhHelp, we implemented a full-3D PIC simulator with OhHelp. The implementation is based on our production level simulator for space plasma[29], a type of pure PIC code without particle-particle interaction although OhHelp itself is capable of it as discussed later.

As shown in Figure 5 which illustrates the outline of the base simulator and the OhHelp application methodology, the main loop of the simulator consists of the following phases.

**particle pushing:** Each node accelerates its primary and secondary particles by electric and the Lorentz force law referring electromagnetic field data $E$ and $B$ associated to the grid points in its primary and secondary subdomains, and then moves particles according to their updated velocities. Particle movements crossing subdomain boundaries will be taken care of by the last phase but crossing particles are counted to build the population histogram for each subdomain. This phase is executed locally.

**current scattering:** Each node locally calculates the contributions of the movement of its primary and secondary particles to the current density $J$ at the grid points in its primary and secondary subdomains. Then an all-reduce communication is performed in each helpand-helper family to sum up the current density in the subdomain of the family. Finally, the boundary values of $J$ are exchanged between primary subdomains and then broadcasted from the helpand to its helpers.

**field solving:** Each node locally updates the value of $E$ and $B$ at the grid points in its primary and secondary subdomains using leapfrog method to solve Maxwell’s equations. Then the boundary values of $E$ and $B$ are exchanged between primary subdomains and then broadcasted from the helpand to its helpers. Note that the base simulator adopts a charge conservation method[28] so that it is unnecessary (or at most very infrequently required) to solve Poisson’s equation explicitly to have electrostatic potential from charge density.
load balancing and particle transferring: The histograms of particle population are exchanged by an all-to-all communication and an all-reduce one, followed by, if in secondary mode, an all-gather one. The first one acquaints each node with the number and the sources of boundary crossing particles into its primary region, while the second one sums up the number of particles in each region. The last one is to make all nodes share the number of primary and secondary particles in each node, with which the balancing algorithm discussed in Section 2 is executed in all nodes\(^5\). If it is necessary to change the secondary subdomain assignments, each node broadcasts its electromagnetic field values to its helpers after new helpand-helper families are established. Then, after each node makes the schedule of particle transfer for its primary region and notifies involved nodes of it, particles are transferred among nodes.

As Figure 5 suggests, applying OhHelp to existent simulators is fairly easy especially when they have been parallelized by domain decomposition method. In fact, required modifications to the main simulation loop of an existent domain-decomposed simulator are, besides attaching OhHelp load balancer module, just as follows.

- Duplicate the execution of the first three phases for primary particles/subdomain and their secondary counterparts. This will be done, for example, by simply calling procedures twice giving them appropriate data structures through their arguments and/or global variables. In fact, our implementation merely needs a 50-line setup procedure for pointer adjustment and so on.
- Add collective communications inside each helpand-helper family, one all-reduce for current scattering and two broadcasts of boundary values of current density and electromagnetic fields. The only attention to be paid to add them is a node may belong to two families, one as a helpand and the other as a helper. Thus if we carelessly perform collective communications by doing them, for example, as a helpand and then as a helper, it may cause unnecessary serialization because the root family must wait the completion of the communications in the second generation families which must wait those in the third ones and so on. This problem is solved by a simple red-black technique which paints families of odd-number generations by red and even ones by black and performs communications of red families first and then of black families. This merely requires 10-line procedure for each type of communications.

4. PERFORMANCE EVALUATION

4.1 Evaluation Setup

We coded the PIC simulator discussed in the previous section using Fortran 90 for the main part of the simulation and C for OhHelp load balancer. The program was compiled by Fujitsu’s Fortran and C compilers version 3.0 with

\(^5\)Replicated execution of the balancing algorithm is logically unnecessary of course, but is a little bit more efficient than executing solely by a node and broadcasting the result.

-\(K\)fast option and then linked with Fujitsu’s MPI library version 3.0. Then the program was executed on our T2K Open Supercomputer[20], which has 416 Fujitsu’s HX600 server nodes each of which is equipped with four quad-core Opteron 8356 processors, 32 GB shared memory and four-way Infiniband 4x DDR links for full-bisection and 8 GB/s per-node high-throughput interconnect. We used up to 16 nodes or 256 cores by mapping each MPI process onto a core.

We evaluated our simulator with two types of scaling, namely strong scaling and weak scaling (Figure 6). For the former, we fixed the space domain size to \(64^3\) grid points and the number of particles to \(2^{27}\) or about 134 million, a half of which have positive charge while the other half are charged negatively with the same absolute value. Then the domain is decomposed for \(2^x \times 2^y \times 2^z\) process arrays where \(2^x : 2^y : 2^z\) is either of 1:1:1, 2:1:1 or 2:2:1. Therefore, the subdomain size of 256-process execution with \(8 \times 8 \times 4\) process array is \(8 \times 8 \times 16\). For weak scaling, on the other hand, we fixed the subdomain size to \(32^3\) and the average number of particles in a subdomain to \(2^{27}\) or about 8.4 million with the shapes of process arrays and charge neutralization same as the strong case. Thus the total domain size of 256-process execution is \(256 \times 256 \times 128\) or about 8.4 million grid points and the number of particles is \(2^{31}\) or about 2.1 billion.

With both scaling types, we examined the performance with two extreme initial particle distributions, balanced and unbalanced. In the balanced case, particles are uniformly distributed in the space domain and have a constant initial velocity toward x-axis. Since no external electromagnetic field is given\(^6\), particles steadily travel to keep simulated in primary mode with perfect load balancing. In the unbalanced case, particles are also uniformly distributed and have the constant initial velocity but reside in a small cubic

\(^6\)Because we did not aim at simulating some physical phenomena but evaluating simulator performance.
Table 1: Performance of Strong Scaling

<table>
<thead>
<tr>
<th>#proc</th>
<th>part.decomp.</th>
<th>balanced</th>
<th>unbalanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.88/ 1.00</td>
<td>2.55/ 0.89</td>
<td>2.55/ 0.89</td>
</tr>
<tr>
<td>2</td>
<td>5.64/ 1.95</td>
<td>4.74/ 1.65</td>
<td>4.31/ 1.50</td>
</tr>
<tr>
<td>4</td>
<td>11.27/ 3.92</td>
<td>9.98/ 3.47</td>
<td>8.57/ 2.98</td>
</tr>
<tr>
<td>8</td>
<td>20.45/ 7.11</td>
<td>18.31/ 6.36</td>
<td>15.70/ 5.46</td>
</tr>
<tr>
<td>16</td>
<td>36.21/ 12.58</td>
<td>34.30/ 11.92</td>
<td>30.54/ 10.61</td>
</tr>
<tr>
<td>32</td>
<td>59.32/ 20.62</td>
<td>68.94/ 23.96</td>
<td>60.42/ 21.00</td>
</tr>
<tr>
<td>64</td>
<td>83.63/ 29.06</td>
<td>137.98/ 47.95</td>
<td>121.42/ 42.19</td>
</tr>
<tr>
<td>128</td>
<td>98.88/ 34.36</td>
<td>269.20/ 93.55</td>
<td>238.67/ 82.94</td>
</tr>
<tr>
<td>256</td>
<td>101.42/ 35.25</td>
<td>519.04/180.38</td>
<td>457.49/158.98</td>
</tr>
</tbody>
</table>

Table 2: Performance of Weak Scaling

<table>
<thead>
<tr>
<th>#proc</th>
<th>balanced</th>
<th>unbalanced</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.55/ 0.89</td>
<td>2.55/ 0.89</td>
</tr>
<tr>
<td>2</td>
<td>4.95/ 1.72</td>
<td>4.63/ 1.61</td>
</tr>
<tr>
<td>4</td>
<td>9.11/ 3.16</td>
<td>8.06/ 2.80</td>
</tr>
<tr>
<td>8</td>
<td>18.31/ 6.36</td>
<td>15.70/ 5.46</td>
</tr>
<tr>
<td>16</td>
<td>34.44/ 11.97</td>
<td>30.58/ 10.63</td>
</tr>
<tr>
<td>32</td>
<td>68.57/ 23.83</td>
<td>59.88/ 20.81</td>
</tr>
<tr>
<td>64</td>
<td>137.13/ 47.66</td>
<td>122.29/ 42.50</td>
</tr>
<tr>
<td>128</td>
<td>274.00/ 95.22</td>
<td>245.64/ 85.36</td>
</tr>
<tr>
<td>256</td>
<td>545.86/189.69</td>
<td>478.93/166.44</td>
</tr>
</tbody>
</table>

4.2 Performance

Measured performance numbers of strong and weak scaling executions with up to 256 processes (cores) are shown in Table 1 and 2, while Figure 7 shows them graphically. The performance numbers in these tables are displayed in the form of "speed/speedup". The simulation speed in the unit of particle-per-second (PPS) is defined as follows.

\[
\text{speed} = \frac{\text{number of particles} \times \text{time steps}}{\text{execution time excluding initialization in second}}
\]

The speedup is the simulation speed normalized by that of the sequential execution of the reference implementation only using particle decomposition method whose performance is also shown in Table 1. Note that this reference sequential execution has no overhead for parallelization, while one-process execution of our simulator incurs self communications of particle movements crossing periodic boundaries and for exchanging boundary field/current values by itself. The absolute execution time of the reference sequential execution is about 83 hours or three days and a half, while 256-process strong scaling executions take 28 minutes and 31 minutes with balanced and unbalanced setting respectively.

By viewing Table 1 and Figure 7, it is obvious that our simulator exerts good scalability and much more scalable than the reference particle-decomposed type simulator as expected. The reason why the performance of the reference simulator is saturated at around 100 million PPS is that the cost of the all-reduce communication in current-scattering phase for the \(64^3 \times 3 \times 8 = 6\) MB array of current density becomes the dominant factor of the execution. In fact, our profiling revealed that, in the reference simulator with 256 processes, the execution time proportional to the number of particles takes only 15% of the total, while 74% is for the all-reduce and the remaining and non-negligible 11% is for the field solving.

Contrary to the results of the reference simulator, our simulator looks exerting unsaturated performance at least up to 128 processes. That is, the parallel efficiency defined as \(\frac{\text{speed}}{\text{speedup}}/\text{number of processes}\) is stable from 16 to 128 processes at about 74% and 65% in balanced and unbalanced cases respectively. Since the problem size is fixed, we cannot expect a sustained efficiency with more processes as we find those of balanced and unbalanced 256-process executions are degraded to 70% and 62% respectively mainly due to the fact that collective communications for particle amount histogram become significant taking about 5% of the total execution time. This means, on the other hand, that our simulator is only 5% slower than the best case of a simple domain-decomposed type simulator because the histogram exchanging and gathering are the sole overhead when the number of particles is balanced well.

The performance difference of about 12% between balanced and unbalanced cases is due to two factors which almost evenly share the degradation. One is small load imbalance of number of particles up to 20% above the average because OhHelp allows it. The other is the all-reduce of the current density in each helpand-helper family, but this is much less significant than that in the reference simulator because the size of the array to be reduced is \(8 \times 8 \times 16 = 1/256\) of that of the reference simulator.

The weak scaling performance shown in Table 2 and Figure 7 is better than the strong scaling as expected. In fact, the parallel efficiencies above 16 processes of balanced and unbalanced cases are both stable even in 256-process executions at 75% and 66% respectively. Although the simu-
lations incur the overhead of the collective communications for the histogram as in strong scaling ones, it is relatively insignificant because the 256-process execution time of one time step is about 17 times as long as that of strong scaling.

The main degradation factors from balanced to unbalanced setting are same as those in strong scaling, small im-balance of particles and all-reduce for current density, but the other factor caused by broadcasts of field boundaries is also as significant as them because we have larger family population and subdomain size. In fact, 14% degradation is almost completely and evenly due to these two factors, because other factors are negligibly small, less than 0.5% in total. That is, factors such as solving field twice, check-
ing the satisfiability of the inequality (2), restructuring the helpand-helper tree, and transferring particles between non-neighboring nodes are commonly insignificant. Note that the tree restructuring itself is a costly operation mainly because it requires to establish new MPI communicators for families, but its low frequency, 23 times out of 6400 time-steps, makes it insignificant. Also note that the negligible overhead due to non-neighboring particle transfers justifies our simple policy of helpand-helper combination.

5. RELATED WORK AND SCALABILITY ISSUES

In this section, we revisit the scalability issues we briefly discussed in Section 1 referring to related work. For the discussion we introduce the notation $D$ to represent the space domain size in terms of the number of grid points, besides $P$ and $N$ which we have defined as the numbers of particles and computation nodes. For $P$ and $D$, we also define their per-node amounts $\bar{P} = P/N$ and $\bar{D} = D/N$ and the worst case per-node amounts $\bar{P}$ and $\bar{D}$ in a certain parallel implementation. It is necessary for a scalable implementation that $\bar{P} = k_{\bar{p}} \bar{P}$ and $\bar{D} = k_{\bar{d}} \bar{D}$ with some constants $k_{\bar{p}}$ and $k_{\bar{d}}$ to make the execution time for a time step $O(\bar{P} + \bar{D})$.

With these notations, it can be shown that particle-decomposed simulations\cite{13, 17, 18, 29} are not scalable because $\bar{P} = \bar{P}$ but $\bar{D} = \bar{D}$. Since $P \gg D$ in usual cases, a small scale parallel executions are feasible but, even if we have a large local memory to accommodate a large $D$, the all-reduce communication, which takes $O(D \log N)$ time with butterfly algorithms or $O(D + \log N)$ even with more sophisticated recursive halving/doubling techniques\cite{26}, should make parallel performance saturated as shown in Section 4.2.

Simple domain-decomposed simulations\cite{1, 9, 11, 16, 27} with $\bar{D} = \bar{D}$ are also unscaleable due to the possibility of $\bar{P} = \bar{P}$ in an extreme case such as the unbalanced setting discussed in Section 4.1. This setting for the 256-process weak scaling simulation, in which all the particles concentrate in a region of 1/256 of the space domain, reveals the limit of dynamic domain partitioning because one-dimensional partitioning\cite{12, 14, 23} makes $\bar{D} \approx D/2$ and even with ORB or Octree\cite{5, 24, 32} we should have $\bar{D} \approx D/8$ for three-dimensional simulations.

Therefore, we need to combine static domain decomposition to make $\bar{D} \approx k_{\bar{d}} \bar{D}$ with dynamic particle decomposition (i.e., load balancing) as our OhHelp does. Thus a simple combination of static or dynamic domain decomposition and static particle decomposition, such as those exploiting shared and distributed memory hierarchy\cite{3, 4, 8} or dimension oriented ones\cite{31}, is not scalable because it cannot keep $\bar{D} = k_{\bar{d}} \bar{D}$ or $\bar{P} = k_{\bar{p}} \bar{P}$.

Another simple combination is to apply two static decomposition independently\cite{22, 30} by assigning a statically partitioned subdomain and a subset of particles irrespective of their position to a computation node. Although this could achieve $\bar{D} = k_{\bar{d}} \bar{D}$ and $\bar{P} = \bar{P}$ by further splitting local particles according to their resident subdomains and fetching field data and scattering current/charge density for each local particle subset, it costs $O(D)$ communication time to make parallel performance saturated. A better algorithm is to make this combination correlated\cite{15} by dynamically decomposing the set of particles so that each subset is spatially localized using a technique similar to one-dimensional border shift or ORB. However it also costs $O(D)$ communication time in the worst case because the members of a particle subset can be distributed in a region almost as large as the space domain.

Thus our OhHelp is an almost unique proposal to achieve $\bar{P} = \bar{P}$ and $\bar{D} = 2\bar{D}$ with $O(\bar{P} + \bar{D})$ time, if we can neglect the $O(N)$ factor for histogram communication and can assume possible $O(\bar{P} \log N)$ time for all-reduce and $O(\bar{D}^{2/3} \log N)$ time for boundary broadcast as $O(\bar{D})$. To the best of our knowledge, an algorithm achieving similar level of scalability is solely found in QUICKSILVER\cite{21, 25} in which a helpand-helpers scheme similar to ours is established. However, since they shift the particles in a heavily loaded subdomain only to lightly loaded nodes in their earlier proposal\cite{25}, a severe load imbalance $\bar{P} \approx 2\bar{P}$ will occur as discussed in Section 2.1. This drawback is solved in their latter proposal\cite{21} by enabling a helper to help two or more nodes but this fails to satisfy $\bar{D} = k_{\bar{d}} \bar{D}$ because the number of helpands for a helper cannot be bounded to a constant in general.

The discussion in this section is summarized in Table 3.

6. CONCLUSION

In this paper, we proposed a new method for PIC simulations, named OhHelp, to achieve both good load balancing and scalability. The OhHelp method simply and equally partitions the space domain for scalability with respect to domain size. It also copes with the imbalance of the num-
ber of particles in subdomains by making each node help another node to which a densely populated subdomain is assigned. Our implementation of a three-dimensional space plasma simulation with OhHelp exhibits a good scalability showing 150–190 speedup with 256 processes compared to the sequential execution of a reference particle-decomposed simulator whose speedup is saturated at about 35-fold.

Table 3: Comparison of PIC Simulation Methods with Respect to Worst Case Per-Node Particle Population and Subdomain Size

<table>
<thead>
<tr>
<th>method</th>
<th>$P$</th>
<th>$D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>particle decomposition</td>
<td>$P/N$</td>
<td>$D$</td>
</tr>
<tr>
<td>static domain decomposition</td>
<td>$P$</td>
<td>$D/N$</td>
</tr>
<tr>
<td>dynamic domain decomposition</td>
<td>$P/N$</td>
<td>$D$</td>
</tr>
<tr>
<td>QUICKSILVER[25]</td>
<td>$2P/N$</td>
<td>$2D/N$</td>
</tr>
<tr>
<td>OhHelp</td>
<td>$P/N$</td>
<td>$2D/N$</td>
</tr>
</tbody>
</table>
We are planning to apply OhHelp to other PIC simulations. One candidate is a plasma simulation whose space domain has a mass of conductor such as an antenna attached to a spacecraft which is exposed to solar-originated space plasma. Since the existence of a conductor requires to solve Poisson’s equation in each time step, this simulation will bring another scalability problem to us. Although our preliminary evaluation with three-dimensional parallel FFT shows that the performance impact to 256-process execution is less than 3%, we should need more scalable algorithms for larger scale parallel executions.

The other target is a simulation involving particle-particle interactions such as so-called Monte Carlo collision[19]. To simulate particle collisions stochastically, we need a little bit more careful particle deputation from a helpand to its helpers because, for example, all particles in a grid cell must be given to a helper as a whole or forming a subset having population large enough to satisfy the law of large numbers. Although this should require more precise histograms of the number of particles taking the grid-level particle residence into account, its implementation will be in the range of natural extensions of OhHelp.

7. ACKNOWLEDGMENTS

This research work is partly supported by Grant-in-Aid of Scientific Research #20300011 of MEXT Japan.

8. REFERENCES


