Parallelizing and Balancing Coupled DSMC/PIC for Large-scale Particle Simulations

Haozhong Qiu, Chuanfu Xu
College of Computer
National University of Defense Technology
{qiuhaozhong16, xuchuanfu}@nudt.edu.cn

Dali Li, Haoyu Wang, Jie Li
College of Aerospace Science and Engineer
National University of Defense Technology
{achhx, wanghaoyu15}@nudt.edu.cn, lijie_gfkd@163.com

Zheng Wang
School of Computing
University of Leeds
z.wang5@leeds.ac.uk

2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS) | 978-1-6654-8106-9/22/$31.00 ©2022 IEEE | DOI: 10.1109/IPDPS53621.2022.00045

***Abstract*—In high-performance and parallel computing, an important application class is particle simulation. Due to massive particle migration among distributed simulation workers across simulation iterations, achieving balanced runtime work distribution is vital for accelerating large-scale realistic particle simulations. This paper proposes a novel approach to enable dynamic load balance for distributed numerical particle simulations, specifically targeting the latest coupled DSMC/PIC method. Unlike prior work, our approach adopts a dual, nested unstructured grid organization to facilitate coupled DSMC/PIC computation and runtime grid distribution. Our implementation leverages both centralized and distributed communication strategies to dynamically migrate particles among arbitrary parallel processes. It then employs a load balancer - driven by a carefully designed analytical model and a grid remapping mechanism - to dynamically redistribute the simulation workloads among parallel simulation workers. By constantly monitoring and redistributing the simulation work across workers, our approach can adapt to the change of particle distribution across simulation iterations, avoiding a few workers becoming the performance bottleneck of the entire simulation process. We integrate our techniques into a coupled DSMC/PIC solver and apply them to simulate the plasma plume with hydrogen atoms and ions. Experimental results show that our approach can scale well up to 1500+ processes with billions of particles, exhibiting the state-of-the-art parallel simulation scalability and efficiency for plasma plume simulation.**

***Index Terms*—Coupled DSMC/PIC, Particle simulation, Dynamic load balance**

I. Introduction

Particle simulation is widely used to analyze the physical phenomena in many scientific and engineering subjects. It is a crucial application class on high-performance computing (HPC) systems. The plasma plume[1] is a typical particle simulation application and is key to enable research in fields like aerospace propulsion [1]–[3], industrial coatings [4], [5], and nuclear fusion reactors [6], [7].

The plasma plume induced by the pulsed vacuum arc has important applications in high-stability plasma devices. It usually flows in the millimeter range, incurring a series of complex thermochemical non-equilibrium reactions and wall interactions within microseconds. Because of the high number of particles evaporated and the high particle density (e.g., $10^{18}/m^3$ to $10^{22}/m^3$), the corresponding plasma plume is very rarefied and the Knudsen number is greater than 0.1. Under such a setup, traditional Computational Fluid Dynamics (CFD) methods based on the Navier-Stokes equations are ill-suit for plasma plume simulation. As a result, the flow problem of plasma plume typically needs to be solved based on the Boltzmann equations [8].

The state-of-the-art approach for simulating plasma plumes combines Direct Simulation Monte Carlo (DSMC) [9] and Particle in Cell (PIC) [10], [11] methods. DSMC is proven to be effective in solving problems at the micro-level for rarefied gas flows, such as simulating the movement and collision between particles. Using the classical Bird's algorithm [8], DSMC can achieve consistent results as the Boltzmann equations. PIC is useful for tracking a large volume of particles and their interactions. By coupling DSMC and PIC, one can model a large number of particles and their behaviors at the micro-level. The combination allows the simulation system to effectively model complex physical phenomena in the plasma plume induced by a pulse vacuum arc.

Although coupled DSMC/PIC can achieve accurate particle simulation, this strategy requires more memory and computation resources than traditional CFD simulation techniques [12], [13]. One of the practical challenges in applying coupled DSMC/PIC in a distributed computing environment is to achieve dynamic load balance across simulation iterations. As we iterate the simulation timesteps, particles can move and may cross their grid cells. As such, the simulation algorithm must re-decompose the grid and then redistribute simulation particles between parallel processes after a certain number of timestep iterations. Without a proper load balancing strategy, the simulation workload can be unevenly distributed among parallel processes, where some are waiting for others to complete while others can be overloaded with too many particles. To improve the performance of large-scale particle simulations, we need to find ways to dynamically rebalance the simulation loads among parallel processes across timestep iterations.

This paper aims to provide a better approach for applying coupled DSMC/PIC to large-scale numerical particle simulations. To have a concrete application context, we choose plasma plume simulations as a case study but our approach can generalize to many other particle simulation workloads.

[1]The *plasma plume* includes steam, plasma, molecular clusters, and surface debris. The steam and plasma are jetted back at ultra-high speeds of tens of kilometres per second.

1530-2075/22/$31.00 ©2022 IEEE
DOI 10.1109/IPDPS53621.2022.00045

Our approach offers several new optimizations for coupled DSMC/PIC simulations. These include a novel grid-based simulation scheme, a load imbalance indicator, a weighted load model to capture the importance of both particles and grid calculations, and the use of the Kuhn-Munkres (KM) algorithm [14], [15] to reduce the overhead of grid remapping.

Unlike prior work that employs two *structured* grids to implement the coupled DSMC/PIC algorithm [2], our approach exploits two *unstructured* grids to facilitate the coupled calculation and parallelization. Unstructured grids permit closer to real-world simulation for many applications with complex geometry configurations. Specifically, our unstructured grid scheme incorporates a coarse and a fine tetrahedral grid. In the coarse tetrahedral grid, we constraint the cell size by the mean free path[2] of particles, and perform DSMC simulations of flow fields. In the fine tetrahedral grid, we constraint the cell size by the Debye length[3], and perform PIC simulations of electric fields. The fine tetrahedral grid is entirely nested in the coarse grid (see Fig. 2 for an example). As a result, we only need to distribute the coarse grid across parallel processes where each process can perform both DSMC and PIC simulations within different grids. Coupled DSMC/PIC along with the combination of two unstructured grids enable us to simulate a wider range of simulation setups involving complex phenomena and realistic geometries, compared to existing approaches [9]–[11].

Our work demonstrates the benefits of alternative implementations for a coupled DSMC/PIC solver. Unlike prior work that solely employs a distributed communication scheme, we offer both centralized and distributed communication schemes based on MPI. We empirically demonstrate that the centralized communication strategy can outperform the distributed counterpart in certain simulation setups (e.g., with fewer simulated particles). A centralized scheme also incurs a low memory footprint, making it suitable for simulating large problems.

Armed with our grid strategy and communication schemes, we implement a dynamic load balancer to adaptively redistribute workloads across MPI processes across simulation iterations. We propose a load imbalance indicator to quantify the degree of imbalance, and dynamically evaluate the imbalance across iterations. If the quantified load imbalance is greater than a pre-defined threshold, we adopt a weighted load model to guide the re-decomposition of the coarse grid and redistribute workloads (i.e., grid cells and particles) to MPI ranks. We employ the KM algorithm to reduce the overhead of work redistribution. We note that our work is the first to use the KM algorithm for dynamic work distribution in coupled DSMC/PIC.

We evaluate our coupled solver by applying it to simulate unsteady plasma plume with particles of *hydrogen atoms* (H) and *ions* (H^+) induced by a pulsed vacuum arc in a 3D cylindrical nozzle. We test the performance of our approach on three HPC platforms, including two distinct Intel CPU-based clusters and an ARM CPU-based system. We showcase that our coupled DSMC/PIC solver delivers state-of-the-art plasma plume simulation performance, scaling well to 1536 processes with billions of particles.

This paper makes the following contributions:

- It is the first work to employ two unstructured grids for coupled DSMC/PIC simulations (Section IV);
- It presents a novel work scheduling scheme for particle simulations by employing a weighted load-balance model and the KM algorithm (Section V);
- It provides quantified analysis, showing the benefits and trade-offs of distributed and centralized communications in distributed particle simulations (Section VII-D).

II. Related Work

Several studies have shown the efficiency of parallelizing coupled DSMC/PIC solvers. Aleph [16] is a coupled DSMC/PIC solver for simulating low-temperature plasma. It can simulate 3D models using unstructured tetrahedral grids [17], [18]. Other works [19] use the parallel coupled PIC/DSMC algorithm to simulate the active plasma flow and chemical reactions in a rarefied condition. The work presented in [20] implemented a parallel high-order DSMC/PIC solver using MPI to simulate the 250ps diffusion process of a laser-driven plasma plume. The solver is based on 3D unstructured hexahedral grids, and the electromagnetic field is discretized using the Discontinuous Galerkin Method [21]. The SUGAR (Scalable Unstructured Gas dynamics with Adaptive mesh Refinement) software [22] uses MPI for parallel computing. SUGAR can simulate the ion thruster plume, including MEX (Momentum Exchange) collision and CEX (Charge Exchange) collision between neutral particles and charged particles. A GPU-based DSMC/PIC solver was presented in [23] using MPI+CUDA. This approach uses an octree to divide the computational domain and perform load balance according to particle numbers during simulations.

The aforementioned parallel implementations of coupled DSMC/PIC solvers mainly focus on the plasma plume generated by the electric thruster. Most of them only target the generation of plasma rather than the process of diffusion. Our work extends the coupled DSMC/PIC boundary to simulate the diffusion, collision, and reaction within a single framework. Due to dynamically changing simulation behaviour resulting from the complex simulation models, the parallelization method must be able to deal with the communications across arbitrary parallel processes. This challenge makes prior work infeasible to the simulation scenario we target. Our work addresses this challenge by implementing two communication strategies to migrate particles among arbitrary parallel processes. We then employ a dynamic load balancer to scale up the simulation performance to thousands of cores - which is the largest simulation setup seen to date for coupled DSMC/PIC. The scalability offered by our approach represents a significant improvement over the hundreds of cores scalability offered by

[2]The *mean free path* is the average distance over which a moving particle (e.g., an atom, molecule, or photon) substantially changes its direction or energy, as a result of one or more successive collisions with other particles.

[3]In plasma, the *Debye length* is a measure of a charge carrier's net electrostatic effect in a solution and how far its electrostatic effect persists.

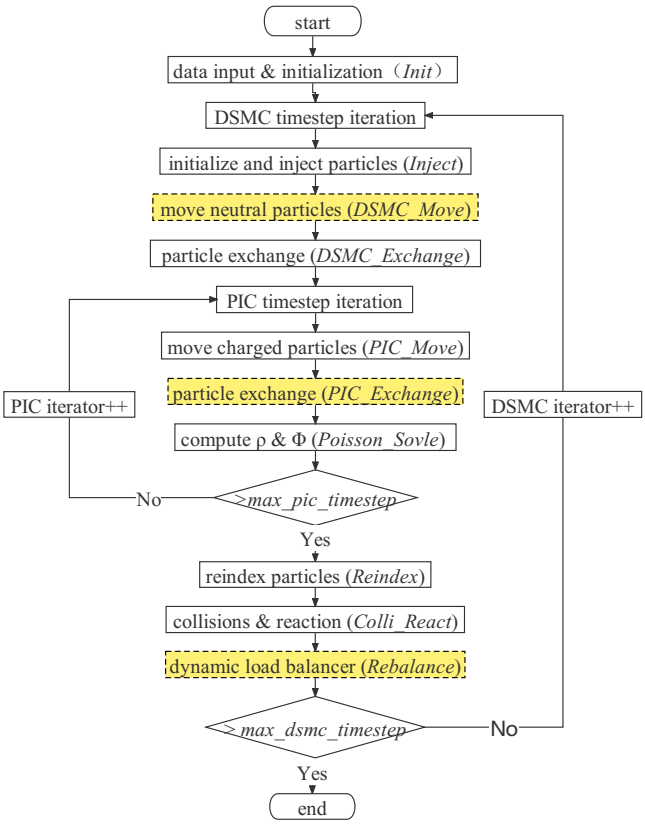


Fig. 1. The workflow of our coupled DSMC/PIC solver. The dotted rectangles highlight the new contributions introduced in this paper.

existing coupled DSMC/PIC. On top of these, we also provide detailed sensitivity analysis on our design choices, which we hope will be useful for further studies in parallel DSMC/PIC solutions.

III. Coupled DSMC/PIC Solver

A. Infrastructures

Our work builds upon an in-house DSMC solver developed based on the seminal algorithm presented in [8]. This DSMC solver implements various models to simulate the processes of collisions, wall interactions and chemical reactions of particles in the flow fields. After the DSMC solver have been used and validated for several years, a PIC solver is introduced to simulate the movement of charged particles according to the effects of electric fields. The two components form the coupled DSMC/PIC solver used in this work. Our prior works [24], [25] have provided compelling evidence, showing the effectiveness and correctness of this coupled solver.

B. Simulation Workflow

Fig. 1 give the overall workflow of the coupled DSMC/PIC solver with the highlighted new extensions introduced in this work. Inputs to the solver are two unstructured grids and simulation parameters. These are passed into the initialization component (*Init*) to instantiate the internal data structures. The solver then iterates through the DSMC and PIC simulation steps, detailed as follows.

DSMC timesteps. During a DSMC timestep, the *Inject* component initializes and injects simulation particles into the inlet randomly. The injection should ensure the velocity is perpendicular to the inlet and complies with the physical law (i.e., the Maxwell distribution in our case). Next, the *DSMC_Move* component starts simulating the movement of neutral particles. In each DSMC timestep, neutral particles will move straightly at a constant speed and can cross different grid cells or even move out of the computational domain.

PIC timesteps. A DSMC timestep typically contains multiple PIC timesteps, and the size of DSMC or PIC timesteps depends on the mean free path of particles [26]. In each PIC timestep, the *PIC_Move* component simulates the movement of the charged particles in the electric field. At the current timestep, the charged particles are driven by the electric field of the previous timestep. A key task of a PIC timestep is to determine the charged particle's velocity and position using the *Poisson_Solve* component. We elaborate on the *Poisson_Solve* component in Section III-C.

Particle numbering. The particle locations are likely to change during iterations. The location change can cause the change of particle distributions, which requires the solver to renumber all particles in a uniform way to ensure each of them has a unique index number. This is performed by the *Reindex* component, which will also remove particles that moved out of the computational domain after a DSMC iteration.

Collision and reaction. Unlike prior work that mainly focuses on the generation of plasma, our DSMC solver also implements various collision and chemical reaction models [25] in the *Colli_React* component. The solver adopts Bird's no time counter (NTC) method [27] to select collision pairs. After selecting the collision pairs, it determines whether the two particles will have a chemical reaction according to the particle types and the energy of the collision. The solver then performs chemical reactions for those particles.

Our extensions. The main contributions of this paper are highlighted using dotted rectangles in Fig. 1. Specifically, the *PIC_Exchange* and *DSMC_Exchange* components are responsible for two parallel communication strategies at the PIC and DSMC iterations, respectively. The communication strategies are described in Section IV-B. The *Rebalance* component is our dynamic load balancer, described in Section V.

C. Determining the Particle Velocity and Position with PIC

The velocity, v, and position, r, of particles can be acquired by solving the following kinetic equation [25] for plasma physics during a PIC timestep, t:

$$\begin{cases} \dfrac{\mathrm{d}r}{\mathrm{d}t} = v \\ m\dfrac{\mathrm{d}v}{\mathrm{d}t} = q(E + v \times B) \end{cases} \tag{1}$$

where m is the mass of the particles (given by the user), q is the charge for particles, B is the magnetic density and E is the electric field intensity. In this paper, we only consider the electrostatic field by assuming that there is no magnetic field (i.e., $B = 0$) or a constant magnetic field (i.e., B is a constant number given by the user). The solution of the kinetic equation determines the distribution function of the dynamical states of each individual particle.

We determine the relevant parameters of the kinetic equation as follows.

Determine v and q. We use the Boris method [28] to calculate the numerical value of the velocity v. We then compute the charge density q, by interpolating the particle charge to the grid nodes, following the movement of charged particles.

Determine E. We compute E by solving Poisson's equation for electrostatics:

$$\nabla \cdot E = \frac{\rho}{\epsilon_0} \tag{2}$$

where ρ is a total volume charge density, ϵ_0 is the permittivity[4] of the medium, and ∇ is the Laplace operator. Here, the electric field intensity E, can be expressed as a negative gradient of the electric potential ϕ:

$$E = -\nabla\phi, \tag{3}$$

With (3), we convert Poisson's equation in (2) to the following formula, by substituting E with $-\nabla\phi$:

$$\nabla \cdot E = \nabla \cdot (-\nabla\phi) = -\nabla^2\phi = \frac{\rho}{\epsilon_0} \tag{4}$$

We then solve (4) by discretizing it using the finite volume method [29] on unstructured grids to construct the following linear system:

$$K\phi = b \tag{5}$$

where K is the diagonally-dominant stiffness matrix constructed according to the grid topology, b is derived from boundary conditions and the electric potential. We use the linear system to derive ϕ, which is then used to compute E through (3). This process is performed by the *Poisson_Solve* component in Fig. 1. We note that solving (5) is a time-consuming process, and hence can benefit from parallel computing. In Section IV-C, we describe our parallelization strategies for solving the linear system.

IV. Parallel Solver Implementation

A. Grid Generation and Decomposition

Our coupled solver is designed and optimized for unstructured grids, focusing on tetrahedral grids in this paper. Under such a setting, each particle belongs to a grid cell, and the cell size is constrained by different type of particles. When applying DSMC to neutral particles, it should meet the mean free path limit of the particles and comply with the Debye length when applying PIC to charged particles for solving the electric potential (see Section III-C). To meet the above constraints, our coupled DSMC/PIC solver first generates a coarse tetrahedral grid for DSMC. It then divides each coarse-grained grid cell into 8 smaller grid cells to form a fine grid for PIC. As shown in Fig. 2, this is realized by halving each edge of the tetrahedron. The fine grid is embedded in the coarse grid to facilitate the coupled computation and grid decomposition for parallel computing. Since each fine-grained cell belongs to a coarse-grained cell, we only need to decompose the coarse grid for parallel computing. We then determine the mapping of fine-grained cells to parallel processes accordingly.

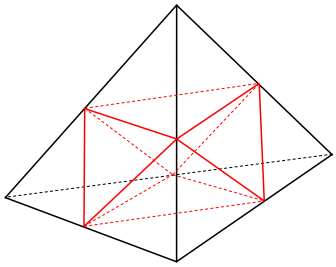


Fig. 2. A coarse tetrahedral DSMC grid cell with 8 fine-grained tetrahedral grid cells for PIC.

We make a use of the graph partition method (*METIS_PartGraphKway*) from the METIS graph and mesh partitioning library [30] to decompose the grid. The end-user can supply a weight array to represent loads of grid cells and achieve specific load balance requirements for their applications. For the first decomposition, we do not provide the weight array to METIS. For the re-decomposition operations in our dynamic load balancer, we derive a load model from calculating a weight for each cell (see also Section V). After grid decomposition, each parallel process can perform simulation independently on its grid cells (and the associated particles).

B. Parallel Communication Strategies

For both PIC and DSMC simulations, the particles may move from one grid cell to another. Since the migration distances of particles in our coupled algorithm can be long, the destination cell of a moving particle may belong to a domain that is far from the original domain. For this reason, the solver must be able to handle the communications between arbitrary parallel processes. Unfortunately, the ghost cell immersed boundary method, which is often used in traditional CFD [31], is not feasible for our problem because it can only deal with the communication between the neighboring parallel processes. To this end, we propose two communication strategies to handle particle migration among arbitrary processes. The two strategies are both implemented in *DSMC_Exchange* and *PIC_Exchange* components in Fig. 1.

1) The centralized communication strategy: Fig. 3 gives a simple illustration of the centralized communication strategy. Here, a centralized process (e.g., rank 0 in Fig. 3) is selected to manage the whole procedure of particle migration. The procedure consists of three stages: *gather*, *classify* and *scatter*.

[4]The permittivity is a measure of the electric polarizability of a dielectric.

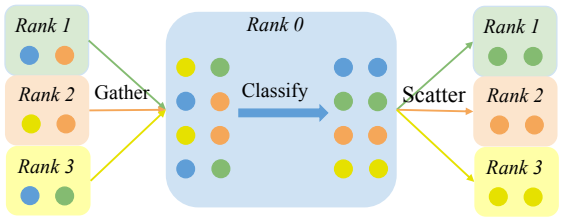


Fig. 3. An simple illustration of 4 processes using the centralized communication strategy. Rank 0 is selected as the centralized process. Lines of different colors represent communications among different processes. Circles of different colors represent particles to be moved to different destinations.

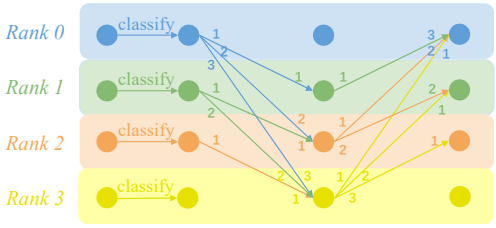


Fig. 4. An simple illustration of 4 processes using the distributed communication strategy. Lines of different colors represent communications among different processes. Circles of different colors represent particles to be moved to different destinations. Numbers on lines represent the communication (sending or receiving) orders.

Each process sends the particles that will move to another process to the centralized process during the gathering stage. After gathering all migrating particles, the centralized process will classify them according to their destination processes. The particles, which may come from different source processes but move to the same destination process, will be packed together. Finally, the centralized process scatters the packed particles to the corresponding processes.

2) The distributed communication strategy: As illustrated in Fig. 4, with a distributed communication strategy, each process classifies and packages its particles that will move out its domain according to their destination processes. Then each process will perform a two-round synchronized *MPI_send/MPI_recv* operation to exchange particles. In the first round, each process will first receive particles from processes with a rank less than itself. Then, each process will send particles to processes with a rank greater than itself. In the second round, each process will first receive particles from the processes with a rank greater than itself. Then, each process will send particles to processes with a rank smaller than itself. An implementation trick to avoid the communication deadlock should be noted that when sending particles, a process should first send to the processes with smaller ranks; when receiving particles, a process should first obtain from the processes with larger ranks.

3) Efficiency analysis: We now perform a theoretical analysis on the performance of the two strategies. Suppose we have M particles to migrate among N parallel processes, and each process has particles moving in and moving out. Under

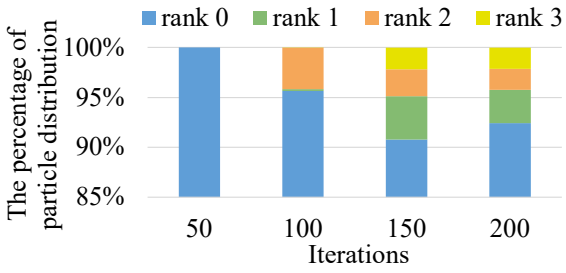


Fig. 5. The percentage of particle distribution when using 4 MPI processes without load balance. Rank 0 is overloaded as it has to simulate 90+% of the particles, highlighting the importance of having a load balancing mechanism.

this setting, the number of communication transactions will be $2N$, and the data transfer size is proportional to $2M$ using the centralized strategy. For the distributed strategy, the number of communication transactions will be around $N(N-1)$, and the data transfer size is approximately proportional to M. A centralized strategy has fewer communication transactions but will incur a higher data transfer size over the centralized strategy. In contrast, a distributed approach will result in a higher number of communication transactions, but it has the benefit of sending fewer data over the network compared to the distributed scheme. As a result, there is not a single winner, and optimal strategy may change depending on the computing and communication capability of the underlying hardware and the simulation setups. Later in Section VII-D, we empirically show that both strategies have their merits.

C. Parallel Solution of Poisson's Equation in PIC

As we have discussed in Section III-C, assembling and solving Poisson's equation is the most time-consuming procedure in PIC. For each process, the charge density of the inner grid nodes can be calculated based on the number of charged particles in their local domain. For boundary nodes belonging to multiple parallel processes, their charge density should be the sum of the charge densities from all neighboring processes. Thus, we need to first apply reduction summation to the charge density of boundary nodes.

After obtaining the charge density, we need to solve Poisson's equation to obtain the electric potential on grid nodes. Directly solving the nonlinear Poisson equation is a complicated and time-consuming process. We take a two-step iterative approach to implement a fast solution to solve Poisson's equation. To this end, we first transform Poisson's equation into a linear system like (5), where K is a global matrix. Then, we use the parallel Krylov subspace (KSP) iterative method [32] from the PETSc library [33] to solve this linear equation. To reduce the memory footprint, we use the Compressed Sparse Row (CSR) format to store the sparse matrix K.

V. Dynamic Load Balance

As we have outlined earlier, particles dynamically migrating between arbitrary processes may cause severe load imbalance,

Algorithm 1 The rebalance algorithm

Input: $cellnum$, $procsnum$, T, $Treshold$, $OriginalMapping$
Output: $FinalMapping$
1: compute *lii* according to (6)
2: $iterator \leftarrow iterator + 1$
3: **if** $iterator < T \parallel lii < Threshold$ **then**
4: **return**
5: **end if**
6: **for** $i = 0 \rightarrow cellnum$ **do**
7: compute wlm_i according to (7)
8: $wlm \leftarrow wlm_i$
9: **end for**
10: $NewPartition \leftarrow$ *METIS_PartGraphKway*($cellnum$, $procsnum$,wlm)
11: $FinalMapping \leftarrow$ *Kuhn_Munkras*($OriginMapping$, $NewPartition$)
12: **return** $FinalMapping$

leading to performance degradation in parallel DSMC/PIC simulations. As we will show later in Section VI, this load imbalance problem can lead to significant performance loss and must be carefully dealt with. Since target unsteady phenomena with particles injected from the inlet in each timestep, the initial distribution of particles is highly uneven among grid cells. As a concrete example, Fig. 5 shows the percentage of particle distribution among 4 parallel processes after 200 PIC timesteps of simulation with no load balance mechanism. For the first 50 timesteps, almost all particles belong to MPI rank 0. This uneven work distribution overloads the rank 0 process while leaving other MPI processes idle, wasting the computation cycle. We also observe that this imbalance issue does not improve over simulation iterations – around 90% of the particles still belong to rank 0 after 200 timesteps.

In light of the above observation, we design and implement a dynamic load balancer in the coupled solver. Algorithm 1 outlines the working mechanism of the dynamic load balancer. Our load balance algorithm quantifies the load imbalance using an analytical model (Section V-A). It then uses a weighted load model to decompose the grids across parallel workers (Section V-B). During runtime, the load balancer periodically checks the imbalance indicator (*lii*) for a user-specified number of timestep iterations (i.e., T). If a $Threshold$ is reached, it then re-decomposes the grid according to the load model and remaps the newly partitioned grid to parallel processes. The algorithm was implemented in the *Rebalance* component in the DSMC iterations as shown in Fig. 1.

A. The Load Imbalance Indicator

Our load imbalance indicator (lii) measures the execution times of different parallel processes and compares the maximum and minimum values. It is formulated as:

$$lii = \frac{Time_{max\_total} - Time_{max\_pm} - Time_{max\_poi}}{Time_{min\_total} - Time_{min\_pm} - Time_{min\_poi}} \quad (6)$$

where subscripts "max" and "min" denote the corresponding processes with the measured maximum and minimum execution times respectively. $Time_{max\_total}$ / $Time_{min\_total}$ is the total execution time, $Time_{max\_pm}$ / $Time_{min\_pm}$ and $Time_{max\_poi}$ / $Time_{min\_poi}$ are the time cost of particle migration and solving the Poisson's equation respectively. Note that the time spending on particle migration (*DSMC_Exchange* and *PIC_Exchange*) and solving Poisson's equation (*Poisson_Solve*) are largely constant.

We use lii to evaluate the load imbalance for a configurable number of DSMC iterations T (defined by the user). We set a load imbalance $Threshold$, and check if lii is larger than the $Threshold$ for T DSMC iterations. If $lii > Threshold$ holds, we will perform the dynamic load balance optimization. T and $Threshold$ can be selected according to specific simulation setups runs using an auto-tuning technique [34].

B. The Weighted Load Model

To initialize the simulation, we use METIS to decompose the grid for parallel computing solely according to the number of grid cells. Our dynamic load balancer may decide to re-decompose the grid during runtime to balance the number of particles simulated by MPI processes. Our optimization strategy also needs to consider the requirement of the coupled computation: DSMC only simulates the neutral particles, and PIC only deals with the charged particle, and a DSMC timestep may contain multiple PIC timesteps. Because of these constraints, we should give different weights to neutral and charged particles when estimating the work for grid decomposition. For example, if each DSMC timestep contains 2 PIC timesteps, then the weight ratio R of charged particles and neutral particles should be 2 because there are twice more charged particles to be simulated than the natural particles within a DSMC step. Besides particles, some computations (e.g., *Colli_React* and *Poisson_Solve* in Fig. 1) are performed on grid cells, and each grid cell should also have its weight. Our design takes these constraints into consideration.

Specifically, our weighted load model (wlm) for the i^{th} grid cell wlm_i is defined as follows:

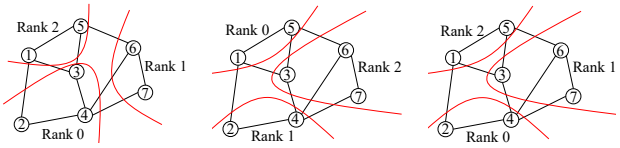
$$wlm_i = N_i + RC_i + W_{cell} \quad (7)$$

where N_i and C_i represents the number of neutral particles and charged particles in the i^{th} grid cell respectively and W_{cell} represents the weight of the grid cell, taking the weight of a neutral particle as the baseline.

With (7) in place, we can calculate the combined weight for each grid cell and then provide the weight to *METIS_PartGraphKway* in *METIS* to re-decompose the grid for parallel computing in the subsequent timesteps.

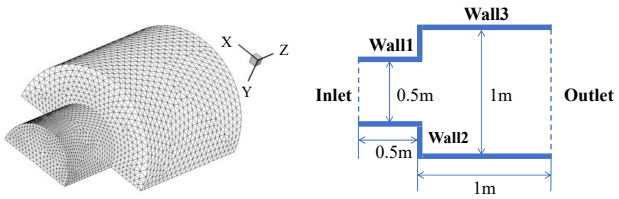
C. Efficient Grid Remapping

After grid re-decomposition, we need to redistribute the grid cells to parallel processes for subsequent simulations. This is also involved migrating particles within a process, or from grid cells to other processes. A naïve approach by randomly remapping grid cells can incur significant overhead and reduce the efficacy of our dynamic load balancer. For the example shown in Fig. 6, the mapping given in Fig. 6(c) is better than the one in Fig. 6(b). Fig. 6(c) requires moving just particles in cell 3 - from rank 0 in Fig. 6(a) to rank 1 in Fig. 6(c). In contrast, Fig. 6(b) requires moving all particles in every cell across MPI ranks - e.g., cells 2 and 4 are moved from rank 0 in Fig. 6(a) to rank 1 in Fig. 6(b) and similarly for other cells.



(a) Original (b) Random (c) Optimized

Fig. 6. An example of different grid mapping strategies. Fig. 6(a) is the original grid decomposition and mapping where the circle represents a grid cell. Fig. 6(b) is a possible random grid remapping after re-decomposition. Fig. 6(c) is an optimized remapping given by our approach.



(a) Grids of the nozzle (b) Simulation setup

Fig. 7. The grids of the 3D cylindrical nozzle and the simulation setup.

Due to the extensive work redistribution across computing nodes, Fig. 6(b) will lead to a higher communication overhead than Fig. 6(c).

This observation suggests that a good remapping strategy should try to remap the re-decomposed cells according to the original mapping. It should also minimize the total number of particles to be migrated during re-decomposition. Therefore, our remapping scheme aims to achieve these goals. To solve the remapping problem, we convert the grid remapping problem into the maximum weight matching problem in a bipartite graph. We then use the classical KM algorithm [14], [15] to solve the maximum weight matching problem. For the example shown in Fig. 6, our KM-based approach gives the optimal solution shown in Fig. 6(c).

VI. Experimental Setup

A. Hardware Platforms

We evaluate our approach on three HPC systems. These include two x86 HPC from the Tianhe-2 supercomputer [35] and the Beijing Beilong Super Cloud Computing (BSCC) infrastructure, and an ARMv8 based system from the Tianhe-3 exascale prototype supercomputer [36]. We run our solver on the CPU host solely as our implementation currently does not support accelerators.

The Tianhe-2 supercomputer. This is our main evaluation platform. Each Tianhe-2 compute node has two 12-core Intel Xeon E5-2692 v2 processors at 2.2GHz and 64GB of RAM. Compute nodes are connected through an in-house developed high-speed interconnection network, with a point-to-point bandwidth of 160 Gbps. Tianhe-2 implements a fat-tree communication topology and runs the Kylin OS with Linux kernel v2.6.32.

The BSCC supercomputer. Each BSCC compute node contains two 48-core Intel Xeon Platinum 9242 CPUs at 2.3 GHz and 384 GB of RAM. Compute nodes are connected via the InfiniBand, with a point-to-point bandwidth of 100 Gbps. This system runs CentOS with Linux kernel v3.10.0.

The ARM-based Tianhe-3 prototype. Each compute node of the Tianhe-3 prototype has a 64-core ARMv8-based Phytium 2000+ CPU at 2.2 GHz and 64GB of RAM. Compute nodes are connected through an in-house developed high-speed interconnection network, with a point-to-point bandwidth of 200 Gbps. This system runs the Kylin OS with Linux kernel v4.4.0.

B. Software Implementations

Our coupled DSMC/PIC solver was implemented in C and compiled with GCC v11.2 with the "-O3" compiler option. We use MPICH v3.4.2 for parallel MPI communication, the KSP solver from PETSc v3.15.0 to solve Poisson's equation and METIS v5.1.0 for domain decomposition.

C. Simulation Setup

In this paper, we simulate the diffusion, collision and reaction of the plasma plume induced by the pulsed vacuum arc in a 3D cylindrical nozzle. Fig. 7 depicts the grids and simulation setup of the nozzle. We use the SALOME numerical simulation platform [37] to generate the tetrahedral grids for the cylindrical nozzle.

As listed in Table I, we used six datasets in our evaluation. We use *Dataset* 1 to validate our implementation and the other five different datasets for performance tests. The solver converts the number density[5] of realistic particles to the corresponding density of simulation particles using a scaling factor given in Table I. The scaling factor indicates the number of real particles to be represented by a simulation particle. By changing the scaling factors, we can easily vary the number of simulation particles and the corresponding computing load for the same number density of realistic particles. Once the grid is determined, we can then work out the timesteps of DSMC and PIC as well as the number density of particles accordingly.

In this work, we are mainly concerned about the dissociation of H and the recombination of H^+. We set the particle velocity to $10000m/s$ and the wall temperature to $300K$. As the setting of particle density and the timestep size depend on the cell size, we will provide this information when discussing the relevant experiments. We run each simulation for 100 DSMC timesteps, where each DSMC timestep contains 2 PIC timesteps. We run each simulation setup 5 times and report the mean execution time across runs. We note that our evaluation setups represent real-life simulation setups.

VII. Experimental Results

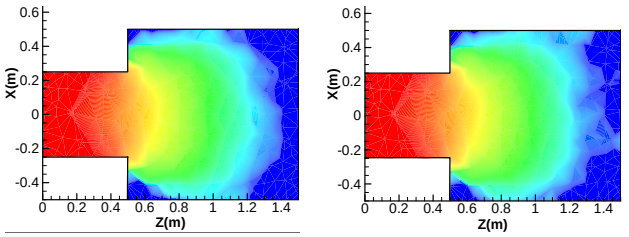
A. Validation of the Parallel Implementation

Setup. We use *Dataset 1* in Table I to validate our parallel implementation of the coupled DSMC/PIC solver. We set the number density of H to 7×10^{18} and the density of H^+ to

[5]The number density quantifies the degree of concentration of countable objects (particles, molecules, phonons, etc.) in physical space.

TABLE I
THE PIC GRID CELL NUMBERS AND SCALING FACTORS OF DIFFERENT DATASETS USED IN VALIDATION AND PERFORMANCE TESTS.

	#PIC Cells	*Scaling Factor* (H)	*Scaling Factor* (H^+)
Dataset 1	55,576	1.000×10^{12}	6,000
Dataset 2	583,386	9.940×10^{10}	0.477
Dataset 3	583,386	9.940×10^{11}	4.77
Dataset 4	583,386	1.988×10^{11}	0.954
Dataset 5	2,242,948	1.400×10^{11}	12,500
Dataset 6	2,242,948	2.800×10^{11}	25,000



(a) Serial simulation results. (b) Parallel simulation results.

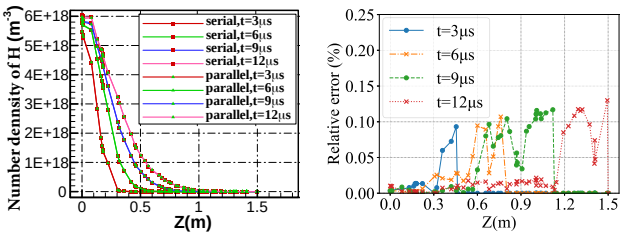
Fig. 8. The number density contours of H after 200 PIC simulation steps produced by a serial (a) and our parallel (b) implementation of the solver.

3×10^8. The timestep sizes of DSMC and PIC are $2 \times 10^{-7}s$ and $1 \times 10^{-7}s$ respectively. The total simulation time is $20\mu s$.

Results. Fig. 8 shows the number density contours of H after 200 PIC timesteps of simulations from our parallel version and a serial version of the solver which was validated in prior work [25]. We observe some minor differences in the contours, and this is mainly due to random seeds used for simulations. To have a closer examination, we further measured the number density of H at several selected points on the central axis of the cylinder. Fig. 9(a) shows the number density of H on the central axis given by the serial and the parallel simulations when time t is $3\mu s$, $6\mu s$, $9\mu s$ and $12\mu s$. As can be seen from this diagram, curves at the same time point coincide, indicating the consistency of parallel and serial runs. To evaluate the differences, we calculate the relative errors of the number density of H on the central axis. As shown in Fig. 9(b), the mean relative errors of four time points t on the central axis are all less than 2.97%. The relative errors become larger when the number density is close to 0. This is because the number density doesn't converge in the marginal area, and it is easier to be affected by the random seeds. The relative standard deviation of 5 runs is less than 5%.

B. Performance Results

Setup. We use *Dataset 2* in Table I to evaluate the strong scalability of the proposed approach on the Tianhe-2 supercomputer. According to the physical constraints of DSMC and PIC models, we set the number density of H and H^+ to 9.94×10^{19} and 4.77×10^7 respectively. In this dataset, the scaling factor of the DSMC and PIC is 9.94×10^{10} and 0.477 respectively, which means we need to deal with 10^9



(a) The number density of H (b) Relative errors

Fig. 9. The number density of H (a) and its relative errors (b) on the central axis at selected time points for the serial and parallel runs.

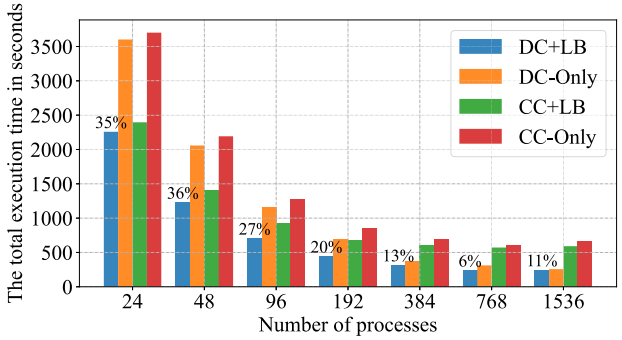


Fig. 10. The total execution times of different implementations scaling up to 1536 processes. The percentages on the bar indicate the performance enhancement when dynamic load balance is enabled

simulation particles of H for DSMC, and 10^8 simulation particles of H^+ for PIC. The timestep of DSMC and PIC is set to $1.2586 \times 10^{-6}s$ and $6.293 \times 10^{-7}s$ respectively. For the parameters of the dynamic load balancer, we set $Threshold$ to 2.0, R to 2, T to 20 and W_{cell} to 1. These parameters were automatically chosen during our pilot study on a different dataset using a sampling script. The effects of different parameters on the overall performance will be analyzed in Section VII-D. We take the results obtained using 24 processor cores as the baseline to measure the strong scalability. We then increase the number of process cores, using up to 1536 cores on Tianhe-2.

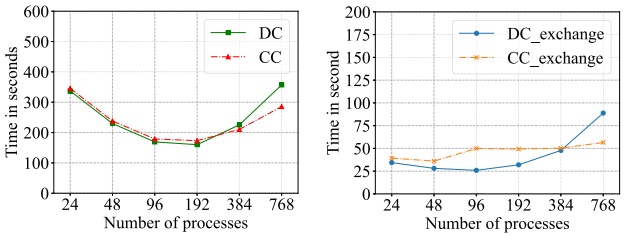
Results. Fig. 10 and Table II summarize the results for distributed (DC) and centralized (CC) communication strategies when they are used together with and without dynamic load balancing (LB). As can be seen from Fig. 10, all four implementation variants give improved performance when scaling from 24 cores to 1536 cores. A maximum speedup of about 14x (DC-only) is achieved, indicating a scaling efficiency of about 22%. On Tianhe-2, the distributed communication strategy always outperforms the centralized communication strategy under different numbers of parallel processes, regardless of whether the dynamic load balancer is enabled or not. When the number of processes increases, the advantage of the distributed communication strategy is even larger. For example, the performance is improved by more than 60% when using the distributed strategy for 1536 cores.

TABLE II
TOTAL EXECUTION TIME IN SECONDS FOR STRONG SCALABILITY TESTS.

	Number of Processes						
	24	***48***	***96***	***192***	***384***	***768***	***1536***
DC+LB	2258.5	1230.3	706.4	445.2	317.4	240.1	245.8
DC-Only	3602.9	2058.5	1156.8	689.1	378.4	311.5	257.1
CC+LB	2396.5	1400.6	919.6	678.8	605.1	571.8	591.4
CC-Only	3702.4	2192.7	1274.3	855.4	698.2	610.7	665.4

TABLE III
THE TOTAL EXECUTION TIMES IN SECONDS FOR PARTICLE MIGRATION

	Number of Processes						
	24	***48***	***96***	***192***	***384***	***768***	***1536***
DC+LB	358.62	208.37	101.81	53.08	29.90	17.95	12.05
DC-Only	1106.51	769.24	408.29	201.56	130.88	69.69	35.17
CC+LB	359.17	192.73	113.42	59.59	41.64	26.96	22.16
CC-Only	1119.21	777.87	408.56	200.13	128.81	69.92	35.37



(a) Total execution time (b) Total communication cost

Fig. 11. Performance comparison of the two communication strategies on the BSCC supercomputer. DC/CC denote the total execution times of the two strategies. DC_exchange/CC_exchange denote the total communication costs using the distributed and the centralized strategy respectively.

C. Further Analysis of Scalability Evaluation

1) Impact of load balancer: For the distributed and centralized communication strategies, we observe significantly improved performance and a better scaling efficiency when our dynamic load balancer is enabled. Our load balancer is particularly useful when a small number of processes is used. For example, our load balancer improves the performance by about 40% and 36% for the distributed and the centralized strategies, respectively, when using 48 cores. As discussed in Section V, load imbalance in our test case is more severe when a smaller number of processes are used in the parallel simulation. In such settings, our dynamic load balancing strategy helps rebalance the simulation load, leading to better overall performance and scalability.

Since load imbalance mainly affects the performance of *DSMC_Move* and *PIC_Move*, we further analyze the effects of dynamic load balance on the two procedures in Table III. It can be seen that the execution time of *DSMC_Move* and *PIC_Move* is reduced to less than one-third of the original implementation without dynamic load balance, which further demonstrates the effectiveness of our rebalancing implementation.

2) Distributed vs centralized communications: While the distributed communication strategy gives better performance over the centralized counterpart on Tianhe-2, the centralized strategy could still be beneficial when the number of simulation particles is small, and the process number is large, as we have discussed in Section IV. A centralized scheme also incurs a low memory footprint, making it suitable for simulating large problems. To validate our hypothesis, we perform a further evaluation on the BSCC supercomputer using *Dataset 3* in Table I. In this dataset, we increase the scaling factors of DSMC and PIC to 9.94×10^{11} and 4.77, respectively. As a result, the numbers of simulation particles are reduced to 10^8 and 10^7 correspondingly. Fig. 11 shows the communication costs as well as the total execution times for the two strategies on the BSCC supercomputer with load balance enabled.

As shown in Fig. 11, for a smaller number of processes (e.g., less than 384), the total execution times of the two strategies are quite close, although the communication costs of the centralized strategy are larger than the distributed strategy. For 768 processes, the communication cost of the distributed strategy dramatically increased to as large as more than twice of the centralized strategy, which makes the whole solver using the distributed strategy slower than using the centralized strategy by about 25%. The comparison provides some interesting insights when choosing parallel communication modes of DSMC/PIC on different HPC platforms for different simulation configurations (e.g., simulation particles, the number of processes). Another disadvantage of the distributed communication strategy is that it demands more memory.

3) Scalability bottleneck: To find out the scalability bottleneck, we further examine the breakdown of the total execution times on Tianhe-2. Some most time-consuming procedures are listed in Table IV for the DC+LB implementation. Most procedures except *Poisson_Solve* show good strong scalability. For example, the parallel efficiency of *DSMC_Move*, *Inject* and *Reindex* remain above 67% when scaling to 1536 cores. The time cost of particle migration is also not dominant. *Poisson_Solve* shows an extraordinary poor scalability to only 24 or 48 processes in our tests, making it a main bottleneck for the parallel performance of the coupled solver. One possible explanation for the poor scalability of *Poisson_Solve* is that the total number of grid cells of the nozzle (i.e., 583386) is not large enough for the large-scale parallel solution of the Poisson's equation. In addition, the matrix K in Poisson's equation is a sparse matrix, and thus the ratio of communication to calculation is very high. Enlarging the grid cell number may improve the parallel scalability of *Poisson_Solve*. Meanwhile, the numbers of simulation particles also have a great impact on parallel scalability. More simulation particles indicate a decreasing communication-to-calculation ratio. In Fig. 10, we scale to 1536 cores with more simulation particles, and in Fig. 11 we only achieve scalability with no more than 192 cores for fewer simulation particles.

D. Sensitivity Tests

This section evaluates the robustness of our approach under different tuning parameters and MPI rank placements. We also test our approach on an additional ARM-based cluster, show-

TABLE IV
BREAKDOWN OF THE TOTAL EXECUTION TIMES IN SECONDS FOR MAIN PROCEDURES ON TIANHE-2 USING DC WITH LOAD BALANCE ENABLED.

	Number of Processes						
	24	***48***	***96***	***192***	***384***	***768***	***1536***
DSMC_Move	283.2	163.1	82.4	42.4	21.6	11.8	6.6
DSMC_Exc[1]	29.8	17.3	11.1	8.3	4.5	2.5	3.3
Inject	1622.6	812.2	420.5	210.8	104.2	52.4	31.2
PIC_Move	74.8	45.3	19.3	10.6	8.2	6.2	5.5
PIC_Exc	68.5	44.5	31.7	25.1	20.5	20.1	28.8
Poisson_Solve	95.2	94.1	99.8	108.4	114.9	116.7	126.2
Reindex	13.9	6.6	2.5	1.3	0.8	0.4	0.2

[1] *Exc* denotes *Exchange*.

TABLE V
OVERHEAD (IN SECONDS) OF DYNAMIC LOAD BALANCING

	Number of Processes						
	24	***48***	***96***	***192***	***384***	***768***	***1536***
DC with KM	9.6	5.4	3.4	1.8	1.1	0.6	0.5
DC without KM	11.5	5.2	3.5	2.0	1.2	0.9	0.6
CC with KM	64.3	63.6	65.1	20.5	8.8	13.0	0.1
CC without KM	121.0	122.0	75.4	49.6	35.2	13.4	0.1

ing that our approach delivers portable performance across computing architectures.

1) Dynamic load balancing: We perform sensitivity tests on the Tianhe-2 supercomputer to further evaluate the impacts of different parameters in our dynamic load balance. This experiment was conducted on *Dataset 2* in Table I, but we observe similar results on other datasets.

Table V quantifies the importance of the KM algorithm. For our test scenarios, this algorithm plays an important role in *Rebalance* by reducing the overhead by 2x over the original implementation without KM. When using more than 768 processes, the benefit of the KM algorithm is less significant. Using more processes reduces the need for rebalancing operations (and the KM algorithm will be used less frequently). For example, when using 1536 processes, we only perform one dynamic load balancing operation for 200 PIC timesteps. As the load balancer gets executed less frequently as we use more processes, the load balancing overhead also declines.

Fig. 12 shows the effects of T. Typically when T is small, the solver will perform more rebalancing operations, and the overhead may possibly exceed the rebalancing benefit; when T is large, the load imbalance will become severe and degrade the overall performance. As we can see from Fig. 12, $T = 20$ is slightly better than $T = 10$ and $T = 30$ when using no more than 96 processes. With the process number increases, $T = 10$ is slightly better than $T = 20$ and $T = 30$.

Table VI shows the effects of various W_{cell} in (7). Since we find that a small change of W_{cell} has little effect in the tests, W_{cell} is increased from 1 to 10000 here. From Table VI we can see that the effects of W_{cell} is more obvious for smaller process numbers (e.g., 24 or 48 processes). For larger process numbers, W_{cell} has a slight effects (no more than 10%) on the overall performance. Generally when W_{cell} is

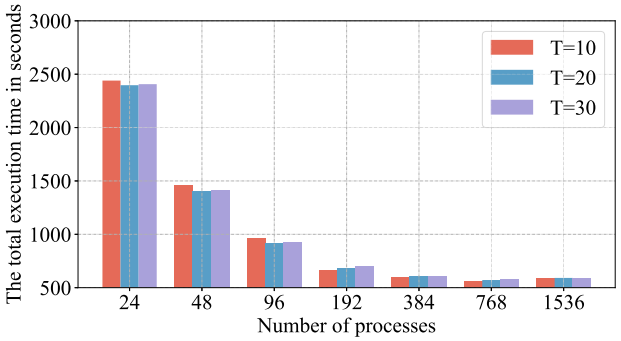


Fig. 12. Impact of T for the DC strategy on Tianhe-2.

TABLE VI
THE TOTAL EXECUTION TIMES IN SECONDS UNDER DIFFERENT W_{cell} FOR THE DC STRATEGY ON TIANHE-2

	Number of Processes						
	24	***48***	***96***	***192***	***384***	***768***	***1536***
$W_{cell} = 1$	2258	1230	706	445	317	240	245
$W_{cell} = 10$	2288	1255	705	437	301	234	239
$W_{cell} = 100$	2258	1240	689	420	292	228	232
$W_{cell} = 1000$	2231	1184	668	408	285	225	228
$W_{cell} = 10000$	2623	1420	771	466	316	243	234

too small, the grid re-decomposition will mainly be based on the particle numbers in cells; when W_{cell} is too large, the impact of particles will be ignored, which can also lead to load imbalance. To achieve better performance, users need to specify an appropriate W_{cell} according to their specific simulation configurations.

Fig. 13 shows the effects of $Threshold$. Similar to the effect of T, when $Threshold$ is small, the iterations required to meet $lii > Threshold$ will be less; when $Threshold$ is large, the iterations required to meet $lii > Threshold$ will be more. As we can see from Fig. 13, smaller $Threshold$ is slightly better than big $Threshold$ when using no more than 96 processes. The reason is that the load imbalance is more severe when the number of processes is smaller. With the process number increases, $Threshold$ has little effect on the results.

2) MPI rank placement: Most of our experiments were conducted on the Tianhe-2 supercomputer. As described in Section VI, this system implements a fat-tree communication topology [35]. This is done by packing 32 computing nodes in a frame where a computing rack has 4 frames. This organization permits three different MPI rank placements: inner-frame, inner-rack and inter-rack. To understand the impact of MPI rank placement, we evaluated our approach under these three MPI placement strategies, using up to 96 processes (16 compute nodes) on *Dataset 2*.

As shown in Fig. 14, different MPI rank placements can have an impact on the performance of the solver. The inner-frame placement outperforms the others because the communication distance between the compute nodes in this placement is smaller than the others. However, the difference is small, around 1% to 2%, indicating the robustness of our approach.

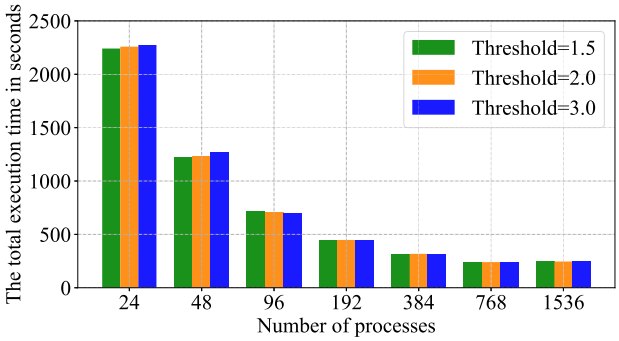


Fig. 13. Impact of $Threshold$ for the DC strategy on Tianhe-2.

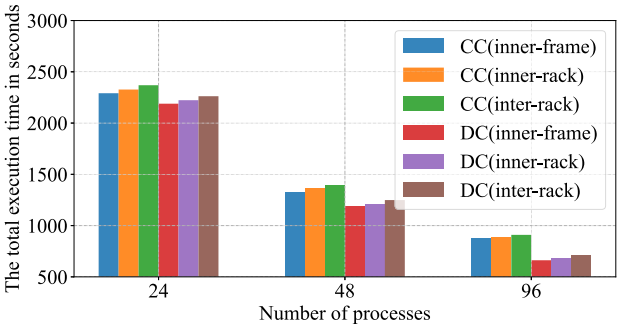


Fig. 14. Impact of different MPI rank placements for the CC and the DC strategies with dynamic load balancing on Tianhe-2.

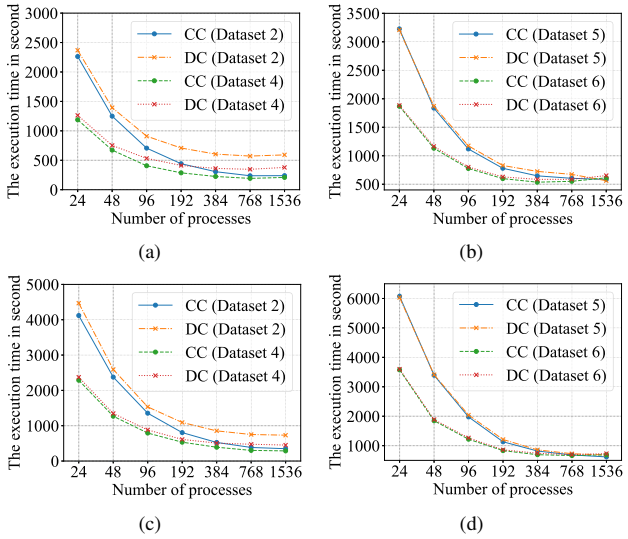


Fig. 15. Performance of the two communication strategies with dynamic load balance on Tianhe-2 ((a) & (b)) and the ARMv8-based Tianhe-3 prototype ((c) & (d)).

3) Hardware portability: We now evaluate the performance of our solver on systems with different hardware architectures and different datasets. In this evaluation, we use four datasets: *Dataset* 2, 4, 5 and 6 in Table I. We test our approach on the x86-based Tianhe-2 supercomputer and the ARMv8-based Tianhe-3 prototype. Among our test datasets, *Dataset* 5 and 6 have a larger size of grids, and *Dataset* 2 and 5 have a larger amount of simulation particles.

Fig. 15 shows the performance results on the two computing systems. Our approach exhibits similar strong scalability results on both architectures, indicating the performance portability of our approach. In addition, when using *Dataset* 5 and 6 (larger grids), the performance difference between the centralized and the distributed communication strategies (Fig. 15(b) and Fig. 15(d)) is smaller than that on *Dataset* 2 and 4 (Fig. 15(a) and Fig. 15(c)). Once again, the results suggest that the centralized communication can outperform the distributed communication in certain simulation setups.

VIII. Conclusion

We have presented a new approach to optimize large-scale coupled DSMC/PIC particle simulation, using the plasma plume simulation as a case study. We empirically show that both centralized and distributed communications can be helpful in particle simulations; each has its advantages in specific simulation setups. Our work addresses a particular challenge in coupled PIC/DSMC simulations – particles can move across grids, leading to load imbalance during dynamic simulation runs. To this end, we design a runtime load balancer to dynamically adjust the work distribution across parallel simulation processes and rebalance the load.

We evaluate our parallel-coupled DSMC/PIC solver by applying it to simulate 3D unsteady plasma plume induced by a pulsed vacuum arc. We test the performance of our solver on three distinct HPC platforms. Experimental results show that our coupled DSMC/PIC solver delivers performance and scaling efficiency, scaling well to thousands of processes with billions of particles.

Acknowledgement

This work was supported in part by the National Natural Science Foundation of China under grant agreements U173024, 61772542 and 61872294, and a UK Royal Society funded international exchange grant (IEC\NSFC\191465). Chuanfu Xu is the corresponding author.

References

[1] B. D. Smith, I. D. Boyd, H. Kamhawi, and W. Huang, "Hybrid-pic modeling of a high-voltage, high-specific-impulse hall thruster," 2013.

[2] B. Korkut, Z. Li, and D. A. Levin, "3-d simulation of ion thruster plumes using octree adaptive mesh refinement," *IEEE Transactions on Plasma Science*, vol. 43, no. 5, pp. 1706–1721, 2015.

[3] L. Brieda, S. Z. Tai, and M. Keidar, "Near plume modeling of a micro cathode arc thruster," in *Aiaa/asme/sae/asee Joint Propulsion Conference*, 2013.

[4] M. Mozetic, K. Ostrikov, D. N. Ruzic, D. Curreli, U. Cvelbar, A. Vesel, G. Primc, M. Leisch, K. Jousten, and O. B. a. Malyshev, "Recent advances in vacuum sciences and applications," *Journal of Physics D Applied Physics*, vol. 47, no. 15, p. 153001, 2014.

[5] Kato, Takahisa, Choi, Junho, Hirata, and Yuki, "Dlc coating on a trench-shaped target by bipolar pbii," *International Journal of Refractory Metals & Hard Materials*, 2015.

[6] F. Taccogna, P. Minelli, D. Bruno, S. Longo, and R. Schneider, "Kinetic divertor modeling," *Chemical Physics*, vol. 398, no. none, pp. 27–32, 2012.

[7] C. Gleason-Gonzalez, S. Varoutis, V. Hauer, and C. Day, "Simulation of neutral gas flow in a tokamak divertor using the direct simulation monte carlo method," *Fusion Engineering & Design*, vol. 89, no. 7-8, pp. 1042–1047, 2014.

[8] B. G. A., "Direct simulation of the boltzmann equation," *Physics of Fluids*, vol. 13, no. none, pp. 2676–2681, 1970.

[9] G. A. Bird, *Molecular gas dynamics*. Molecular Gas Dynamics, Oxford, 1976.

[10] C. K. Birdsall and A. B. Langdon, *Plasma Physics Via Computer Simulation*. Plasma Physics Via Computer Simulation, 1985.

[11] R. W. Hockney and J. W. Eastwood, "Computer simulation using particles," *Institute of Physics*, vol. 76, 1988.

[12] C. Xu, L. Zhang, X. Deng, J. Fang, and L. Wei, "Balancing cpu-gpu collaborative high-order cfd simulations on the tianhe-1a supercomputer," in *IEEE International Parallel & Distributed Processing Symposium*, 2014.

[13] C. Xu, X. Deng, L. Zhang, J. Fang, G. Wang, Y. Jiang, W. Cao, Y. Che, Y. Wang, and Z. a. Wang, "Collaborating cpu and gpu for large-scale high-order cfd simulations with complex grids on the tianhe-1a supercomputer," *Journal of Computational Physics*, vol. 278, pp. 275–297, 2014.

[14] H. Yaw, "The hungarian method for the assignment problem," in *Naval Res Logist Quart*, 1955.

[15] J. Munkres, "Algorithms for the assignment and transportation problems," *SIAM. J*, vol. 10, 1962.

[16] Hopkins, M. Matthew, Boerner, J. Jeremiah, Moore, C. Hudson, Crozier, and P. Stewart, "Addressing challenges to simulating breakdown and arc evolution in vacuum and low pressure systems." 2013.

[17] M. M. Hopkins, J. J. Boerner, C. H. Moore, P. S. Crozier, and M. T. Bettencourt, "Ppps-2013: Accommodating large temporal, spatial, and particle weighting demands for simulating vacuum arc discharge," in *Abstracts IEEE International Conference on Plasma Science*, 2013.

[18] M. M. Hopkins, R. P. Manginell, J. J. Boerner, C. H. Moore, and M. W. Moorman, "Fully kinetic simulation of atmospheric pressure microcavity discharge device," in *IEEE International Conference on Plasma Sciences*, 2015, pp. 1–1.

[19] P. Ortwein, T. Binder, S. Copplestone, A. Mirza, and C. D. Munz, "Parallel performance of a discontinuous galerkin spectral element method based pic-dsmc solver," 2015.

[20] S. Copplestone, P. Ortwein, C. D. Munz, T. Binder, and S. Fasoulas, "Coupled pic-dsmc simulations of a laser-driven plasma expansion," *Springer International Publishing*, 2016.

[21] C., Johnson, J., and Pitkgranta, "An analysis of the discontinuous galerkin method for a scalar hyperbolic equation," *Math. Comp*, 1986.

[22] B. Korkut and D. A. Levin, "Three dimensional dsmc-pic simulations of ion thruster plumes with sugar," in *Aiaa/asme/sae/asee Joint Propulsion Conference*, 2014.

[23] J. Revathi and D. A. Levin, "Chaos: An octree-based pic-dsmc code for modeling of electron kinetic properties in a plasma plume using mpi-cuda parallelization," *Journal of Computational Physics*, vol. 373, pp. 571–604, 2018.

[24] J. Li, D. Ingham, L. Ma, N. Wang, and M. Pourkashanian, "Numerical simulation of the chemical combination and dissociation reactions of neutral particles in a rarefied plasma arc jet," *IEEE Transactions on Plasma Science*, vol. 45, no. 3, pp. 461–471, 2017.

[25] Y. Su, J. Li, H. Wang, and A. Xu, "Numerical simulation of chemical reactions on rarefied plasma plume by dsmc method," *IEEE Transactions on Plasma Science*, vol. PP, no. 99, pp. 1–13, 2021.

[26] G. A. Bird, "Definition of mean free path for real gases," *Physics of Fluids*, vol. 26, no. 11, pp. 3222–3223, 1983.

[27] G. Bird, "Molecular gas dynamics and the direct simulation of gas flow," *Clarendon Press*, 1994.

[28] J. P. Boris, "Relativistic plasma simulations – optimization of a hybrid code," 1970.

[29] E. Lima, F. W. Tavares, and E. B. Jr, "Finite volume solution of the modified poisson-boltzmann equation for two colloidal particles," *Physical Chemistry Chemical Physics*, vol. 9, no. 24, pp. 3174–3180, 2007.

[30] K. L. U. of Minnesota. (2013) Metis - serial graph partitioning and fill-reducing matrix ordering. [Online]. Available: http://glaros.dtc.umn.edu/gkhome/metis/metis/overview

[31] Y. H. Tseng and J. H. Ferziger, "A ghost-cell immersed boundary method for flow in complex geometry," *Journal of Computational Physics*, vol. 192, no. 2, pp. 593–623, 2003.

[32] J. Liesen and Z. Strakos, *Krylov subspace methods: principles and analysis*. Oxford University Press, 2013.

[33] A. N. Laboratory. (2021) Petsc 3.16 — petsc 3.16.0 documentation. [Online]. Available: https://petsc.org/release/

[34] Z. Wang and M. O'Boyle, "Machine learning in compiler optimization," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879–1901, 2018.

[35] X. Liao, L. Xiao, C. Yang, and L. U. Yutong, "Milkyway-2 supercomputer: system and application," *Frontiers of Computer Science*, vol. 8, no. 3, 2014.

[36] Y. S. Li, X. H. Chen, J. Liu, B. Yang, and H. Xu, "Ohtma: an optimized heuristic topology-aware mapping algorithm on the tianhe-3 exascale supercomputer prototype," *Frontiers of Information Technology & Electronic Engineering*, vol. 21, no. 6, pp. 939–949, 2020.

[37] O. CASCADE. Salome - the open source integration platform for numerical simulation. [Online]. Available: https://www.salome-platform.org/