Scaling to 150K cores: recent algorithm and performance engineering developments enabling XGC1 to run at scale

Mark F Adams
Department of Applied Physics and Applied Mathematics, Columbia University
E-mail: mark.adams@columbia.edu

Seung-Hoe Ku
Courant Institute of Mathematical Sciences, New York University
E-mail: sku@cims.nyu.edu

Patrick Worley
Computer Science and Mathematics Division, Oak Ridge National Laboratory
E-mail: worleyph@ornl.gov

Ed D’Azevedo
Computer Science and Mathematics Division, Oak Ridge National Laboratory
E-mail: dazevedoef@ornl.gov

Julian C Cummings
Center for Advanced Computing Research, California Institute of Technology
E-mail: cummings@cacr.caltech.edu

C-S Chang
Courant Institute of Mathematical Sciences, New York University
E-mail: cschang@cims.nyu.edu

Abstract. Particle-in-cell (PIC) methods have proven to be effective in discretizing the Vlasov-Maxwell system of equations describing the core of toroidal burning plasmas for many decades. Recent physical understanding of the importance of edge physics for stability and transport in tokamaks has lead to development of the first fully toroidal edge PIC code – XGC1. The edge region poses special problems in meshing for PIC methods due to the lack of closed flux surfaces, which makes field-line following meshes and coordinate systems problematic. We present a solution to this problem with a semi-field line following mesh method in a cylindrical coordinate system. Additionally, modern supercomputers require highly concurrent algorithms and implementations, with all levels of the memory hierarchy being efficiently utilized to realize optimal code performance. This paper presents a mesh and particle partitioning method, suitable to our meshing strategy, for use on highly concurrent cache-based computing platforms.
1. Introduction

The effective simulation of burning plasmas is essential in the development of commercially viable fusion power, and the physical understanding and prediction capability of the edge plasma pedestal are very important in this tokamak plasma research. Many important phenomena in tokamak plasmas require a kinetic description as given by the Vlasov equation. The 6-dimensional Vlasov equation can be approximately reduced to a 5D (3 real space + 2 velocity) gyrokinetic Vlasov equation, which models the particle gyro-motion as a charged ring.[1, 2, 3] This simplification gives us orders of magnitude increase in timestep size in the simulation and makes it possible to simulate kinetic turbulence physics in tokamaks with modern-day supercomputers.

Particle-in-cell (PIC) methods have proven to be effective in discretizing the gyrokinetic Vlasov-Maxwell system for core tokamak plasmas for many decades[4, 5]. Recent physical understanding of the importance of edge physics in tokamaks has lead to the development of the first fully toroidal gyrokinetic edge PIC code to include the magnetic separatrix – XGC1[6]. XGC1 in its present form is a full-f electrostatic gyrokinetic ion-electron particle code specifically designed for edge plasmas. The simulation region of XGC1 can be from the magnetic axis to the material wall. The equations of motion from the discretized gyrokinetic Vlasov equation are

\[ \dot{X} = \frac{1}{D} [v \hat{b} + (v^2_B/B) \nabla B \times \hat{b} + \{B \times (\mu \nabla B - \bar{E})\}/B^2] \]
\[ \dot{v}_\parallel = -\frac{1}{D} (B + v \nabla B \times \hat{b}) \cdot (\mu \nabla B - \bar{E}), \]

where \( \dot{X} \) is the time derivative of ion gyro-center position, \( v\parallel \) is the speed of the particle parallel to the local magnetic field vector \( B \), \( \hat{b} = B/B \), \( \mu = v^2_B/2B \) is the magnetic moment, \( \bar{E} \) is the gyro-averaged electric field, and \( D = 1 + (v\parallel/B) \hat{b} \cdot (\nabla \times \hat{b}) \). The following gyrokinetic Poisson equation is solved for electrostatic potential \( \Phi \) with the four-point averaging technique[3, 7]:

\[ -\nabla_\perp \frac{\rho^2_i}{\lambda^2_{Di}} \nabla_\perp \Phi = e \left( 1 - \nabla_\perp \rho^2_i \nabla_\perp \right) (n_i - n_e), \]

where \( \rho_i \) is the ion gyroradius, \( \lambda_{Di} \) is the ion Debye length, and \( n_i \) is the ion gyro-center density in real space,

\[ n_i = \frac{1}{2\pi} \int f_i(x, \mu, u) \delta(X - x + \bar{p}_i) dX d\mu d\alpha. \]

Here \( x \) is the particle position vector in real space and \( \alpha \) is the gyro-phase angle.

XGC1 solves the above equations with particles and electric field data on a spatial grid. Each timestep of an XGC1 execution includes, minimally, the following stages:

(i) Collect particle charge density on underlying grid
(ii) Solve gyrokinetic Poisson equation on grid
(iii) Compute electric field and any derivatives needed in particle equations of motion
(iv) Calculate and output diagnostic quantities
(v) Update particle positions and velocities

This is a simplified view of the algorithm in that these steps are used within a Runge-Kutta or predictor-corrector time integration method. Depending on the experiment configuration, the particles may be ions, electrons, or both. Experiments can also include collisions and other physical processes important to full-device simulations.

The core tokamak plasma has well confined single-particle orbits without a particle sink or source. In this situation, the plasma is well conserved in phase space, and the perturbed...
distribution function (delta-f) method has been popular. The edge tokamak plasma has some critical differences from the core region. First, the magnetic coordinate system diverges in the presence of a divertor, which all modern tokamaks have been designed to have for controlling wall damage due to escaping hot plasma. The magnetic field geometry with divertor has a separatrix and X-point (see Figure 1). The poloidal field is vanishingly small near the X-point, and the magnetic coordinate system is undefined on the separatrix. Second, the existence of the magnetic separatrix and the material wall creates unconfined single-particle orbits[8, 9]. In the scrape-off layer, the plasma is ill-confined. Third, in the edge pedestal the gradient scale length is on the same order as the radial particle excursion width, and a large self-consistent radial electric field is formed. Due to these differences, XGC1 uses the full-f PIC method and a cylindrical coordinate system.

Strong magnetic guide fields generally make the turbulence structure approximately aligned along the field. Therefore, field aligned meshes allow a significant reduction in the number of cells (via fewer grid points in the toroidal or parallel direction) in tokamak simulations. The field aligned mesh can be easily obtained from a magnetic coordinate system. As previously noted,
however, the magnetic coordinate system has a singularity at the separatrix and is problematic in the diverted edge region. This paper introduces a meshing strategy that allows for semi-field aligned meshes with a cylindrical coordinate system that is both naturally axisymmetric and also effective in achieving the efficiencies of field aligned meshes.

One of the core computational kernels of a PIC code is the charge deposition process. Charge deposition requires that the interpolates of a charge at an arbitrary position in the tokamak domain be determined and the charge be deposited onto the grid that is used in the gyrokinetic Poisson equation to compute the electrostatic potential. This charge deposition can account for a significant fraction of the computation time in a PIC code. Our mesh method requires a 2D unstructured triangular mesh (e.g., Figure 2) that is used for each of the 16-64 poloidal planes (toroidal cross-sections) in the simulation. These unstructured meshes pose significant challenges in terms of computational efficiency due to the relatively complex nature of unstructured data. One of the computation bottlenecks of XGC1 is the determination of these interpolates. To ameliorate this problem, we have developed an efficient search method and code to find the triangles that a particle is contained in, once the particle has first been interpolated within the toroidal domain along a local magnetic field line.

Parallelization of XGC1 is based on decompositions of both the spatial grid and the particle data. In particular, the assignment of particles to processes is based on a decomposition of the spatial domain. Minimally, both the spatial grid and the particle decompositions utilize a one-dimensional decomposition in the toroidal direction of the tokamak geometry.

All but one of the basic stages during an XGC1 timestep requires MPI communication in the parallel implementation. The charge density collection requires MPI communication between processes assigned adjacent parts of the spatial grid in neighboring poloidal planes (neighboring slices in the one-dimensional toroidal decomposition), as well as two distributed reductions. The gyrokinetic Poisson solver uses parallel numerical algorithms supplied by the PETSc library [10]. MPI communication in the diagnostic routines is limited to gathers and reductions. The update of particle positions requires reassigning some of the particles to different processes.

The remainder of this paper is organized as follows. Section 2 presents a new mesh design for edge gyrokinetic methods. Section 3 describes the triangle search method implemented in XGC1. Section 4 describes a 3D particle and grid partitioning scheme suitable for XGC1 applications. Section 5 describes performance optimizations of the parallel implementation. Section 6 presents numerical experiments on leadership class Cray computers. Finally, we conclude in Section 7.

2. Meshing and coordinate system methods
The unperturbed magnetic field in XGC1 is symmetric in the toroidal direction, which is the direction of positive or negative cylindrical angle \( \phi \). The magnetic field can be decomposed into toroidal field \( \mathbf{B}_T = (\mathbf{B} \cdot \hat{\phi}) \hat{\phi} \) and poloidal field \( \mathbf{B}_P = \nabla \phi \times \nabla \psi \), where \( \psi = RA_\phi \), \( A_\phi \) is the \( \phi \) component of the vector magnetic potential \( \mathbf{A} \), and \( R \) is the radial coordinate. We can define a field-line following poloidal variable

\[
\lambda \equiv \int \frac{B_T}{RB_P} ds,
\]

where \( ds \) is the infinitesimal line segment on the surface of constant \( \phi \) and \( \psi \) [11]. The magnetic field can then be written as \( \mathbf{B} = \nabla \psi \times \nabla (\lambda - \phi) \), and \( \lambda - \phi \) is constant along the B-field. Unlike a magnetic coordinate system, \( \lambda \) is defined both inside and outside of the separatrix with a proper reference line, and it does not diverge on the separatrix except at the X-point. \( \lambda \) is used for the field-line following property of mesh generation, and it is calculated using the method of characteristics and an ordinary differential equation solver in a separate code from XGC1.

The mesh of XGC1 is chosen to be identical in each poloidal plane for simplicity, and the field-line following property is applied approximately. When we follow the field line, the change
in toroidal angle $\phi$ implies a change in $\lambda$, and $\Delta \phi = \Delta \lambda$ is satisfied. The grid points are the intersection of the field line with a poloidal plane. In the mesh of XGC1, $\Delta \phi = 2\pi/n$, where $n$ is the number of poloidal planes. Therefore, generating a grid point at every $\Delta \lambda = 2\pi/n$ on a flux surface will give a field-line following grid. Usually the grid spacing of $\Delta \lambda = 2\pi/n$ is much larger than the characteristic grid spacing except in the regions near the magnetic axis or X-point where the poloidal magnetic field vanishes. Reducing the grid distance in $\lambda$ space to $2\pi/nm$ (where $m$ is an integer) will still give the approximate field-line following property and keep the grid distance to the order of the characteristic grid spacing. In the ITER tokamak example, $m$ is around 10 in the middle of the magnetic axis and separatrix.

In the core region, the grid should satisfy poloidal periodicity, which means that the generation of grid points should return to the original point as it makes a complete loop in the poloidal plane. In the grid of XGC1, the grid spacing $\Delta \lambda$ is set to $\lambda_{2\pi}/N$, where $\lambda_{2\pi}$ is the change in $\lambda$ as it makes a loop on the poloidal plane, and $N$ is the nearest integer to $mn\lambda_{2\pi}/2\pi$. $N$ for the ITER grid is on the order of 100 to 1000 except in the region near the magnetic axis, and the field-line following error is less than 1%.

Even though $\lambda$ is well behaved on the separatrix and outside of it, it diverges at the X-point. The field-line following grid becomes too fine near the X-point. The field-line following property of the turbulence structure cannot be satisfied to arbitrary accuracy, so we can apply a cut-off of $1/B_P$ in Eq. 3 and redefine $\lambda$ as

$$\lambda \equiv \int \frac{B_T}{R \max(B_P, B_{P0})} ds. \quad (4)$$

This cut-off keeps the size of the grid near the X-point finite and reasonable without breaking the field-line following property, and the grid size is fully determined by the cut-off value $B_{P0}$.

The grid points generated in the above procedure are unstructured in the cylindrical coordinate system. We have used a triangle mesh to connect the grid points. The connectivity between nearest grid points in a flux surface is enforced. The TRIANGLE code\cite{12} is used for mesh generation. Figure 2 shows a sample triangle mesh for a very large grid size.

3. Mesh search algorithm

One key computational kernel is the deposition of charges by particles into this unstructured mesh. The Poisson solver computes the effective electric field using these accumulated charges as equivalent sources. The primary concern is, given a particle location $(x, y)$, to find the enclosing triangle and corresponding barycentric coordinates. If $(x_i, y_i)$ are the vertices of the triangle, then the barycentric coordinates (or area coordinates) of an interior point satisfy $0 \leq \lambda_i \leq 1$, $1 = \sum_{i=1}^{3} \lambda_i$, and $(x, y) = \sum_{i=1}^{3} \lambda_i(x_i, y_i)$. The barycentric coordinates can also be used for linear interpolation of a function $f(x, y) \approx \sum_{i=1}^{3} \lambda_i f_i$ where $f_i = f(x_i, y_i)$.

To speed up this search kernel, XGC1 uses geometric hashing into a background rectangular grid to locate a close initial triangle. Further testing of a small number of neighbor triangles is used to determine the target triangle. An optional preprocessing can relabel or reorder the vertices and triangles numbers by a Hilbert or Z-Morton space-filling curve to further improve the cache performance by encouraging better spatial locality.

4. Particle partitioning scheme

Modern supercomputers require highly concurrent (i.e., highly parallel) algorithms and implementations, with all levels of the memory hierarchy being efficiently utilized, to realize optimal performance. PIC codes present unique computational challenges in that the equations

\footnote{1 The TRIANGLE Delaunay triangulation software is developed by Jonathan Shewchuk and available at http://www.cs.cmu.edu/~quake/triangle.html}
to be solved are discretized with two very different methods: relatively common grid (cell) based
methods such as finite elements for Maxwell’s equations and particle methods for the Vlasov
equation. This leads to the need to optimize two very different processes and more importantly
the interaction between the two. The majority of the operations in a PIC code for tokamak
plasmas, especially with the full-f methods used in XGC1, are in particle processing. That is,
there are many more particles than grid points. Thus, the purely mesh-based tasks, such as
the Poisson solve, smoothing and field calculations, do not produce computational bottlenecks.
They can still impact performance, however, as discussed in the next section.

The particle processing is fairly straightforward, both in terms of serial and parallel
performance. One of the most challenging aspects of performance in PIC codes is the interaction
between particles and the grid. This is because the particles are distributed randomly throughout
the mesh, and the deposition of the charge on the grid requires (some degree of) random access
to an unstructured grid data structure. This random access poses significant performance
challenges due to inefficient use of the cache of common modern computer architectures.
Managing the cache effectively is critical to achieving high performance on our cache-based
machines. A common technique for managing caches effectively is to partition or block data into
pieces that can fit comfortably in as high a level of the cache as possible. (This suggests recursive
or multi-level methods to accommodate the multiple levels of cache in typical processors.)

We have developed an approach for particle decomposition that is effective in providing good
data locality, can be implemented efficiently, and is the first step in fully parallelizing the grid
in XGC1. Our approach takes advantage of the fact that our grid points are ordered using the
"natural" ordering on flux surfaces as discussed above. Grid points are ordered first by flux
surface, starting from the inner surface, and within each flux surface grid points are ordered by
poloidal angle. This results in an approximate spiral space-filling curve within the separatrix
that provides good data locality, and similarly good ordering in the region outside the separatrix.
Our algorithm defines a non-overlapping partitioning of the computational domain – the Voronoi
diagram of the grid points (or a nearest grid point partitioning). This algorithm is practical
because the triangle search method discussed in Section 3 is fast and can be used to compute
the nearest grid point to a coordinate.

Our algorithm first assigns weights to each grid point by counting the number of particles
in its domain. This weight is approximately a third of the area of all triangles that touch a
grid point with a uniform particle distribution, but we elect to explicitly compute the weight
to accommodate nonuniform particle distributions that develop as the simulation evolves. The
array of grid points is segmented to keep about the same total weight in each segment or
partition. Each partition is then mapped to an MPI process. After a particle is moved, in what
is called the push phase, its new coordinate is used to find its nearest grid point; the partition
or segment to which the grid point belongs determines the MPI process to which the particle
belongs. The particle data is then sent, if necessary, to a new processor in what is called the
shift phase. This particle decomposition strategy is an effective blocking strategy to improve
the data locality for cache sensitive processes such as the charge deposition, and it is the first
step in a fully parallelized XGC1 implementation.

5. Performance engineering
5.1. OpenMP
As mentioned in the previous section, computation solely on the mesh, such as the Poisson solve,
smoothing and field calculations, is dominated by the processing of the particles. However,
the runtime of phases associated with the mesh-only computation can limit performance when
running with large numbers of MPI processes.

For a given experiment (and a corresponding fixed mesh), the runtime for the mesh-only
computation stops decreasing when using more than a certain number of MPI processes, a
number dependent on the computing system and on the problem particulars. This follows simply from there being a fixed amount of work to do. The lower bound is a function of the MPI communication overhead, the amount of redundant computation, limited parallelism bottlenecks (such as serial fraction), load imbalance, etc. It is still profitable to use more parallelism by applying it to the particle processing phases, especially when the particle count grows with the processor count in a weak scaling mode.

Problems arise when the MPI communication overhead of the mesh-only phases begins growing with the MPI process count. This clearly holds for the Poisson solve. A mitigation technique that has been effective in other PIC codes within the fusion simulation community is to exploit OpenMP parallelism within SMP compute nodes, decreasing the total number of MPI processes to the total number of processor cores divided by the number of OpenMP threads per process. We are currently utilizing this approach on computer systems with multiprocessor compute nodes that support shared memory parallel programming paradigms such as OpenMP. In our experience, decreasing the number of MPI processes also improves stability and performance in all MPI communication phases when running on large numbers of processors, as it lessens the demand for MPI and related system resources.

Some of the particle processing is logically independent, and OpenMP can be applied to the loops over particles in these phases without code modifications. Other loops, such as those related to charge deposition, are reductions onto the mesh. In these cases, the particles are divided between the threads and thread-specific contributions to the reduction are computed. A final loop over the nodes of the mesh sums the thread-specific contributions. This loop over nodes can itself be parallelized using OpenMP. In fact, many loops over nodes or triangles of the mesh can be parallelized with OpenMP, further parallelizing the mesh-only work even when MPI overhead limits the amount of MPI parallelism that can be exploited.

This discussion glosses over a number of issues that arose during the introduction of OpenMP parallelism. In one case, the data structures being manipulated were arrays referenced via pointers and the compiler was not able to determine that these arrays were nonoverlapping, disabling any OpenMP parallelism. Passing in the arrays as subroutine arguments eliminated this particular problem. A more difficult issue arises from the memory-intensive nature of much of the particle processing work. As described earlier, a significant part of the work involves identifying the location of each particle in the mesh. Over time, the particles assigned to a process are effectively randomized, and performance is constrained by the corresponding randomized access to memory. Periodic sorting can improve the locality of memory accesses, but this in itself introduces additional cost. The decomposition of particles over MPI processes is an approximate bin sort, and improves performance at high process count. Using OpenMP to decrease the MPI process count runs counter to this, and many of the particle processing loops take less time when using MPI-only decompositions than with hybrid OpenMP/MPI parallel implementations. Currently the optimal number of MPI processes and OpenMP threads per process for a given total number of processor cores is determined empirically for a given system and problem specification.

5.2. MPI Communication Algorithms
With respect to performance, the most important MPI communication algorithms outside of those used by PETSc in the Poisson solve are those used to reassign particles between MPI processes. After the location of each particle is advanced, it is determined whether a particle has left the domain assigned to the process that currently owns the particle, and which process it needs to be reassigned to. The original algorithm for this communication was very efficient for small numbers of processes, but it failed, for example, on the Cray XT4 and XT5 systems when using more than 8000 processes.

The problem arose because the original algorithm posted send requests, did some local
processing, then posted receive requests. Many of the messages were arriving before the receive requests were posted and were going into MPI system buffer space. Eventually there was no more space available to receive the messages, and the program failed. Note that due to the nature of the simulations, the number of processes sending particles to any given process is typically not large, but the number of particles received can be large.

An alternative algorithm was developed that is equally efficient for small numbers of processes, and has been demonstrated to work on up to 64,000 processes on the Cray XT5. This new algorithm introduces explicit flow control: handshaking messages guarantee that receive requests on the destination processes are posted before send requests on the source processes, and the total number of outstanding send and receive requests on any given process at any given time is kept below a specified threshold.

6. Numerical results
The previous sections discussed the major algorithmic innovations that have been introduced into XGC1 to enable good computer performance on leadership class computing systems. This section describes recent results in benchmarking XGC1 performance on one of the largest computers available – the Cray XT5 jaguarpf machine at Oak Ridge National Laboratory. This system includes 18,722 compute nodes and a custom, three-dimensional torus (25x32x24) interconnect. Each compute node contains two 2.3 GHz quad-core Opteron processors (AMD 2356 Barcelona) linked with dual HyperTransport connections and 16 GB of DDR2-800 memory. Thus each compute node contains 8 processor cores and 2 GB memory per core, and an aggregate of 149,776 processor cores and 299,552 TB of memory for the entire system. Memory access performance is non-uniform (NUMA) at the node level, but uniform memory access can be enforced by restricting thread placement and associated memory to a single socket in the dual socket node architecture.

To investigate parallel scalability we conducted a series of weak scaling or scaled speedup studies in which the number of particles per processor core is kept constant (at 900,000). These studies use one mesh, a 3 mm mesh of the ITER edge region. So the mesh part of the work is a strong scaling study. The grid has 893,884 grid points and 32 poloidal planes are used in all cases. Only 10 time steps are used in these simulations; this is adequate to understand the performance of the code. While there is the potential for load imbalance to develop during real physics runs (with more than 10,000 time steps), we do not address this issue here.

The left graph in Figure 3 shows the time spent in the main time stepping loop of XGC1 and in the Poisson solver within that loop for a number of different processor core counts, from 2048 to 149,248 cores. Data is presented for MPI-only experiments, for experiments with 4 OpenMP threads per MPI process, and for experiments with 8 OpenMP threads per process. The total number of computational threads and the total number of particles are fixed for a given processor core count.

From these results we see that using OpenMP does decrease the cost of the Poisson solve. While the OpenMP parallelization of the loops over particles is not as efficient as using MPI only, using 4 OpenMP threads results in an overall increase in performance compared to the MPI-only experiments. MPI-only experiments attempting to use 131,072 or higher process counts failed. The source of the failure is still under investigation, but it is repeatable and is strictly a function of the process count.

When using 4 OpenMP threads, two MPI processes are assigned to each node. In this case, each process and its 4 threads are assigned to a single quad-core processor, and the memory subsystem as seen by each process and its threads is essentially Uniform Memory Access (UMA). In contrast, when using 8 OpenMP threads, a single process and 8 threads are assigned to a compute node, and performance suffers from the Non-Uniform Memory Access (NUMA) performance characteristics of the compute node memory subsystem. Improved memory
placement may address this performance problem for the 8 OpenMP thread experiments.

The right graph in Figure 3 breaks down where time is spent for the experiments with 4 OpenMP threads per process. This shows the maximum time that a processor spent in the four major phases of the code: 1) the charge deposition (charge), 2) the particle evolution (push), 3) the particle communication (shift) and 4) the Poisson solver (solve). Note that the solve phase is dominated by large all-to-all communication, which results in slightly increasing run times. These data show that the code is scaling very well with very nearly constant run time as the particle and processor count is increased.

Many of the optimizations discussed in this paper, such as the fast search routine and the particle decomposition strategy, are for serial performance and are not directly related to scalability. However, we have chosen to implement the particle decomposition strategy by placing one domain on each MPI process. Thus, as the number of processors is increased the number of particle domains increases as well, resulting in better data locality and better performance. This results in a decrease in memory-bound routines like the charge deposition routine shown in the right graph in Figure 3. In particular, time spent in the primary loop over particles in this phase decreases from 176 seconds with 2K cores to 130 seconds with 64K cores.

7. Conclusions
XGC1 is a new gyrokinetic particle code for tokamak plasma simulation which differs from other gyrokinetic codes in that it is a full-f code and includes the magnetic separatrix and scrape-off region, thus requiring a computationally intricate unstructured mesh. We have described here recent algorithmic advances in the XGC1 particle code that have resulted in our being able to scale to the edge of current extreme scale computing using about 150,000 cores. We have demonstrated new decomposition strategies for unstructured PIC methods and the effectiveness of using multi-threaded programming on a common extreme scale computer architecture of today – the Cray XT5. Future work will include continuing to parallelize the grid operations and data storage via a fully parallel or non-redundantly stored mesh.

Acknowledgments
This research was sponsored by the Office of Fusion Energy Sciences and by the Office of Advanced Scientific Computing Research, both in the Office of Science, U.S. Department of Energy. The research of D’Azevedo and Worley was performed at the Oak Ridge National Laboratory.

Figure 3. Run times for weak scaling with an ITER mesh and 900,000 particles per core, from 2048 to 149,248 cores.
Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05-00OR22725. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes. Mark Adams was funded under DOE grant DE-FC02-06ER54862.

This research used resources (Cray XT5) of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] Lee W W 1983 *Phys. Fluids* **26** 556  
[2] Lee W W 1987 *J. Comput. Phys.* **72** 243  
[3] Hahm T S 1988 *Phys. Fluids* **31** 2670  
[4] Lee W W and Tang W M 1988 *Phys. Fluids* **31** 612  
[5] Lin Z, Hahm T S, Lee W W, Tang W M and White R B 1998 *Science* **281** 5384  
[6] Chang C S, Ku S, Diamond P H, Lin Z, Parker S, Hahm T S and Samatova N 2009 *Phys. Plasmas* **16** 056108  
[7] Lin Z and Lee W W 1995 *Phys. Rev. E* **52** 5646  
[8] Chang C S and Ku S 2004 *Phys. Plasmas* **11** 2649  
[9] Ku S, Baek H and Chang C S 2004 *Phys. Plasmas* **11** 5626  
[10] Balay S, Buschelman K, Eijkhout V, Gropp W D, Kaushik D, Knepley M G, McInnes L C, Smith B F and Zhang H 2008 PETSc users manual Tech. Rep. ANL-95/11 - Revision 3.0.0 Argonne National Laboratory  
[11] Weitzner H 1981 *Phys. Fluids* **24** 2280  
[12] Shewchuk J R 1996 *Applied Computational Geometry: Towards Geometric Engineering (Lecture Notes in Computer Science* vol 1148) ed Lin M C and Manocha D (Springer-Verlag) pp 203–222 from the First ACM Workshop on Applied Computational Geometry