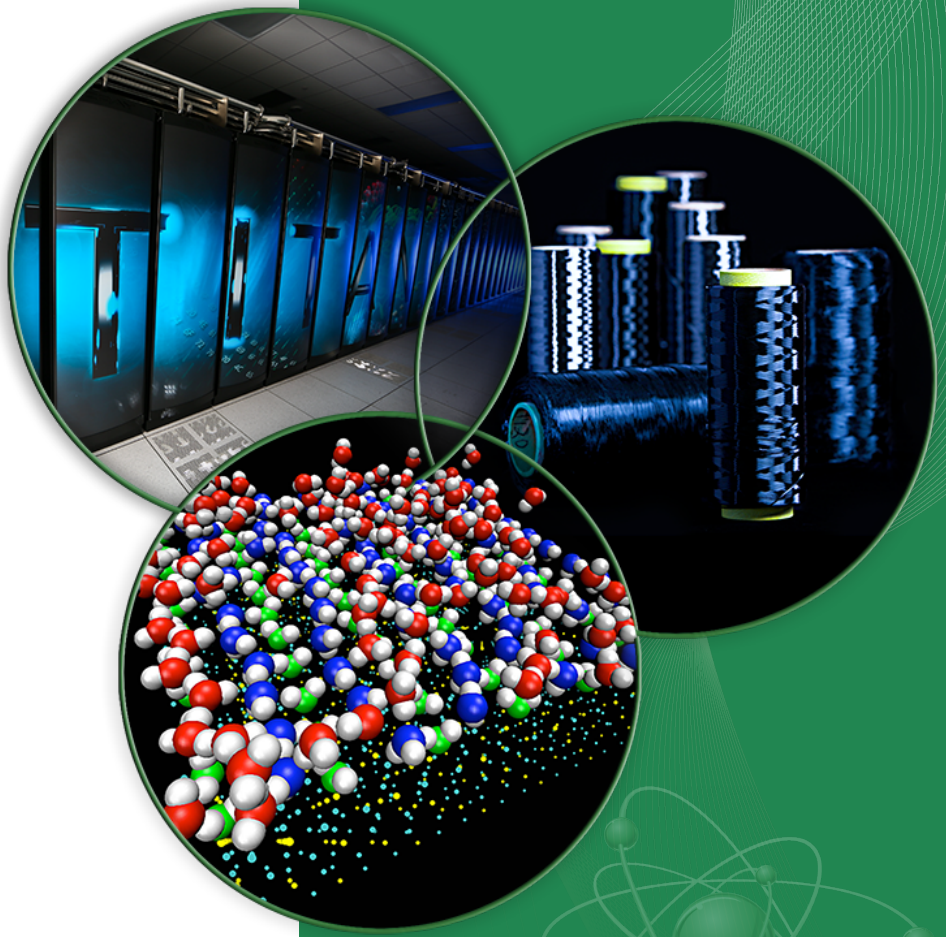
Balancing Particle and Mesh Computation in a Particle-In-Cell Code

Patrick H. Worley

Oak Ridge National Laboratory



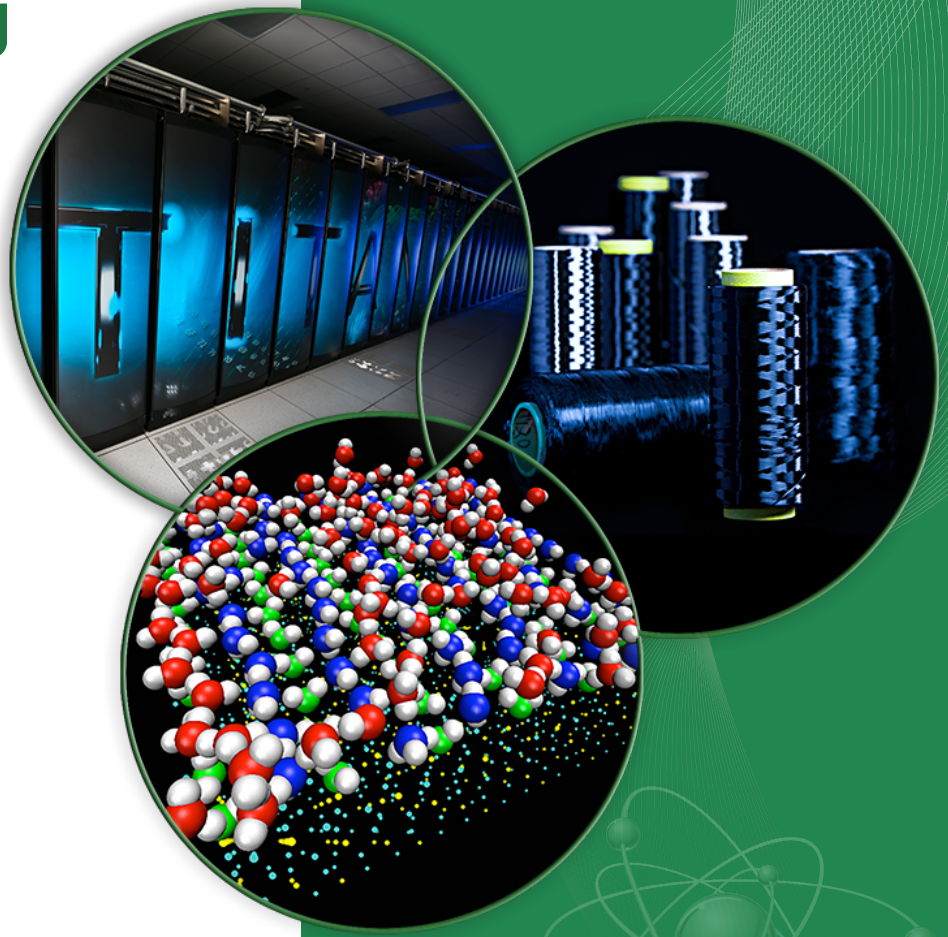
May 12, 2016

CUG 2016

Grange Tower Bridge Hotel

London, UK

Alternate Title: Issues in Maintaining Scalability as Application and (Cray) Architectures Evolve



Patrick H. Worley

Oak Ridge National Laboratory

May 12, 2016

CUG 2016

Grange Tower Bridge Hotel

London, UK

Co-Authors

- Eduardo F. D'Azevedo (Oak Ridge National Laboratory)
- Robert Hager (Princeton Plasma Physics Laboratory)
- Seung-Hoe Ku (Princeton Plasma Physics Laboratory)
- Eisung Yoon (Rensselaer Polytechnic Institute)
- Choong-Seock Chang (Princeton Plasma Physics National Laboratory)

Acknowledgements

- Support for this work was provided through the Scientific Discovery through Advanced Computing (SciDAC) program funded by the **U.S. Department of Energy** (DOE) Office of Advanced Scientific Computing Research and the Office of Fusion Energy Sciences, as part of the projects "**Center for Edge Physics Simulation (EPSi)**" and "**Institute for Sustained Performance, Energy, and Resilience (SUPER)**". The work was performed at Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract No. De-AC05- 00OR22725, at Princeton Plasma Physics Laboratory, which is managed by Princeton University under Contract No. DE- AC02-09CH11466, and at Rensselaer Polytechnic Institute under Contract No. DE-SC0008449.

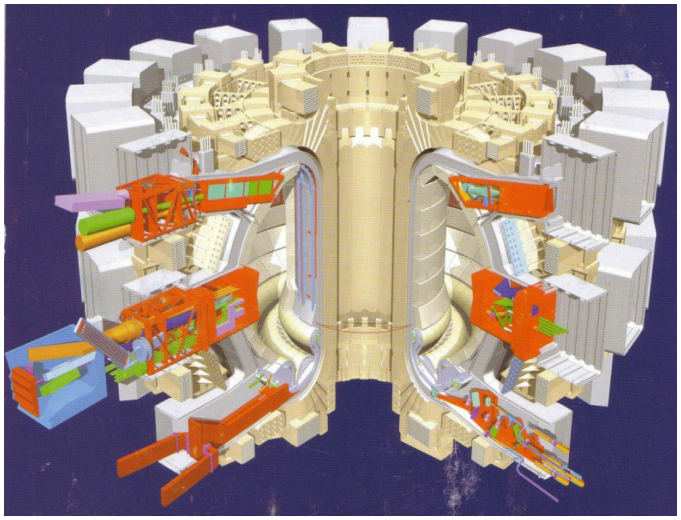
- This work used resources of the **Oak Ridge Leadership Computing Facility** (OLCF), which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This research also used resources of the **Argonne Leadership Computing Facility** (ALCF), which is a DOE Office of Science User Facility supported under contract DE-AC02-06CH11357. This research also used resources of the **National Energy Research Scientific Computing Center**, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 . Awards of computer time at ALCF and OLCF were provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program.

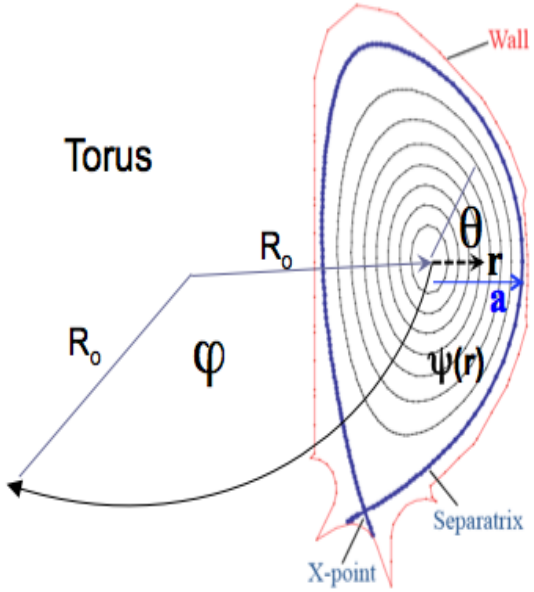
Background Science

- Edge simulation (near outer wall) is critical for tokamak-based fusion reactor design because the edge plasma conditions determine (i) core plasma quality, thus the fusion efficiency, and (ii) wall deterioration, thus the reactor lifetime
- Accurate simulation is difficult and requires effective exploitation of leadership computing systems:
  - Edge plasma is in contact with wall and density has a very steep gradient, invalidating fluid approx. and requiring treatment of non-equilibrium physics
  - Complicated geometry, currently addressed using unstructured triangular mesh
  - Multiscale physics: background physics, microturbulence, neutrals and atomic physics

  ⇒ Unlike for simulations of just the core plasma, the scale-separated "delta-f" perturbation method is invalid.

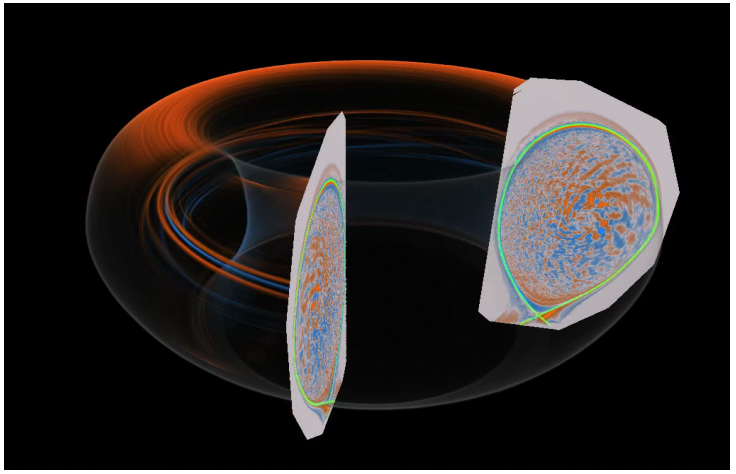
ITER fusion reactor

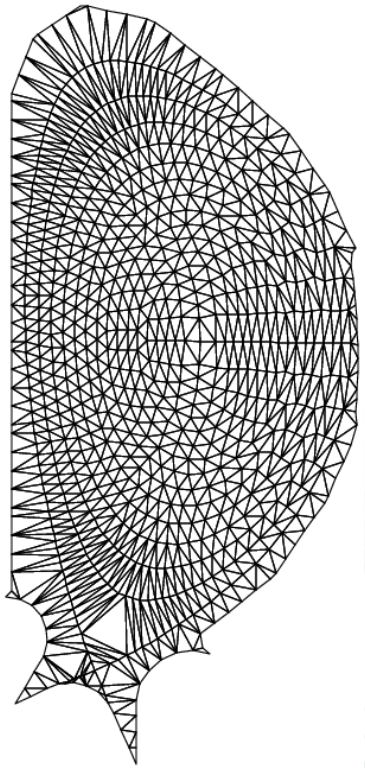




Target Application: XGC1

- Full-function (as opposed to perturbative delta-f) gyrokinetic particle-in-cell code designed for simulating edge plasmas in tokamaks
- Solves 5D gyrokinetic equations via
  - ordinary differential equations for time advance of particles in unstructured triangular physical space grid
  - finite difference discretization of partial integro-differential Fokker-Planck collision operator on rectangular velocity-space grid
  - Maxwell's (partial differential) equations on unstructured triangular physical space grid, solved using PETSc
- The whole volume is simulated with a coarse mesh in the core to capture the large scale turbulence interaction between core and edge





Large scale turbulence in the whole volume simulation in DIII-D geometry

XGC1: Key Features

- diverted magnetic geometry
- gyrokinetic ions
- **drift-kinetic electrons: small gyroradius limit** *(new as of 2013)*
- Monte-Carlo neutral particle with wall-recycling coefficients
- multi-species impurity particles
- plasma heating in the core
- torque input in the core
- **nonlinear Fokker-Planck-Landau collision operator** *(new as of 2014)*
- logical Debye-sheath: code determines wall sheath from ambipolar loss constraint
- reads in an experimental geometry and plasma data

XGC1: Program Flow

I. Initialization

II. Timestep Loop:

1. For each step of a Runge-Kutta algorithm:
   a. Collect particle charge density on underlying mesh.
   b. Solve gyrokinetc Poisson equation on mesh.
   c. Compute electric field and any derivative 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 runtime parameter, N, every Nth timestep:
   a. **Calculate particle collisions and apply this** and other source terms to adjust fields.

III. Finalization

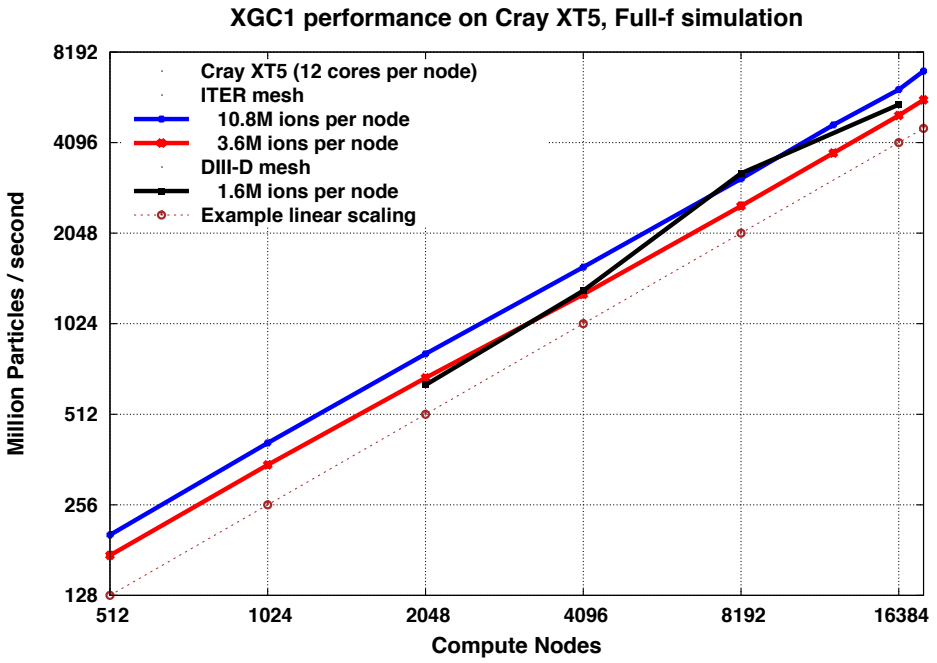
XGC1: Prior Art

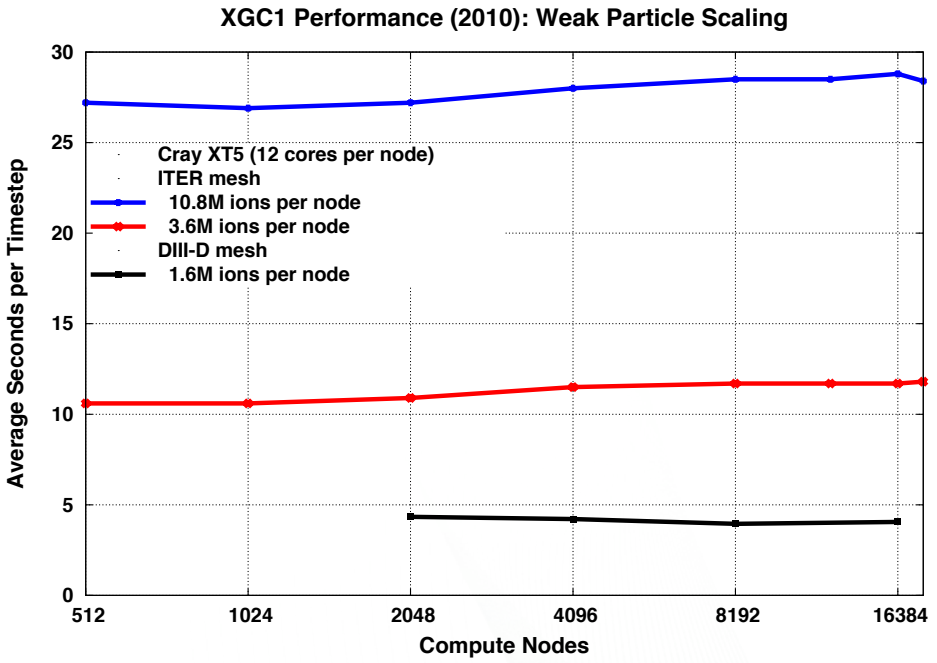
XGC1, equipped with electrostatic turbulence capability to begin with, was further developed during the 2005-2012 DOE project Center for Plasma Edge Simulation (CPES):

- 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
- Utilized a number of sophisticated computational methods and tools, including PETSc, geometric hashing, Hilbert space filling curve, hybrid MPI-OpenMP parallelization, bicubic spline, etc,
- New computer science technologies were also developed, in particular the Adios adaptable IO system, DataSpaces data-staging substrate, and the eSiMon dashboard, all included in the End-to-end Framework for Fusion Integrated Simulation (EFFIS).

XGC1: Pre-2013

This pre-2013 version of XGC1 scaled efficiently up to the full Cray XT5 at the Oak Ridge Leadership Computing Facility at the time (up to 223,488 cores), and routinely used over 70% of the system for production runs.





XGC1: 2013

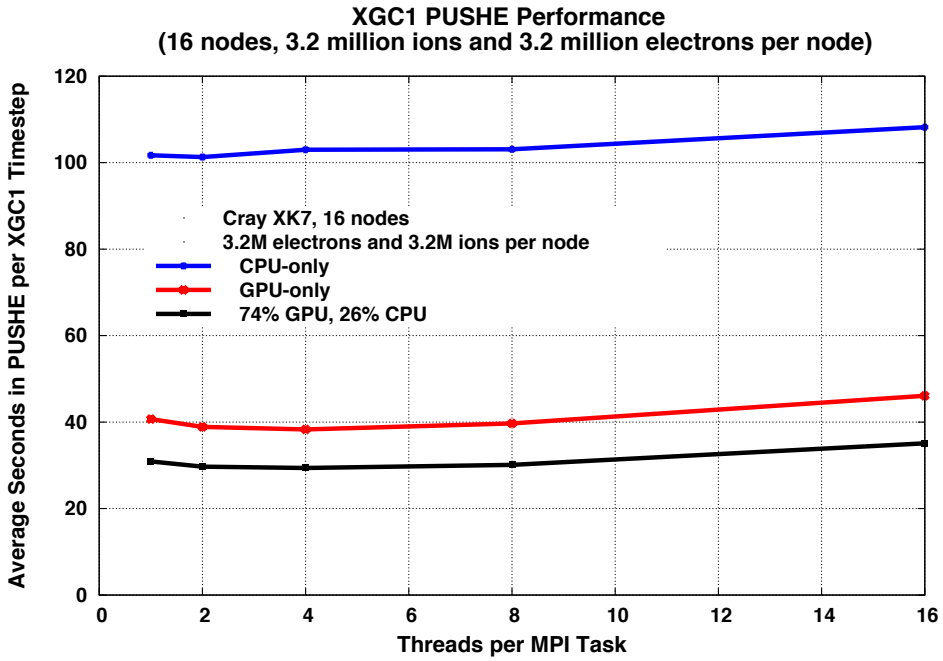
- The introduction of drift-kinetic electrons changed the performance characteristics of XGC1 simulations significantly. The electron mass is much smaller than the ion mass, and the timestep for each electron push (calculating new particle positions) needs to be smaller.
  - XGC1 uses an electron sub-cycling method and ~60 electron pushes per ion push.

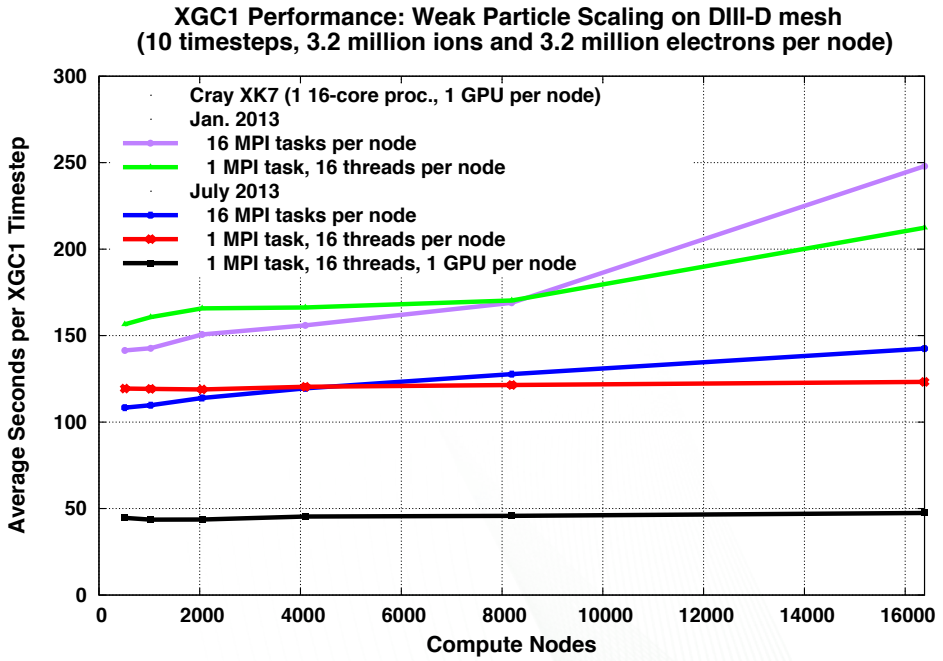
  The actual cost of the electron push is even larger than this implies, reaching more than 90% of the computation time in CPU-only runs (and an even higher percentage of flops). Table below is data from a 10 timestep run for a production DIII-D mesh on 32 XK7 nodes, without and with drift-kinetic electrons. Ion/electron flop ratio is representative across scales. On 16384 nodes (same particle count per node), 'Main Loop'/'Electron Push' flops ratio is 0.98, so slightly more than here.

	Seconds		Flop Count	
	(Max over procs.)		(Total over procs.)	
Main Loop	54	1246	8.1e+12	3.9e+14
Ion Push	13	13	4.0e+12	4.0e+12
Electron Push	-	1142	-	3.8e+14

XGC1: 2013 Performance

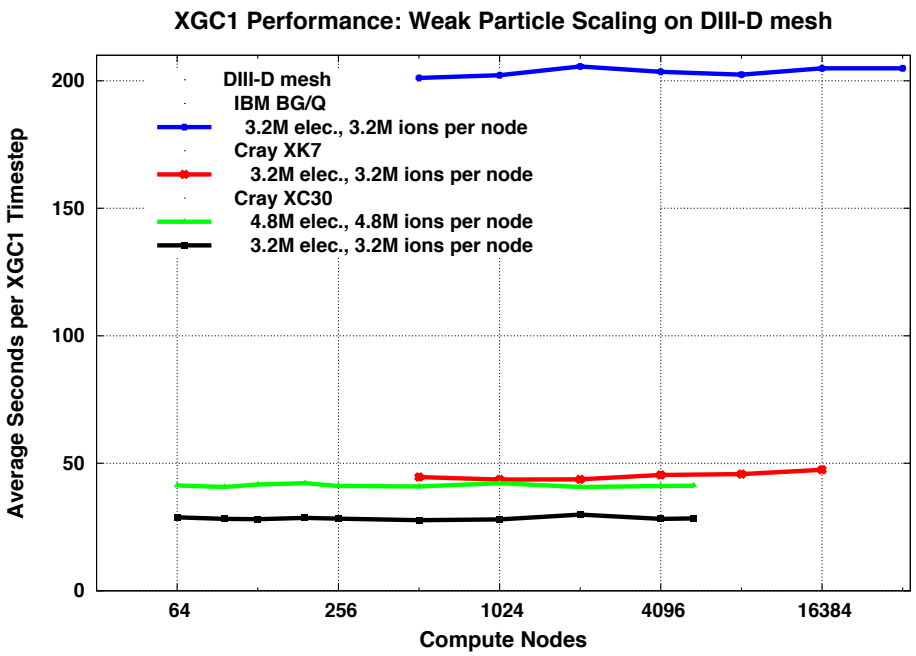
- The drift-electron push was an obvious target for acceleration using the GPGPU available on the Cray XK7 architecture. This was implemented using PGI CUDA Fortran, and achieved approximately 4X speed-up. This did require further optimizations to the associated communication algorithms to preserve good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

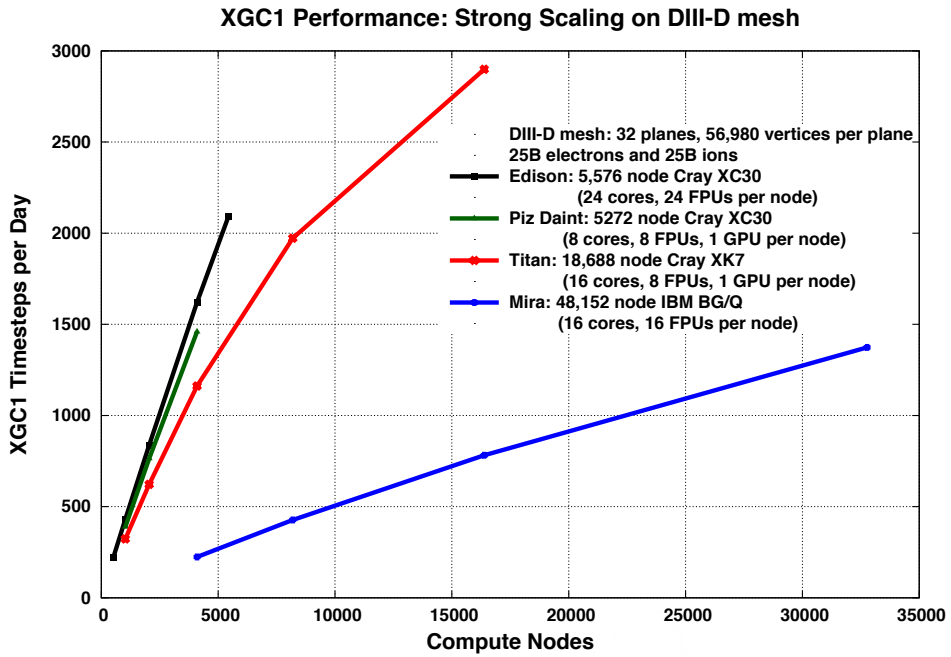




XGC1: 2013 Performance

- XGC1 also ran and scaled well on other systems.





- Mira: IBM BG/Q ("16"-core; 4-way HT per core; 1.6 GHz PowerPC)
- Titan: Cray XK7 (8 FPUs, 16 integer cores; 2.2 GHz AMD; NVIDIA K20X GPU)
- Piz Daint: Cray XC30 (8 cores, 2-way HT per core; 2.6 GHz Intel "Ivy Bridge"; NVIDIA K20X GPU)
- Edison: Cray XGC30 (24 cores, 2-way HT per core; 2.4 GHz Intel "Ivy Bridge")

XGC1: 2014

- The nonlinear Fokker-Planck-Landau collision operator (potentially) changes the performance characteristics of XGC1 simulations significantly (again).
  - The cost of the collision operator scales with the mesh size, not the particle count, and is also a function of how often it is called (every timestep, every third timestep, …) and whether axisymmetry (between the mesh planes) is assumed.
  - For 5 XGC1 timesteps of a representative science configuration (DIII-D mesh, 32 planes, 26B electrons, 8192 nodes on Titan), computing collisions every timestep, and not exploiting axisymmetry:

Flop Count	Main Loop	14.89e+15
	Electron Push	5.37e+15
	Collision	7.49e+15

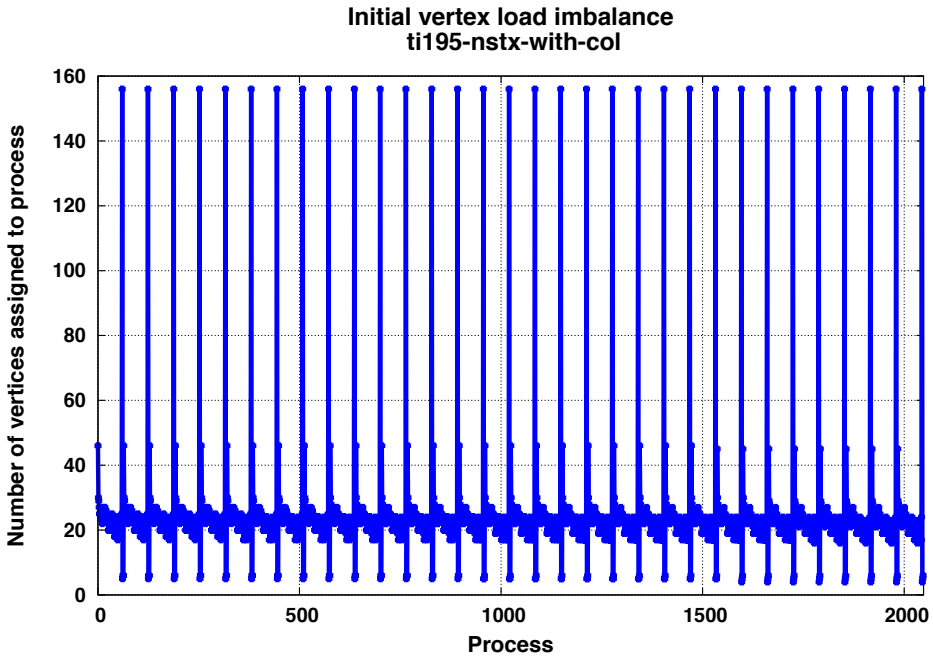
  - Collision flop count would be divided by 32 if assuming axisymmetry, and by 3 if, e.g., computing every third timestep. However, electron push is implemented on the GPGPU on the XK7, and achieves a higher flop rate than Collision running on the CPU.
  - Collision is also a prime candidate for porting to the GPGPU, and work on using OpenACC for this is in progress. Currently OpenMP is used effectively for thread level paralllelization.

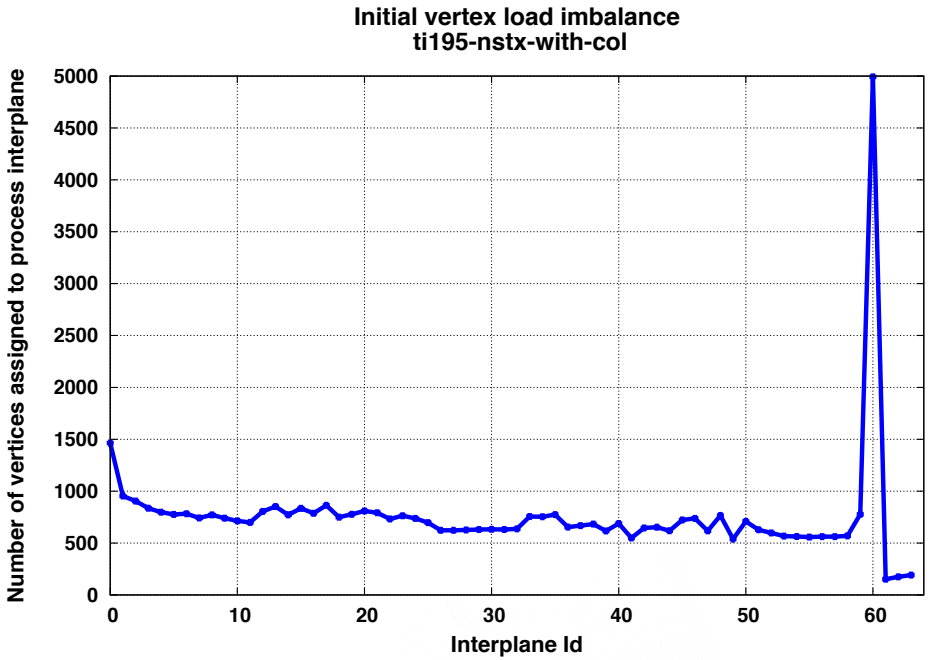
Hybrid Load Balancing

- 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 simple 1-D decompositions of a space-filling-curve ordering of the mesh points has proven very effective for load balancing particles, and thus both particle-related computational costs and memory requirements.
- The collision cost associated with a mesh cell is not uniform across the mesh, depending on the temperature of the plasma in the cell. Because the temperature distribution evolves with the simulation, the corresponding cost will also vary. 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.
- The same mesh decompositions are used for both the particle pushing and for the collision operator calculation, and what is optimal for one is not optimal for the other.
- So, when neither particle push nor 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.

Collision Load Balance Diagnosis

- Even if the collision cost was uniform across the mesh, the optimal decomposition for particles would be inefficient.

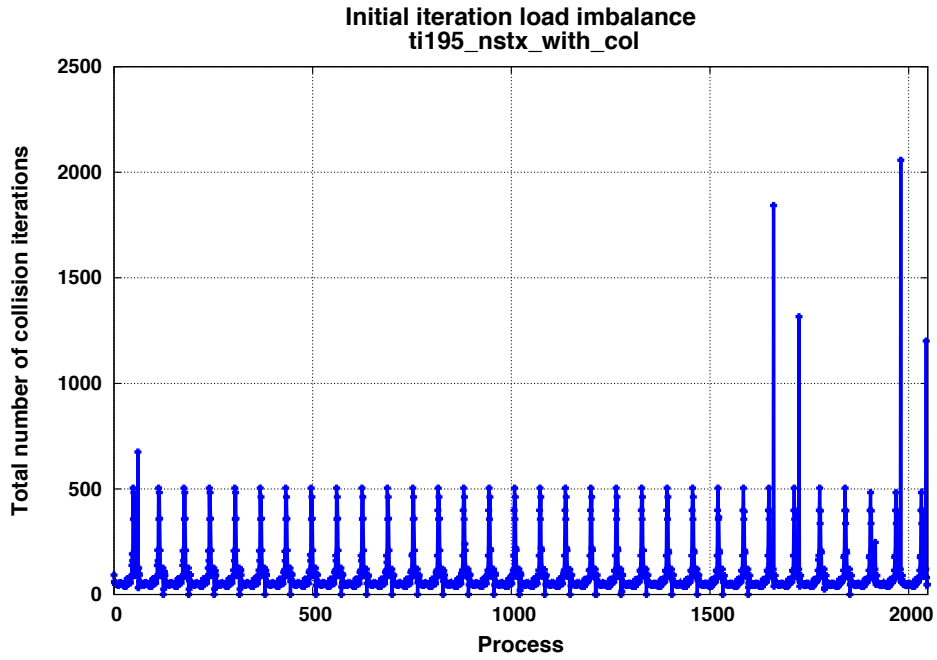


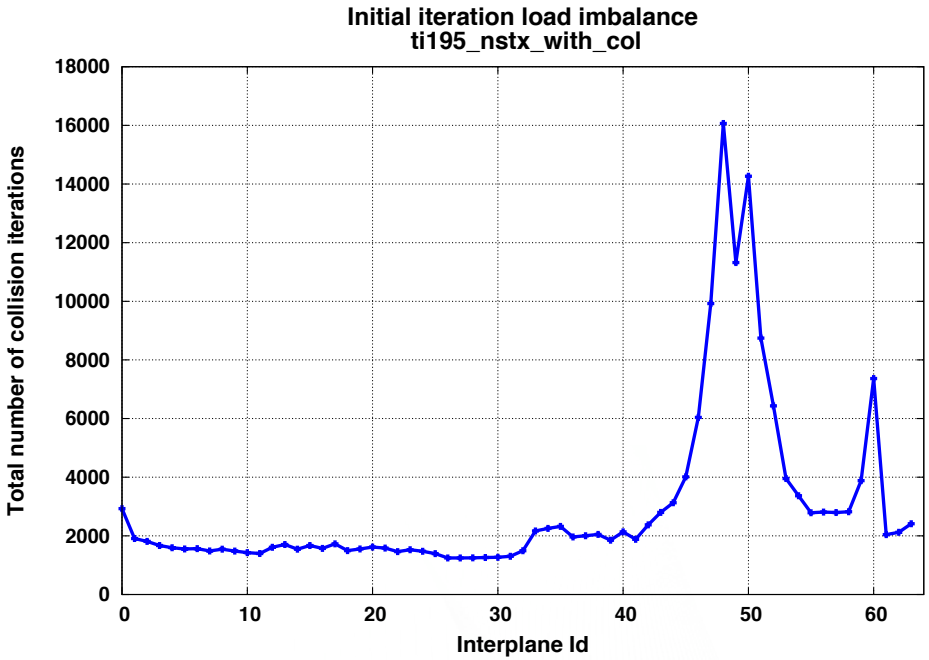


- For this one example problem, even at the very beginning of the run, there is a significant discrepancy in the number of nodes (mesh vertices) assigned to each process (left curve). The pattern is more obvious when looking at the number of nodes assigned to the subset of processes representing each “interplane” in the grid decomposition (same process index in a plane).

Collision Load Balance Diagnosis

- But the collision cost is not uniform. One issue is that an iterative (Picard) algorithm is used to solve in each mesh cell, and the number of iterations required for convergence varies.

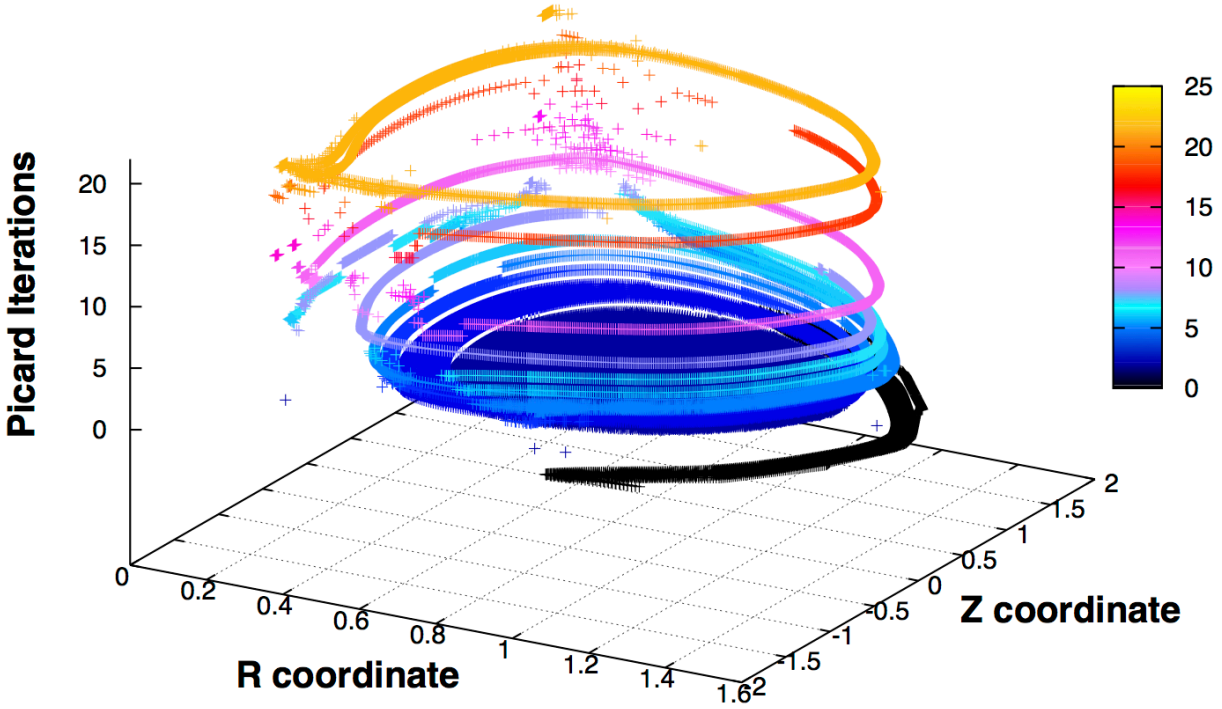




- The number of iterations to convergence summed over all nodes assigned to a process or to an “interplane” of processes, is even greater than that of the node distribution imbalance, and the maximum cost is NOT on the processes with the maximum number of nodes.

Collision Load Balance Diagnosis

- As mentioned earlier, there is a strong domain dependence, due to the standard temperature profile of the plasma.



Black Box – First Attempt

- Diagnosis generated lots of pretty pictures, but did not lead to any physics-inspired solutions. Started over.
- First attempt: Simultaneous optimization
  - Can measure collision cost per cell directly. Can determine 1-D decomposition to load balance this.
  - Can measure particle count per cell directly. Can determine 1-D decomposition to load balance this.
  - How to combine?

Was not successful in combining particle count and collision cost metrics into a single function to be minimized.

  - *Load balancing particle distribution does load balance particle-associated cost, but the actual function mapping count to cost is complicated.*
  - *Measuring cost (walltime) directly as a function of mesh cell was also infeasible because of the large number of particle-loops and synchronization points that exacerbate the impact of the particle count load imbalances.*

Black Box – Second Attempt

- Second attempt: Constrained optimization
  - Add a new input parameter β that specifies the maximum permissible particle load imbalance. (See below.)
  - Add a new input parameter to specify how often to load balance (in increments of collision timesteps).
  - Measure number of particles per cell each ion timestep.
  - Measure collision cost per cell directly each ion timestep that collision is calculated.
  - When load balancing,
    - Determine 1-D decomposition to load balance particle count. Record associated worst case particle load imbalance ratio (maximum over processes / average per process). Call it α. (Typically not possible to load balance perfectly.)
    - Determine 1-D decomposition to load balance collision cost as well as possible subject to a maximum particle load imbalance of α*β.
  - Also update load balance whenever particle load imbalance exceeds α*β*γ (where γ is another already existing parameter, and this logic is already in the code).

This works, but is somewhat unsatisfactory in that choosing β is not obvious. It also does not adapt to changes in the distribution of the collision operator.

Black Box – Third and Current Attempts

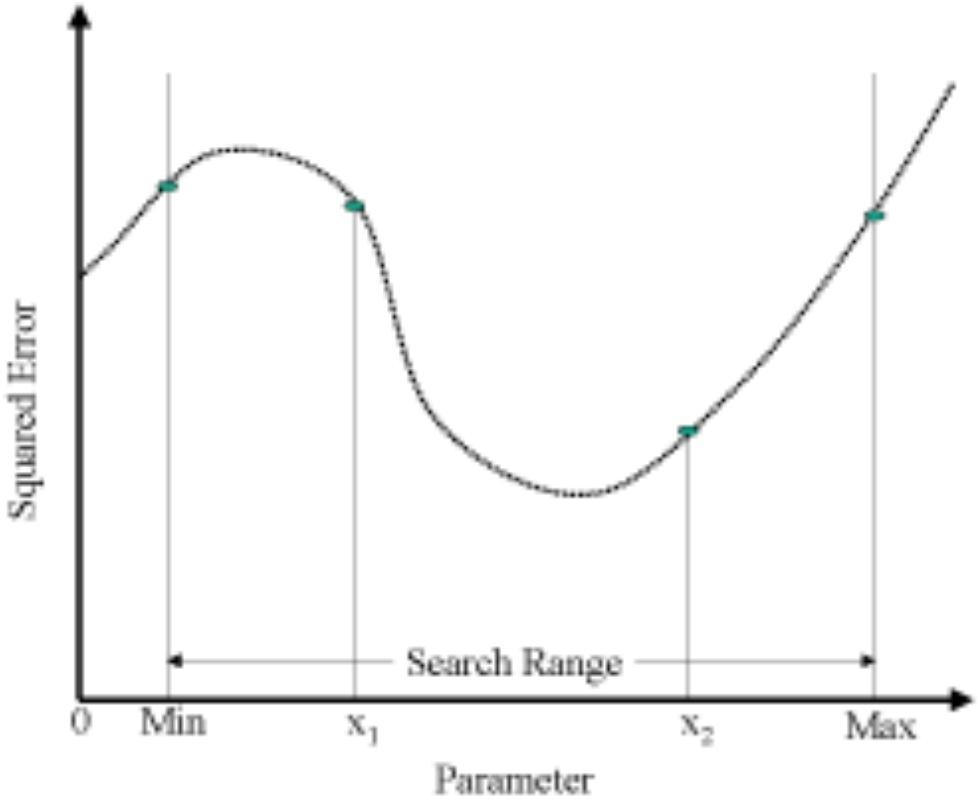
- Third attempt: Two level scheme
  - Record elapsed walltime from one collision timestep to the next, and refer to this as γ.
  - Use Golden Section Search to modify β each time load balancing algorithm is invoked, looking to minimize γ as a function of β. (Note that the input parameter is now the initial value used.)

This does not address the issue of the collision cost distribution varying with the simulation, so walltime associated with a given value of β may not be accurate later in the simulation. Performance is also variable due to external factors (interconnect contention, etc.), and so the optimization algorithm needs to be robust with respect to perturbations.

- Fourth attempt: Modified Golden Section Search
  - Golden Section Search keeps history (ζ) corresponding to 3 β values. If any of these values is not replaced after three updates, reuse this β value and force an update to the associated ζ wallclock time.
  - Add new parameter to specify the largest permissible β value.

Experience usually provides a good estimate of the appropriate search interval, especially if a job is being restarted from a checkpoint. A tight upper bound on β accelerates convergence.

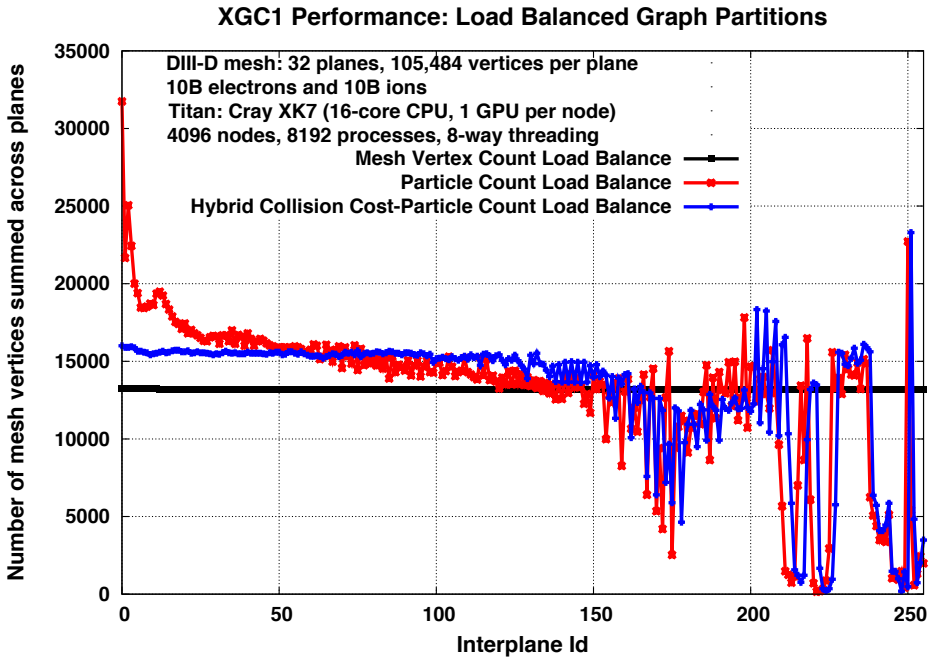
Golden Section Search Cartoon

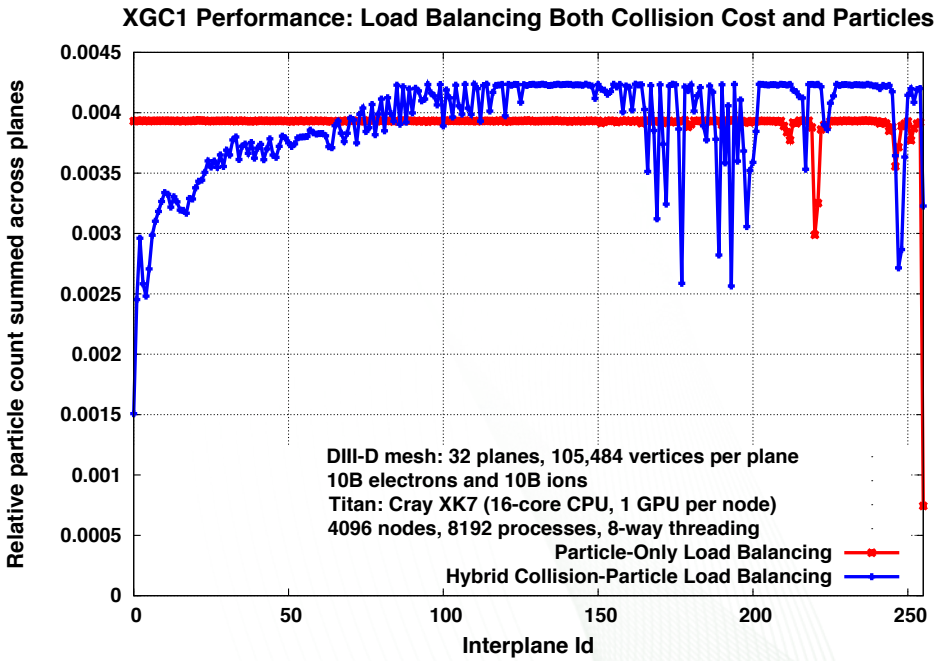


From http://ops.fhwa.dot.gov/trafficanalysistools/tat_vol3/sectapp_d.htm

Black Box Results

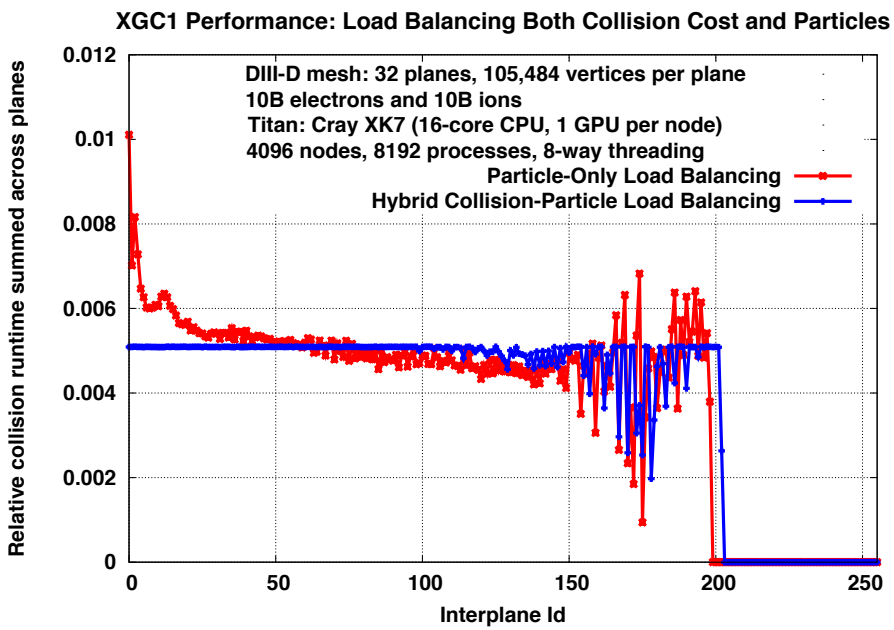
- Example optimal decompositions
  - Uniform mesh decomposition
  - Particle-only load balanced decomposition
  - Hybrid particle count, collision cost decomposition

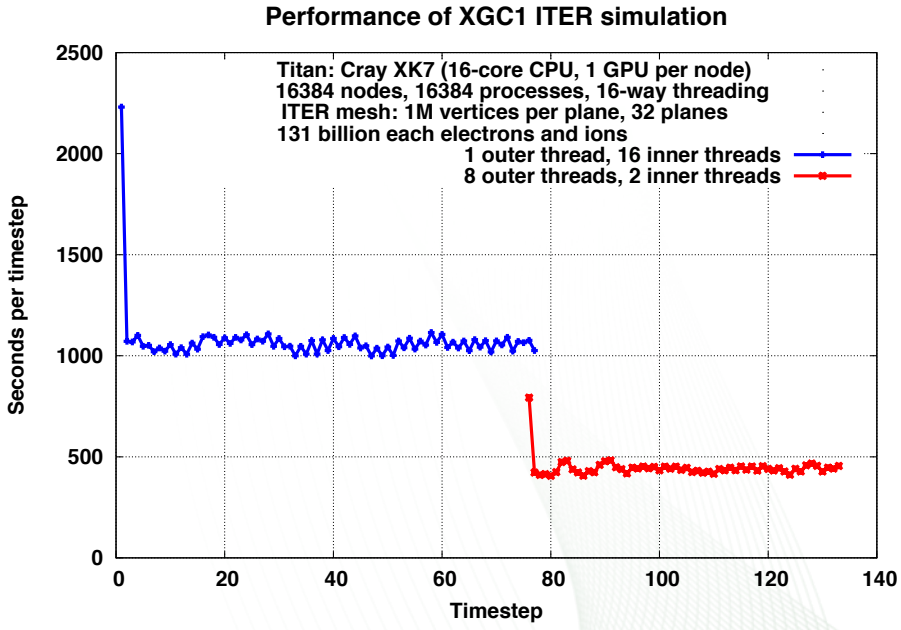




Black Box Results

- Example performance impacts
  - Left: impact on collision load imbalance if only addressing particle load imbalance
  - Right: Here an initial run is followed by restart run, where the number of outer threads used in the collision operator is increased from 1 to 8. The initial run starts with particle-only load balancing. The restart inherits the decomposition from the previous run, which is inefficient here, but the hybrid load balancing algorithm quickly doubles performance in both cases.





Black Box Results: 2nd Example

Back Story: Nested OpenMP parallelism is implemented in the collision operator for performance portability and to expose additional parallelism for future processor architectures. Outer OpenMP is usually a little more efficient, but requires more memory, and so a mixed strategy works best on the IBM BG/Q. On Edison (Cray XC30), both inner and outer are equally good. For the Titan runs, we started with an Edison-like specification (all 16 inner threads) but neglected to specify

```
export OMP_MAX_ACTIVE_LEVELS=2
```

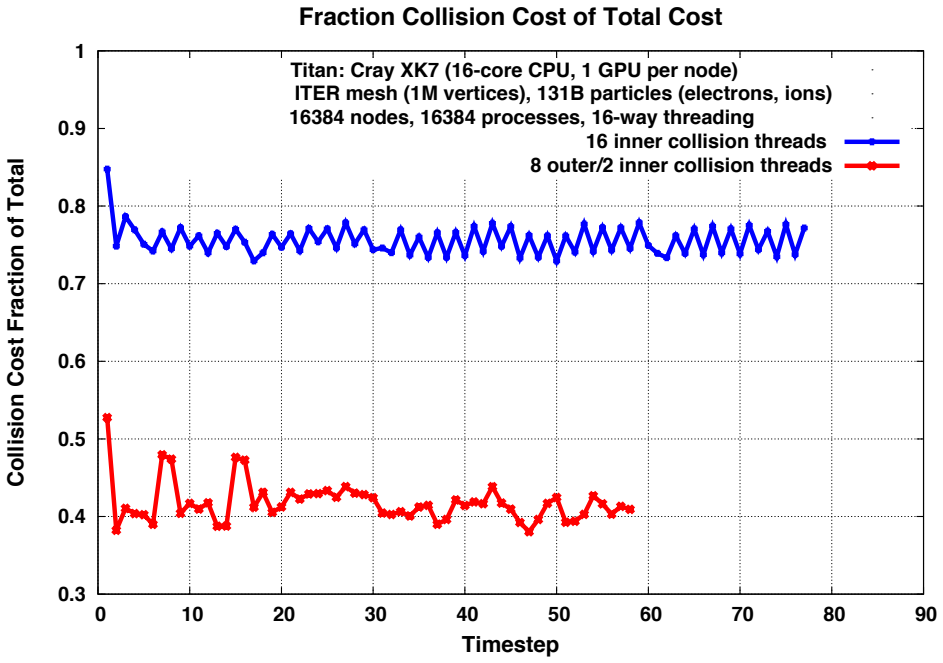
In consequence, the job began without any OpenMP threading in the collision operator. We noticed the problem, and adjusted the settings to 8 outer threads and 2 inner threads (still not realizing the source of the problem), and this improved performance significantly. It also changed the collision cost to particle cost ratio, and thus changed the optimal load balance decomposition.

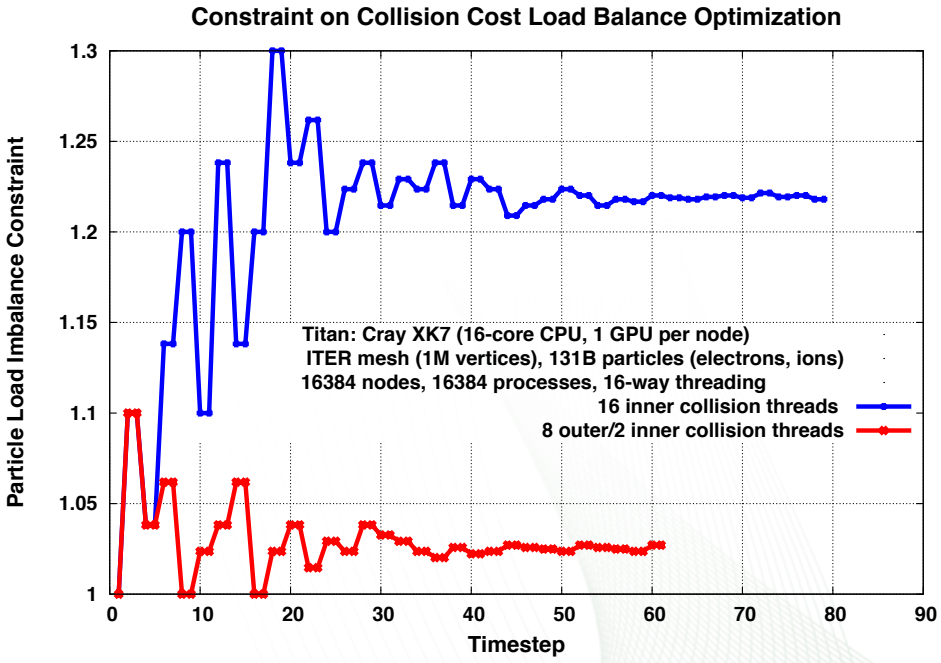
Details:

- Collision operator computed every ion timestep
- Hybrid load balancing scheme applied every second collision operator timestep (so every second ion timestep)

Black Box Results: 2nd Example

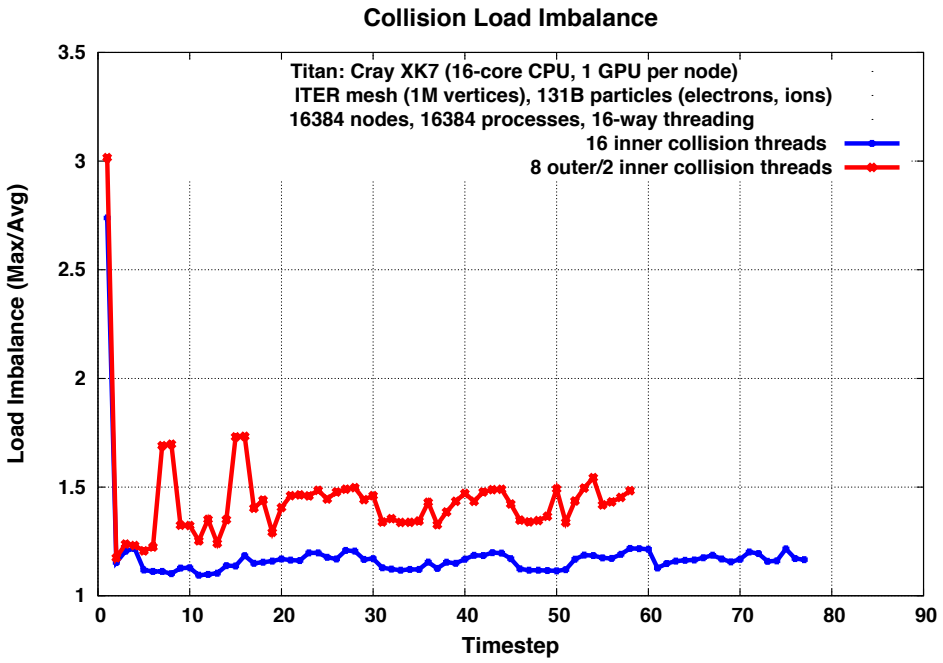
- Left: Fraction of total runtime represented by collision operator. Using 8 OpenMP threads almost decreases this by half. Note that hybrid load balancing also decreases this metric.
- Right: Impact of Golden Section Search on particle load imbalance constraint. Note that Golden Section Search History is lost when restarting, so both runs have the same first three values. After that, the higher collision cost (blue) leads to a looser particle load imbalance constraint when optimizing runtime.

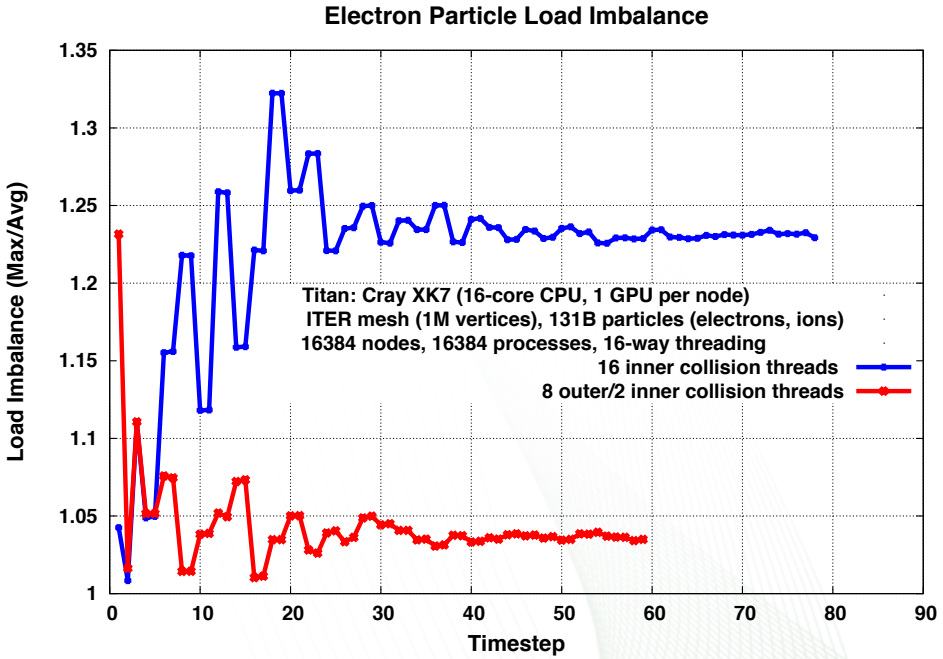




Black Box Results: 2nd Example

- Left: Impact of hybrid load balancing on collision cost load imbalance. Both initial and restart runs start off with very large load imbalances, which are quickly addressed.
- Right: Impact of hybrid load balancing on electron particle distribution load imbalance. Particle load imbalance seems to be less variable than collision cost (particle load imbalance does not change much between hybrid load balancing algorithm adjustments).





OAK RIDGE National Laboratory

Summary

- New science capabilities are continuing to increase cost and change performance characteristics of the XGC1 code, and characteristics are very sensitive to choice of the many options.
- Recent addition of new nonlinear collision operator required re-engineering load balance scheme to address both grid-based and particle-based load imbalances simultaneous, to respond to evolutionary changes, and to deal robustly with performance outliers (whether due to simulation or to externally caused performance variability).
- 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 need to set upper bounds.