New Advances in the Gyrokinetic Toroidal Code and Their Impact on Performance on the Cray XT Series

Nathan Wichmann, Cray Inc.; **Mark Adams**, Columbia University; **Stephane Ethier,** Princeton Plasma Physics Laboratory.

ABSTRACT: *With the recent agreement to build the multi-billion dollar international burning plasma experiment known as ITER, fusion simulations will be growing dramatically in both complexity and size. The Gyrokinetic Toroidal Code (GTC) is a 3D Particle-In-Cell code used for studying the impact of fine-scale plasma turbulence on energy and particle confinement in the core of tokamak fusion reactors. To tackle global ITER-size simulations with full kinetic ion and electron physics, GTC will require new algorithms and supercomputers will have to grow in capabilities. We will review recent code modifications made to prepare for these new, exciting simulations and examine the performance of GTC on both the Cray XT3 and Cray XT4 systems using the latest Cray tools. Finally, we will look to the future and discuss plans for GTC and how that will effect its performance on future computers.*

KEYWORDS: Cray XT4, GTC, Gyrokinetic Toroidal Code, PIC, Fusion

Introduction

An effective method for analyzing the Vlasov-Poisson system of equations, used in simulating fully ionized plasmas, is the particle in cell (PIC) method [6]. PIC methods evolve plasma dynamics selfconsistently by alternately pushing charged particles and solving the fields governed by Maxwell's equations. These methods enable the study of plasma micro turbulence on global scales. This micro turbulence (eg, scales a few millimeters) is important in understanding transport phenomenon in magnetically confined plasmas and is best understood with a model that includes the effect of gyro motion of charged particles in magnetic fields. The basic idea behind the gyrokinetic simulation method is to time average rapid precessing motions, and only to push the guiding center motion for the particles [3].

The gyrokinetic toroidal code (GTC) is an implementation of the gyrokinetic PIC method used for toroidal magnetic confined burning plasma devices. GTC has been run effectively, with scaling up to 32K cores, on all of the recent high performance computational architectures. The GTC programming models uses FORTRAN 95 and MPI, and uses PETSc for the potential solves in Poisson's equation, which is discretized with linear finite elements. Note, standard gyrokinetic ordering implies that Poisson's equation need only be solved on each *poloidal plane* (perpendicular to the toroidal direction in the torus), resulting in series of independent 2D grid linear solves.

Gyrokinetic PIC algorithm.

The basic PIC algorithm consists of depositing the charge from the particles onto the grid, computing and smoothing the potential, computing the electric field and pushing particles with Newton's laws of motion. Figure 1 shows a schematic of the PIC algorithm.

$(x, v) \leftarrow Initialize$ – Initialize (ion) particle position and velocity
time loop $t = 1 : \delta t : T$
$\delta\bar{n}_i \leftarrow Charge(x, q)$ – Deposit ion charge (q) on grid
$\Phi \leftarrow \left(1 - \nabla_\perp^2\right)^{-1} \delta\bar{n}_i$ – Potential solve (small parameter expansion)
$\Phi \leftarrow Smooth(\Phi)$ – Smooth potential
$E \leftarrow \nabla\Phi$ – Compute electric field on grid
$v \leftarrow v + \delta t \cdot qE/m$ – Update velocity (interpolate E from grid)
$x \leftarrow x + \delta t \cdot v$ – Update ion position
end loop

Figure 1. Schematic of the PIC algorithm

Performance is governed by three basic types of operations in this algorithm: 1) grids work (ie, Poisson solve), 2) particle processing (eg, position and velocity updates and 3) interpolation between the two (ie, charge deposition and field calculation in particle pushing). The dominate computational cost depends on the number of particles use; the delta F methods used in GTC the grid work is a significant minority of the overall cost of the simulation, the cost being dominated by particle pushing. Earlier version of GTC parallelized the grid work with a shared memory model (Open MP) which is adequate if the grid is small enough and the size of the shared address space for each poloidal plane is large enough. As devices get larger, and finer grids are desired for some types of simulations, performance degrades with this model because more address spaces are required for each plane, and the larger grid that needs to be stored puts more pressure on the cache and the entire memory system. This performance degradation is due to increased pressure on the cache in the charge deposition and redundant work required in the solve, field calculation and smoothing phases. The large fusion devices that now need to be modeled and the small amount of shared memory parallelism available on the newest large machines (ie, Cray XT4 and IBM Blue Gene have essentially no shared memory) requires a domain decomposition of the grid with MPI parallelism. This report discusses an BPI parallel decomposition method for the grids in GTC.

GTC mesh and decomposition

Figure 2 shows a diagram of a, typical magnetic fusion tokamak device. GTC stimulates the core plasma; this is a, toroidal domain with magnetic field lines with strong components in the toroidal direction but with some component in the poloidal plane resulting in a "twisting" magnetic field. GTC generates meshes for poloidal planes where Poisson's equation is solved. Thus, charge deposition first interpolates a particles charge to the two planes on each side of the toroidal domain in which it, is located. This charge on the poloidal plane is then interpolated onto the mesh points. Figure 3 (left) shows a sample set of poloidal mesh points on a poloidal plane. Figure 3 (right) shows a sample global grid with the field line following grid lines.

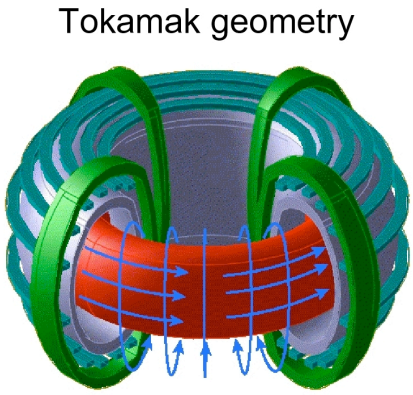

Figure 2. Typical Tokamak Device

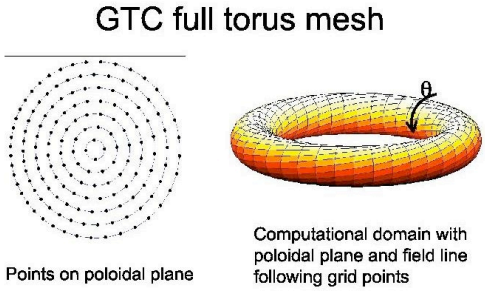

Figure 3. GTC Meshing

These planes form a natural 1D decomposition of the computational domain, which is the primary decomposition in GTC. The next section describes a new 2D (radial) decomposition of the particles and grid. This significantly complicates the parallel model in that the grid on the poloidal plane must be decomposed with explicit MPI parallel model.

Radial grid decomposition

The optimal grid decomposition for GTC is not obvious in that several efficiencies are impacted by the decomposition in different ways. Additionally, the optimal method may be more complex to implement than is necessary for acceptable performance in the range of device sizes (up to 10K radial grid cells) and the computers of interest in the next several years (say 640K cores and 128 poloidal planes, resulting in 4K processes per plane). The first thing to note is that the particles move primarily along the magnetic field lines and thus do not move much in the radial direction. Thus, a radial partitioning will result in minimal communication of particles within the poloidal plane. A fully 2D grid decomposition has the advantage that the same decomposition can be used for the Poisson solver – currently a separate unstructured 2D decomposition is used. A difficulty with an unstructured 2D decomposition, however, is that a structured decomposition is essential required for fast computation of a particle's processor. Also a structured 2D decomposition is not as efficient for the solver as an unstructured one when large amounts of parallelism are required. Given these tradeoffs and the relative simplicity in implementation we have opted for a structured 1D radial grid decomposition for the grid/particle computations and a 2D unstructured decomposition for most of the grid computations (ie, the Poisson solver).

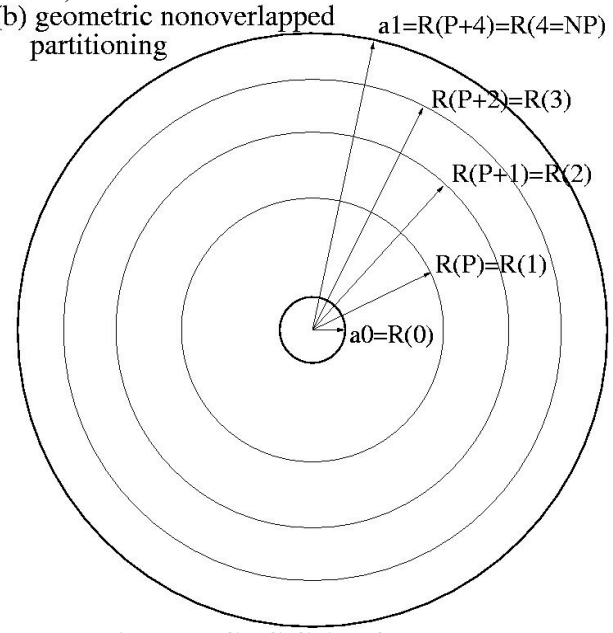

Figure 4. GTC Grid of one Plane

Our approach is to assume that the density of particles is a known function of radius (eg, a constant) and to compute a nonoverlapping decomposition of the domain (ie, a geometric partitioning) that balances the particles exactly (given the assumed distribution). Figure 4 shows a small GTC grid of one plane (left) and a schematic of a geometric partitioning with four radial domains (processors or cores). Note the GTC computational domain has an inner hole of radius "a0" and an outer radius "a1" in Figure 4. The local domain, that needs to be stored on each processor, must be extended to accommodate the charge deposition. The particle position stored in the gyrokinetic method is the guiding centers of the particles – the gyrokinetic formulation models the gyro motion as a charged ring around particles guiding center. This charged ring is discretized with a few points (eg, four) on the ring; the charge at these points is deposited on the grid with linear or bilinear interpolation. A small but trivial optimization is to extend each radial domain to line up with the radial grid points of the mesh before the grid extension for the gyro radius is computed. This enlarges the size of the domain that a particle can occupy on each processor. This (overlapped) radial decomposition defines the valid region in which a particles guiding center must reside for a processor to be able to, in general, deposit the particle charge. That is, a particle's guiding center must be in this region for it to be processed locally and must be sent to another processor of it strays outside of this region.

New Baseline Benchmark

GTC has often been used to test out new systems. Its scaling behaviour was well know and the problem sizes could be easily adaptable to a wide range of machine sizes and problems. Because its behaviour was well understood, it provided good feedback on the performance of anything from new processor types to new interconnects. Results could also be used to project out to much larger science problems.

Today's GTC has changed significantly in terms of both capabilities and performance characteristics. It can now simulate much larger fusion reactors using more and different kinds of particles. It also uses the PETSc solver, which can solve the Poisson equation in parallel. Finally the new decomposition

strategy eliminates the need to do a major mpi_allreduce inside all the processor in a plane, potentially dramatically changing the scaling to high processor counts.

One of the early goals of this study was to define a new baseline benchmark that not only preserved the same beneficial characteristics of the old benchmark, but also demonstrated the new capabilities of the code, namely to simulate much larger fusion devices. This benchmark would be used act as a comparison across generations of both machines and processors and project performance to larger machines and problem sizes.

The new weak scaling benchmark that we created "weakly" scales the problem in terms of both the size of the reactor being simulated as well as the total number of particles in the reactor. It runs on 64 to 16K processors, increase the processor count by a factor of 4 with each step. We assume that one will always use a constant decomposition of 64 slices in the direction of the torus as the code is known to scale well up to that point and the science rarely allows one to go beyond that number of slices. If necessary, it is still possible to reduce the number of slices so one can run on fewer than 64 processors. A critical feature of the benchmark is that at 16K processors we are simulating a reactor the size of ITER, an important reactor to simulate in the coming years.

Scaling Results

Data was collected on the 11,508 socket XT3/XT4 system at Oak Ridge National Laboratories (ORNL) called jaguar. Each socket is a dual core AMD Opteron running at 2.6 Ghz. The main performance different between the XT3 and XT4 half is the XT4 has approximately double the main memory bandwidth. We collected data by running the benchmark from 64-4096 processor targeting either the XT3 or the XT4 exclusively, and from 64 to 16K processors allowing the scheduler to choose which processor type to use for any given PE. Finally, we ran the old version of GTC up to 4K processors, the highest it could go given the size of the reactors being simulated.

Weak Scaling to 16K processors

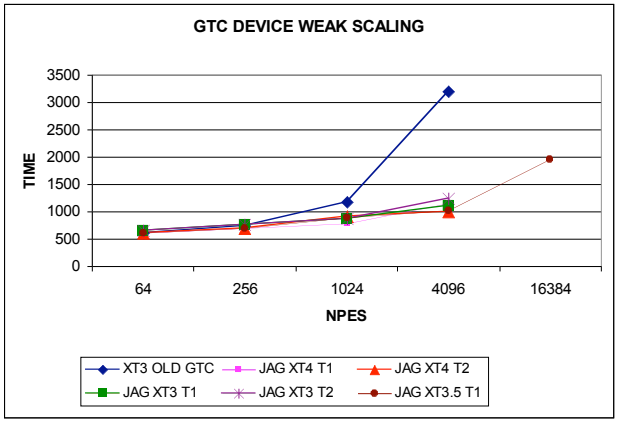

Figure 2: Weak Scaling Results

Figure 2 above shows the results of the weak scaling study running on 64 to 16K processors. The first observation is that the new version of GTC is performing much better than the older version as the processor count, and thus the device size grows. This is primarily the result of the new decomposition scheme's and solver's ability to distribute the work associated with large grids. Scaling of the older version of GTC stopped at 4096 PEs because it could not run and ITER size device using 16K processors.

Scaling to 4096 processors is good, but not as good as hoped for. Time approximately doubles while ideally it would stay flat. There does not appear to be much difference between the XT3 and

the XT4, an early indication that the majority of the code is not sensitive to memory bandwidth. Moving to 16K PEs time almost doubles again. While a tremendous amount of science can be accomplished at this number of PEs, performs has degraded sufficiently that the we suspected a problem.

Component Times

To get a better idea of where the time was being spent, we graphed the time spent in each of the major components at a function of the number of PEs. Figure 3 below shows the time spent in each component. Not surprisingly, the pusher and charge routines dominate the time. Each is increasing in time as the number of PEs increase, but do not seem to be enough to cause the scaling that we observed. The shifter starts out very small relative the main two components, but at the number of PEs increase it begins to increase in time, with a dramatic jump at 4K PEs.

The shifter is where all of communication is performed. There are two attributes to the communication that could potentially cause the behavior seen. First, it could be the result of the communication itself, either because of network or injection bandwidth contention. The second possibility is that the communication effectively acts like a barrier, if there was load imbalance in the shifter or prior to the shifter, we could see it as extra time spent in the shifter even though that was not the true cause.

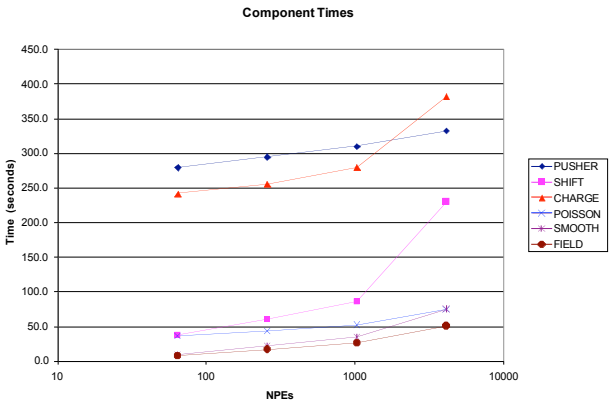

Figure 3: Component Times

Load Imbalance

To test whether the cause was the communication or load imbalance, we inserted a barrier just inside the shifter drive and then timed the three components. The results can be seen in figure 4 below.

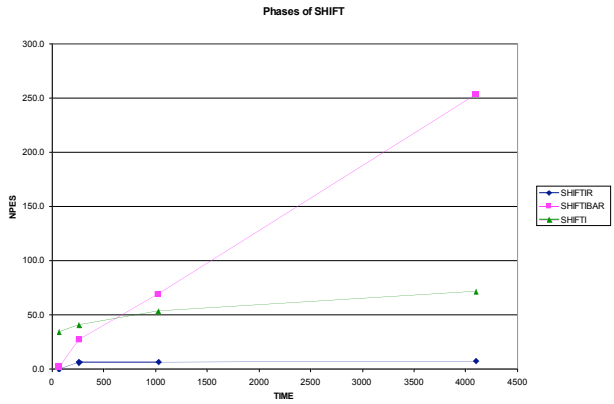

Figure 4: Breakdown of time in shift

All of the communication is done in shifti, the routine that moves particles in the direction of the torus, or shiftir, the routine that moves particles radially. The two remain quite flat across all PE counts. On the other hand, the barrier time is climbing dramatically as PE count increases.

Since this barrier is not necessary for the communication, the barrier itself and the time associated with it could just have easily been counted in the PUSHI routine. It is thus interesting to replot the components graph with the barrier time counted at part of PUSHI.

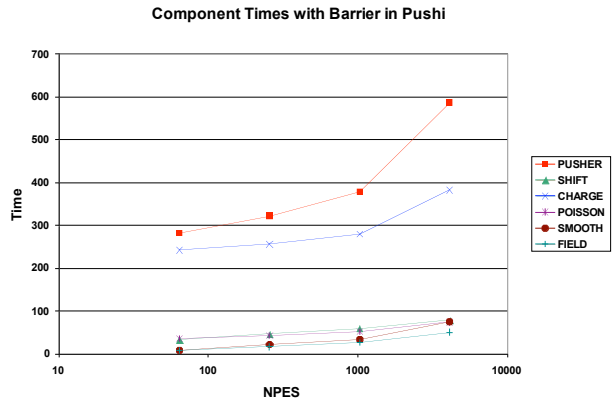

Figure 5: Component Times

Figure 5 now clearly shows that our problem is actually in the main two computational kernels. Both PUSHI and CHARGEI are increasing in time as the number of PEs increase.

Instrument PUSHI

With scaling apparently limited by load balance problem in PUSHI, we decided to try collect as much information as we could about that routine. To do that, we collected Opteron counter data using the Cray Pat performance tool. The first step was to simply plot the time spent in PUSHI as a function of PE number.

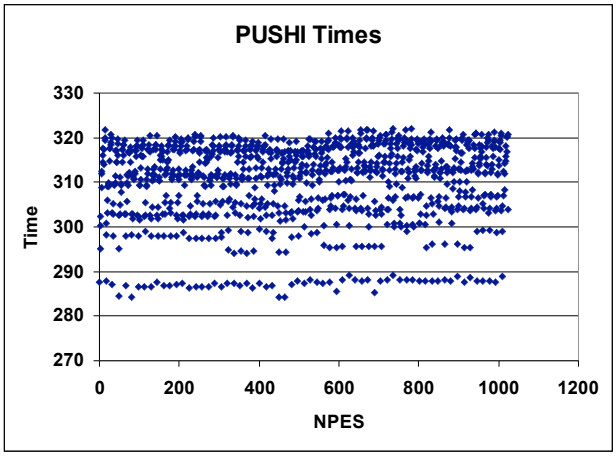

Figure 6: Time spent in PUSHI for different PEs

One can see in Figure 6 that there is about a 15-20% difference in time depending on the PE number. Initial examination of the FLOP count cache hit rate data that they was a not sufficient variable in computational or memory work load to explain the variable in times. When we graphed the number of TLB misses we did observe significant variation.

Figure 7 shows almost a factor of four difference in TLB misses depending on PE number, a variation that could potentially explain the variation in time. The question was if there was a correlation between TLB misses and PUSHI time.

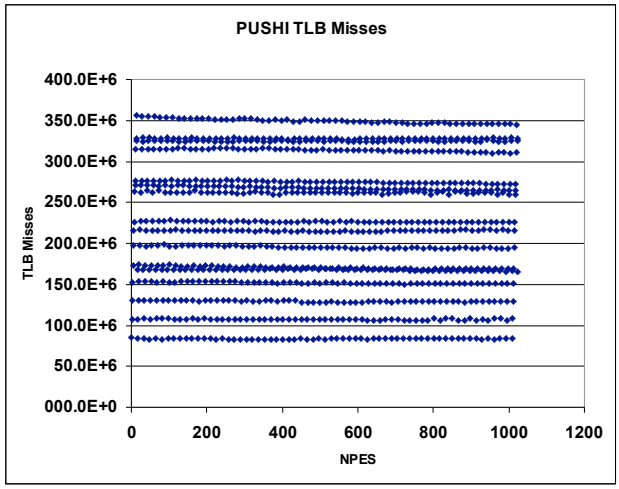

Figure 7: TLB misses on different PEs

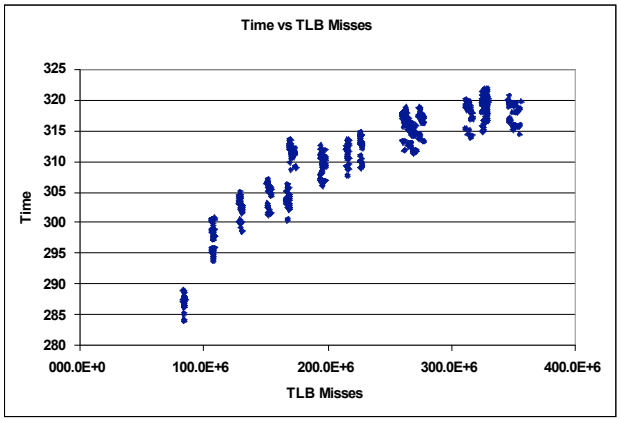

Figure 8: PUSHI Time and a function of TLB misses

Figure 8 is a plot of PUSHI time and a function of TLB misses for every PE. One can see a VERY strong correlation. Any PE that took less than 100 million TLB misses ran in less than 290 seconds, and any PE that took more than 300 M TLB misses to more than 312 seconds.

With this latest data, much of the focus of performance studies are trying to explain both the frequency of TLB misses and the distribution of TLB as a function of PEs. Unfortunately we do not have a good explanation for this data. PUSHI does contain some multi-dimensional gathers, but it is not clear if that is the problem, of even if there might be more than one problem.

Multi Core versus Single Core

A clear trend in micro processor design is the introductions of multi-core sockets. Today the XT4 is a dual core system, but it will be possible to upgrade that to quad core Opterons once then become available. While vendors are constantly improving memory bandwidth, cores share all of the memory bandwidth on and off the socket. Given the trend toward more and more cores sharing memory bandwidth, it is important to understand how one's program responds to a multi-core environment.

We wanted to run GTC in both a single core and a dual core mode and examine the change in performance. To do this we decided run a problem on 512 Opteron sockets using only 1 core per socket, and then to run the exact same problem using the same number of socket but using both cores on every socket. While we realized that by using twice and many MPI processes to solve the same problem we are also testing the ability of the code to "strongly" scale, we feel that this is a reasonable data point. Basically we want to know how much faster our science will be solved as we get access to more and more cores.

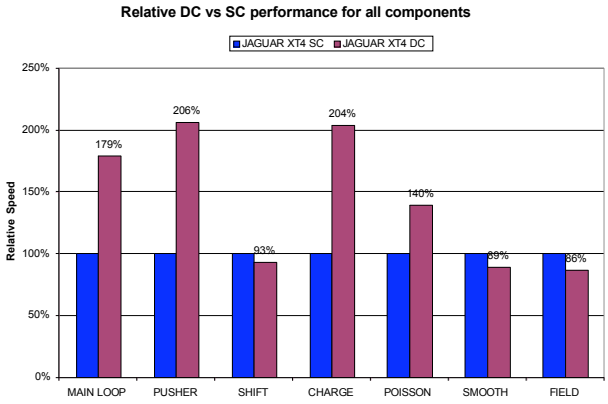

Figure 9: Relative dual core speed up for each component

Figure 9 shows the relative performance of the entire code as well as broken down on a per component basis. We see that the "MAIN LOOP", the entire computational portion of the code, gets a very nice speed increase of 179% moving to dual core from single core run. This is already at the high end of the range of dual core speed up.

To really understand what is going on we want to look at the speed up for each component. The first order information is very good; both of the two main computation kernels get an excellent speed up of 200%. This is a strong indication that neither of this routines are memory bandwidth intensity. The only caveat might be that if we are limited by TLB misses, and can eventually improve that TLB miss rate, the routine might speed up to the point that they start to pressure memory bandwidth.

The PETSc solver does get a respectable speed up of 140%. In the case of the solver the speed increase could be limited by local memory bandwidth, network bandwidth, of scaling of the algorithm itself. At this point we do not have sufficient data to know which it is.

The SHIFT, SMOOTH, and FIELD all actually go slower in dual core mode. In the chase of SHIFT, it is this is not surprising, the kernel is either moving data around locally, or communicating that across the network. In either case both cores will be competing for a shared resource. We have not examined SMOOTH of FIELD, but this data strongly suggests that they are bandwidth limited.

Taken together, Figure 9 is telling us that we should be able to utilize quad core systems. The main computations should get a very nice speed up. This figure also contains a warning. As the number of cores continue to increase, routines that today are not considered important, could dominate in the future if memory bandwidth does not increase or if corrective action is not taken.

Future Work

The work done for this paper thus far has been extremely valuable. We have established a new baseline benchmark, collected scaling data, and begun the process of predicting the performance of the code on future systems. That said there is still much work to be done. Below is a list of what we hope to study over the next several months.

- Explain and eliminate the load imbalance in PUSHI. If TLBs are really the problem, what is the fundamental cause? How do we ultimately fix the problem so that we not only scale better, but hopefully single CPU performance improves.

- Switch back to 32 precision for the most of the computation in GTC. While the PETSc solver needs to remain a 64 bit precision solver, the rest of GTC can be computed using only 32 bit floats. Advantages include potentially better TLB performance, less memory bandwidth consumed, better utilization of the cache, and double the peak performance of the SSE floating point units.

- Examine the PETSc solver for performance improvements. We did not directly examine the performance of the PETSc solver in this study, but the Dual Core study showed the PETSc performance may become more important in the future. We plan on working with the Cray Scientific Libraries group to improve the performance of the solver.

- Examine the use of OMP. With the advent of multi-core chips there is a renewed interest in OpenMP to help everything from attaching different levels of parallelism to reducing the number of messages on the network. Will OpenMP make sense for GTC?

- Other Weak Scaling Studies. We plan on performance other weak scaling studies to enhance our ability to predict the performance of future science problems. One in particular is simulating the ITER reactor with trillions of particles. The new code makes this plausible, but how will it perform?

Conclusions

The new GTC provides a substantial increase in both performance and scientific capabilities. It can run larger science problems then ever before and the new PETSc solver allows one to perform more complex simulations. While performance for large devices is much better than the older version and scaling is good to 16K cores, there is still a strong desire to improve scaling further. Finally, GTC performs very will on Dual Core Opteron chips, and should be able to continue to perform well as the number of cores per socket increases.

About the Authors

Nathan Wichmann is currently a Senior Applications Engineer in the Cray Supercomputing Center of Excellence working with Oak Ridge National Laboratories. During his nearly ten years at Cray Inc. he has work on a range of applications include automotive, climate and weather, and fusion research. Nathan has a particular interest in compilers and processor micro-architecture and has worked in determining the direction of compiler and processor development. He can be reached at wichmann@cray.com.

Dr. Mark Adams is Research Scientist in the Applied Physics and Applied Mathematics department at Columbia University. His career has focused on high performance finite element simulation systems, in particular, parallel multigrid equation solvers for large unstructured finite element problems in solid mechanics and plasma physics. He has a BA and a Ph.D, both from U.C. Berkeley

Dr. Stephane Ethier is a Computational Physicist in the Computational Plasma Physics Group at the Princeton Plasma Physics Laboratory (PPPL). He received a Ph.D. from the Department of Energy and Materials of the Institut National de la Recherche Scientifique (INRS) in Montreal, Canada, and an M.Sc. from the University of Montreal, Canada. His current research involves large-scale gyrokinetic particle-in-cell simulations of microturbulence in magnetic confinement fusion devices.
This work is funded in part by three DOE SciDAC projects and by the DOE Plasma Sciences Advanced Computing Institute (PSACI).