ANL/ALCF/ESP-13/15

Global Simulation of Plasma Microturbulence at the Petascale & Beyond (Optimizing the GTC Code for Blue Gene/Q)

ALCF-2 Early Science Program Technical Report

Argonne Leadership Computing Facility

ANL/ALCF/ESP-13/15

Global Simulation of Plasma Microturbulence at the Petascale & Beyond (Optimizing the GTC Code for Blue Gene/Q)

ALCF-2 Early Science Program Technical Report

prepared by
William Tang[1,2], Stephane Ethier[1], Bei Wang[2], Timothy Williams[3], Khaled Ibrahim[4], Kamesh Madduri[5,4], Samuel Williams[4], and Leonid Oliker[4]
[1]Princeton Plasma Physics Laboratory
[2]Princeton University
[3]Argonne Leadership Computing Facility, Argonne National Laboratory
[4]Lawrence Berkeley National Laboratory
[5]The Pennsylvania State University

May 7, 2013

Optimizing the GTC Code for Blue Gene/Q

ALCF-2 Early Science Program Technical Report

William Tang,[1, 2] Stephane Ethier,[1] Bei Wang,[2] Timothy Williams,[3]
Khaled Ibrahim,[4] Kamesh Madduri,[5, 4] Samuel Williams,[4] and Leonid Oliker[4]

[1]*Princeton Plasma Physics Laboratory*
[2]*Princeton University*
[3]*Argonne Leadership Computing Facility, ANL*
[4]*Lawrence Berkeley National Laboratory*
[5]*The Pennsylvania State University*

I. SCIENCE

Figure 1 shows, schematically, the *tokamak* device, whose intent is to magnetically confine a high-temperature plasma and produce energy through nuclear fusion. The confined plasma, shown in pink, is toroidal in shape. For several decades now, scientists have been studying and improving these devices, working toward a successful fusion *ignition*—creating a burning plasma—and sustaining it to generate more power than it took to create it. The next big experimental step will be the International Thermonuclear Experimental Reactor, a twenty billion dollar burning plasma device under construction in France, involving the partnership of seven governments.[1] The human in Figure 2 illustrates the scale of ITER.

The magnetic field in the tokamak confines the plasma—ions, electrons, and their heat and momentum. Various kinds of instabilities in the plasma work against that confinement. Turbulent fluctuations can cause transport of particles and energy across the magnetic field lines, toward the outside of the plasma, where it is lost. Loss of plasma and energy, of course, works against confinement and successful fusion. This *microturbulence* is something the fusion community needs to understand and control.

In past and present tokamaks, measurements have shown that the transport of energy and particles caused by microturbulence depends on the size of the tokamak—larger means more transport. This is *Bohm* scaling[2]. However, theoretical arguments predict that beyond a certain size, which ITER will be beyond, the size dependence goes away, leading to relatively smaller turbulent losses and thus better confinement. This is *GyroBohm* scaling[3]. The key target of this ESP project is to simulate microturbulent transport for devices of different sizes up through ITER size, and validate and understand GyroBohm scaling.

II. NUMERICAL METHOD

To study low-frequency microturbulence for magnetically confined plasmas, we start from the Vlasov equation in six-dimensional phase space for each particle species. In the gyrokinetic approach[4], by removing the high frequency motion of the particles that is not important to turbulent transport, we reduce the six-dimensional equation to a five-dimensional Vlasov equation:

$$\frac{df_\alpha}{dt} = \frac{\partial f_\alpha}{\partial t} + \frac{d\mathbf{R}}{dt} \cdot \frac{\partial f_\alpha}{\partial \mathbf{R}} + \frac{dv_\parallel}{dt} \frac{\partial f_\alpha}{\partial v_\parallel} = 0, \tag{1}$$

where $f_\alpha(\mathbf{R}, v_\parallel, \mu)$ is the five-dimensional phase space distribution function for species α in the gyrocenter coordinates $\mathbf{R}$ and $v_\parallel$ is the velocity parallel to the magnetic field.

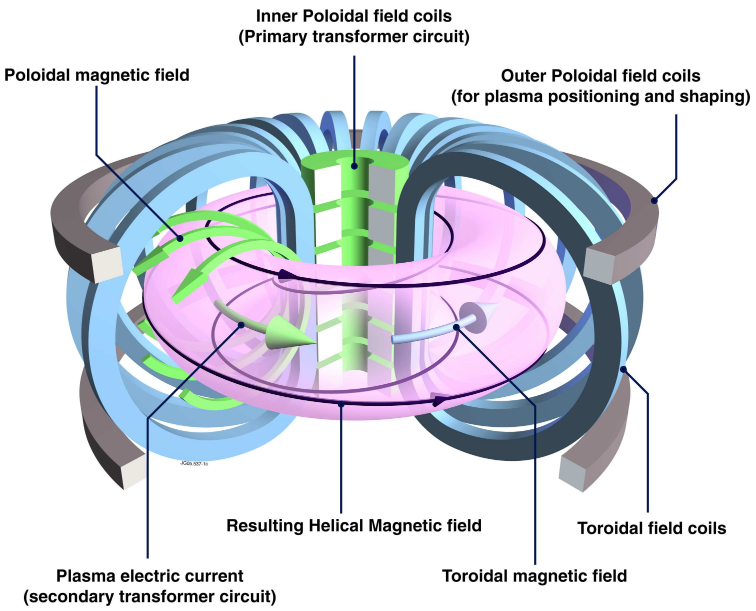
Inner Poloidal field coils
(Primary transformer circuit)
Poloidal magnetic field
Outer Poloidal field coils
(for plasma positioning and shaping)
Resulting Helical Magnetic field
Toroidal field coils
Plasma electric current
(secondary transformer circuit)
Toroidal magnetic field

FIG. 1: **Tokamak. (Source: EFDA-JET)**

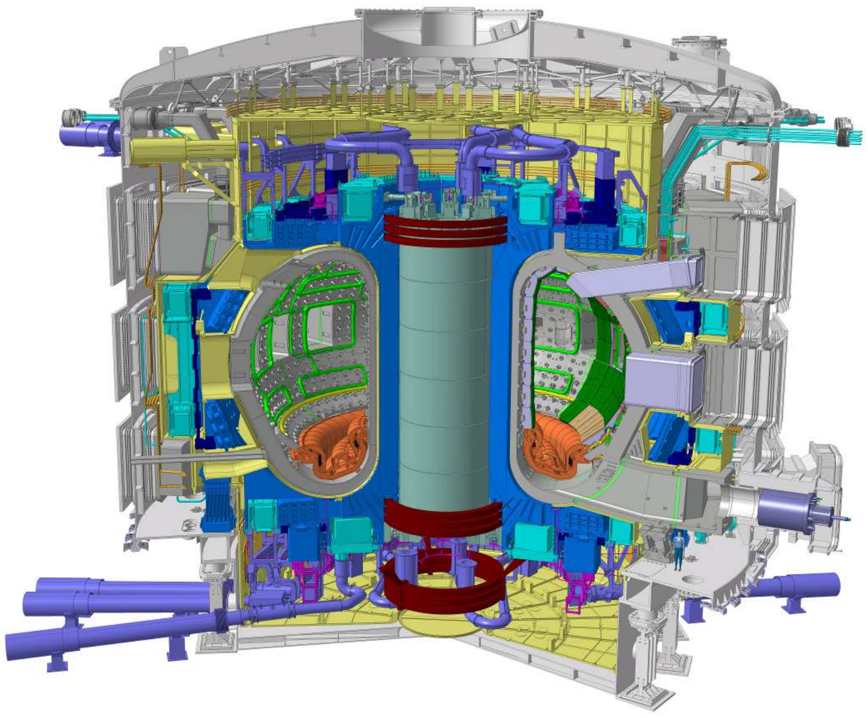

FIG. 2: **ITER. Human figure toward lower right corner indicates scale. (Image credit: © ITER Organization, http://www.iter.org/)**

Since plasmas are charged, we must also solve the relevant electromagnetic field equations. In the electrostatic gyrokinetic regime we're studying, the full Maxwell equations reduces to a single gyrokinetic Poisson equation.

In this project, we solve gyrokinetic Vlasov-Poisson system of equations using the Particle-In-Cell (PIC) approach.[4] Put simply, the ions are represented by a set of particles having the shape of charged rings perpendicular to the magnetic field, having radius equal to the ion gyroradius. The electric field is represented on a grid, through which the particles move continuously. The field is interpolated from the grid to each of four points on the gyro-orbit, which then is averaged to get the force that moves the particles (the particle *push*). Given the position of the particles, the electric charge of each particle is deposited onto a charge density array defined on the grid (one particle contributes to a small neighborhood of grid points around each of the 4 gyro-orbit points on the ring. Given the charge density on the grid, the nonlinear gyrokinetic Poisson equation is solved using a custom solver. The gradient of this potential field gives the electric field, which is then used to push the particles again, and so on. Figure 3 illustrates the process for a time step. In the work discussed here, the electrons in the plasma are treated as *adiabatic*, meaning they follow a simple Boltzmann response function (*i.e.,* not treated as particles).

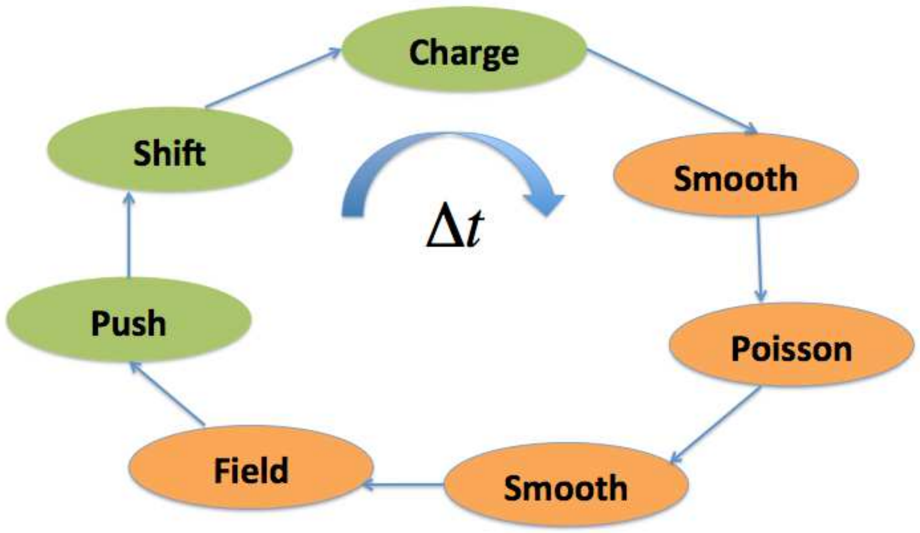

FIG. 3: Timestep elements for self-consistent evolution of plasma particles and electrostatic field. *Push:* interpolate electric field to positions of 4 gyro-orbit points of particles, use these to compute force and move particles. *Shift*: Send and receive particles that move between separate subdomains in the parallel decomposition of particles and fields. *Charge:* for 4 points on gyro-orbit of each particle, accumulate fractional charge density onto a neighborhood of grid points. *Smooth:* Smooth charge density and potential with a filter on the grid. *Poisson:* Solve gyrokinetic Poisson equation to get potential on the grid.

The grid geometry we use approximates the toroidal cross section as circular. Grid lines the long way around the torus follow magnetic field lines, which wind around helically. Figure 4 illustrates the coordinates, geometry, and several particles. The particles are rings with the 4 gyro-orbit points indicated, along with the neighborhood of grid points used to accumulate the charge density from or interpolate the force (electric field) from those 4 points. Not shown is the interpolation along the zeta direction. Psi is the radial coordinate, theta is the poloidal angle, and zeta is the toroidal coordinate.

III. CODES

The implementations we discuss here derive from the Gyrokinetic Toroidal Code (GTC)[5]. The variants we use and compare are two implementations of GTC-P, an electrostatic code with a circular cross section. Most newer tokamaks, and ITER, have D-shaped plasma cross sections, but

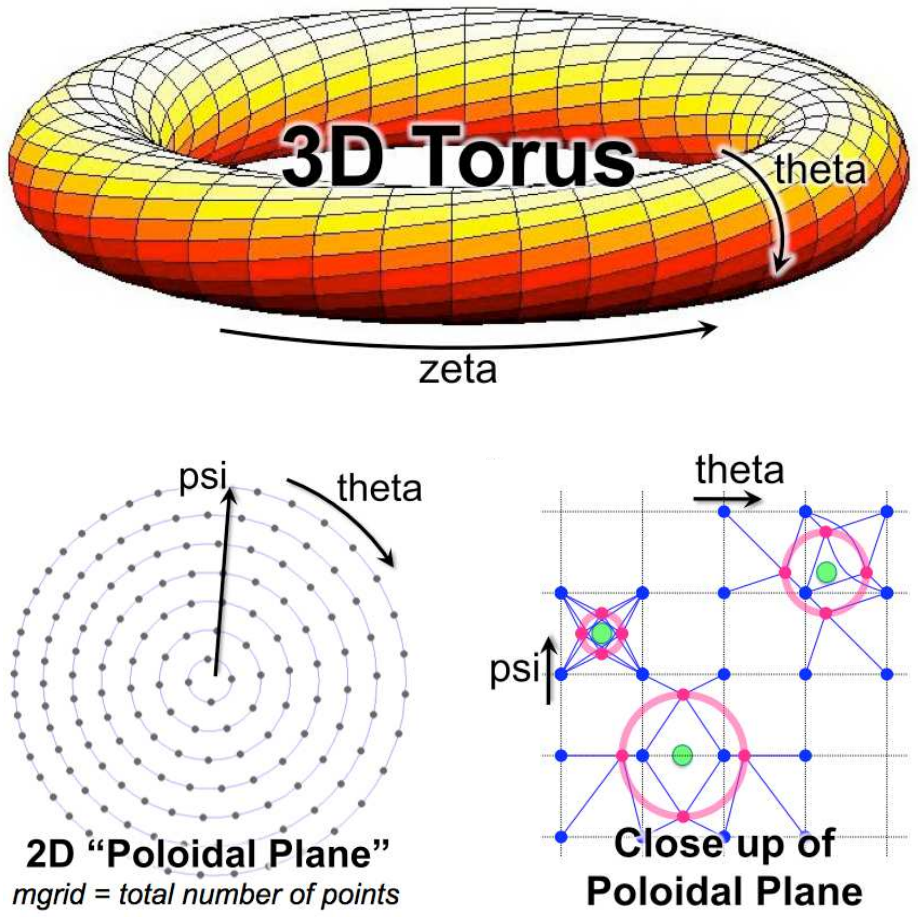

FIG. 4: An illustration of the 3D toroidal grid (top), a cross section (lower left), and the 4-point gyrokinetic averaging scheme employed in the charge deposition and push steps (lower right).

the effects of that geometry are not necessary to the basic physics we're investigating. When the ESP project started, until recently, we used a Fortran implementation, which we'll call GTC-P Fortran. More recently, we have moved to a new, from-scratch implementation in C, called GTC-P C.

IV. PARALLELIZATION

The original GTC code used 3 levels of parallelization:

1. One-dimensional domain decomposition in the toroidal (zeta) direction. Because of the physics of the plasma in the regime of interest, and because of the efficiency of the magnetic-field-line-following grid, we need only 64 grid zones toroidally, even for the largest problems we run. This limits parallelism in this dimension to 64; the MPI ranks are divided into a maximum of 64 subsets, one per toroidal subdomain.

2. Particle decomposition. Within each toroidal subdomain, the particles in that domain are distributed among the MPI ranks. Each rank maintains its own copy of the whole sub grid for the subdomain—all the poloidal and radial grid points, usually on two poloidal planes

bounding one toroidal grid zone. For the charge deposition from particle gyro-orbit points to the grid, it is necessary to use an `MPI_Allreduce` to sum up the contributions from all the MPI ranks' local grid copies.

3. Thread parallelism. For particle and grid operations within local toroidal subdomains, we use OpenMP to parallelize relevant loops over both particles and grid points.

More recently, we added an additional level of MPI parallelism: radial decomposition. Radial domain decomposition begins by partitioning a poloidal plane to non-overlapping domains with equal area. Assuming particle density is uniform, this partitioning divides all particles in one toroidal section equally across multiple processes. Next, the non-overlapping domain is extended to line up with the mesh boundary in the radial direction (shown as valid grid in Figure 5). Finally, the valid grid is extended on each side with ghost cells accounting for charge deposition with 4-point approximation (shown as local grid in Figure 5). In general, 3 to 8 ghost cells are sufficient. The 2D domain decomposition is implemented with MPI using two different communicators: a toroidal communicator and a radial communicator. The particles move between domains with nearest-neighbor communication in a circular fashion. Since the number of particles moving in the radial dimension is much smaller than the particles moving in the toroidal dimension, radial partitioning results in minimal communication. Within radial subdomains, we still allow partitioning of particles among more than one MPI rank.

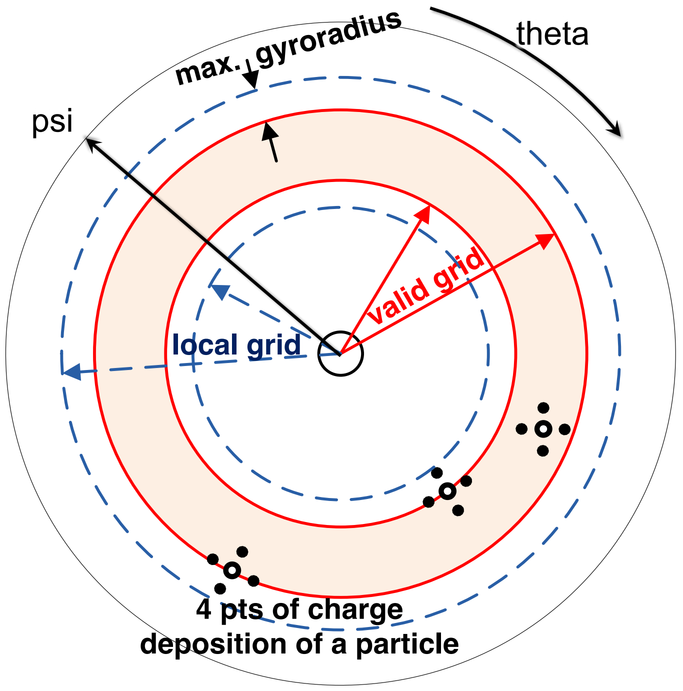

FIG. 5: A geometric radial partitioning with extended ghost zones. The valid region contains guiding centers for the MPI rank owning the radial subdomain. The full local region includes a number of ghost zones based on the maximum gyroradius of the particles (constant, typically 8, and the same for all radial subdomains).

V. KERNELS

Figure 3 shows 6 basic kernels in GTC-P, which can be grouped into 4 groups:

Charge Deposit charge from the 4 gyro-orbit points for each particle onto a neighborhood of grid points (neighborhoods are up to 32 grid points for each gyro-orbit point). This *scatter-add* operation must be managed when multiple ranks and/or threads are updating the same grid zones—either through using locks or using separate local copies of the grid, which then must be summed across ranks and/or threads.

Poisson/Field/Smooth Solve the gyrokinetic Poisson equation, compute the electric field, and smooth the charge density and potential fields. These are all grid-only operations. Because of the physics modeled (Debye shielding term much smaller than ion polarization term), it is sufficient to solve only independent 2D Poisson equations, one for each poloidal-radial plane.

Push Gather the force at each gyro-orbit point for each particle and use it to compute the force and advance the particle's phase space coordinates. The gather operation is the inverse of the scatter-add operation in the **charge** kernel. Here, we're only reading from grid arrays, so there is no locking or synchronization needed.

Shift As particles (that is, their gyro-orbit centers) move through space, they will cross subdomain boundaries into regions owned by other MPI ranks. they must be buffered and sent to the appropriate neighboring rank.

VI. OPTIMIZATIONS

Since the beginning of the Early Science Program, we have worked on two basic types of optimization of GTC-P in preparation for running on *Mira*:

1. Generic optimizations in single-core computations, parallel decomposition and message passing, and OpenMP threading.

2. IBM Blue Gene/Q specific optimizations

Here we will briefly touch on the generic optimizations, then discuss the BG/Q-specific optimizations. The full set of optimizations have been made only in the C version, GTC-P C, though any of them could also have been made to GTC-P Fortran. We discuss the performance impact of the optimizations in section VII.

A. Generic Optimizations

GTC-P Fortran used arrays of structures for particle data (each element stores all data quantities carried by a particle, including extras to handle two time levels in time integration). One optimization in GTC-P C is to change this to structures of arrays, to improve data locality of access when streaming through all the particles. This optimization benefits SIMD architectures in generally, including BG/Q and also GPGPU machines.[6–8]

As discussed in section III, GTC-P Fortran required each MPI rank to store a complete copy of the 2D radial-poloidal grid. The radial decomposition discussed there means that storage can be reduced as needed for optimal performance on a given architecture by tuning the radial decomposition.

To increase data locality and improve cache hits on grid accesses for the **charge** and **push** kernels, we periodically sort particles into radial bins. This tends to improve cache reuse for data loaded from the grid arrays. The binning algorithm is multithreaded.[6] Locality can further be improved by accounting for the 4 gyro-orbit points associated with each particle (charged ring). We bin the coordinates of the gyro-orbit points radially, using a preliminary pass through the particles to store the 4 points. The second pass through all these binned points only touches field grid points in a band of narrow width radially; compared to a gyro-radius-wide band of points if you are going through all 4 points for a single particle. In this way, data locality and cache reuse is improved. We use this two-level binning for the **charge** kernel; it doesn't improve performance on present-day machines in the **push** kernel.

We have also used loop-fusion to improve computational intensity. We fused OpenMP loops where possible to minimize thread creation overhead. We flattened 2D and 3D grid arrays to 1D arrays. Additionally, we pre-allocated memory buffers that are used for temporary storage in every time step.

Grid-based subroutines in GTC-P included nested loops with *psi* is the outer loop and *theta* is the inner. Near the center radially, the number of *theta* grid points changes significantly (as a percentage) with each increasing radial grid level. To mitigate load imbalance in multithreading these loop nests, we flattened the nest into a single loop over all grid points. Finally, GTC-P Fortran used version 2.3.3 of the PETSc library[9–11], which is not multithreaded, to solve the gyrokinetic Poisson equation; we replaced this with a multithreaded, hand-coded solver in GTC-P C. (*N.B.:* Newer versions of PETSc have added thread support.)

B. Blue Gene/Q Specific Optimizations

Many of the generic optimizations discussed previously led to improvement in GTC-P performance on *Mira*. The general goal was scaling up to much higher levels of concurrency, which is what we got with *Mira*, both in total number of cores ($\sim$ 750 million)and in total number of threads (4 hardware threads per core).

One BG/Q optimization that is important when running on large numbers of nodes is the mapping of MPI ranks onto the physical nodes and cores on the machine. IBM defines a naming convention for the communication dimensions in the torus: "ABCDET" means that MPI ranks are mapper to the system in a particular order: Each letter is associated with a dimension in the 5D torus (with "T" being an index across the MPI ranks (processes) in a single node).[12] The last dimension (T in this example) is fastest varying, so, if running with one MPI rank per node:

- MPI rank 0 is assigned to coordinates $\langle 0,0,0,0,0,0\rangle$
- MPI rank 1 is assigned to coordinates $\langle 0,0,0,0,1,0\rangle$
- MPI rank 2 is assigned to coordinates $\langle 0,0,0,0,2,0\rangle$
- ....

The second-to-last dimension (E in this example) is the next-fastest varying, and so on. The highest value for each dimension depends on the number of nodes in the partition you're using within the machine. For anything 512 nodes or larger, you have your own isolated partition on BG/Q, generally itself a torus. The optimized GTC-P C code uses a two-dimensional topology for point to point communication, where the first dimension (toroidal dimension) has fixed dimensionality

64. On a BG/Q system with 5D torus network, we can thus group two or three torus dimensions together to match 64 for an optimized placement layout by setting the environment variable RUNJOB_MAPPING. Table I shows some examples of configurations and groupings when we run the application with one process per node. This explicit process mapping leads up to a 45% communication improvement for particle shift in the toroidal dimension using 8 racks (8192 nodes) of *Mira*.

Configuration	**Torus Shape**	**Grouping**
256 nodes (1/4 rack)	4 2 4 4 2	ABCE×D×T
512 nodes (1/2 rack)	4 4 4 4 2	ABC×DE×T (default)
1024 nodes (1 rack)	4 4 4 8 2	ABC×DE×T (default)
2048 nodes (2 racks)	4 4 4 16 2	ABC×DE×T (default)
4096 nodes (4 racks)	4 4 8 16 2	ACE×BD×T
8192 nodes (8 racks)	4 4 16 16 2	AC×BDE×T

TABLE I: Process Mapping on BG/Q. Not shown in the torus shape are the on-node"T" dimensions, whose maximal value depends on the chosen number of MPI ranks per node.

As BG/Q is a highly multithreaded architecture (up to 64 OpenMP threads per MPI rank), efficient use of OpenMP is essential for attaining high performance. Some of the generic optimizations in section VI A, especially those for grid-based calculations, improved the OpenMP performance and scalability. It is very important to be aware of environment variables controlling thread behavior on BG/Q. The settings BG_SMP_FAST_WAKEUP=YES and OMP_WAIT_POLICY=active make threads use an atomic-based spin barrier instead of a slower sleep-based approach. For some of the grid-based routines, these settings resulted in a 10× speedup when using 64 threads per MPI rank.

VII. PERFORMANCE

Here we present some performance measurements on a set of different problem sizes. These are the four problem sizes used to study the Bohm-to-GyroBohm scaling transition discussed in section I. The parameters are shown in Table II; the number of particles is based on the *micell* parameter, which is the number of particles per grid cell. Grid sizes A and B correspond to the majority of existing tokamaks in the world, C corresponds to the JET tokamak, the largest device currently in operation [10], and D corresponds to to ITER.

Grid Size	**A**	**B**	**C**	**D**
mpsi	90	180	360	720
mthetamax	640	1280	2560	5120
mgrid (grid points per plane)	32449	128893	513785	2051567
chargei grid (MB)	0.5	1.97	7.84	31.30
evector grid (MB)	1.49	5.90	23.52	93.91
Total particles *micell*=100 (GB)	0.29	1.16	4.64	18.56

TABLE II: The GTC numerical settings for different plasma sizes. The grid and particle memory requirement are for one toroidal domain only. A simulation typically consists of 64 toroidal domains. *mpsi* and *mthetamax* are the number of grid points in the radial (*psi*) and poloidal (*theta*) coordinates.

A. Strong Scaling

First, we consider strong scaling—running a fixed problem size on an increasing sequence of compute nodes. We denote the problem as D100. The problem size is D (ITER size), and we use the typical production setting of 100 particles per cell. (This is typical when running a *delta-f* simulation, where the particles represent only the difference of the phase space distribution with respect to a constant Maxwellian background distribution.) In the toroidal direction, we use *ntoroidal*=64 grid cells. This is distributed among 64 sets of MPI ranks, so each toroidal subdomain has two poloidal planes having 2 million grid points each. This global simulation is 130 million grid points and 13 billion particles.

For the first strong scaling test, we increase the number of radial partitions from 32 to 512. We run on *Mira* with 4 MPI ranks per node and 16 OpenMP threads per rank; the simulations scale from 512 to 8192 nodes (8192 to 131,072 cores). Table III shows the results. The GTC-P C code uses the optimizations detailed in section VI, which combine to give a substantial improvement in overall runtime ($\sim 2\times$ on 32768 nodes) and a substantial improvement in parallel efficiency. Note that these runs did not use particle decomposition; this allows strong scaling up to even larger node counts.

MPI Ranks	**Radial Partitions**	**GTC-P Fortran**	**Parallel Efficiency**	**GTC-P C**	**Parallel Efficiency**	**Speedup**
2048	32	9.282	1.0	5.275	1.0	1.76
4096	64	4.685	0.99	2.651	0.99	1.76
8192	128	2.536	0.92	1.373	0.96	1.85
16384	256	1.453	0.80	0.726	0.91	2.00
32768	512	0.873	0.60	0.414	0.80	2.21

TABLE III: Wall-clock time (sec) for one time step with strong scaling in radial domain decomposition for D100 using GTC-P Fortran and GTC-P C. In all experiments, we use 4 processes/node and 16 threads/process. GTC-P C attains a $2\times$ speedup due to our optimizations.

For the second strong scaling test, we consider scaling with respect to the number of OpenMP threads per MPI rank. Table IV shows the results. We hold the number of MPI ranks (processes) constant at 32768. As we increase from 2048 nodes to 32768 nodes by doubling, the amount of available hardware thread concurrency doubles, since we're reducing freeing up more cores. With perfect thread scaling, the run times would halve each step. For the D100 problem, the parallel efficiency at the largest number of nodes has dropped to 59%.

D100: 13 Billion particles, 32768 total processes, with 400556 particles per process											
Nodes	**Processes×Threads**	**Charge**	**Push**	**Shift_t**	**Shift_r**	**Binning**	**Poisson**	**Field**	**Smooth**	**Total**	**Eff.**
2048	16×4	0.6691	0.4569	0.3073	0.0326	0.0602	0.0197	0.0050	0.0069	1.558	1.0
4096	8×8	0.3397	0.2313	0.1536	0.0169	0.0340	0.0078	0.0027	0.0047	0.791	0.99
8192	4×16	0.1822	0.1178	0.0779	0.0094	0.0167	0.0044	0.0021	0.0039	0.414	0.94
16384	2×32	0.1110	0.0591	0.0722	0.0063	0.0087	0.0033	0.0022	0.0039	0.267	0.73
32768	1×64	0.0709	0.0298	0.0487	0.0050	0.0039	0.0022	0.0018	0.0035	0.166	0.59

TABLE IV: Strong scaling of threads per process for D100 using GTC-P C on *Mira*. The time is the wall-clock time (sec) for one time step. We use 64-way toroidal and 512-way radial partitioning.

For the third strong scaling test, we look at how the optimizations made in GTC-P C improve fine-grained parallelism with respect to GTC-P Fortran. Table V breaks down the relative performance improvement by numerical kernel. The problem and set of runs are the same as in Table IV. The biggest speedups are in the grid-based kernels, because of the loop flattening and other optimizations in section VI. The speedups in **charge** and **push** are mainly because of binning, and avoiding synchronization in **charge**. Both the Fortran and C codes employ the private grid replication strategy on a per thread basis for charge deposition. However, GTC-P Fortran uses a `critical section` to merge charges from all copies private of threads. This serial portion of the code can be easily avoided by carefully reorganizing the summation order.

Nodes	**2048**	**4096**	**8192**	**16384**	**32768**
Thread/Process	**4**	**8**	**16**	**32**	**64**
charge	1.14×	2.06×	1.95×	3.52×	8.80×
push	1.71×	1.71×	1.69×	1.72×	1.70×
shift	1.37×	2.05×	2.23×	2.23×	2.85×
poisson	6.84×	12.05×	13.57×	13.55×	16.45×
field	41.02×	35.74×	28.43×	20.32×	20.11×
smooth	28.87×	23.87×	21.72×	16.46×	16.40×
overall speedup	1.43×	1.78×	2.11×	2.87×	5.56×

TABLE V: Speedup (GTC-P C vs GTC-P Fortran) on D100 problem by kernels with different threads per process. The total processes (32768), the number of particles in each process (400556) and the number of radial partitions (512) are fixed. Shift includes shift_t, shift_r and binning in Table IV.

B. Weak Scaling

Next, we consider weak scaling—running a sequence of problem size on an increasing sequence of compute nodes, keeping the amount of work per node constant. We use the four problem sizes used to study the Bohm-to-GyroBohm scaling transition discussed in section I. We use 100 particles per cell, one MPI rank per node, and 64 threads per rank.

The parallelism in GTC is denoted as three parameters: the number of toroidal partitions (*ntoroidal*), the number of radial partitions (*nradiald*), and the number of particle partitions in one spatial subdomain (*npe_radiald=npartdom/nradiald*, where *npartdom* is the total number of particle partitions in one toroidal partition). Table VI shows the settings for the weak scaling study. Figure 6 shows the results, comparing GTC-P Fortran and GTC-P C. Some efficiency loss is still seen for GTC-P C, but keep in mind that here we are using 2/3 of *Mira*: 524,288 cores.

Problem Size	**Nodes**	***ntoroidal***	***npartdom***	***nradiald***	***npe_radialid***
A	512	64	8	1	8
B	2048	64	32	1	32
C	8192	64	128	1	128
D	32768	64	512	1	512

TABLE VI: Parameters for weak scaling study

Acknowledgments

Dr. Wang was supported by the NSF OCI-1128080/G8 Initiative: G8 Research Councils Initiative on Multilateral Research Funding. Authors from Princeton Plasma Physics Laboratory were by the DOE

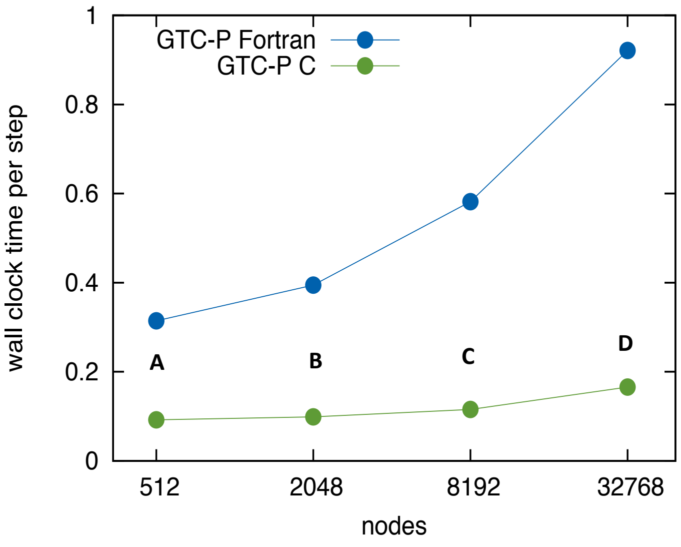

FIG. 6: Weak scaling from A to D size plasmas using GTC-P Fortran and GTC-P C on *Mira*.

contract DE-AC02-09CH11466. Authors from Lawrence Berkeley National Laboratory were supported by the DOE Office of Advanced Scientific Computing Research under contract number DE-AC02-05CH11231. Dr. T. Williams was supported by the DOE contract number DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is supported by the Office of Science of the U.S. Department of Energy under contract DE-AC02-06CH11357. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

[1] Y. Shimomura, R. Aymar, V. Chuyanov, M. Huguet, R. Parker, and ITER Joint Central Team. Iter overview. *Nuclear Fusion*, 39(9Y):1295, 1999.

[2] Andrew Guthrie and Raymond Kornelious Wakerling. *The characteristics of electrical discharges in magnetic fields*, volume 5. McGraw-Hill, 1949.

[3] G. Manfredi and M. Ottaviani. Gyro-Bohm Scaling of Ion Thermal Transport from Global Numerical Simulations of Ion-Temperature-Gradient-Driven Turbulence. *Physical Review Letters*, 79(21):4190–4193, November 1997.

[4] W.W Lee. Gyrokinetic particle simulation model. *Journal of Computational Physics*, 72(1):243–269, September 1987.

[5] Z. Lin. Turbulent Transport Reduction by Zonal Flows: Massively Parallel Simulations. *Science*, 281(5384):1835–1837, September 1998.

[6] Kamesh Madduri, Eun-Jin Im, Khaled Z. Ibrahim, Samuel Williams, Stphane Ethier, and Leonid Oliker. Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms. *Parallel Computing*, 37(9):501 – 520, 2011.

[7] Kamesh Madduri, Khaled Z. Ibrahim, Samuel Williams, Eun-Jin Im, Stephane Ethier, John Shalf, and Leonid Oliker. Gyrokinetic toroidal simulations on leading multi- and manycore hpc systems. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '11, pages 23:1–23:12, New York, NY, USA, 2011. ACM.

[8] Kamesh Madduri, Samuel Williams, Stéphane Ethier, Leonid Oliker, John Shalf, Erich Strohmaier, and Katherine Yelicky. Memory-efficient optimization of Gyrokinetic particle-to-grid interpolation for multicore processors. In *Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis - SC '09*, SC '09, pages 48:1–48:12, New York, New York, USA, November 2009.

ACM Press.

[9] Satish Balay, Jed Brown, Kris Buschelman, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc Web page, 2012. http://www.mcs.anl.gov/petsc.

[10] Satish Balay, Jed Brown, , Kris Buschelman, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Barry F. Smith, and Hong Zhang. PETSc users manual. Technical Report ANL-95/11 - Revision 3.3, Argonne National Laboratory, 2012.

[11] Satish Balay, William D. Gropp, Lois Curfman McInnes, and Barry F. Smith. Efficient management of parallelism in object oriented numerical software libraries. In E. Arge, A. M. Bruaset, and H. P. Langtangen, editors, *Modern Software Tools in Scientific Computing*, pages 163–202. Birkhäuser Press, 1997.

[12] IBM Redbooks — IBM System Blue Gene Solution: Blue Gene/Q Application Development - Update, April 2013.