PID Feedback for Load-Balanced Parallel Gridless DSMC

Spencer E. Olson*

Air Force Research Laboratory, Directed Energy Directorate,
3550 Aberdeen Ave. SE, Kirtland Air Force Base, NM 87117-5776

Andrew J. Christlieb

Mathematics Department, Michigan State University
D304 Wells Hall, East Lansing, Michigan 48824-1027

Fredrik K. Fatemi

Naval Research Laboratory, Optical Sciences Division
4555 Overlook Ave. SW, Washington D.C. 20375

Abstract

Parallel code presents a non-trivial problem of load balancing computational workload throughout a system of hardware and software resources. The task of load balancing is further complicated when the number of allowable processors changes through time. This paper presents a two-component load-balancing mechanism using optimal initial workload distribution and dynamic load maintenance. The initial guess is provided by inversion of the workload distribution function. Workload distribution inversion enables efficient domain decomposition for arbitrary workloads and easily compensates for changes in system resources. Dynamic load balancing is provided by process feedback control as used, for example, in control mechanisms of physical processes. Proportional, integral, and differential (PID) feedback readily allows controls to compensate for runtime-changes of the workload distribution function. This paper demonstrates a one-dimensional realization of the ideas presented here. We apply this load-balancing technique to our gridless direct simulation Monte Carlo algorithm. We demonstrate that the method does indeed maintain uniform workload distribution across available resources as the workload and usable system resources undergo change through time.

Key words: Load balancing, parallel, DSMC, gridless
PACS: 2.70.-c, 47.11.-j
2000 MSC: 68W10, 68W15

1. Introduction

Rarefied gas flows are found in a range of technologically important applications and experiments. These include the design of vacuum chambers for plasma aided manufacturing [1–3], space environment test facilities [4, 5], micro-/nano-electro-mechanical systems (MEMS/NEMS) [6, 7], spacecraft thrusters [8, 9], expanding gas jets of [10] high altitude/velocity aircraft [11–13], and cold atom physics experiments [14].

A common approach to simulating rarefied gas flows is to use Direct Simulation Monte Carlo (DSMC) [15]. DSMC recasts the system in terms of a statistical representation of N test particles. The motion of these particles is kept completely independent except for collisions between only nearest-neighbor pairs of particles. Thus, by operating on small, localized groups of particles, DSMC lends itself well to excellent parallelization.

Early work on efficient DSMC methods centered on the development of algorithms ideal for vector supercomputers [16–18]. While this work demonstrated the advantage of parallel operations for DSMC, important limitations for DSMC on vector supercomputers arose. DSMC requires frequent random access to memory and stochastic interaction between fundamental particle elements. Furthermore, multiple levels of branching that occur in DSMC tend to increase the need for random variable access. Thus, taking greater advantage of vector hardware is an increasingly difficult problem [19, 20]. McDonald [19] demonstrated greater performance gains by targeting hybrid multi-processor, vector machines. Message passing provided synchronization and was used to transfer data (particles) between processors. McDonald further demonstrated parallel performance gains using multi-processor, scalar systems.

There are a few basic architectures to consider for implementing a massively parallel DSMC code: 1) a large multi-core shared memory system, 2) a network of distributed memory, lower-cost, few-core machines, or 3) more recently, multi-

*Corresponding author.

core Graphics Processing Unit (GPU) hardware. Because of the lower cost as compared to large shared memory systems, distributed-memory, multi-processor machines have become the most prevalent architecture for parallel processing, although multi-core GPU systems are currently being investigated [21]. As a result, high performance DSMC has evolved to fit into networks of independent computational nodes. Rather than sharing access among many processors to the same particles in memory, simulations are broken up into smaller tasks that are made as fundamentally independent as possible. By distributing the smaller tasks among a group of distributed memory machines, synchronization is made less local and CPU-memory bandwidth limits are less restrictive. One DSMC code to take advantage of the new distributed architecture is MONACO [22]. Dietrich showed that the distributed architecture can be employed to produce excellent performance and parallel efficiency [22].

Dietrich also demonstrated that parallel efficiency strongly depends on the workload uniformity across processors in the system. While workload distribution based on information known a priori does improve the load balance throughout a system of processors, a dynamic redistribution process is usually needed to achieve optimal performance. Nicol and Saltz discussed one strategy for dynamic load balancing of DSMC based on a method called Stop At Rise (SAR) [20, 23]. This method defines a degradation function, which is used to perform a cost-benefit analysis of re-decomposing the computational domain versus leaving the decomposition unchanged. Nance et al. [24] showed that the choice and implementation of the degradation function for SAR is very dependent on the physical system being simulated.

Another dynamic load-balancing technique uses a strategy analogous to heat diffusion [25]. Heat diffusion mathematics are used to determine the proper amount of information to transfer from one compute node to another. This load-balancing technique was proven useful in a parallel DSMC code called SMILE [26]. As reported by Wu and Lian [27, also see references therein], a dynamic domain decomposition can also be implemented using a graph partitioning technique. Wu and Lian demonstrated such a dynamic, load-balance code in three dimensions using an unstructured mesh.

In this paper, we discuss an efficient dynamic load-balancing scheme based on process feedback control. This scheme is optimally applied to systems that manage frequent and non-discrete changes to the workload distribution. Consequently, the first tests of the techniques presented here were done using gridless DSMC. As described in Ref. [28], gridless DSMC is an implementation of DSMC that automatically adapts to the real-time distribution of particles in a simulation. Hence gridless DSMC can easily accommodate continuous changes to the workload distribution. The gridless DSMC code has proven to be extremely flexible and has been applied to a range of rarefied gas flow problems: Couette flow, thermal Couette flow, low-velocity flow past a thin plate, high Mach flow past cylinders of square or double-flare profiles [28], and very-low-velocity flow of magnetically guided, evaporatively cooled, ultracold atoms [29].

In the following sections, we describe our load-balancing algorithm. We first discuss background information for load balancing and describe the test applications that will be used to demonstrate the techniques put forth in this paper. We then present our approach for establishing the two key aspects of load balancing: (1) dynamic load redistribution and (2) the initial guess. We also describe the application of our load-balancing algorithm to gridless DSMC to create a parallel processing code. Finally, we present the results of and conclusions from applying this work to the test applications described in Sec. 2.

2. Background

A typical approach for a parallel operation involves dividing a large task into smaller, less-demanding tasks which can be executed independently without affecting simulation integrity. Load-balanced, parallel computation implies that the total given workload of the major task is shared evenly across system resources. This efficient use of system resources minimizes wasted idle time resulting in quickest job completion.

For particle simulations, each of the smaller tasks must represent contiguous regions of space wherein collision statistics are calculated. Thus, the division of labor involves a partitioning of computational space such that each partition can be executed independently without affecting collision statistics. In addition, load balancing for particle simulations becomes a determination of the partition boundaries such that each partition represents an equal share of the total workload.

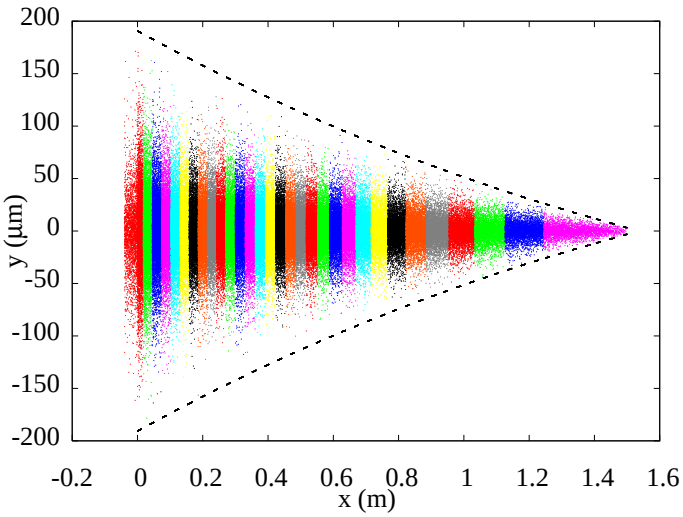

Figure 1: (Color online) An atomic beam being evaporatively cooled as it moves through a long, thin guiding channel. Particle groups, defined by small, contiguous regions of simulation space, are assigned to different processors to balance the workload on each of the processors. The different colors in this figure represent particles that are simulated on various processors. The thin, enveloping curves indicate the evaporation threshold as a function of x.

Fig. 1 shows a snapshot of a particle simulation near steady state that is executed in parallel. In this simulation, particles enter a longitudinal, magnetic, particle guide at $x \approx 0$ with an

average longitudinal velocity $\langle v_x \rangle > 0$ and average transverse velocity $\langle v_y \rangle \approx 0$. The configuration confines the particles in the transverse direction and guides them along the minimum of the magnetic field. The dotted line in Fig. 1 represents an evaporation surface. The most energetic particles travel beyond this evaporation surface and are immediately removed from the simulation. The remaining particles collide and redistribute their energy into the missing tail of a truncated thermal Boltzmann distribution. This process results in changing a rarefied particle beam at $x \approx 0$ into a much colder, density-enhanced, particle beam at the end of the guide. For more details concerning this particular application, see Ref. [14].

The different colors for the particles shown in Fig. 1 indicate the different parallel nodes to which particles are assigned. As the simulation progresses, particles cross the boundaries of the contiguous colored regions and are transfered from node to node as they travel down the length of the beam. The size of each nodal domain shown in Fig. 1 is chosen such that the workload per node is kept uniform.

3. Technique

In order to maintain balance, we must first define a measurable that will be used to quantify workload. For simplicity, we define the local workload size as the execution time τ_i as measured on the i^{th} node, or rather the time required for the i^{th} node to execute an arbitrary number of time steps. It is important to note that strict particle number per node is not necessarily a good indicator of computational workload. This is because calculating other gas dynamics can be computationally expensive. For example, in a hypersonic simulation such as presented in Ref. [28], some sections of the simulation domain exhibit very high collision rates compared to other sections. As another example, for simulations of trapped particles (*e.g.* in magnetic traps [14, 30] as shown in Fig. 1, optical traps [31], or Penning traps [32]), calculating accurate collisionless particle trajectories can be computationally diverse throughout the simulation domain.

We base our load-balancing scheme for DSMC on the premise that small changes to the partition boundaries between parallel nodes can be made from time step to time step. This premise enables incremental response to changes in the workload distribution as the particle simulation evolves. Incremental response causes the overall workload distribution to become and then remain uniform across the CPUs in the system. In this paper, we demonstrate a one-dimensional implementation of this technique.

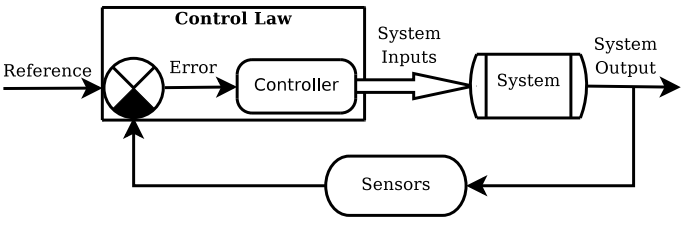

Figure 2: Diagram of a generic feedback control circuit.

Control theory provides ideal solutions for this incremental change problem. Control theory includes the study of maintaining a desired outcome from a system by continuously manipulating a set of system input parameters. The conceptual schematic in Fig. 2 shows a basic configuration used in control theory. The input to the control loop, marked as **Reference** on the left side of the diagram, represents the desired outcome of the system. At the beginning of the control loop, the reference point is compared to a value that represents the current state of the system. The difference between the reference point and the system point is known as the control loop error ϵ. As indicated in Fig. 2, the error ϵ is fed into the controller box. This box converts the error ϵ into a set of system input parameters. These input parameters are intended to change the system such that the desired outcome is more closely matched by the actual outcome of the system. The system response to this change of input parameters is then measured by appropriate sensors and fed back into the beginning of the control loop. For a more in-depth discussion of control theory, see Ref. [33].

The conversion of ϵ into a set of input parameters inside the controller box determines the effectiveness of the control loop for achieving and maintaining the desired outcome. One generic implementation of the controller box that has seen widespread use is called Proportional, Integral, and Differential (PID) feedback control [33, 34]. PID feedback control combines the result of proportional, integral, and differential operations on the error ϵ creating a total feedback response that follows

$$\mathrm{K_P}\epsilon + \frac{1}{\mathrm{T_I}}\int_0^t \epsilon\, dt - \mathrm{T_D}\frac{\partial \epsilon}{\partial t}. \tag{1}$$

The coefficients $\mathrm{K_P}$, $\mathrm{T_I}$, and $\mathrm{T_D}$ are called proportional, integral, and differential gain respectively.

The proportional term in Eq. 1 changes the system inputs in a way that is directly proportional to the current value of ϵ. This term tends to decrease both the steady-state magnitude of ϵ and the timescale over which the control loop attempts to correct the system output. Conversely, proportional feedback also tends to cause the system to overshoot the desired outcome, resulting in an oscillatory system point around the reference point.

The contribution from the integral term in Eq. 1 drives the system inputs based on both the magnitude and duration of the error ϵ. An undiminished ϵ causes the integral term to increase in magnitude and more significantly affect the system response. Thus, the primary benefit of the integral term is to sense and correct for steady state error. Similar to the proportional term, the integral term tends to decrease the control timescale while also causing an oscillatory system output.

The differential term in Eq. 1 drives the system inputs based on the derivative of ϵ. This term tends to slow the timescale of the controller especially when ϵ is near zero. Thus, differential control reduces the magnitude of the overshoot caused by the proportional and integral feedback terms and increases the stability of the control loop. However, because differentiation amplifies signal noise, this term can cause instability in a process if the noise and/or the differential gain are sufficiently large.

The process of optimizing the gain parameters for PID feedback control is called tuning. There are several methods known to successfully tune PID circuits. These methods include mathematical analysis, manual tuning, as well as automatic software tuning [34]. Analogous to other feedback control circuits, it can be expected that a set of well-optimized gain parameters will significantly improve the transient as well as steady state response of the feedback control loop. Conversely, we observed that some values of the gain parameters in Eq. 1 caused large steady state error and/or large oscillations of the system error ϵ. For simulations with large memory requirements, this corresponded to out-of-memory errors occurring and simulations crashing as the workload became too great for some system nodes. For this paper, gain parameters were found via rough manual tuning and not via a full tuning optimization. Although higher control performance was certainly possible, the gain parameters were tuned only to the point that waste of computational resources and wild oscillations in the system response were significantly diminished. The gain parameters used in this work were also satisfactorily used in several other simulations, the results of which are not presented here.

In the following subsections, we first describe the use of PID feedback control to perform dynamic load balancing during a simulation. Second, we discuss the effect of initial conditions on the PID controller. We present sub-optimal and optimal techniques for priming the load balancer. Finally, we briefly discuss the implementation of the load balancer as applied to gridless DSMC.

3.1. Load Balancing

As stated earlier, we define the workload measurable of the i^{th} node as the execution time τ_i required per simulation time step. The goal of load balancing is to equalize the execution time τ of any two neighboring nodes without adding significant overhead cost (discussed later in Sec. 4.2). Thus, for two neighboring nodes CPU1 and CPU2, the PID error signal ϵ is defined as

$$\epsilon \equiv \tau_1 - \tau_2 \tag{2}$$

where τ_1 and τ_2 are the computational times required for one time step as measured on CPU1 and CPU2 respectively. Note that the procedure for dynamic load balancing described here implies communication between neighboring nodes only.

If $\epsilon \neq 0$ at some time step of the simulation, a PID controller determines the proper relative spatial boundary shift $d\mathfrak{L}_{\text{SHIFT}}$ between CPU1 and CPU2 such that the resulting sub-domains require the same amount of work. A schematic for the PID control law is shown in Fig. 3. As shown in Fig. 3, ϵ is fed into the three term PID control box. The PID feedback result is scaled by the lower portion of the diagram in Fig. 3 so as to correlate computational time τ to sub-domain side length $\mathfrak{L}$. CPU1 and CPU2 then move the respective boundary by an amount $\pm d\mathfrak{L}_{\text{SHIFT}}$.

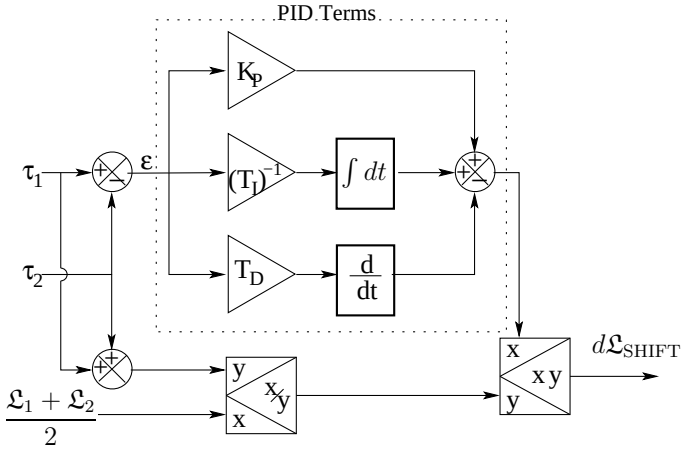

Figure 3: A feedback control diagram describing the load-balancing algorithm. The error signal is $\epsilon = \tau_1 - \tau_2$, where τ_1 and τ_2 are the computational times recorded on CPU1 and CPU2 respectively. ϵ is used to derive a relative shift, $d\mathfrak{L}_{\text{SHIFT}}$, in the common boundary between CPU1 and CPU2. CPU1 and CPU2 then move the respective boundary by an amount $\pm d\mathfrak{L}_{\text{SHIFT}}$.

The operations in Fig. 3 can be more formally written as

$$d\mathfrak{L}_{\text{SHIFT}} = \frac{1}{2}\frac{\mathfrak{L}_1 + \mathfrak{L}_2}{(\tau_1 + \tau_2)} \cdot \left(K_P\epsilon + \frac{1}{T_I}\int_0^t \epsilon\, dt - T_D\frac{\partial \epsilon}{\partial t}\right) \tag{3}$$

where K_P, T_I and T_D are the gain constants for the proportional, integral, and differential terms in Eq. (3). K_P is unitless whereas T_I and T_D are in pseudo-units of simulation time-steps and determine the length of time over which the respective controls affect the feedback loop. The leading factor of $1/2$ in Eq. (3) is to account for the fact that both CPU1 and CPU2 undergo a boundary move by an amount $d\mathfrak{L}_{\text{SHIFT}}$. Typical values of our control parameters are $K_P = 0.1$, $T_I = 100$, and $T_D = 1/80$. As indicated earlier, these values have not been determined by a rigorous tuning optimization procedure. We expect that higher performance can be attained by executing such a tuning optimization. Furthermore, the exact tuning optimization is likely to depend on the particular simulation.

As an illustration of the PID feedback load balancer, consider a simple gas expansion problem: A one-dimensional, thermal gas with a positive average velocity ($\langle v_{\parallel} \rangle = 1$ m/s) is allowed to freely expand from a source at the origin. A set of restrictions, including a minimum number of particles per CPU, is specified such that only one CPU is allowed at $t = 0$, as shown in Fig. 4. Fig. 4 represents the sub-domain of each active processor as indicated by the different colored regions. The solid lines in Fig. 4 show the actual boundaries of the domain that contains all particles.

As the source injects more and more particles into the system, it becomes more appropriate to divide the workload between more than one processor. At $t \approx 20$ ms, one additional processor is added and the domain is evenly divided between the two active processors. The simulation then continues and grows in size until yet another processor is made active at $t \approx 35$ ms. Fig. 4 demonstrates that the PID feedback load

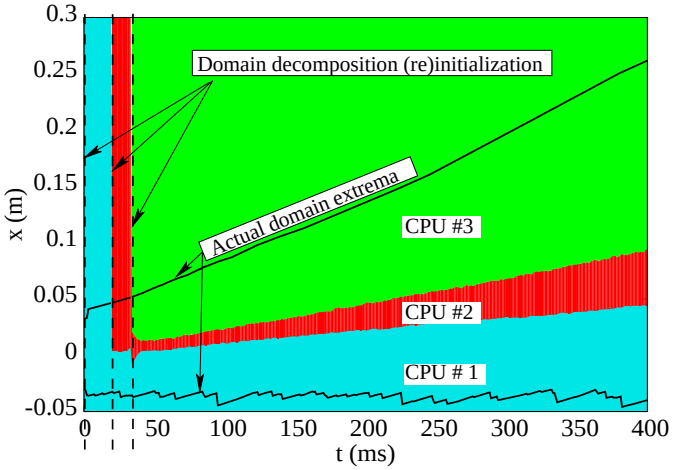

Figure 4: (Color online) The computational domain is divided up into sections, which are assigned to specific processors. After global domain decomposition takes place, local shifts of CPU boundaries occur in order to perform locally controlled load balancing. The colored regions indicate the computational sub-domain as assigned to a given CPU. The solid lines indicate the boundaries of the domain that contains all particles.

balancer continuously realigns the local sub-domain boundaries while it attempts to properly distribute the workload. Fig. 5 further shows that the computational time τ consumed by each CPU in this toy problem is indeed uniform across the active processors. Fig. 6 shows the number of particles assigned to each CPU. From Figs. 5 and 6 it is apparent that particle count is indeed not a good measure of workload as discussed in the beginning of Sec. 3.

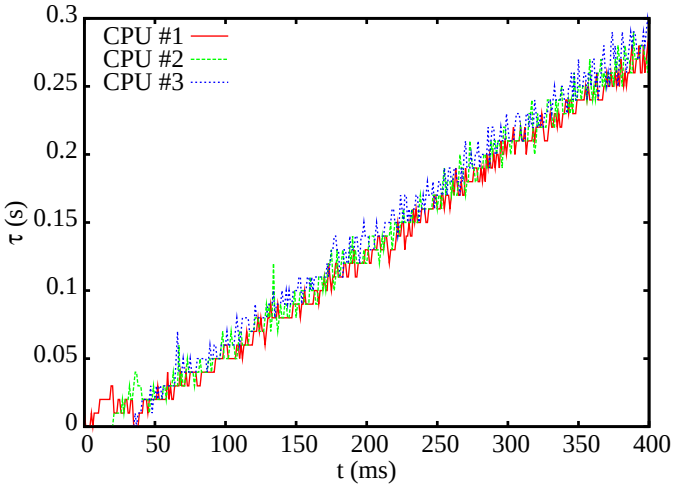

Figure 5: (Color online) Computational time expended by each processor.

3.2. Initial Domain Decomposition

Even a load-balancing scheme that hones the workload distribution during operation can waste significant system resources if not placed in correct initial conditions. Initial domain decomposition is the process whereby simulation space, or the domain, is initially allocated among the available computational

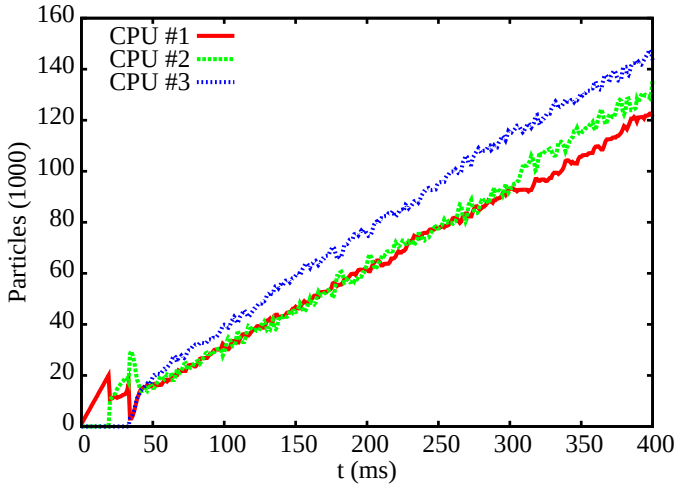

Figure 6: (Color online) Particles per CPU.

nodes. For many DSMC simulations, the number of processors that can be utilized can undergo change throughout the length of the simulation. It is thus necessary to re-initialize the workload distribution as processors are either added or removed from a simulation. To achieve optimal performance and minimal wait time in the system after each time (re)initialization is performed, it is imperative that the initial decomposition represents a best estimate of the appropriate starting load balance.

As an illustration of a suboptimal or naïve initial domain decomposition, consider a system with an arbitrarily contrived workload function as shown in Fig. 7. One might make the naïve assumption that the workload is evenly distributed throughout the simulation space. Following this assumption, processors would be assigned to equally sized sub-domains spaced on an even grid of the complete domain, as shown in Fig. 7. We call this *flat profile decomposition*, referring to the processor distribution function. Using this decomposition with the workload function shown in Fig. 7 results in an imbalanced system where a few processors perform most of the work. The weakly loaded processors spend most of their wall-clock time in an idle state.

To more accurately divide the workload, predictive knowledge of the workload is required. Assuming that such information is available, it is possible to generate a very accurate domain decomposition that initially evenly balances load among a set of processors. To generate a predictive description of the system workload, we rely on the assumption that the number of processors that can be used by a simulation varies with time. Thus, for each time the number of active processors changes, a more sophisticated initial domain decomposition can be executed using workload information as collected in prior simulation time steps. For simplicity, *flat profile decomposition* is used at $t = 0$.

To describe the more sophisticated domain decompositions that occur for $t > 0$, we first define the workload distribution

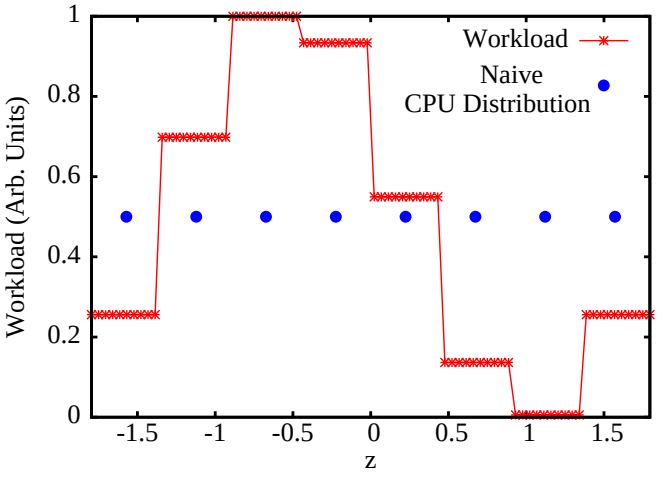

Figure 7: (Color online) Workload and original CPU distribution based on *flat profile decomposition*.

function $W(z)$:

$$W(z) = \sum_i W_\mathrm{i} \int \delta\left(\left\lfloor \frac{z_\mathrm{i} - z}{\mathfrak{L}_\mathrm{i}} + \frac{1}{2} \right\rfloor\right) dz \qquad (4)$$

where $\mathfrak{L}_\mathrm{i}$ is the one-dimensional length of the i^{th} computational sub-domain in the z direction, z_i and W_i are the central position and workload per unit length of the same i^{th} sub-domain respectively, and $\delta(X)$ is the Dirac delta function. W_i is given as

$$W_\mathrm{i} = \tau_\mathrm{i} / \mathfrak{L}_\mathrm{i} \qquad (5)$$

where τ_i is the computational time or the amount of CPU time required to execute a given time step. $W(z)$ describes the workload throughout the complete one-dimensional computational domain. To efficiently use computational resources we attempt to distribute the work to different nodes according to $W(z)$. Thus, the centers of neighboring nodes in regions of high workload will be closer together in simulation space than those in regions of low workload.

There is one caveat when assigning initial nodal boundaries: for simplicity and scalability, we seek to maintain nearest-neighbor-only communication. A problem arises if nodes are assigned sub-domains that are small enough for any given local particle to traverse a local sub-domain entirely within one time step. To protect against this, we modify $W(z)$ by a function that describes the maximum density of processors allowed $N_\mathrm{p}(z)$ given by

$$N_\mathrm{p}(z) = \sum_i N_{\mathrm{p_i}} \int \delta\left(\left\lfloor \frac{z_\mathrm{i} - z}{\mathfrak{L}_\mathrm{i}} + \frac{1}{2} \right\rfloor\right) dz \qquad (6)$$

where $N_{\mathrm{p_i}}$ is given by

$$N_{\mathrm{p_i}} = \mathfrak{L}_\mathrm{i} / \left[(v\Delta t)_{\max}\right]_\mathrm{i} \qquad (7)$$

and $(v\Delta t)_{\max}$ is the maximum distance traveled by any given local particle during a single time step. The total number of usable processors is found by $\max\left[1\,, \int N_\mathrm{p}(z) dz\right]$. Thus, we end up with the processor distribution function $P_{\mathrm{CPU}}(z)$ given by

$$P_{\mathrm{CPU}}(z) = \sum_i W_\mathrm{i}\, N_{\mathrm{p_i}} \int \delta\left(\left\lfloor \frac{z_\mathrm{i} - z}{\mathfrak{L}_\mathrm{i}} + \frac{1}{2} \right\rfloor\right) dz\,. \qquad (8)$$

Processors will be assigned sub-domains according to Eq. (8) to reflect both the need to concentrate resources in heavy workload regions as well as to avoid neighboring sub-domain centers with insufficient separation. To apply Eq. (8), we use distribution inversion. We call this technique *fitted profile decomposition*.

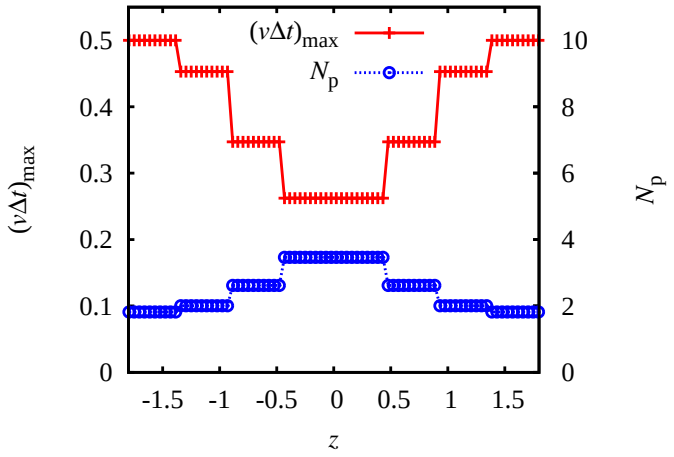

Figure 8: (Color online) Maximum transit $(v\Delta t)_{\max}$ of particles throughout each node of the simulation and the resulting number of supportable processor density $N_\mathrm{p}(z)$.

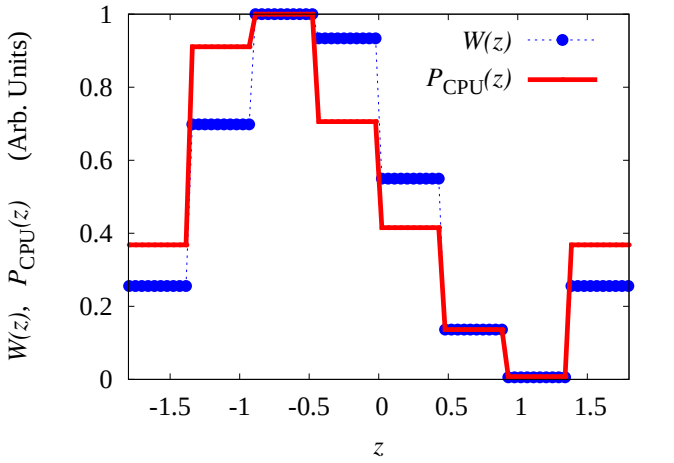

Figure 9: (Color online) Processor distribution function $P_{\mathrm{CPU}}(z)$ is a product of the workload ($W(z)$) and supportable processor ($N_\mathrm{p}(z)$) functions.

To illustrate this technique, consider the situation shown in Fig. 7 where *flat profile decomposition* places eight processors evenly throughout simulation space. Although the decomposition shown in Fig. 7 does waste system resources, it can be used to gain information about the workload function $W(z)$. In fact, the measured workload will be exactly the curve shown in

Fig. 7. For the sake of demonstration, assume that maximum transit $(v\Delta t)_{\max}$ as measured within each node is given by the contrived upper curve in Fig. 8. Using Eqs. (6) and (7), the density of nodes that can be supported $N_{\mathrm{p}}(z)$ is given by the lower curve in Fig. 8. Integrating the N_{p} curve in Fig. 8 shows that up to nine processors can be supported by P_{CPU}. The resulting processor distribution function $P_{\mathrm{CPU}}(z)$ is compared to $W(z)$ in Fig. 9.

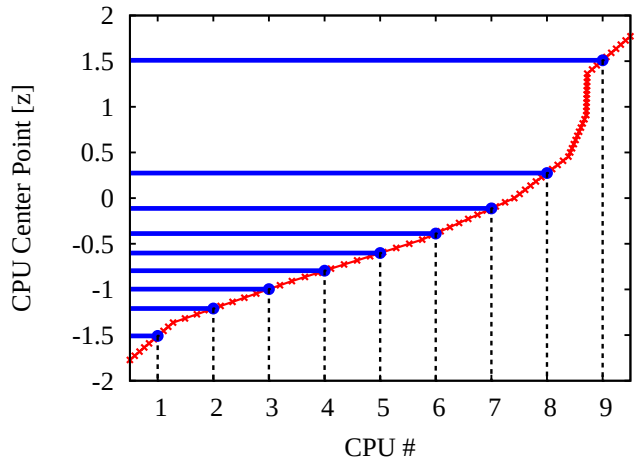

Figure 10: (Color online) Inverted desired CPU distribution.

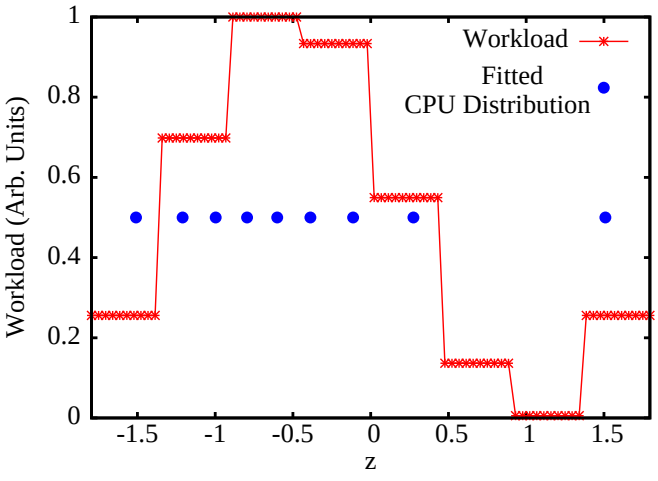

Figure 11: (Color online) Workload and new CPU distribution for decomposition example.

To assign new sub-domains, we invert the distribution $P_{\mathrm{CPU}}(z)$: first we integrate Eq. (8) as

$$I(z) = \int_0^z P_{\mathrm{CPU}}(\mathfrak{z}) d\mathfrak{z} \ ,$$

then swap axes, and assign processor locations by mapping from evenly spaced x-axis coordinates as shown in Fig. 10. This process results in new processor locations as shown and compared to the originally measured workload function $W(z)$ in Fig. 11. Supplemental material to this paper includes a Matlab demonstration of this decomposition logic.

3.3. Application to Gridless DSMC

The independence of gridless DSMC from the simulation geometry allows a parallel management code to easily wrap around the DSMC code. In this way, the gridless DSMC component within each processor of the parallel system runs almost independently. Communication between the various processors is handled by the wrapping parallel code only at the end of each time step. Using the boundary mechanism, described in Sec. 2.3.4 of Ref. [28], virtual boundaries are installed within each DSMC instance to represent the boundaries between the sub-domains of the CPUs. Particles that cross these boundaries during the move operation are subsequently shuffled (via network communication) to the appropriate neighboring sub-domain at the end of the time step. By ensuring that the length $\mathfrak{L}_{\mathrm{i}}$ of any i^{th} computational sub-domain satisfies $\mathfrak{L}_{\mathrm{i}} \gg (v_{\parallel}\Delta t)_{\max}$, the shuffling is limited to nearest-neighbor communication. This simplifies the implementation and minimizes both network traffic (as many neighboring nodes are often on the separate cores of a single multi-processor machine) and overhead.

It should also be noted that to apply the algorithms presented here to any DSMC code, care should be taken to ensure that the i^{th} sub-domain width $\mathfrak{L}_{\mathrm{i}}$ is somewhat greater than the mean free path λ_{MFP} of the particles in the i^{th} sub-domain. Particles within a distance of λ_{MFP} from each other should generally be allowed to collide. Enforcing a minimum sub-domain size based on λ_{MFP} would ensure that particles are not synthetically isolated from each other. Thus, it will likely be necessary to replace occurrences of $(v\Delta t)_{\max}$ in Sec. 3.2 by

$$\max\left((v\Delta t)_{\max} \ , \ m \cdot \lambda_{\mathrm{MFP}}\right)$$

where m generally satisfies $m \gtrsim 1$ and depends on the time step size Δt. In the results presented in this paper, care was indirectly taken by requiring a minimum number of particles per sub-domain such that each sub-domain size was in excess of λ_{MFP}.

4. Results

To demonstrate the methods presented in this paper more fully, we return to the simulation described in Sec. 2 and depicted in Fig. 1. As described earlier, this calculation simulates forced evaporative cooling and compression of a particle beam.

There are several major factors that contribute to the workload distribution for this simulation besides the density of particles. Particle guiding involves particles traveling through regions with a significant range of forces. Furthermore, in the physical system modeled by this simulation, the magnetic guiding potential increases as a function of longitudinal position. An increasing effort is therefore needed through the length of the guide to obtain accurate trajectories. Since this simulation attempts to show real cooling, it is necessary to use an integrator that does not synthetically dissipate energy. Thus, we must exert considerable effort to calculate accurate particle trajectories. To achieve this, this simulation uses a fifth-order, adaptive Runge-Kutta integration. In addition to the variable forces, the

workload is modified by the evaporative cooling. Evaporative cooling plays a role in increasing both the collision cross section, as well as changing the density and hence collision rate. With all of these effects, we expect the workload density to exhibit a large dependence on longitudinal position.

To simulate the guided particle beam in a parallel context, the code automatically divides the guide in the longitudinal direction as shown in Fig. 1. For demonstrating the load-balancing approach of this paper, we use up to 20 processors. To start the simulation, the guide is initially empty. This allows for only one active processor out of the available pool of 20. A constant flux of particles is injected into the guide at $x = 0$. As the particle count increases, more processors are added to the simulation. The following two subsections will demonstrate the load-balancing code while contrasting the fitted initial domain decomposition logic with the simple, flat distributed model.

4.1. Decomposition and Load Balancing

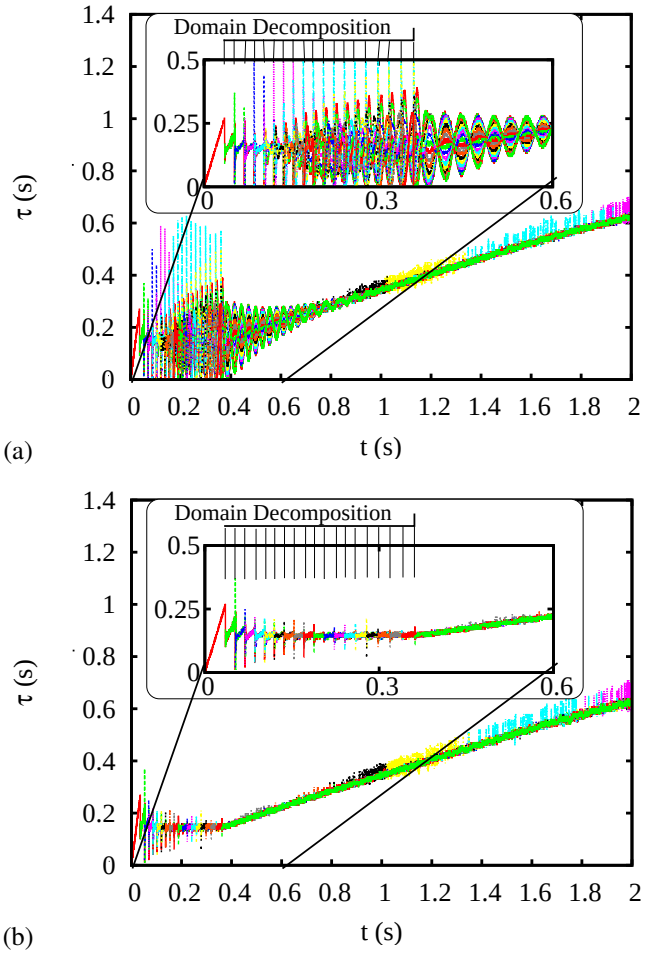

Figure 12: (Color online) Computational time per CPU τ as a function of simulation time t for a 20 processor simulation. (a) Simple flat and (b) fitted decomposition results are compared.

Fig. 12(a-b) shows the computational time per each CPU required for simulating the particle guide as a function of simulation time. Each figure uses the same dynamic, PID feedback load balancer preceded by either the simple *flat profile decomposition* model as in Fig. 12(a) or the *fitted profile decomposition* model as in Fig. 12(b). At the beginning of the simulation, there are no particles and thus only one CPU is active. As more and more particles are added to the system, additional nodes are made active and the computational domain is reassigned. Fig. 12(a-b) shows each repeat decomposition until all twenty of the available processors have been added. After each iteration of the given decomposition logic, boundaries are allowed to evolve under the load-balancing PID control. The insets in Fig. 12(a-b) show a closeup of each domain decomposition and the results of the initial PID control.

From Fig. 12(a), it is clear that the *flat profile decomposition* hardly distributes the workload properly. This is evident by the large adjustments that are necessary as the PID logic attempts to correct the imbalance. Although the dynamic load balancing PID control does eventually reign in the imbalance, significant time is wasted as the system undergoes large oscillations in the workload distribution.

On the other hand, as shown in Fig. 12(b), the *fitted profile decomposition* results in the workload being distributed evenly before the PID control takes over. In contrast to the large oscillations in Fig. 12(a), Fig. 12(b) shows a continual even workload distribution and smooth transitions even as new processors are added to the calculation.

Fig. 13(a-b) shows the particle number per processor as a function of simulation time. The wide range in particle numbers per CPU as the PID control approaches steady state reflects the higher computational load for regions where collisions play a larger role as well as the integrator requiring more calculations due to larger magnetic forces. This shows that the PID control does indeed balance the actual workload such that each processor in the system spends roughly an equal portion of time calculating each step as shown in Fig. 12(a-b).

4.2. Scalability

For any parallel processing scheme, there is a certain fraction of the total computational time that must be used for managing the parallelization. This overhead often involves communication between nodes, explicit synchronization (which can result in idle processors) between nodes, sub-domain boundary calculations/decomposition, and so on. To justify the use of a parallel approach for a given calculation, the cost of the overhead must not outweigh or be equal to the benefit gained through using the additional computational resources.

Fig. 14(a-b) shows the overhead $\Delta\tau$ per each processor as a function of simulation time for the examples above. Fig. 14(a) shows that the *flat profile decomposition* causes considerable overhead during the initial PID control. In spite of this, it is also clear that after the PID control balances the workload, the remaining overhead is less than 5%. As expected from the smooth transitions via the *fitted profile decomposition* method, Fig. 14(b) shows that the overhead can be minimized, even while activating new processors for the simulation.

The scalability of the parallel approach presented in this paper depends on the application. The overhead using the one-dimensional splitting, as described in this paper, depends on

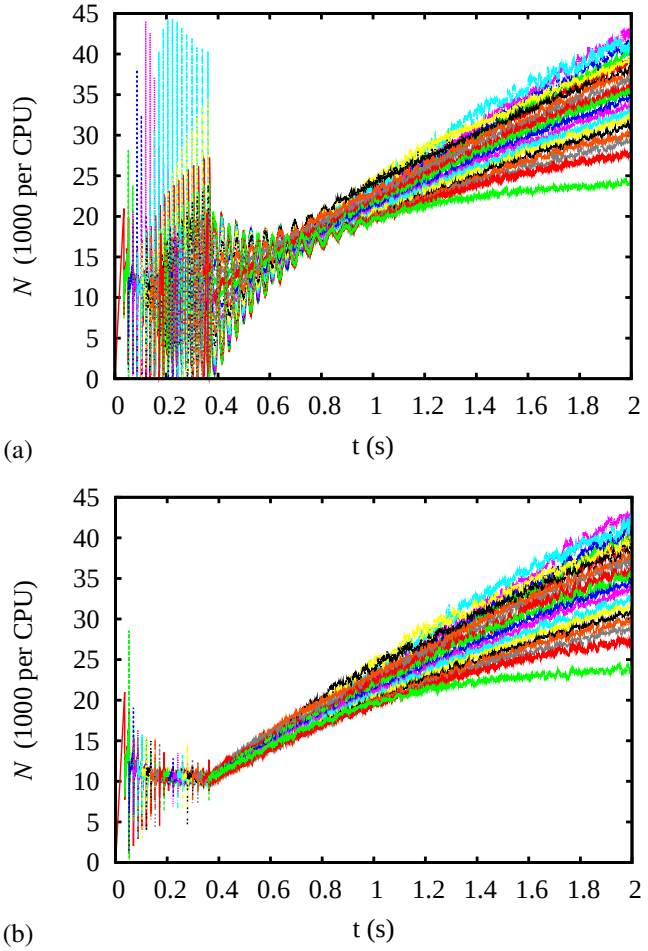

Figure 13: (Color online) Number of particles N assigned to each of 20 processors as a function of simulation time t. (a) Simple flat and (b) fitted decomposition results are compared.

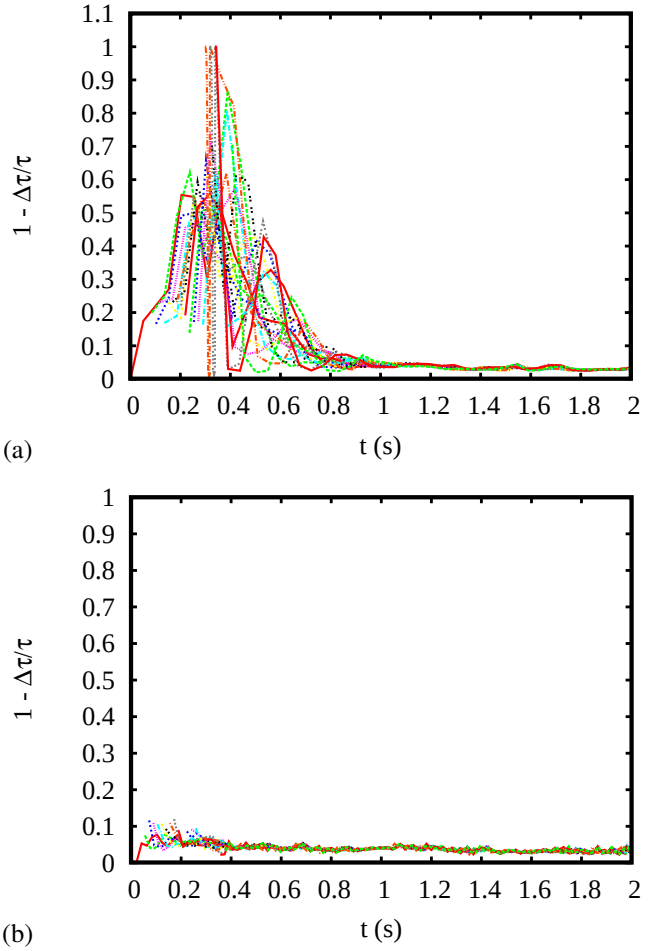

Figure 14: (Color online) Average overhead ($\Delta\tau$) for each of 20 processors in fraction of total computational time per time step (τ) as a function of simulation time t. (a) Simple flat and (b) fitted decomposition results are compared.

whether the virtual boundaries between the sub-domain nodes must make large excursions. Large movements in the boundaries cause a dramatic increase in necessary particle exchange between neighboring nodes. In the examples presented here, the splitting is done along the direction of the stream velocity. In addition, because we only present the non-steady state turn-on phase of the simulations, significant movement among the internal virtual parallel boundaries is necessary to achieve load balance. As the system approaches steady state, the overhead will decrease.

In practice, the overhead can be significantly lower than seen in the examples here. Because of the nearest-neighbor-only communication paradigm, the fractional overhead per CPU does not increase with the number of processors. In practice, we have seen overhead values as low as 0.1% for as many as 256 active processors.

5. Conclusion

We have demonstrated an adaptive technique for managing load balancing in a parallel simulation. The results show that the overhead is near negligible, usually around 1-5%. The results further show that the load among system resources can be easily balanced even with a dynamically changing simulation.

5.1. Future Direction

The algorithms presented in this paper assume that workload change generally occurs slowly. Simulations can exist where instantaneous workload change can occur at any time in the simulation. For instance, a simulation might include a process that begins and ends at arbitrary simulation times. A reset condition to detect sudden large change in the PID controller should be defined whereupon the initial workload distribution process described in Sec. 3.2 is re-initiated.

For more complex workload configurations, domain decomposition in more than one dimension may be optimal. It would thus be of interest to determine whether PID feedback control and workload distribution inversion can be used efficiently in concert or separately for two and three dimensions. One possible method of extending this work to two and three dimensions is to use a hierarchical approach as described by Campbell et al. [35]. A hierarchical approach dictates that one-dimensional

methods be repeated over each dimension. In two dimensions, this results in rows of one-dimensional load balancing groups. Ideally, the dimension corresponding to the most intense load-balancing effort would also correspond to the individual cells of the rows of load balancers. Thus, most communication and sub-domain movement is isolated separately within each one-dimensional load balancing group. For three dimensions, this results in grouping rows together into a single load balancing group that divides the entire computational domain in the direction of the final dimension.

Though we anticipate that extending PID feedback control into higher dimensions using the hierarchical approach is straightforward, enforcing alignment to coordinate axes may prove detrimental for global efficiency. An alternate approach could include measures to separate global domain structure from local domain structure. For example, a hierarchical tree could be constructed to represent the entire computational domain, where the size and location of leaf nodes in the tree represent local computational demands. In each of these local zones, a single one-dimensional PID feedback controller could be aligned along an arbitrary axis. The direction of the local feedback controller would be determined by some method that dynamically measures the direction of the most demanding load-balancing effort. Allowing the local domains to be aligned arbitrarily will enable higher local efficiency without globally effecting the efficiency of other local balancers. This approach may perhaps be considered a hybrid strategy between the graph decomposition as presented by Wu and Tseng [36] and the geometrical PID-feedback-based decomposition presented here.

Acknowledgments

This work was partially supported by the Army Research Office and the Office of Naval Research (Project number 42791-PH). SEO was partially supported for this work by a National Research Council research associateship. Computational resources for this work were provided by the Naval Research Laboratory as a part of the Defense High Performance Computing Modernization Program.

References

[1] A. J. Christlieb, W. N. G. Hitchon, and E. R Keiter. A computational investigation of the effects of varying discharge geometry for an inductively coupled plasma. *Plasma Science, IEEE Transactions on*, 28(6): 2214–2231, 2000. doi: 10.1109/27.902250 .

[2] A. V. Vasenkov and M. J. Kushner. Angular anisotropy of electron energy distributions in inductively coupled plasmas. *J. Appl. Phys.*, 94:5522, 2003. doi: 10.1063/1.1614428 .

[3] C. Cai, I. D. Boyd, and Q. Sun. Rarefied background flow in a vacuum chamber equipped with one-sided pumps. *J. Therm. Heat Trans.*, 20(3): 524–535, 2006. doi: 10.2514/1.19178 .

[4] I. D. Boyd, C. Cai, M. L. R. Walker, and A. D. Gallimore. Computation of Neutral Gas Flow From a Hall Thruster Into a Vacuum Chamber. *AIP CONFERENCE PROCEEDINGS*, pages 541–548, 2003.

[5] D. F. G. Rault and M. S. Woronowicz. Application of direct simulation Monte Carlo to satellite contamination studies. *J. Spacecraft and Rockets*, 32:392–397, May 1995. doi: 10.2514/3.26627 .

[6] A. J. Christlieb, W. N. G. Hitchon, I. D. Boyd, and Q. Sun. Kinetic description of flow past a micro-plate. *J. Comp. Phys.*, 195(2):508–527, 2004. doi: 10.1016/j.jcp.2003.10.027 .

[7] W. Yuan, H. Chang, W. Li, and B. Ma. Application of an optimization methodology for multidisciplinary system design of microgyroscopes. *Microsystem Technologies*, 12(4):315–323, 2006. doi: 10.1007/s00542-005-0054-2 .

[8] I. D. Boyd. Review of Hall Thruster Plume Modeling. *J. Spacecraft and Rockets*, 38:381–387, May 2001. doi: 10.2514/2.3695 .

[9] I. D. Boyd and J. P. W. Stark. Modeling of a small hydrazine thruster plume in the transition flow regime. *J. Prop. Pow.*, 6(2):121–126, 1990. doi: 10.2514/3.23232 .

[10] J. S. Wu, S. Y. Chou, U. M. Lee, Y. L. Shao, and Y. Y. Lian. Parallel DSMC simulation of a single under-expanded free orifice jet from transition to near-continuum regime. *J. Fluids Eng.*, 127:1161, 2005. doi: 10.1115/1.2062807 .

[11] J. F. Padilla and I. D. Boyd. Assessment of Rarefied Hypersonic Aerodynamics Modeling and Windtunnel Data. *9 th AIAA/ASME Joint Thermophysics and Heat Transfer Conference*, 2006.

[12] M. S. Ivanov and S. F. Gimelshein. Computational Hypersonic Rarefied Flows. *Annual Review of Fluid Mechanics*, 30:469–505, 1998. doi: 10.1146/annurev.fluid.30.1.469 .

[13] J. N. Moss, R. C. Blanchard, R. D. Braun, and R. G. Wilmoth. Mars Pathfinder Rarefied Aerodynamics: Computations and Measurements. *J. Spacecraft and Rockets*, 36:330–339, May 1999. doi: 10.2514/2.3475 .

[14] S. E. Olson, R. R. Mhaskar, and G. Raithel. Continuous propagation and energy filtering of a cold atomic beam in a long high-gradient magnetic atom guide. *Phys. Rev. A*, 73(3):033622, March 2006. doi: 10.1103/PhysRevA.73.033622 .

[15] G. A. Bird. *Molecular Gas Dynamics*. Oxford University Press, 1976.

[16] D. Baganoff and J. D. McDonald. A collision-selection rule for a particle simulation method suited to vector computers. *Phys. Fluids*, 2:1248–1259, July 1990. doi: 10.1063/1.857625 .

[17] S. Dietrich. An efficient computation of particle movement in 3D DSMC calculations on structured body-fitted grids. In *Proceedings of 17th Int. Symp. On Rarefied Gas Dynamics, Aachen, Germany*, 1991.

[18] I. D. Boyd. Vectorization of a Monte Carlo simulation scheme for nonequilibrium gas dynamics. *J. Comp. Phys.*, 96:411–427, October 1991. doi: 10.1016/0021-9991(91)90243-E .

[19] J. D. McDonald. Particle Simulation in a Multiprocessor Environment. *AIAA Paper*, 9(1):1366, 1991.

[20] G. J. LeBeau. A parallel implementation of the direct simulation Monte Carlo method. *Comp. Meth. Appl. Mech. Eng.*, 174(3–4):319–337, 1999. doi: 10.1016/S0045-7825(98)00302-8 .

[21] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008. doi: 10.1145/1365490.1365500 .

[22] S. Dietrich. Scalar and Parallel Optimized Implementation of the Direct Simulation Monte Carlo Method. *J. Comp. Phys.*, 126:328–342, July 1996. doi: 10.1006/jcph.1996.0141 .

[23] D. M. Nicol and H. Saltz. Dynamic Remapping of Parallel Computations with Varying Resource Demands. *IEEE TRANSACTIONS ON COMPUTERS*, 37(9):1073, 1988.

[24] R. P. Nance, H. Hassan, R. G. Wilmoth, B. Moon, and J. Saltz. Parallel Monte Carlo simulation of three-dimensional flow over a flat plate. *J. Therm. Heat Trans.*, 9(3):471–477, 1995. doi: 10.2514/3.689 .

[25] S. Taylor, J. R. Watts, M. A. Rieffel, and M. E. Palmer. The Concurrent Graph: Basic Technology for Irregular Problems. *IEEE parallel and distributed technology: systems and applications*, 4(2):15–25, Summer 1996. doi: 10.1109/88.494601 .

[26] M. Ivanov, G. Markelov, S. Taylor, and J. Watts. Parallel DSMC strategies for 3D computations. *Parallel Computational Fluid Dynamics: Algorithms and Results Using Advanced Computers*, 1996.

[27] J. S. Wu and Y. Y. Lian. Parallel three-dimensional direct simulation Monte Carlo method and its applications. *Computers and Fluids*, 32(8): 1133–1160, 2003. doi: 10.1016/S0045-7930(02)00083-X .

[28] S. E. Olson and A. J. Christlieb. Gridless DSMC. *J. Comp. Phys.*, 227 (17):8035–8064, September 2008. doi: 10.1016/j.jcp.2008.04.038 .

[29] S. E. Olson. *Long, High-Gradient Magnetic Atom Guide and Progress Towards an Atom Laser*. PhD thesis, University of Michigan, 2006.

[30] H. Wu and C. J. Foot. Direct simulation of evaporative cooling. *J. Phys. B*, 29:L321–L328, April 1996. doi: 10.1088/0953-4075/29/8/003 .

[31] S. E. Olson, M. L. Terraciano, M. Bashkansky, Z. Dutton, and F. K. Fatemi. Cold atom confinement in an all-optical dark ring trap. *Phys. Rev. A*, 76(6):061404, 2007. doi: 10.1103/PhysRevA.76.061404 .

[32] A. J. Christlieb, R. Krasny, and J. P. Verboncoeur. A Treecode Algorithm for Simulating Electron Dynamics in a Penning–Malmberg Trap. *Comp. Phys. Comm.*, 164(1–3):306–310, December 2004. doi: 10.1016/j.cpc.2004.06.076 .

[33] W. L. Brogan. *Modern Control Theory*. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1991.

[34] K. J. Åström and T. Hägglund. The future of PID control. *Control Engineering Practice*, 9(11):1163–1175, 2001. ISSN 0967-0661. doi: 10.1016/S0967-0661(01)00062-4 .

[35] P. M. Campbell, E. A. Carmona, and D. W. Walker. Hierarchical domain decomposition with unitary load balancing for electromagnetic particle-in-cell codes. In *Distributed Memory Computing Conference, 1990., Proceedings of the Fifth*, volume 2, pages 943 –950, apr 1990.

[36] J. S. Wu and K. C. Tseng. Parallel DSMC method using dynamic domain decomposition. *Int. J. Numer. Meth. Eng*, 63:37–76, 2005.