Heterogeneous Programming and Optimization of Gyrokinetic Toroidal Code Using Directives

Wenlu Zhang[1,2], Wayne Joubert[3], Peng Wang[4], Matthew Niemerg[5], Bei Wang[6], William Tang[6], Sam Taimourzadeh[1], Lei Shi[1], Jian Bao[1], Zhihong Lin[1]

[1] Department of Physics and Astronomy, University of California, Irvine, California, USA
[2] Institute of Physics, Chinese Academy of Sciences, Beijing, China
[3] Oak Ridge National Lab, Oak Ridge, TN, USA
[4] NVidia, USA
[5] IBM, USA
[6] Princeton University, Princeton, NJ, USA

zhihongl@uci.edu

Abstract. The latest production version of the fusion particle simulation code, Gyrokinetic Toroidal Code (GTC), has been ported to and optimized for the next generation exascale GPU supercomputing platform. Heterogeneous programming using directives has been utilized to fuse and thus balance the continuously implemented physical capabilities and rapidly evolving software/hardware systems. The original code has been refactored to a set of unified functions/calls to enable the acceleration for all the species of particles. Binning and GPU texture caching technique have also been used to boost the performance of the particle push and shift operations. In order to identify the hotspots, the GPU version of the GTC code was the first benchmarked on up to 8000 nodes of the Titan supercomputer, which shows about 2–3 times overall speedup comparing NVidia M2050 GPUs to Intel Xeon X5670 CPUs. This Phase I optimization was followed by further optimizations in Phase II, where single-node tests show an overall speedup of about 34 times on SummitDev and 7.9 times on Titan. The real physics tests on Summit machine showed impressive scaling properties that reaches roughly 50% efficiency on 928 nodes of Summit. The GPU+CPU speed up from purely CPU is over 20 times, leading to an unparalleled speed.

Keywords: Massively Paralleled Computing, Heterogeneous Programming, Directives, GPU, OpenACC, Fusion Plasma, Particle in Cell.

1 Introduction

Fusion energy would ensure a safe, environmentally friendly, resource conserving power supply for future generations. In an operating fusion reactor, part of the energy generated by fusion itself will serve to maintain the plasma temperature as fuel is introduced. However, to achieve the desired levels of fusion power output, the plasma in a reactor has to be heated and maintained to its operating temperature of greater than 10 keV (over 100 million degrees Celsius) and additional current induction must be applied. Confinement of such a high density and high temperature burning plasma poses big challenges to both science and technology researches. One critical mission for the fusion energy research and development is the timely achievement of the capability to understand, predict, control, and mitigate performance-limiting and integrity-threatening instabilities in the burning plasmas. The excitation and evolution of the most important instabilities can be expected to depend on kinetic effects and the nonlinear coupling of multiple physical processes spanning disparate spatial and temporal scales.

In the research of fusion plasma physics, simulations have always been an effective tool due to the complexity of theoretical analysis and the high cost of experiments. After several decades of fast development in the capability of high-performance-computing, it becomes feasible to conduct the consuming massively paralleled simulations to investigate the complex physics using equilibrium and profiles close to realist discharges in fusion devices. Along with the progress in computing power, a set of gyrokinetic theory[1–11] have been proposed and established to construct a set of simple theoretical and numerical model by eliminating the fine-scale gyro-phase dependence through gyro-averaging, which reduces the original phase space dimensionality from six to five. This not only

assists in the comprehension of the low frequency physics in magnetized plasmas, such as the anomalous transport that is critical for the magnetic fusion, but also facilitates the development and application of massively paralleled simulation codes.

As a well benchmarked massively parallel gyrokinetic toroidal code, GTC[12,13] is built upon the first-principles and adopts efficient low-noise numerical simulation methods for integrated simulations of key instabilities. This is of great significance since these instabilities not only limit the burning plasma performance but also threaten device integrity in magnetically-confined fusion systems such as the International Thermonuclear Experimental Reactor (ITER)[14], which is a crucial next step in the quest for the fusion energy on earth. The particle-in-cell method is utilized so that particles are treated with a Lagrangian scheme while fluid moments and field information are calculated with an Eulerian scheme. The capability of GTC has been extensively expanded and verified to deal with a wide range of physical problems such as neoclassical and turbulence transport[15,16], energetic particle transport by microturbulence[17,18], Alfven eigenmodes[19–22], radio frequency heating[23], static magnetic island[24] and current-driven instabilities[25,26]. Over the years, the GTC code has grown from a single-developer code to a prominent code being developed by an international collaboration with many users and contributors from the magnetic fusion energy and high performance computing communities.

GTC is the key production code for several multi-institutional U.S. Department of Energy (DOE) Scientific Discovery through Advanced Computing (SciDAC) project and the China National Magnetic Confinement Fusion Science Program, for example. GTC is currently maintained and developed by an international team of core developers who have the commit privilege and receives contributions through the proxies of core developers from collaborators worldwide[13]. GTC continuously pushes the frontiers of both physics capabilities and high-performance computing. It is the first fusion code to reach the teraflop in 2001 on the Seaborg computer at NERSC[27] and the petaflop in 2008 on the Jaguar computer at ORNL[28] in production simulations which is also the benchmark and early application code that fully utilizes the computing power of a list of TOP500 machines including but not limit to Tianhe-1A[29] and Titan with a CPU and GPU heterogeneous architecture and Tianhe-2[30] with an Intel Xeon Phi accelerator architecture.

In the pursue of extreme performance from the high computing community, many excellent pioneer works have been carried by computer scientists and developers by porting and optimization the GTC and its derived codes to the GPU on variety of machines. The work of Madduri[31,32] et al. discussed the porting of an earlier version of GTC to GPU at that time. They concluded that the GPU was slower than the CPU for their version of GTC, which only included kinetic ions with adiabatic electrons. Then the GTC GPU[29] version, which was the Tianhe-1A benchmark code developed on the production version using NVidia CUDA libraries, showed excellent speedup and scaling in the whole machine test with the actual physics simulation parameters. The weak scaling to 3072 nodes of Tianhe-1A was obtained with 2X-3X overall speedup comparing NVidia M2050 GPUs to Intel Xeon X5670 CPUs. A "companion" version of the electromagnetic GTC code, the electrostatic GTC-P code is a modern, highly portable GTC code now operational on the top 7 supercomputers worldwide[33]. Over the years, GTC-P has been ported and optimized on different supercomputers such as IBM Blue Gene/P (BG/P) at Argonne Leadership Computing Facility (ALCF), IBM Blue Gene/Q (BG/Q) of Mira at ALCF, Sequoia at Lawrence Livermore National Laboratory, the Cray XT4, Cray XE6, and later Cray XC30 at Lawrence Berkeley National Laboratory, et al[33,34]. The scalability up to 131,072 BG/P and 32,768 XT4 cores were attained with as little as 512MB memory per core by incorporating a new radial decomposition method, developed first by Ethier et al that features a dramatic increase in scalability for the grid work and decrease in the memory footprint of each core[34]. Later, Kamesh et al. made further optimizations of the code, such as multi-level particle and grid decompositions, particle binning, and memory-centric optimizations. As a result, they delivered 1.22x, 1.35x, 1.77x, and 1.34x performance improvement on BG/P, the Cray XE6, and Intel Cluster, and a Fermi Cluster respectively[35]. Recently, the radial domain decomposition was optimized by Wang et al, which enables the GTC-P code scale up to the full capability of Sequoia (98,304 nodes), and Mira (49,152 nodes)[33]. The performance was increased from nearly 50 billion particles per second per step (BPST) to more than 100 BPST on 98,304 Sequoia nodes. GTC-P was also weak scaling to 32,768 Fujitsu K nodes, and about 50 BPST was achieved [36].

In this work, the associated R&D has been focused toward the goal of delivering a comprehensive and modern production version of the fusion GTC code capable of greatly accelerating progress toward a realistic predictive capability for ITER experiments. The technical advances are aimed at providing the computational foundations needed for simulating nonlinear interactions of multiple physical processes covering disparate spatiotemporal scales in burning plasmas. This is a part of efforts to develop the next generation applications for exascale supercomputing platforms. For vast portability and easy maintenance, the directive approach is chosen to lower the technical requirement for students and researchers of fusion plasma physics.

GTC is one of a small but growing number of production applications run on leadership class systems to employ compiler directives to access modern accelerated node hardware. Use of compiler directive programming models such as OpenACC and OpenMP is of increasing importance to achieve performance portability across multiple target computer architectures. We believe the lessons learned described in this paper will be useful to other developers wishing to use directives for programming to accelerated architectures.

This paper is organized as follows. Section 2 briefly introduces the benchmark platforms of Titan and SummitDev. Section 3 discusses the technical basis of the GTC code, which is followed by the porting and optimization strategies in Section 4. Section 5 reports the status of the porting and optimization and Section 6 shows the performance benchmarks. The conclusions are given in Section 7 at the end.

2 Simulation Platforms: Titan, SummitDev and Summit

All the benchmark runs in the following sections were performed on the Titan and SummitDev supercomputers, both hybrid massively parallel processing (MPP) systems with CPUs and GPUs.

The Titan system at Oak Ridge National Laboratory (ORNL) is a Cray XK7 system composed of 200 cabinets containing 18,688 compute nodes, each equipped with a 16-core Advanced Micro Devices AMD Interlagos processor with 32 GB of memory and an NVidia Kepler K20X GPU accelerator with 6 GB memory, with Gemini interconnect. Titan's peak speed is in excess of 27 petaflops. The GPU attains a peak double precision rate of 1.311 TF/sec with main memory bandwidth of 250 GB/sec and is connected to the CPU by a PCI Express Gen 2.0 bus with an 8 GB/s data transfer rate[43].

SummitDev is an early access system at ORNL used by developers to prepare applications for the 200 PF Summit system to be available in 2018. SummitDev is comprised of 54 IBM Power8 S822LC compute nodes connected with a Mellanox EDR Infiniband network, each node containing two IBM POWER8 processors with 10 cores and 80 hardware threads each. Each CPU is connected by an 80 GB/sec NVLINK connection to two NVidia P100 GPUs with peak double precision rate of 5.312 TF/sec and with 16 GB of on-package high bandwidth memory with peak speed of 732 GB/sec[44].

Summit is the next generation leadership supercomputer at ORNL, which is the 200PF system built upon IBM AC922 architecture. It consists of 4,608 nodes linked with Mellanox EDR 100G InfiniBand network, each node host 2 22-core IBM Power 9 CPUs, 6 Nvidia Volta GPUs, 512GB DDR4 memory and 96GB HBM2 memory on GPU.

3 Scientific Methods of GTC

As a gyrokinetic particle-in-cell[37,38] (PIC) code, GTC tracks individual charged marker particles in a Lagrangian frame in a continuous phase-space[10,11], whereas the moments of particle distribution of different species (thermal ion, thermal electron, fast ion, fast electron, etc.) are simultaneously computed on a stationary Eulerian field mesh. This field mesh is also used to interpolate the local electromagnetic fields at the marker particle positions in phase-space. The trajectories of charged marker particles (guiding centers) in a strong magnetic field are calculated by integrators of the equations of motion in the self-consistent electromagnetic fields computed on the field mesh. The number density and current density carried by each marker particle is then projected to the field mesh through

interpolations. The moments of the distributions of species, such as number density, charge density and current density, are then calculated by accumulating the projected quantities of marker particles. The electromagnetic fields are then solved on mesh grids using proper combinations of Poisson equation, Ampere's law, Faraday's law and force-balance equations with finite difference methods[39] and finite element methods[40].

The PIC approach implemented in GTC dramatically reduces the computation complexity from $O(N^2)$ to $O(N + MlogM)$, where N is the number of particles, and M is the number of grid points[36,41]. The use of spatial grids and the procedure of gyro-averaging reduce the intensity of small-scale fluctuations (particle noise). Particle collisions can be recovered as a "sub-grid" phenomenon via Monte Carlo methods. The system geometry simulated in GTC is a torus with an externally-imposed equilibrium magnetic field[35]. In order to capture and take advantage of the characteristics of this curvature geometry, GTC employs the magnetic flux coordinate system (ψ, θ, ζ)[42], where ψ is the poloidal magnetic flux, θ is the poloidal angle and ζ is the toroidal angle. This is the base coordinate used for mesh construction, on which the equilibrium and profiles are built. It is also used to construct an intermediate field-line-following coordinate (ψ, θ, α) by a simple transformation$\alpha = \zeta - q(\psi)\theta$. The introduction of such a field-line coordinate system makes it convenient to decompose a vector into components parallel and perpendicular to the direction of magnetic field and to separate the rapid guiding center motion along the magnetic field lines from the slow motion across the lines, which promotes the simplicity in theory analysis and efficiency in numerical simulation. On the other hand, the field-line coordinate system drastically reduced computational complexity in the parallel direction. The Poisson equation can be simplified and solved in the (ψ, θ) plane perpendicular to the equilibrium magnetic field in this field-line coordinate system.

Quantities and variables in GTC can be divided into categories.

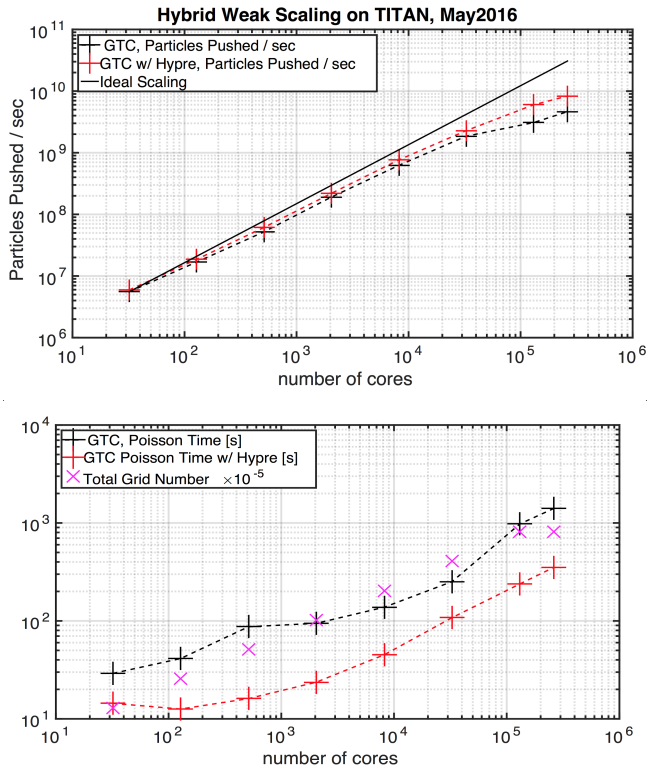

Figure 1. Phase-I weak scaling of GTC on Titan (top), with the number of nodes ranging from 32 to 16384 (88% of the whole machine). Both grid number and total particle number are increased, but the number of particles per core remains constant. The Poisson time (bottom) shows the improved performance due to the Hypre multigrid solver. The total grid number for the scaling is also shown.

The first one includes the field quantities bounded to the stationary mesh, such as electrostatic potential, vector potential, magnetic fields, and accumulated number density and current density distributions on mesh. Originally, the field solver was built on the Portable, Extensible Toolkit for Scientific Computation (PETSc), which was the

best choice in the dual core and multiple code age and has been the major solver for daily electromagnetic simulations. However, it gradually emerges as a serious performance hot spot later in the many-core and heterogeneous architecture era due to its lack of in-node accelerations for many core architectures, for instance for general purpose GPU and Intel Xeon Phi.

The other category includes marker particle related quantities for every species, such as physical position, velocity or momentum, particle number and electric current carried by each marker particle. Depending on the physics studied, a typical production run in fusion plasma research may have multiple species with different governing equations, such as thermal ions, thermal electrons, energetic ions, energetic electrons, impurities, etc. Originally, each species had its own set of functions and subroutines which used to calculate the particle trajectories (push subroutine), manage and exchange particle information between computing devices and processes (shift subroutine), and aggregate number density and current density as well as the thermal pressure carried by each particle (charge subroutine).

GTC has successfully transferred the physical models into computing power by employing a multi-level palatalization technique, which utilizes the Message Passing Interface (MPI) to manage and balance the distributed computing resources cross computing nodes or devices on the top level, and utilizes shared memory multiprocessing (SMP) techniques via OpenMP and OpenACC/CUDA inside each node or device on the lower level so that it can take the advantage of the hardware hierarchy of modern massively paralleled computers and reach a scale up to millions of conventional CPU cores and heterogeneous accelerating devices such as NVidia GPU and Intel Xeon Phi (MIC) chips.

4 Porting and Optimization Strategy

When porting the GTC code to the next generation supercomputing machines powered by accelerators or co-processors such as the GPU or Intel Xeon Phi (MIC), significant challenges are anticipated. Achieving high parallel efficiency on complex modern architectures is in general a formidable task facing PIC codes because of potential fine-grained data hazards, irregular data access, and low arithmetic intensity. Attaining high performance becomes an increasingly complex challenge as HPC technology evolves towards vast on-node parallelism in modern multi- and many-core designs. In order to harness the computing power of advanced systems such as Summit, application codes, including GTC, need to be carefully designed such that the hierarchy of parallelism provided by the hardware is fully utilized. To this end, the multithreading capabilities in the GTC code will be enhanced.

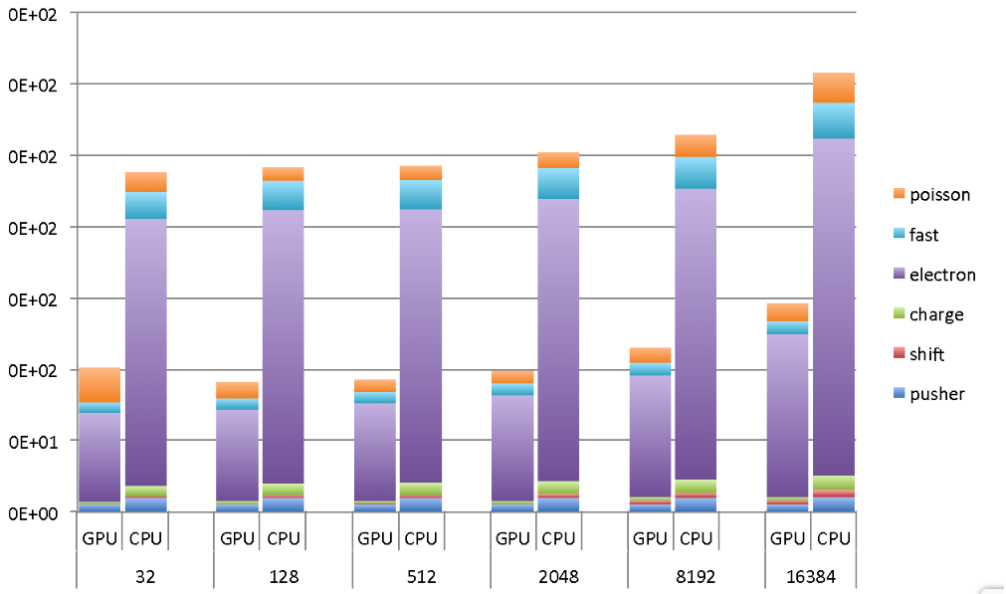

Figure 2. The Phase I timing breakdown for GTC particle weak scaling study on Titan. Note: x-axis is the number of nodes and y-axis the total wall-clock time. The GPU delivers up to 3.0X speedup compared with the CPU.

GTC was originally written in Fortran 90. The current GTC version has four species of particles: thermal ions, fast ions, fast electrons and kinetic thermal electrons. Many routines are shared between those particle types.

In fusion simulations using GTC, the number of particles per mesh cell varies from tens to thousands in a typical production run for each particle species, which means that every cell would have *O(10)* - *O(10^3)* of particles. In other words, the total number of particles is *O(10)* - *O(10^3)* larger than the total number of cells (with field data on cells). Most of the data, either on disk or in memory, and runtime---including I/O time and computing time---are accordingly consumed by the particle routines instead of field routines, which has been consistent with our benchmarking results.

The preceding analysis and observations suggest that particle related routines are the key for optimizing the PIC code like GTC. An appropriate effective strategy for porting GTC to a CPU-GPU heterogeneous architecture would be as follows: migrate all particle relevant data and computing onto the GPU. This approach will not only enable the utilization of the most powerful computing unit of the heterogeneous architecture but also minimize the data transfer between the CPU and the GPU which can be the most challenge part when utilizing GPU in high performance computing. Instead of porting each particle species one by one, all the particle related routines are replaced with a set of unified push, charge and shift routines, which can then be ported to the GPU using OpenACC. After the successful port of particle related part, the field solvers will also be ported onto the GPU to boost the computing performance of field solvers.

Given the existing MPI-OpenMP framework, the most natural parallel framework for GTC on CPU-GPU nodes would be using one MPI rank per GPU. Since the CPU version is already parallelized using OpenMP, OpenMP threads should also be enabled to utilize all the available CPU cores.

A large part of the performance optimization work will thus focus on multithreading for NVidia GPU and Intel Xeon Phi architectures (MIC), as well as current multicore processors. Fortunately, the GTC code has been using multithreading for more than a decade and has already had initial porting efforts to advanced heterogeneous architecture systems that deploy the GPU and Intel MIC.

To satisfy the needs for performance portability across multiple HPC system architectures, GTC will initially support multiple programming models via conditional compilation. For shared memory multi-core and Intel Xeon Phi many-core processors, OpenMP parallel directives are used. Support for NVidia GPUs will be deployed using OpenACC directives. An alternative conditionally compiled CUDA code path will be available for cases when

compilers are not yet able to generate well-performing code for OpenACC. Later as compiler support becomes available, OpenMP 4.5 and 5.0 target directives will be evaluated as a potential performance portability solution.

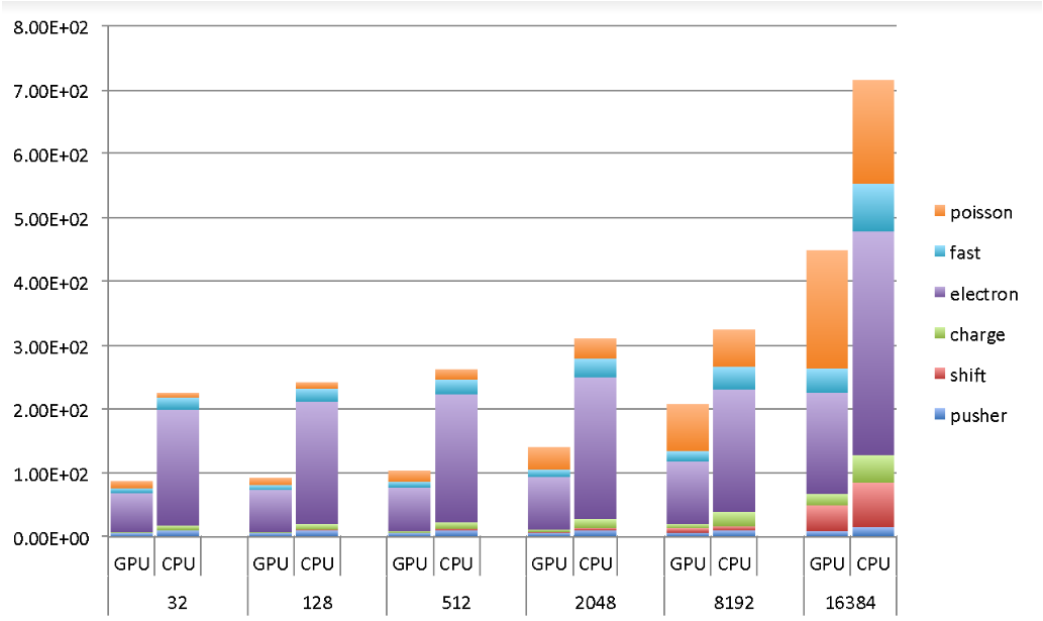

Figure 3. The Phase I timing breakdown for GTC hybrid weak scaling study on Titan. Here the work per processor is increased as node count is increased. Note: x-axis is the number of nodes and y-axis the total wall-clock time. GPU delivers up to 3.0x speedup compared with CPU.

GTC currently uses the DOE-funded PETSc toolkit to implement the electromagnetic parallel solvers. PETSc is a well-established MPI-based framework through which many different solvers can be exercised without having to change the source code. Advanced third-party packages, such as LLNL's Hypre multigrid solver, can also be used via the PETSc framework with a simple parameter change in a runtime configuration file. Nevertheless, in spite of its successful use in GTC, PETSc has some limitations with respect to today's advanced computer architectures. The main drawback is its lack of multithreading support, which especially impacts global PIC codes like GTC since they run routinely in mixed-mode MPI+OpenMP. It would clearly be beneficial to introduce OpenMP multithreading at the lowest level, for example, in the PETSc functions. This would help us avoid having to deal with non-thread-safe issues in higher-level functions.

In order to identify the hotspots and performance issues, code profiling was performed to reveal performance characteristics and to identify performance issues. The code's timer instrumentation for major parts of the computation was revised to provide performance data needed for the project.

5 GPU Porting Status

Baseline code versions were extracted from current production version of GTC (Fortran) as a starting point of the code porting and optimization work.

First, all the particle routines (push, charge and shift) for thermal ions, fast ions and fast electrons and kinetic thermal electrons have been replaced by the set of unified routines, which can operate on every species controlled by the calling parameters like

push(species_name, and other parameters)

charge(species_name)

shift(species_name)

where species_name is the description keyword which can be any of "thermal-ion", "thermal-electron", "fast-ion" or "fast-electron". Such species, including both the thermal and fast particles, are described by diverse physical models such as fully-kinetic, gyrokinetic, drift-kinetic, or fluid-kinetic hybrid. This makes it possible that all species benefit from optimizations, for example OpenACC optimizations for the GPU, through changing only one subroutine.

	SummitDev GPU w/ AmgX	SummitDev GPU w/ PETSc	Titan GPU w/ PETSc	SummitDev CPU w/ PETSc	Titan CPU w/ PETSc (Ideal OMP)
pushi	0.66	0.65	2.37	23.97	17.3
shifti	0.26	0.26	0.61	21.07	7.8
chargei	0.66	0.65	1.03	9.59	2.0
electron	8.40	8.37	22.40	370.23	266.0
fast	1.53	1.54	4.74	55.47	28.7
Poisson	2.64	14.67	10.19	9.54	8.1
pushfield	0.27	0.27	0.53	0.26	1.0
total	14.42	26.41	41.87	490.13	330.9

Table 1. Phase II GPU and CPU timings (in seconds) from the preliminary SummitDev benchmarks. For comparison, the same physics case, for both GPU and CPU, is shown for Titan. All runs use 32 MPI ranks; GPU runs have 1 GPU / MPI rank; the SummitDev CPU run has 5 OMP / MPI rank; and the Titan CPU case has 8 OMP / MPI rank, however in the data shown here, we assume an ideal scaling in OMP from 8 to 16 threads, and so the data here is the real time divided by 2. This latter point is done to yield a lower bound in the possible speed up. Also, so as to keep the GPU to MPI rank ratio unity, there are 2 ranks / CPU and 1 rank / CPU on SummitDev and Titan, respectively; hence, SummitDev CPUs have a larger compute load.

Second, these unified routines have been successfully ported to the GPU using OpenACC directives supported by PGI compiler. GTC's main data structure is allocatable arrays within modules. The "acc declare" directive was used in the module file to specify all the arrays that the GPU needs to access. Then, the CPU code for allocating the array typically will not require any change since the OpenACC runtime will automatically allocate a GPU copy if an array is specified in "acc declare". Whenever data needs to be copied between the CPU and the GPU, the "acc update" directive was used. Finally, the "acc host_data" directive was used to interoperate with the CUDA kernels.

The unified push routine was ported to the GPU using OpenACC. Most of the push time is spent in two loops. The first loop performs a gather operation from grid points to particles. By porting this loop to CUDA, it was identified that using texture cache for the grid arrays will lead to ~3X speedup compared to the base OpenACC version. So, enabling texture cache in OpenACC will be an important next step for optimizing this loop. The second loop updates the particle locations. It was identified that the memory access of the private array "dx" was the main bottleneck. This "dx" array stores the coefficients used to interpolate the local field quantities from the Euclidian meshes. The optimization was to move the array bound variable for dx to a module file as a parameter and rewrite some of the loops involving dx using the array bound parameter. Those changes enabled the compiler to put this array in local memory, which led to ~4X speedup compared to the base OpenACC version. So, this made a case for adding texture cache support to OpenACC. Experimental support of texture cache is now being added to PGI's OpenACC compiler, and we will test it when available.

The unified charge routine was ported to the GPU using OpenACC. Because different particles may write to the same grid points, the OpenACC atomic directive was used to handle write collisions. This strategy looked to be working well.

The shift routine was ported to CUDA before the US DOE Center for Accelerated Application Readiness (CAAR) program. Since shift routine is not modified by developer often at all, the GTC team thinks it's fine to use the CUDA version for this routine. So, the CUDA port in previous version was used for shift routine.

A binning subroutine, based on the out-of-place counting sort algorithm, was implemented in GTC (radial_bin.F90). The first version of the binning algorithm bins all particle species in the radial dimension

periodically to improve data locality for charge deposition and field interpolation. For linear problems, where spatial change is small in the radial dimension from one time step to the next, up to 10% overall speedup is observed. It is expected that binning will speed up the performance significantly for nonlinear problems. Later, a cell-based binning was developed and improved the performance by 70% for electron subroutines. Overall, over 10% performance improvement is observed by enabling the cell-based binning.

Both Array of Structure (AoS) and Structure of Array (SoA) data layouts for particles have been implemented on a simplified version of GTC. For GPU, performance analysis is conducted using CUDA profiling toolkit nvprof on a single Titan node. Higher bandwidth and transactions are observed for the AoS layout. Overall no significant speedup is obtained with the SoA data structure for all versions including CPU, GPU (OpenACC) and GPU (CUDA) of the code. We thus decide to use AoS layout for all particle species (as before). The SoA alternative will be available in the future for architectures for which this data layout might improve performance.

	SummitDev GPU w/ AmgX, Speed up	SummitDev GPU w/ PETSc, Speed up	Titan GPU w/ PETSc, Speed up
Pushi	36.2	36.6	7.3
Shifti	82.5	80.5	12.7
chargei	14.6	14.7	1.9
pushe	27.4	27.6	14.0
Shifte	76.1	75.9	9.6
chargee	10.2	10.2	2.7
Fast	36.2	36.0	6.0
Poisson	3.6	0.7	0.8
Pushfield	1.0	1.0	1.8
total	34.0	18.6	7.9

Table 2. Phase II GPU speedups, for 15 time steps. SummitDev speedups are relative to the SummitDev CPU w/ PETSc & 5 OMP thread / rank case. The Titan GPU speedup is relative to the Titan CPU w/ PETSc & Ideal OMP case (see Table 1 caption). All GPU runs use 32 MPI, with 1 GPU / rank.

Due to increasing relative costs of the Poisson field solve, the PETSc standard solver has been replaced with several alternatives. The Hypre algebraic multigrid solver, whether used standalone or as part of PETSc, runs up to 11X faster than the PETSc standard solver on SummitDev. An early GPU-enabled version of Hypre gave up to 15X improvement over PETSc, and furthermore the NVidia AmgX solver executed up to 27X faster than PETSc. The new solvers also scale much better than PETSc, an increasingly important property as larger and more complex problems are attempted.

GTC uses explicit OpenACC directives to manage GPU data. Unified memory has been introduced since CUDA 6.0 for reducing the complexity of GPU programming and improving performance through data locality. Though typical unified memory implementation has lower performance than explicit memory management, it is interesting to port GTC to unified memory to evaluate the tradeoff between productivity and performance. The initial experiments have suggested that using unified memory in GTC incurred a significant performance penalty due to page fault of Fortran automatic arrays. It is expected that the performance of the unified memory will be improved as PGI provides optimized pool allocator.

6 Performance

A set of test problems was developed for evaluating performance (see scaling studies below). The physics case[21] in the 2013 Physical Review Letters by Z. Wang et al was prepared as a base case to measure improvements in performance. This choice is appropriate since it is a good representation of future production runs and GTC's capabilities, since it employs all particle species, electromagnetic capabilities, experimental profiles and realistic tokamak equilibrium.

6.1 Solver performance improvement

The GTC Poisson solver currently runs on the CPU. Though it is presently not the most time-consuming part of GTC simulations, the solver time requirements have become more significant since other parts of the code have been accelerated using GPUs. We have replaced the standard PETSc solver with a Hypre multigrid solver. This solver is threaded to effectively use the CPUs and is also scalable to many compute nodes. Figure 1 shows comparative timings of the PETSc solver and the Hypre multigrid solver for a representative set of GTC test cases. The Hypre solver for these cases is ~4X faster than the standard PETSc solver and has better scaling properties.

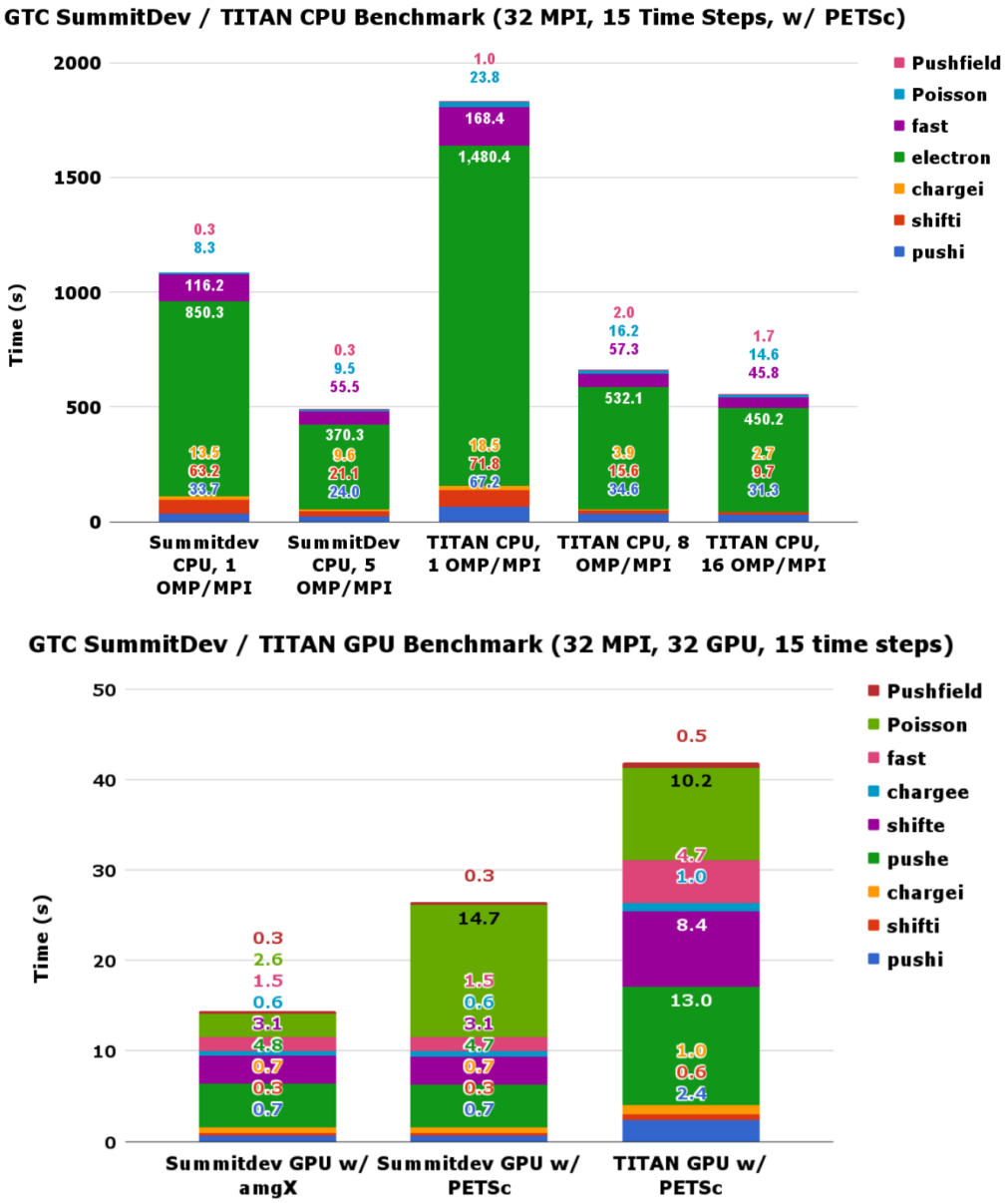

Figure 4. The Phase II timing breakdown for GTC performance study on SummitDev and Titan for 15 time steps and 32 MPI processes. Note that in order to keep the GPU to MPI ratio unity, there are 2 ranks / CPU and 1 rank / CPU on SummitDev and Titan, respectively. Hence, SummitDev CPUs have a larger load. (Top) Pure CPU tests with a scan of OMP thread / rank. (Bottom) GPU tests. All GPU runs use 1 GPU / rank.

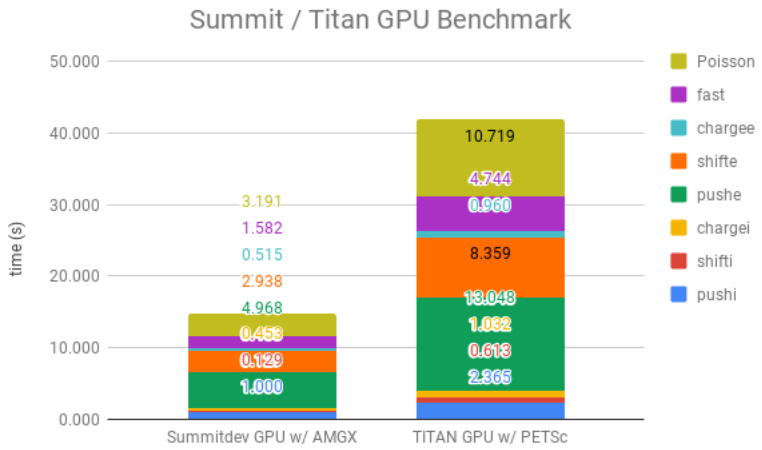

Figure 5. The Phase II timing breakdown for GTC performance study on Summit and Titan for 15 time steps and 32 MPI processes.

6.2 Scaling performance

Two sets of weak scaling studies were carried out on Titan up to nearly the full system (16,384 nodes; at the time of this study, many Titan nodes were unavailable, making it impossible to run on all 18,688 nodes). The first test set is called "particle weak scaling study", where we fix the grid size, but scale the total number of particles. The second set of tests is called "hybrid weak scaling study", where we scale both the grid size and total number of particles. The first study holds the number of particles per MPI rank and the number of grid cells per MPI rank nearly constant, thus reflecting a conventional weak scaling study; the second study is a more realistic scaling study based on typical production run of the code: grid size is proportional to the square root of number of nodes). For both sets of weak scaling study, the number of particles per processor is fixed at 3.2 million. Compared with CPU (16 cores AMD 6274), GPU (NVidia K20x) has boosted the overall performance by 1.6-3.0X. The decrease of the performance speedup in large processor counts is mainly due to the increased portion of the non-GPU accelerated subroutines as well as MPI time. These tests were conducted in May 2016.

6.3 Preliminary Tests on SummitDev

To foreshadow the performance of GTC on the next generation supercomputer, Summit, a set of dedicated benchmarks have been executed on SummitDev and Titan, again employing the physics case used in section 6.1. The scaling study of section 6.2 was executed in May 2016, and since then much effort has been placed into GTC's GPU optimizations, such as removing unnecessary CPU to GPU data transfer and an increase in compiler support for texture, for use on Titan, and these additions have subsequently been ported for use on SummitDev. Hence, the speedups presented here are larger than those shown above.

Since SummitDev is a small prototype machine, 32 MPI processes were used per test. Tests covered both pure CPU runs and GPU dominant heterogeneous runs. Table 1 tabulates the results of 5 tests: 2 CPU runs, one on Titan and one on SummitDev, and 3 GPU runs, one on Titan and two on SummitDev. The CPU runs aimed to utilize both MPI and OpenMP parallelization with realistic job configurations, employing 8 OMP threads[1] / MPI rank on Titan and 5 OMP threads / MPI rank on SummitDev. This configuration leads to there being 2 ranks / CPU and 1 rank / CPU on SummitDev and Titan, respectively; hence, the SummitDev CPUs have a larger load in these runs, which explains why the SummitDev CPU timings are not as dramatically better than the Titan CPU timings. The GPU runs used 1 GPU / MPI rank and no OpenMP. Again, two GPU runs were carried out on SummitDev, each with a different library to solve the Poisson equation. One used PETSc, and the other used AmgX, the latter of which can take advantage of GPU acceleration. The Titan GPU run also uses PETSc to solve the Poisson equation. With AmgX, the total number of particles pushed per second on the SummitDev GPU run is 1.29×10^9.

The tabulated data is also presented in Figure 4. The upper panel shows CPU only tests on both SummitDev and Titan for a range of OMP threads / MPI rank. The scaling from 8 to 16 OMP threads / MPI rank in Titan was poor. This is in part due to there being a decrease in efficiency when using OMP threads across cores on Titan--hence we assumed an ideal scaling from 8 to 16 OMP threads / MPI rank in Table 1 to obtain a lower bound in the possible speedup attainable. The lower panel presents the GPU timing data.

Table 2 shows the GPU speedups obtained. SummitDev GPU speedups are relative to the SummitDev CPU case with 5 OMP threads / MPI rank, and Titan GPU speedups are relative to the Titan CPU case with ideal OMP scaling from 8 to 16 threads / MPI rank. The overall speedups were 34.0 and 18.6 on SummitDev, for the AmgX and PETSc libraries, respectively, and 7.9 on Titan. The most notable speedups came from the particle push and shift routines on SummitDev, with a roughly 36 and 82 times speed up for the ion push and shift, respectively; and a roughly 27 and 76 times speed up for the electron push and shift, respectively. The high speedup factors are to large degree enabled by the very effective use of texture cache as described earlier, as well as need to further optimize the OpenMP threading version for CPU. Moreover, the utilization of the AmgX library decreases the Poisson time by 5.5 times. It is noteworthy that the SummitDev/GPU/AmgX to Titan/PETSc performance ratio is about 3X, roughly in line with the 4X flop rate ratio and 3X memory bandwidth ratio of SummitDev vs. Titan GPUs.

6.4 Performance and Scalability on Summit

For testing the performance and scalability on Summit and the followed early science applications thereafter, a set of test problems was developed for evaluating performance. The physics simulation reported in the paper45 was prepared as a base case to measure improvements in performance. This choice is appropriate since it is a good representation of Summit early science simulations with all particle species, electromagnetic capabilities, experimental profiles and realistic tokamak equilibrium. As shown in Table 3 and Figure 6. Wall-clock time for one trillion particle pushes in the GTC weak scaling test on Summit., GTC CPU-only runs scale almost perfectly up to 928 nodes (about 20% of the whole Summit) in the weak scaling test (i.e., by keeping constant number of particles per node). The simulation on 928 nodes uses 2x106 grids and $2x10^{11}$ particles utilizing 2/3 of the GPU memory. GTC simulations using all GPUs and CPUs also show good scaling, with a ~50% efficiency at 928 Summit nodes when compared with the ideal scaling. The GTC speed up from CPU-only to GPU+CPU is over 20 at 928 Summit nodes, leading to an unprecedented speed of one trillion particle pushes in 2 seconds wall-clock time. Furthermore, GTC performance on each Summit GPU is about 8 times faster than each Titan GPU. Finally, as part of the Summit acceptance benchmark simulations, preliminary results of GTC running on 4576 Summit nodes (by Dr. Wayne Joubert of OLCF) show good scaling and similar performance, as shown in Figure 7. The impressive GTC performance on Summit would enable integrated simulation of multiple physical processes as proposed in this project.

Summit nodes	16	64	256	512	928
GPU+CPU	58.43	15.37	4.99	2.92	2.00
CPU only	2167.56	525.98	150.45	71.53	41.76

Table 3 Wall-clock time for one trillion particle pushes in the GTC weak scaling test on Summit.

1 The timings for the TITAN CPU w/ PETSc case in Table 1 assume an ideal scaling in OMP threads from 8 threads to 16. i.e. the times presented in Table 1 for this case are those of the 8 OMP threads case, but they are divided by 2. The motivation for this is to set a lower bound in the possible GPU speedup attainable in TITAN.

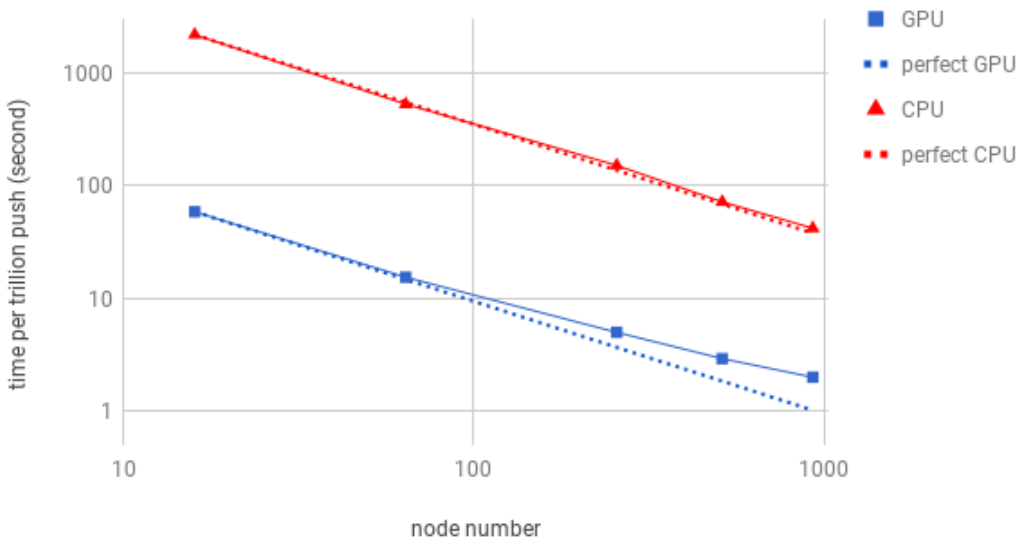

Figure 6. Wall-clock time for one trillion particle pushes in the GTC weak scaling test on Summit.

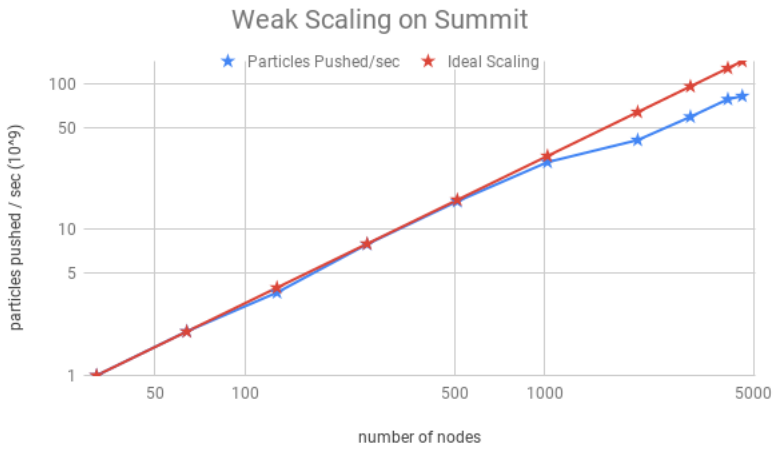

Figure 7. Phase-II weak scaling of GTC on Summit, with the number of nodes ranging from 32 nodes to 4576 nodes (almost the whole machine). Total particle number are increased by increasing the number of particles per node.

7 Conclusion

We have successfully restructured the current production version of gyrokinetic toroidal code (GTC) to a more modularized format with unified routines for all particle species, including thermal ions, thermal electrons, fast ions and fast electrons. This followed by the optimizations using OpenACC directives to enable the GPU accelerations to the code, which is also relatively friendly for fusion physics researchers and students. Other techniques have also been introduced to boost the performance to a higher level, which includes the binning technique where particle data storage is optimized for access. Hypre and Amgx have been adopted as alternatives to the PETSc field solver, which make the code benefit from the accelerations of many core CPUs (Hypre) and GPUs (AmgX).

The GPU version of code was benchmarked on up to 8000 nodes of the Oak Ridge Titan supercomputing machines, which is equipped with NVidia M2050 GPUs and Intel Xeon X5670 CPUs. Realist profiles and parameters from

fusion experiments have been used in these benchmarks to provide insights into technical interests and scientific significance. Technically, the strong and weak scaling studies have been performed and the overall speedup is about 2-3 times; and preliminary tests on SummitDev show an overall speedup of about 34 times. GTC showed very good scalability on the whole machine tests on Titan.

The real physics tests on Summit machine have also been conducted to tackle the self-consistent energetic particle physics in fusion plasmas, especially for ITER. These tests showed impressive scaling properties that reaches roughly 50% efficiency on 928 nodes which is 20% of total nodes of Summit. The GPU+CPU speed up from purely CPU is over 20 times, leading to an unparalleled speed.

ACKNOWLEDGMENTS

The authors would like to thank Eduardo D'Azevedo for his many useful suggestions in the optimizations. This work was supported by the US Department of Energy (DOE) CAAR project, DOE SciDAC GSEP center, and China National Magnetic Confinement Fusion Science Program under Grant Nos. 2013GB111000, the National Natural Science Foundation of China under Grant Nos. 11675257, and the External Cooperation Program of Chinese Academy of Sciences under Grant No. 112111KYSB20160039. This research used resources of the Oak Ridge Leadership Computing Facility (OLCF) at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

1. W. W. Lee, The Physics of Fluids 26, 556 (1983).
2. W. Lee, Journal of Computational Physics 72, 243 (1987), ISSN 0021-9991.
3. R. G. Littlejohn, Journal of Plasma Physics 29, 111 (1983).
4. A. Brizard and T. Hahm, Reviews of modern physics 79, 421 (2007).
5. T. Hahm, Physics of Fluids (1958-1988) 31, 2670 (1988).
6. E. Frieman and L. Chen, Physics of Fluids (1958-1988) 25, 502 (1982).
7. A. Rogister and D. Li, Physics of Fluids B: Plasma Physics (1989-1993) 4, 804 (1992).
8. Z. Lin and L. Chen, Physics of Plasmas (1994-present) 8, 1447 (2001).
9. Y. Lin, X. Wang, Z. Lin, and L. Chen, Plasma physics and controlled fusion 47, 657 (2005).
10. I. Holod, W. L. Zhang, Y. Xiao, and Z. Lin, Physics of Plasmas 16, 122307 (2009).
11. P. Liu, Z. Wenlu, D. Chao, L. Jingbo, L. Zhihong, and C. Jintao, Submitted to Nuclear Fusion (2017).
12. Lin, Z., Hahm, T.S., Lee, W.W., Tang, W.M., White, R.B.: Turbulent Transport Reduction by Zonal Flows: Massively Parallel Simulations. Science 281, 1835 (1998).
13. http://phoenix.ps.uci.edu/GTC
14. http://www.iter.org
15. Z. Lin, I. Holod, L. Chen, P. H. Diamond, T. S. Hahm, and S. Ethier, Phys. Rev. Lett. 99, 265003 (2007).
16. Y. Xiao and Z. Lin, Phys. Rev. Lett. 103, 085004 (2009).
17. W. Zhang, Z. Lin, and L. Chen, Phys. Rev. Lett. 101, 095001 (2008).
18. W. Zhang, V. Decyk, I. Holod, Y. Xiao, Z. Lin, and L. Chen, Physics of Plasmas 17, 055902 (2010).
19. W. Zhang, I. Holod, Z. Lin, and Y. Xiao, Physics of Plasmas 19, 022507 (2012).
20. C. Zhang, W. Zhang, Z. Lin, and D. Li, Physics of Plasmas 20, 052501 (2013).
21. Z. Wang, Z. Lin, I. Holod, W. W. Heidbrink, B. Tobias, M. Van Zeeland, and M. E. Austin, Phys. Rev. Lett. 111, 145003 (2013).
22. J. Cheng, W. Zhang, Z. Lin, I. Holod, D. Li, Y. Chen, and J. Cao, Physics of Plasmas 23, 052504 (2016).
23. A. Kuley, Z. Lin, J. Bao, X. S. Wei, Y. Xiao, W. Zhang, G. Y. Sun, and N. J. Fisch, Physics of Plasmas 22, 102515 (2015).
24. J. Peng, L. Zhihong, I. Holod, and X. Chijie, Plasma Science and Technology 18, 126 (2016).
25. J. McClenaghan, Z. Lin, I. Holod, W. Deng, and Z. Wang, Physics of Plasmas 21, 122519 (2014).
26. D. Liu, W. Zhang, J. McClenaghan, J. Wang, and Z. Lin, Physics of Plasmas 21, 122520 (2014).

27. Z. Lin, T. S. Hahm, S. Ethier, and W. M. Tang, “Size scaling of turbulent transport in magnetically confined plasmas,” Phys. Rev. Lett., vol. 88, pp. 195004, 2002.
28. Y. Xiao and Z. Lin, “Turbulent transport of trapped electron modes in collisionless plasmas,” Phys. Rev. Lett., vol. 103, pp. 085004, 2009.
29. Xiangfei Meng, Peng Wang, Yang Zhao, Xin Liu, Bao Zhang, Xiaoqian Zhu, Yong Xiao, Wenlu Zhang, Zhihong Lin, Heterogeneous Programming and Optimization of Gyrokinetic Toroidal Code and Large-scale Performance Test on TH-1A, International Supercomputing Conference, Leipzig, Germany, 2013.
30. Endong Wang, Shaohua Wu, Qing Zhang, Jun Liu, Wenlu Zhang, Zhihong Lin, Yutong Lu, Yunfei Du, Xiaoqian Zhu, The Gyrokinetic Particle Simulation of Fusion Plasmas on Tianhe-2 Supercomputer, Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems (ScalA) 2016, International Conference for High Performance Computing, Networking, Storage and Analysis (SC2016), Salt Lake City, USA.
31. Madduri, K., Ibrahim, K.Z., Williams, S., Im, E.J., Ethier, S., Shalf, J., Oliker, L.: Gyrokinetic toroidal simulations on leading multi- and manycore HPC systems. In: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (2011)
32. Madduri, K., Im, E.J., Ibrahim, K.Z., Williams, S., Ethier, S., Oliker, L.: Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms. Parallel Computing 37(9), 501–520 (2011)
33. B. Wang, S. Ethier, W. Tang, T. Williams, K. Zibrahim, K. Madduri, S. Williams, L. Oliker, “Kinetic turbulence simulations at extreme scale on leadership-class systems,” SC’13 Proc. Int’l Conf. High Performance Computing, Networking, Storage and Analysis, 2013, No. 82.
34. S. Ethier, M. Adams, J. Carter, L. Oliker, “Petascale parallelization of the gyrokinetic toroidal Code,” LBNL Paper LBNL-4698, 2012.
35. K. Madduri, K. Z. Ibrahim, S. Williams, Eun-Jin Im, S. Ethier, J. Shalf, L. Oliker, ”Gyrokinetic Toroidal Simulations on Leading Multi- and Manycore HPC systems,” SC’11, Proc. Int’l Conf. High Performance Computing, Networking, Storage and Analysis, 2011.
36. W. Tang, B. Wang, S. Ethier, “Scientific Discovery in Fusion Plasma Turbulence Simulations at Extreme Scale,” Comput. Sci. Eng., vol.16, pp. 44, 2014.
37. J. M. Dawson, Reviews of modern physics 55, 403 (1983).
38. C. K. Birdsall and A. B. Langdon, Plasma physics via computer simulation (CRC Press, 2004).
39. Y. Xiao, I. Holod, Z. Wang, Z. Lin, and T. Zhang, Physics of Plasmas 22, 022516 (2015).
40. Hongying Feng, Wenlu Zhang, Zhihong Lin, Xiaohe Zhufu, Jin Xu, Jintao Cao, and Ding Li, Development of Finite Element Field Solver in Gyrokinetic Toroidal Code, Communications in Computational Physics (submitted).
41. S. Ethier, Z. Lin, “Porting the 3D gyrokinetic particle-in-cell code GTC to the NEC SX-6 vector architecture: perspectives and challenges,” Computer Physics Communications, vol. 164, pp. 456-458, 2004.
42. R. B. White and M. S. Chance, The Physics of Fluids 27, 2455 (1984).
43. Joubert, Wayne, et al. "Accelerated application development: The ORNL Titan experience." Computers & Electrical Engineering 46 (2015): 123-138.
44. Veronica Vergara Larrea, Wayne Joubert, Mark Berrill, Swen Boehm, Arnold Tharrington, Wael Elwasif, Don Maxwell, "Experiences evaluating functionality and performance of IBM Power8+ systems," Proceedings of the International Workshop on OpenPOWER for HPC '17, Frankfurt, Germany, 22 June 2017.
45. Zhixuan Wang, Zhihong Lin, Ihor Holod, W. W. Heidbrink, Benjamin Tobias, Michael Van Zeeland, and M. E. Austin, “Radial Localization of Toroidicity-Induced Alfven Eigenmodes”, Phys. Rev. Lett. 111 (2013) 145003.