Balancing Particle and Mesh Computation in a Particle-In-Cell Code*

P. H. Worley
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
Email: worleyph@ornl.gov

E. F. D'Azevedo
Oak Ridge National Laboratory
Oak Ridge, Tennessee, USA
Email: dazevedoef@ornl.gov

R. Hager
Princeton Plasma Physics Laboratory
Princeton, New Jersey, USA
Email: rhager@pppl.gov

S-H. Ku
Princeton Plasma Physics Laboratory
Princeton, New Jersey, USA
Email: sku@pppl.gov

E. S. Yoon
Rensselaer Polytechnic Institute
Troy, New York, USA
Email: yoone@rpi.edu

C-S. Chang
Princeton Plasma Physics Laboratory
Princeton, New Jersey, USA
Email: cschang@pppl.gov

***Abstract*—The XGC1 plasma microturbulence particle-in-cell simulation code has both particle-based and mesh-based computational kernels that dominate performance. Both of these are subject to load imbalances that can degrade performance and that evolve during a simulation. Each separately can be addressed adequately, but optimizing just for one can introduce significant load imbalances in the other, degrading overall performance. A technique has been developed based on Golden Section Search that minimizes wallclock time given prior information on wallclock time, and on current particle distribution and mesh cost per cell, and also adapts to evolution in load imbalance in both particle and mesh work. In problems of interest this doubled the performance on full system runs on the XK7 at the Oak Ridge Leadership Computing Facility compared to load balancing only one of the kernels.**

***Keywords*-particle-in-cell; load balancing; optimization**

I. Introduction

The target code, XGC1, has been developed to model edge plasma (and its effect on the core plasma) in tokamak fusion reactors. XGC1 has many runtime options, but in one important configuration the primary computational kernels are the electron push, determining where electrons move during a timestep, and the nonlinear Fokker-Planck collision operator, calculating the effect of collisions between particles using the particle distribution functions in each mesh cell. For some production-like scenarios for this problem configuration the computational complexities of the electron push and the collision operator are comparable, but the optimal load balancing strategies are different, and optimizing just for one introduces significant load imbalances in the other, degrading performance. Both electron push and collision operator load imbalance evolve with the simulation, and the load balance for each may need to be adjusted to maintain good performance.

Load balancing the electron push has been dealt with effectively for a number of years, but the recent development and inclusion of the nonlinear collision operator forced us to revisit our approach. This included first determining the nature of the collision operator load imbalance, and then determining how to load balance using incommensurate metrics. Finally, the mechanism for determining when to update the electron push load imbalance did not generalize to the new combined algorithm, and a new approach was developed. The MPI communication algorithms used to implement the new algorithm also required careful attention.

The resulting algorithm has proven to be very effective at improving performance for production scenarios of XGC1 running on nearly the full Titan system, the Cray XK7 at the Oak Ridge Leadership Computing Facility (OLCF) system, as well as on Cray and IBM systems at the National Energy Research Scientific Computing Center and the Argonne Leadership Computing Facility, respectively.

II. XGC1

XGC1 is a full-F 5D gyrokinetic PIC/finite element code that models the whole volume plasma dynamics in an experimentally realistic tokamak magnetic confinement fusion device geometry, with a special strength in modeling the edge plasma in tokamak fusion reactors at a first-principles level.

XGC1 solves the gyrokinetic Vlasov equations[1], [2], [3] with particles and electric field data on a spatial computational mesh. For the purposes of this paper, each timestep of an XGC1 execution includes the stages described in Table I.

* This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan(http://energy.gov/downloads/doe-public-access-plan).

1) For each step of a Runge-Kutta algorithm:
   a) Collect particle charge density on underlying mesh.
   b) Solve gyrokinetic Poisson equation on mesh.
   c) Compute electric field and any derivatives needed in particle equations of motion.
   d) Calculate and output diagnostic quantities.
   e) Update particle positions and velocities.
      i) For electrons, subcycle (typically at least 60 times) with a fixed electric field (*electron push*).
      ii) For ions, advance one step (*ion push*).
   f) Move particles between processes, as required by updated positions (*particle shift*).
2) Based on a runtime parameter N, every Nth timestep:
   a) Calculate particle collisions and apply this and other source terms to adjust fields.

Table I
XGC1 TIMESTEP LOOP

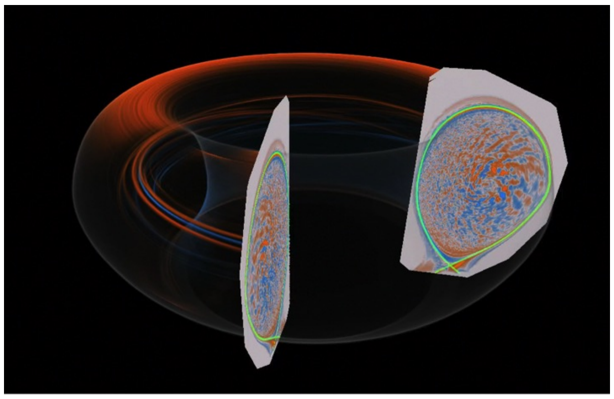


Figure 1. Cutaway view of simulated plasma flow in a tokamak fusion reactor.

(It can also be run with some options disabled and other options not mentioned enabled.)

Parallelization of XGC1 is based on decompositions of both the spatial computational mesh and the particle data across processes, and MPI is used to communicate between processes. The particle decomposition is based on the location in the spatial domain, and so is defined by the spatial domain decomposition, described in the next section. OpenMP is used to parallelize loops over particles and loops over mesh vertices or triangles. CUDA Fortran is also used to accelerate the electron particle position update on systems with Graphical Processing Units (GPU).

III. XGC1 COMPUTATIONAL MESH AND DECOMPOSITION

The XGC1 computational mesh for a toroidal domain consists of uniformly spaced poloidal planes (toroidal cross-sections) and an unstructured triangular tesselation within each poloidal plane, where each poloidal plane tesselation is identical. See Figures 1 and 2 for cartoons of the complete physical domain, a single poloidal plane, and the computational mesh for the poloidal plane.

As described in [4], mesh vertices in a poloidal plane are ordered first by *flux surface*, starting from the inner surface. Within each flux surface vertices are ordered by poloidal angle. This results in an approximate *spiral* space-filling curve ordering that preserves locality of the associated triangles and is roughly monotonic in the radial direction.

The full computational mesh is partitioned over a virtual two-dimensional processor array, with a "column" subset of processes sharing responsibility for a poloidal plane, and a "row" subset of processes sharing responsibility for one element of a one-dimensional partition of the mesh vertex index space. Load imbalances are controlled by adjusting the one-dimensional partition of the vertex indices, resulting in particle and vertex-related data being moved between processes in the same "column" for each adjustment. While some load

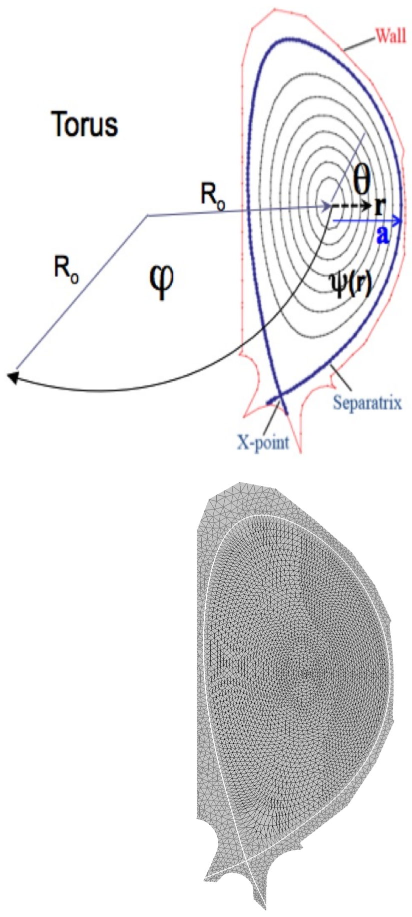


Figure 2. Geometry of poloidal plane for a tokamak fusion reactor and an example XGC1 computational mesh (coarsened for viewing).

imbalances occur in the toroidal direction (between poloidal planes), these develop and dissipate very quickly and are difficult to address effectively. Fortunately, addressing load imbalances in the radial and poloidal directions has been sufficient to enable good performance scalability.

IV. Target Systems

Results here were collected on the following systems (but primarily the Cray XK7).

- Titan: Cray XK7 sited at the Oak Ridge Leadership Computing Facility (OLCF), consisting of a Gemini interconnect and 18,688 computational nodes. Each node contains
  - one 2.2GHz 16-core AMD Opteron 6274 processor (with 8 floating point units, 1 per every two cores),
  - one NVIDIA K20 Kepler GPU with 6 GB memory,
  - 32 GB (non-GPU) memory.
- Edison: Cray XC30 sited at the National Energy Research Scientific Computing Center (NERSC), consisting of an Aries interconnect and 5,576 computational nodes. Each node contains
  - two 2.4GHz 12-core Intel Xeon ”Ivy Bridge” processors,
  - each of the 24 cores supporting 2-way hyperthreading,
  - 64 GB memory.
- Mira: IBM BG/Q sited at the Argonne Leadership Computing Facility (ALCF), consisting of a five-dimensional proprietary network and 49,152 computational nodes. Each node contains
  - one 1.6GHz 16-core IBM PowerPC A2 processor,
  - each of the 16 cores supporting 4 hardware threads,
  - 16 GB memory.
- Jaguarpf: Cray XT5 that was sited at the Oak Ridge Leadership Computing Facility (OLCF) before being de-commissioned. It consisted of a Seastar2+ interconnect and 18,688 computational nodes. Each node contained
  - two 2.6GHz 6-core AMD Opteron 2435 (Istanbul) processors per node,
  - 16 GB memory.

V. Prior Performance and Current Issues

The simulation capabilities, and computational complexity, of XGC1 have increased over the years.

A. Pre-2012

XGC1, equipped with electrostatic turbulence capability to begin with, was further developed during the 2005-2012 Department of Energy (DOE) project “Center for Plasma Edge Simulation” (CPES), resulting in a code with the following characteristics and capabilities.

- Lagrangian particle-in-cell code in 5D gyrokinetic phase space (3D configuration + 2D velocity)
- Solved gyrokinetic Vlasov equation with particle-momentum-energy conserving linear and nonlinear Coulomb collisions
- Used realistic geometry and boundary condition
  - World’s only kinetic code to include magnetic separatrix and material wall

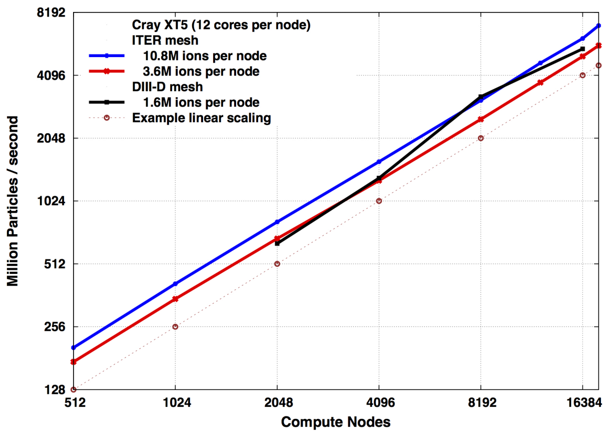
In this earlier work excellent performance scalability was achieved for both (a) weak scaling in particle count and strong scaling in mesh size and (b) strong scaling in both particle count and mesh size, out to the full system size of a series of Cray XT systems at the OLCF. This was made possible by utilizing a number of sophisticated computational methods and tools, including PETSc, geometric hashing, Hilbert space filling curve, hybrid MPI-OpenMP parallelization, bicubic spline, etc. New IO and workflow technologies were also developed to support production runs, in particular the ADIOS adaptable IO system [5], DataSpaces data-staging substrate, and the eSiMon dashboard (all included in the End-to-end Framework for Fusion Integrated Simulation (EFFIS)).

An example of this is shown in Figure 3, which describes weak scaling (in particle count) for two different computational meshes, one for the DIII-D [6] device and one for the ITER [7] device. For the ITER device mesh, weak scaling with two different particle counts per compute node was examined.

B. Introduction of Drift Kinetic Electrons (2012-2013)

In 2012, support for drift kinetic electrons was added to XGC1. When enabled, this changes the performance characteristics of XGC1 simulations significantly. The electron mass is much smaller than the ion mass, and the timestep for each electron push (calculating new particle positions) needs to be smaller. XGC1 uses an electron sub-cycling method and approximately 60 electron pushes per ion push. The actual cost of the electron push is even larger than this implies, reaching more than 90% of the computation time in CPU-only runs (and an even higher percentage of floating point operations (*flops*)). For example, the data in Table II are from 10 timestep runs using a production DIII-D device mesh on 32 nodes of the Cray XK7, without and with drift kinetic electrons. Each compute node started with 3.2 millions ions and, when using drift kinetic electrons, 3.2 million electrons. This is a representative particle workload per Cray XK7 compute node, and thus the ion/electron flop ratio described here is representative of production runs using more nodes and more total particles. (These runs do, however, underemphasize the communication overhead and overemphasize the mesh-based computation that would be found in production runs.)

For a similar configuration but on 16384 compute nodes (3.2 millions electrons and 3.2 millions ions per compute



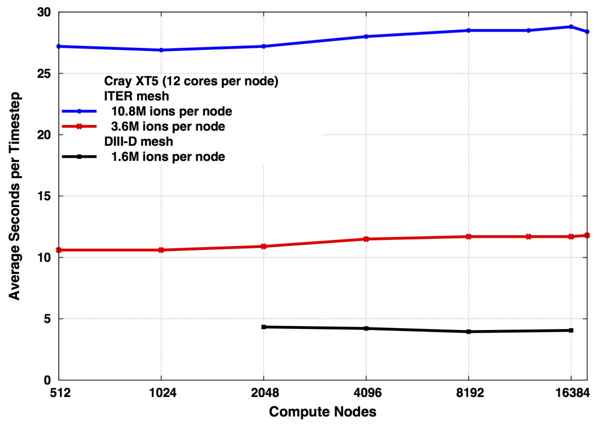


Figure 3. 2010 Weak Particle Scaling Performance of XGC1 performance on Cray XT5.

	Seconds (max. over processes)		Flop Count (total over processes)	
	without elec.	with elec.	without elec.	with elec.
timestep loop	54	1246	8.1×10^{12}	3.9×10^{14}
ion push	13	13	4.0×10^{12}	4.0×10^{12}
electron push		1142		3.8×10^{14}

Table II
WALLCLOCK TIME AND TOTAL FLOATING POINT OPERATION COUNTS WITHOUT AND WITH DRIFT KINETIC ELECTRONS FOR 10 TIMESTEPS OF XGC1 ON 32 NODES OF A CRAY XK7 WHEN USING 3.2 MILLION ELECTRONS AND 3.2 MILLION IONS PER NODE AND USING A PRODUCTION DIII-D DEVICE MESH.

node using a production mesh for the DIII-D device), the timestep loop total flop count for 10 timesteps was 2.01×10^{17} while the electron push total flop count for 10 timesteps was 1.97×10^{17}. So 98% of the floating point operations were in the electron push stage, similar to the results on 32 nodes.

This drift-electron push operation was an obvious target for acceleration using the GPUs available on the Cray XK7 architecture. This was implemented using PGI CUDA Fortran, and achieved approximately 4X speed-up. This did, however, require further optimizations to the associated communication algorithms to preserve good scalability for the whole code.

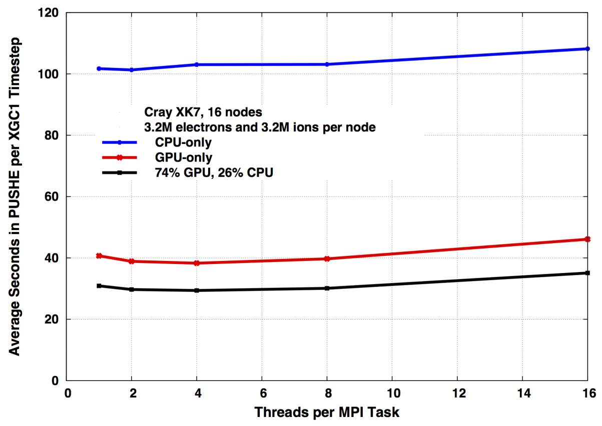


Figure 4. Comparison of performance of electron push routine (PUSHE) as a function of threads per MPI process for CPU-only, GPU-only, and hybrid CPU/GPU implementation

Figure 4 describes the performance of the electron push (PUSHE) for a CPU-only implementation, a GPU-only implementation, and an implementation that used both (74% of the particles pushed on the GPU and 26% on the CPU). The data are from XGC1 running on 16 nodes of a Cray XK7 and performance is reported as the maximum time spent in the electron push across all processes. (Existing MPI barrier calls within XGC1 indicate that this is an accurate metric for PUSHE performance.) The CPU implementation uses OpenMP to parallelize the electron push. For the GPU implementation, multiple MPI processes on a node can assign work to the GPU. The data also indicates the relative insensitivity to the choice of number of OpenMP threads per MPI process as long as the total number of computational threads is constant. This is important, as it allows the OpenMP thread count to be set so as to optimize other aspects of XGC1 performance.

The impact of the port to the GPU on whole code performance is described in Figure 5. Note the importance of also optimizing the MPI communication algorithms after the introduction of drift kinetic electrons. Performance of this version of the code also scaled well on other platforms. For example, Figure 6 describes the results of a strong scaling study. The x-axis here is compute node count, even though the compute nodes are different enough to make direct comparison between platforms problematic.

Note that, for these science cases, equidistributing the electrons between the processor rows (minimizing the maximum count) adequately addressed electron push-related load imbalances.

C. *Introduction of New Collision Operator (2014-2015)*

In 2014, support for the nonlinear Fokker-Planck collision operator was added to XGC1 [8], [9]. When enabled, this (once again) changes the performance characteristics of XGC1 simulations, potentially significantly. Unlike the

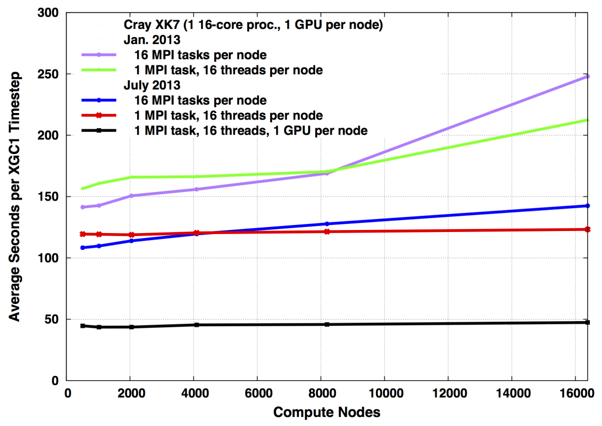


Figure 5. Evolution of XGC1 performance on Cray XK7 from Jan. 2013 to July 2013 as MPI communication algorithms were optimized and electron push operator was ported to GPU. Performance is for weak scaling in particles (3.2 million ions and 3.2 million electrons per compute node) and strong scaling in mesh (for DIII-D device), for 16 MPI processes per node and for 1 MPI process and 16 OpenMP threads per node.

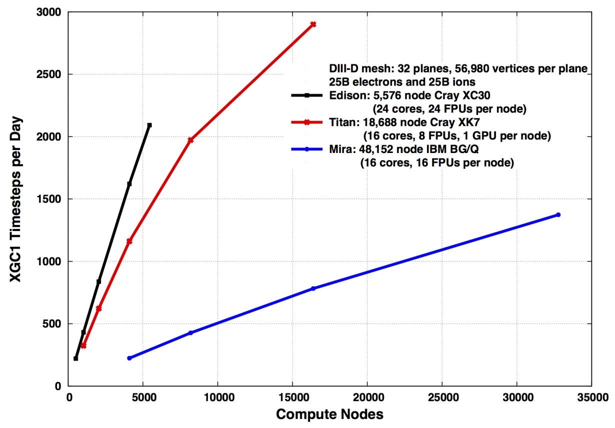


Figure 6. Strong scaling study for XGC1 on Cray XC30, Cray XK7, and IBM BG/Q using representative production size problem.

electron push, the cost of this collision operator scales with the mesh size, not the particle count. It is also a function of how often it is called (every XGC1 timestep, every third timestep, ...) and whether axisymmetry (between the mesh planes) is assumed. The choice of frequency and whether to exploit symmetry is determined by the physics question being addressed, and there are simulations that require computing the collision operator every timestep and without symmetry.

For example, Table III lists the total floating point operation count for 5 timesteps of a representative science configuration (DIII-D device mesh using 32 planes, 26 billion total ions, and 26 billion total electrons run on 8192 compute nodes of a Cray XK7) when computing collisions every XGC1 timestep and not exploiting axisymmetry. In this example the collision operator is more expensive, in terms of floating point operations, than the electron push. However, collision floating point operations would be divided by 32 in this example if assuming axisymmetry, or by 3 if, for example, computing every third timestep.

	Floating Point Operation Count
timestep loop	14.9×10^{15}
electron push	5.4×10^{15}
nonlinear collision operator	7.5×10^{15}

Table III
TOTAL FLOATING POINT OPERATION COUNTS FOR 5 TIMESTEPS OF A REPRESENTATIVE XGC1 PRODUCTION RUN USING DRIFT KINETIC ELECTRONS, NONLINEAR COLLISION OPERATOR, 26 BILLION ELECTRONS, 26 BILLION IONS, AND A DIII-D DEVICE MESH, ON 8192 NODES OF A CRAY XK7.

The collision operator is local to each mesh cell. Disregarding load imbalance, performance then scales with the number of processes for a fixed size mesh. Nested OpenMP parallelism is implemented in the collision operator for performance portability and to expose additional parallelism. OpenMP at the outermost loop is usually a little more efficient, but requires more memory, and so a mixed strategy works best on the IBM BG/Q. On the Cray XC30 both inner and outer are equally efficient. Given the computational intensity, the nonlinear collision operator is a prime candidate for acceleration on the GPU. This is currently being investigated using OpenACC.

VI. LOAD IMBALANCES

As ions and electrons are pushed, load imbalances will appear for any fixed mesh decomposition. The particle velocity is such that load balancing in the toroidal direction is impractical, but also of little consequence in current simulations. The movement of particles in the radial or poloidal directions is much slower, and updating the one-dimensional decomposition of the space-filling-curve ordering of the mesh vertices has proven very effective for load balancing particle counts, and thus both particle-related computational costs and memory requirements.

While the cost of computing the nonlinear collision operator is local to a mesh cell, it is not uniform across the mesh. One aspect of the cost is the number of iterations required for convergence of a Picard iteration, and the number of iterations is dependent on the temperature of the plasma in the cell. Because the temperature distribution evolves with the simulation, the corresponding cost also varies. However, temperature will generally be higher in some parts of the domain than in others throughout the simulation, and does not typically change its distribution very quickly. See, for example, Figure 7 for snapshot of Picard iteration count as a function of location during a simulation. The data are from a run assuming axisymmetry, but the results are representative of non-axisymmetric cases as well.

Like the particle distribution load imbalance, the collision cost load imbalance can be treated very effectively by updating the one-dimensional decomposition of the mesh. The collision cost (wallclock time to compute) can be measured per mesh cell for each poloidal plane, the maximum across

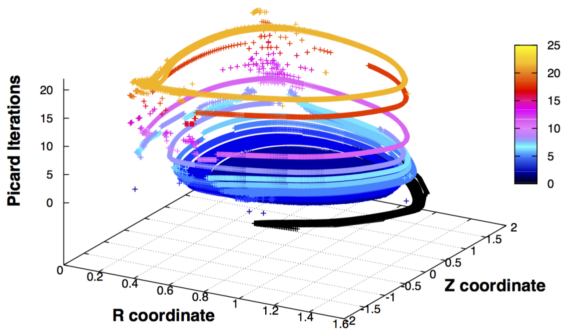


Figure 7. Picard iterations required for convergence in nonlinear collision operator as a function of location in poloidal plane for an example problem and simulation time for a production DIII-D device mesh.

the planes computed, and the one-dimensional decomposition adjusted to approximately equidistribute this metric of the cost across processes.

The same mesh decomposition is used for both the particle pushing and for the nonlinear collision operator calculation, and what is optimal for one is not optimal for the other. This would be true even if collision cost was uniform across the mesh. For example, see Figure 8. Here the initial particle distribution is load balanced, and we plot the number of mesh vertices assigned to each process. As this is identical for processes assigned the same element of the mesh decomposition but in different poloidal planes (having the same *interplane id* in the virtual processor array), it is more convenient to plot the sum of these vertices by the interplane id.

Figure 9 is a similar analysis, but now looking at the number of Picard iterations required per process and per the subset of processes with the same interplane id, using the same axisymmetric example. Note that when assuming axisymmetry the Picard iteration is calculated only for one poloidal plane, and the vertices in a given element of the partition are further decomposed across the processes with the same interplane id (and thus there is no redundant calculation). As these figures demonstrate, this part of the collision cost is neither uniform over the mesh nor similar to the particle count distribution, and load balancing particles only introduces yet a different load imbalance.

VII. Hybrid Load Balancing

When neither particle push nor the nonlinear collision operator dominate the computational cost (as measured in wallclock time), a compromise mesh decomposition must be determined, and updated as both collision and particle cost evolve, otherwise performance will be lost. The type of analyses described in the previous section did not provide sufficient insight for us to develop a physics-based *hybrid* load balancing scheme.

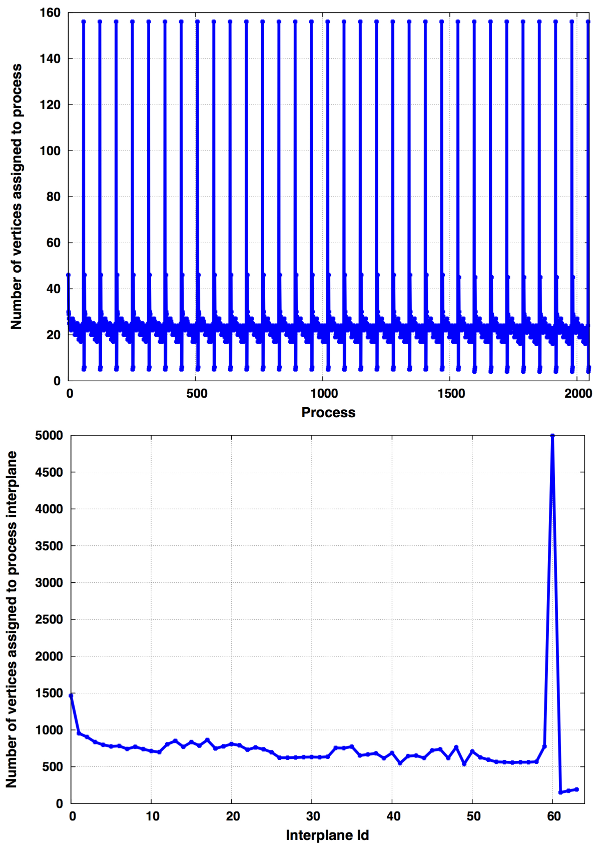


Figure 8. Number of vertices assigned to each process, and then to each subset of processes with the same interplane id, when load balancing the particle distribution for an example problem and simulation time for a production DIII-D device mesh.

The next approach considered was equidistibuting some weighted measure of particle-related and collision computation cost. We were also unsuccessful with this. It did not make sense to try to combine particle count per mesh cell and collision wallclock time per mesh cell. And, while the particle count load imbalance is an excellent predictor of particle computation cost load imbalance, predicting the actual particle computation wallclock time from the particle distribution is not simple (and we were not successful in doing so accurately enough for use in such an optimization strategy.). There are many loops over particles throughout the code, with synchronization points in between, and the particle load imbalance also affects the MPI communication overhead in the code.

The next approach was to equidistribute the nonlinear collision wallclock time, but subject to an upper bound on the particle load imbalance. This was a simple generalization of the existing one-dimensional optimization algorithm, and it worked. However, there is no guarantee that any particular upper bound on the particle load imbalance will improve performance significantly over just load balancing particles or load balancing collision wallclock time. This approach did however provide a single variable to use in yet another

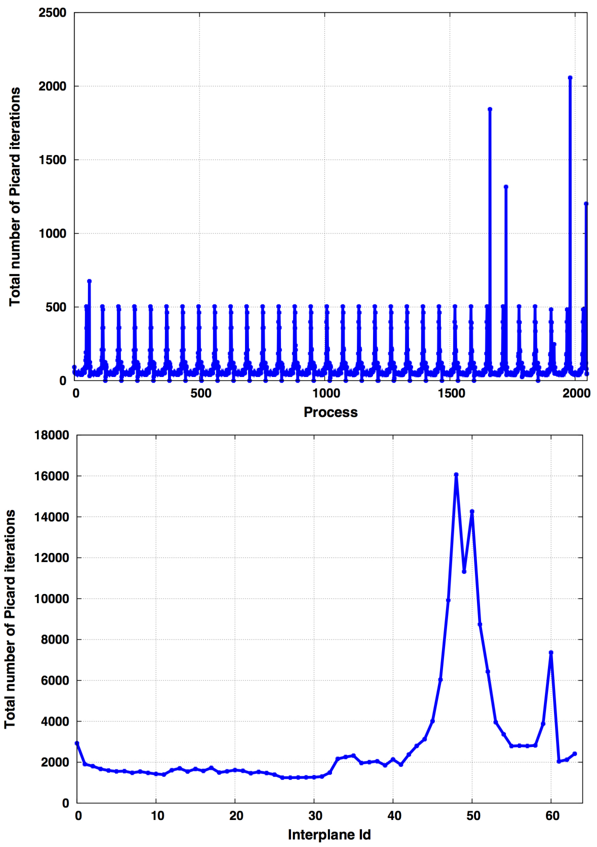


Figure 9. Number of Picard iterations calculated by each process, and then by each subset of processes with the same interplane id, when load balancing the particle distribution for an example problem and simulation time for a production DIII-D device mesh.

optimization algorithm.

Our almost final approach was to use a Golden Section Search algorithm [10] to adjust the upper bound on the particle load imbalance. The metric to be optimized was the observed wallclock time for the entire code between collision timesteps. So, if nonlinear collisions are computed every third timestep, for example, then the optimization takes into account the cost of the particle-related computation for the intermediary timesteps as well. Note that this means that the Golden Section Search algorithm proceeds with the XGC1 simulation, saving wallclock time data associated with previous particle load imbalance constraints (saving history for the three previous values), and we do not optimize exactly for any given timestep. It is not possible to optimize the performance based on the information for a given timestep because the impact of changing the particle load imbalance bound can only be determined empirically, and it would be a significant waste of resources to recompute an existing timestep. Rather, the Golden Section Search algorithm is used to estimate improvements to the constraint for future timesteps.

The algorithm as currently implemented differs from the above description in the following ways.

- The constraint on the particle load imbalance is actually a constraint on a relative increase in the minimum particle load imbalance possible. So the first step is to calculate the particle-only load balanced mesh decomposition, determine the maximum load imbalance for this (a perfect load balance is almost never possible by partitioning the mesh vertex ordering), and multiply this by α, where α is the variable manipulated by the Golden Section Search. Note that this implies that a lower bound on α is 1.
- As both particle and collision cost distributions change over time, the wallclock time measured for a previous value of α, call it α_i, will not be accurate in the future, leading to nonoptimal solutions from the Golden Section Search. If the wallclock time for this α_i has been used 3 times without being recalculated, then the next iteration of the Golden Section Search reuses this value, so as to update the associated wallclock time. Note that this also enables the optimization process to recover from a performance perturbation that would otherwise skew the α selection process.
- Similarly, to avoid finding a minimum from which the Golden Section can not escape even when the distributions evolve, the implemented algorithm looks outside of the bounding interval once the interval becomes too small.

The current inputs to the algorithm are

1) lower bound on α. (Default is 1.0.)
2) upper bound on α. (Default is 2.0.)
3) initial value for α. (Default is the user specified particle load imbalance constraint, which default is 1.1)
4) frequency of Golden Section Search updates, in terms of number of nonlinear collision timesteps. (Default is 0, so disabled.)

Note that there is still a user-specifed upper bound on the particle load imbalance, if only to prevent memory issues. (This bound is relative to the maximum particle load imbalance after the last rebalance.) If this bound is exceeded, then the hybrid load balancing algorithm is invoked, using the existing constraint value (α). That is, the Golden Section Search algorithm is only invoked to update the constraint at the user indicated timesteps, not whenever the load balancing algorithm is executed.

These input parameters, and the somewhat awkward relationship to the “other” upper bound on particle load imbalance, are topics for future improvements. However, reasonable settings for these parameters have been determined through experimentation. At worst, nonoptimal settings of the parameters will either slow the convergence (because the interval is unnecessarily large or the frequency is too low) or miss the optimum by the interval not containing the optimum, neither of which will affect the validity of the simulation. Too frequent invocations of the Golden Section Search algorithm also do not improve the solution, and there is some overhead to the redistribution of particles and mesh information after repartitioning.

Figure 10 contains three plots, the first comparing the distribution of mesh vertices for an example problem for a uniform decompostion, for a particle-only load balance, and for the hybrid particle count and collision cost load balance. As before, this is summed across the processes with the same interplane id. The second compares the particle distribution of the particle-only and hybrid load balanced decompositions. The third compares the collision wallclock time distribution for the particle-only and hybird load balanced decompositions. The second plot indicates how much particle count load imbalance is being introduced by the hybrid load balancing scheme, and the third indicates how much collision cost load imbalance exists if only load balancing particles. The hybrid load balancing data were collected after three Golden Section Search adjustments where the adjustments occured every other timestep that nonlinear collisions were calculated. Other experiment details are listed in the figure legends.

The previous examples confirm that the hybrid load balancing is doing something that appears reasonable. The next example documents the performance advantage of the new load balancing scheme. A production XGC1 run on the Cray XK7 at the OLCF (Titan) was simulating the plasma on an ITER device computational mesh (approximately 1 million vertices per plane, and 32 planes) with 131 billion electrons and 131 billion ions total. It was run using 16384 compute nodes, with 16384 processes, 16 OpenMP threads per process, and the GPU to accelerate the electron push (75% of the particles on the GPU and 25% on the CPU). To accurately simulate the transient at start-up, the non-axisymmetric version of the nonlinear collision operator was computed every XCG1 timestep. Hybrid load balancing was invoked every other timestep. Based on tuning results on the XC30 system at NERSC (Edison), the simulation began with using all 16 OpenMP threads for the inner loops in the nested OpenMP. Delays in getting the job started (tracking down a bad compute node) precluded tuning this on Titan before the simulation started. However, it was noticed during the run that the performance of the collision operator was poor. The job was stopped after 77 timesteps, and then restarted from the most recent checkpoint (after timestep 75). For the restart, 8 OpenMP threads were assigned to the outer loop (leaving 2 inner threads per outer thread for the inner loops). This improved performance significantly. (Note however that the reason that the original configuration was so very slow was that the jobs scripts mistakenly did not include the setting

```
export OMP_MAX_ACTIVE_LEVELS=2
```

and thus no threading was enabled in the collision operator during the first run except what was exploited in the thread-aware math library routines. Only 8 threads were active during the second run. Later runs changed the number of outer threads to 16.)

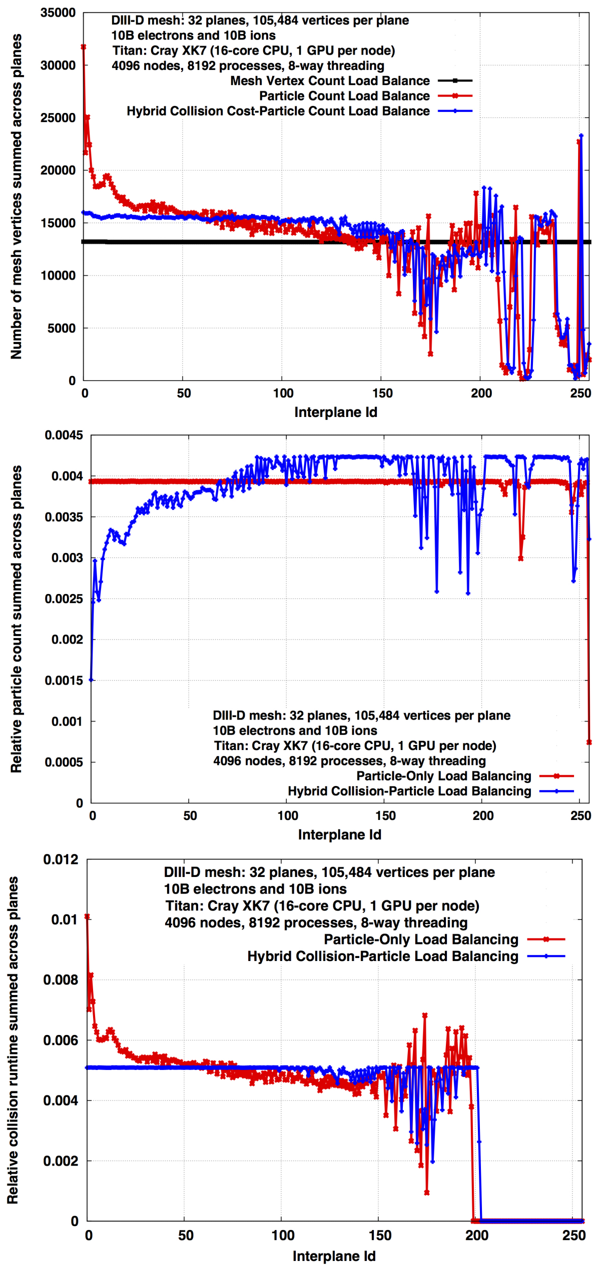


Figure 10. Comparison of impact of particle-only and hybrid particle count and collision cost load balancing on load balance for mesh vetex, particle count, and collision wallclock time distributions, respectively. Counts and time are summed across the planes for the same interplane id. Experiment particulars described in figure legends and in text.

This unusual situation provided a way to demonstrate the impact of the hybrid load balancing scheme. The first step of a start-up run uses particle-only load balancing. The first step of the restart run uses the mesh decomposition captured in the checkpoint. Since the cost of the collision

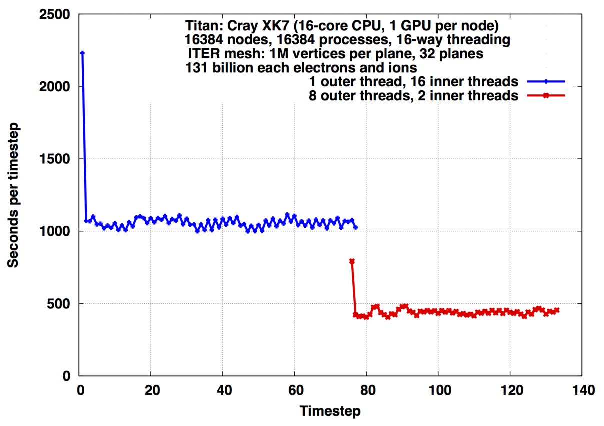


Figure 11. Wallclock time per XGC1 timestep as a function of the timestep for two runs of the ITER device science case. The two runs differ by the amount and type of thread-level parallelism exploited in the nonlinear collision operator. The second run is a restart of the first at timestep 75.

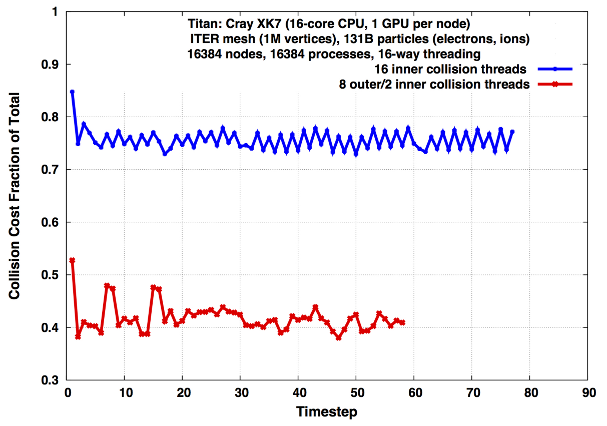


Figure 12. Fraction of total wallclock time spent in the nonlinear collision operator as a function of the XGC1 timestep for the two runs of the ITER device science case.

operator changed dramatically between the two runs, the hybrid load balancing scheme was presented with significant load imbalances at the beginning of both runs. (The Golden Section Search history is not checkpointed, which was an advantage in this case as the data from the first run was not representative of the second run.)

Figure 11 describes the performance of these runs in terms of seconds per XGC1 timestep as a function of timestep. It can be seen immediately the impact of the hybrid load balancing (and the impact of changing the number of outer OpenMP threads in the collision operator from 1 to 8). Performance was not particularly sensitive to the Golden Section Search parameter after the first step, but the overhead of load balancing every other step was also not significant in these experiments.

Figure 12 contains a plot of the fraction of the total cost represented by the collision operator. Note that load balancing decreases this somewhat. For ease of comparison, the second run is plotted with its first timestep labelled as timestep 1. Figure 13 is a plot of the particle load imblance

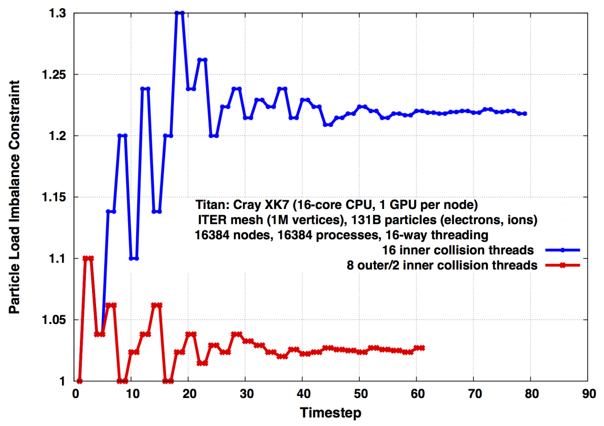


Figure 13. Evolution of the Golden Section Search optimization parameter (constraint on maximum electron particle load imbalance) as a function of the XGC1 timestep for the two runs of the ITER device science case.

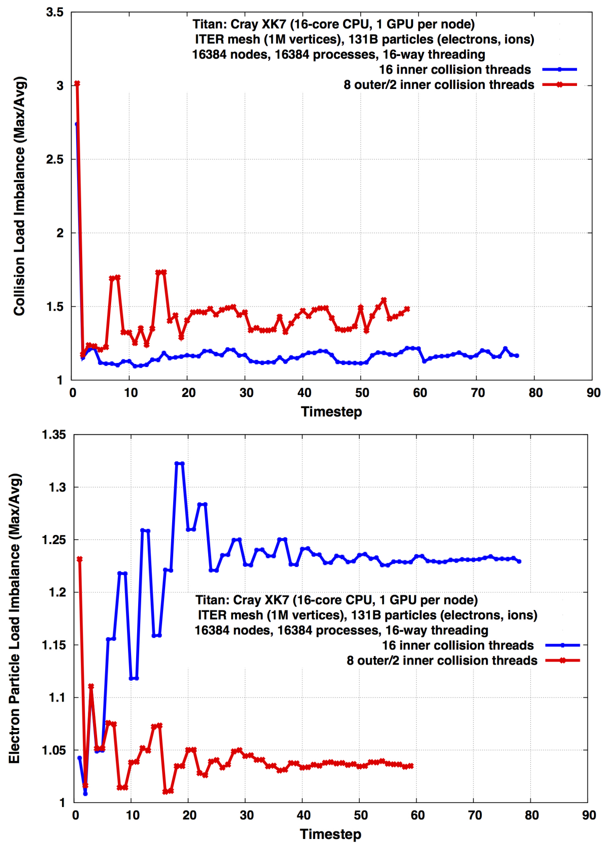


Figure 14. Evolution of the electron particle load imbalance and of the nonlinear collision operator wallclock time load imbalance as a function of the XGC1 timestep for the two runs of the ITER device science case.

constraint used by the Golden Section Search. Note that the first three values are the same for the two runs, as these are purely a function of the user input describing the search domain and the initial parameter. After that they diverge. Despite the very different values for later iterations of the Golden Section Search, most of the benefit in these runs was from the constrained optimization using a fairly tight particle load imbalance constraint (as can be seen in Figure 11). Figure 14 contains two plots, one documenting the evolution of the collision wallclock time load imbalance (ratio of the maximum over processes to the average) and the evolution of the electron particle count load imbalance. Note that the Golden Section Search parameter is closely related to the electron particle count load imbalance, but is not identical.

In summary, for this experiment, the hybrid load balancing was very effective, doubling performance in both runs, and was able to adapt to the extreme change in performance characteristics when transitioning between the two. When the particle push cost or the collision operator cost dominates the other, the Golden Section Search will not do much, once it determines which of the two is dominant. However, it also will do no harm, as long as the overhead of tweaking the mesh partition is not significant (which is true in these examples).

VIII. Conclusion

New science capabilities are continuing to increase cost and change performance characteristics of the XGC1 code, and these characteristics are very sensitive to the choice of the many options. In particular, the recent addition of the nonlinear Fokker-Planck collision operator required re-engineering the load balance scheme to address both mesh-based and particle-based load imbalances simultaneously, to respond to evolutionary changes, and to deal robustly with performance outliers (whether due to simulation or to externally caused performance variability). The current algorithm has more than doubled performance for example production runs. Future work will investigate automatic determination of appropriate load balancing frequency, and elimination of the need to set upper and/or lower bounds.

Acknowledgements

Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research and Fusion Energy Sciences, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC, No. DE-AC02-09CH11466 with Princeton University, and DE–SC0008449 with Rensselaer Polytechnic Institute.

Awards of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program. This research used resources of the Oak Ridge Leadership Computing Facility (OLCF), Argonne Leadership Computing Facility (ALCF), and National Energy Research Scientific Computing Center (NERSC), which are U.S. DOE Office of Science User Facilities supported under contracts DE-AC05-00OR22725, DE-AC02-06CH11357, and DE-AC02-05CH11231, respectively.

References

[1] W. W. Lee, "Gyrokinetic approach in particle simulation," *Phys. Fluids*, vol. 26, no. 1, pp. 556–562, 1983.

[2] ——, "Gyrokinetic particle simulation model," *J. Comput. Phys.*, vol. 72, no. 1, pp. 243–269, 1987.

[3] T. S. Hahm, "Nonlinear gyrokinetic equations for tokamak microturbulence," *Phys. Fluids*, vol. 31, pp. 2670–2673, 1988.

[4] M. Adams, S. Ku, E. D'Azevedo, J. Cummings, and C.-S. Chang, "Scaling to 150k cores: recent algorithm and performance engineering developments enabling XGC1 to run at scale," *Journal of Physics: Conference Series*, vol. 180, no. 1, p. 012036, July 2009. [Online]. Available: http://stacks.iop.org/1742-6596/180/i=1/a=012036

[5] Q. Liu, J. Logan, Y. Tian, H. Abbasi, N. Podhorszki, J. Y. Choi, S. Klasky, R. Tchoua, J. Lofstead, R. Oldfield *et al.*, "Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014.

[6] "DIII-D." [Online]. Available: https://fusion.gat.com/global/DIII-D

[7] "The ITER Tokamak." [Online]. Available: https://www.iter.org/mach

[8] E. S. Yoon and C. S. Chang, "A Fokker-Planck-Landau collision equation solver on two-dimensional velocity grid and its application to particle-in-cell simulation," *Physics of Plasmas (1994-present)*, vol. 21, no. 3, p. 032503, 2014.

[9] R. Hager, E. Yoon, S. Ku, E. D'Azevedo, P. Worley, and C. Chang, "A fully non-linear multi-species Fokker-Planck-Landau collision operator for simulation of fusion plasma," *Journal of Computational Physics*, vol. 315, no. 15, pp. 644–660, 2016.

[10] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press, 1988.