Extreme Scale Plasma Turbulence Simulations on Top Supercomputers Worldwide

William Tang[1,2], Bei Wang[1], Stephane Ethier[2],
Grzegorz Kwasniewski[3], Torsten Hoefler[3], Khaled Z. Ibrahim[4],
Kamesh Madduri[5], Samuel Williams[4], Leonid Oliker[4],
Carlos Rosales-Fernandez[6], Tim Williams[7]
[1]*Princeton University, Princeton Institute for Computational Science and Engineering, Princeton, NJ, USA*
[2]*Princeton Plasma Physics Laboratory, Princeton, NJ, USA*
[3]*ETH, Computer Science Department, Zürich, Switzerland*
[4]*Lawrence Berkeley National Laboratory, Computational Research Division, Berkeley, CA, USA*
[5]*The Pennsylvania State University, Computer Science and Engineering Department, University Park, PA, USA*
[6]*The University of Texas, Texas Advanced Computing Center, Austin, TX, USA*
[7]*Argonne National Laboratory, Argonne Leadership Computing Facility, Argonne, IL, USA*

***Abstract*—The goal of the extreme scale plasma turbulence studies described in this paper is to expedite the delivery of reliable predictions on confinement physics in large magnetic fusion systems by using world-class supercomputers to carry out simulations with unprecedented resolution and temporal duration. This has involved architecture-dependent optimizations of performance scaling and addressing code portability and energy issues, with the metrics for multi-platform comparisons being "time-to-solution" and "energy-to-solution". Realistic results addressing how confinement losses caused by plasma turbulence scale from present-day devices to the much larger $25 billion international ITER fusion facility have been enabled by innovative advances in the GTC-P code including (i) implementation of one-sided communication from MPI 3.0 standard; (ii) creative optimization techniques on Xeon Phi processors; and (iii) development of a novel performance model for the key kernels of the PIC code. Results show that modeling data movement is sufficient to predict performance on modern supercomputer platforms.**

I. Fusion Plasma Turbulence Simulations: Grand Scientific Challenge and Relevance to Society

As the global energy economy makes the transition from fossil fuels toward cleaner alternatives, fusion becomes an attractive potential solution for satisfying the growing needs. Fusion energy, which is the power source for the sun, can be generated on earth, for example, in magnetically-confined laboratory plasma experiments (called tokamaks) when the isotopes of hydrogen (e.g., deuterium and tritium) combine to produce an energetic helium alpha particle and a fast neutron with an overall energy multiplication factor of 450:1. Building the scientific foundations needed to develop fusion power demands high-physics-fidelity predictive simulation capability for magnetically-confined fusion energy (MFE) plasmas. To do so in a timely way requires utilizing the power of modern supercomputers to simulate the complex dynamics governing

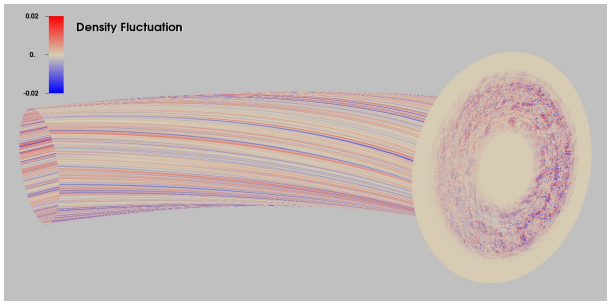

Fig. 1. Density fluctuations during the non-linear phase of an ITER-size GTC-P simulation of plasma microturbulence.

MFE systems, including ITER, a multi-billion dollar international burning plasma experiment supported by 7 governments representing over half of the world's population. Given the increasingly dangerous consequences for air pollution caused, for example by dependence on energy produced from fossil fuels, the temporal urgency for accelerated progress in this critical grand challenge mission is very clear [1].

Unavoidable spatial variations in fusion systems produce micro-turbulence, fluctuating electric and magnetic fields, which can grow to levels that significantly increase the transport rate of heat, particles, and momentum across the confining magnetic field in tokamak devices. Since the balance between these energy losses and the self-heating rates of the actual fusion reactions will ultimately determine the size and cost of an actual fusion reactor, understanding the underlying physical processes is key to achieving the efficiency needed to help ensure the practicality of future fusion reactors. The associated motivation drives the pursuit of realistic high-resolution calculations of turbulent transport at scale, an MFE scientific grand challenge that can be rapidly advanced only by making effective use of top supercomputing systems. For example, the long time behavior of turbulent transport in

SC16; Salt Lake City, Utah, USA; November 2016
978-1-4673-8815-3/16/$31.00

ITER-scale plasmas can now be studied using simulations with unprecedented phase-space resolution to address the reliability/realism of the influence of increasing plasma size on confinement in tokamaks (see Figure 1 for visualization of density fluctuations at high resolution for an ITER-size fusion system). ITER is in fact the largest MFE experiment to be constructed by a huge margin. Since it is a factor of 8 times bigger in volume than the largest tokamak ever built, we have no experimental knowledge of fusion at that level and accordingly require reliable simulations.

The major challenge addressed in this paper is the development of modern software capable of using the most powerful supercomputers to carry out reliable first-principles-based simulations of multi-scale tokamak plasmas. In particular, a key fusion physics task is to significantly improve the decade-long MFE estimates of confinement scaling with device size (the so-called "Bohm to Gyro-Bohm" rollover trend caused by the ion temperature gradient instability). This demands much higher resolution to be reliable. The problem becomes significantly more demanding when the complex electron dynamics associated with the trapped-electron instability is also taken into account. This is a proto-typical increasing problem size issue which is discussed in [2] with more details of the advances achieved provided in §VI.

II. Computational Approach

A. Current State of The Art for GTC-P code

The scientific application software developed and exercised in this paper is the GTC-Princeton (GTC-P) code, a highly scalable Particle-in-Cell (PIC) code used for studying micro-turbulent transport in tokamaks. Micro-turbulent transport in tokamaks is mathematically described using the nonlinear gyro-phase-averaged Vlasov-Poisson equations (or "gyrokinetic equations") and the PIC methodology can be applied to solve these high-dimensional equations. We use discrete, charged rings to represent the fast cyclotron motion of the ions in a strong magnetic field averaged over several periods. The rings or particles interact with each other through a self-consistent field evaluated on a grid that covers the entire 3D simulation domain. The charge of each ring is deposited on the grid by selecting four points on the ring and interpolating each to its nearest grid points. The resulting charge densities are then used in the evaluation of the field by solving the Poisson equation [3]. The field is then evaluated at the position of each of the four points, again by interpolation, and used in the equations of motion to advance the particles. Achieving high parallel and architectural efficiency is very challenging for the standard PIC method due to potential fine-grained data hazards, irregular data access, and low arithmetic intensity. In addition, gyrokinetic PIC codes are confronted by major pressure on the memory sub-system caused by poor spatial and temporal locality in the requisite gather and scatter operations. This is due to the fact each ion ring needs to access up to 32 individual grid points. Such issues can become more formidable as the HPC community moves into the future to address even more radical changes in computer architectures as the multi-core and many-core revolution progresses.

Prior research involving large-scale PIC simulations, both standard PIC and gyrokinetic PIC, on leading supercomputer systems includes, for example, VPIC on Roadrunner [4], OSIRIS on Sequoia [5], PIConGPU on Titan [6], GTC-P on Mira and Sequoia [7], GTC on Tianhe-1A [8] and XGC1 on Titan [9]. The research carried out in this paper using the latest version of GTC-P—the fastest gyrokinetic PIC code worldwide with respect to time-to-solution of large problem size physics challenges—is the first to explore the portability, optimizations, and scalability across a wide range of multi-petaflop platforms at the full or near-full capability, including both homogeneous and heterogeneous systems involving GPUs and Intel Xeon Phis. In addition, motivated by the most relevant modern HPC issues, we focus on time-to-solution (seconds) and energy-to-solution (KWh) as the performance metrics of interest that are guided by a novel performance model that is described in detail in §III-A. Note that metrics such as flop/s or percentage-of-peak are less relevant for the predominantly memory-bound gyrokinetic PIC methods, as modern architectures require 10 flops per byte moved from DRAM in order to be compute-limited.

Prior to the work presented in the current paper, the GTC-P code had included all of the important physics associated with ion temperature gradient driven turbulence captured in numerous global PIC simulation studies of plasma size scaling over the years that had concentrated on comprehensive inclusion of ion dynamics and the electrons treated using a simple adiabatic model [2, 10, 11]. In this model, the biggest impediments to performance are the data hazards and the lack of data locality in charge deposition (**charge**) and field interpolation (**push**). As such, the associated optimizations were focused on reducing data hazards and improving locality for multicore and manycore architectures [12–14]. The GTC-P code has now been significantly upgraded with a more complete electron drift-kinetic dynamics capability that has been verified with systematic benchmarks against the comprehensive electromagnetic GTC code [15, 16] at UC-Irvine. The introduction of kinetic electron dynamics has shifted the emphasis of time consuming kernels to field interpolation (**push**) and particle communication (**shift**) because of electron subcycling where each ion step includes the calling of these two kernels 60 times. Besides the data locality challenge, the performance of the code is also largely influenced by the network performance and the specific implementation of the communication. In this paper, we have developed a performance model to evaluate the most time consuming subroutines, **push** and **shift** (along with particle **sort** to improve data locality). To leverage the capabilities of hardware enabled Remote Direct Memory Access (RDMA), we have implemented the **shift** with MPI-3 One-sided communication (see §III-D). It is expected that "lessons learned" achieved from such research that focuses on operations fundamental to all gyrokinetic PIC codes will prove most valuable to ongoing and future efforts in advancing the more comprehensive and complex fusion micro-turbulence

codes such as GTC to the multi-petaflop range on the path to exascale and beyond.

B. Parallelization

In an earlier version of GTC-P, the 3D tokamak mesh was partitioned uniformly along the toroidal dimension, leading to a 1D domain decomposition. Within each of the toroidal domains, particle decomposition was introduced to distribute particle-related work among multiple processors. This approach was shown to lead to nearly-perfect scaling with respect to the number of particles [17]. However, simple 1D domain decomposition causes replication of the toroidal domain across particle domains, and the increased memory footprint due to this replication hinders simulations of large fusion devices, or small-size devices with a higher grid resolution. To address this problem, an additional decomposition in the radial dimension [18, 19] was introduced. As a result, particle- and grid-based routines are distributed across all processes. We further exploit multicore parallelism using shared-memory multithreading which provides an additional multiplicative source of speedup [7].

The multilevel particle and domain decompositions also provide significant flexibility in distributed-memory task creation and layout. While the ranks in the toroidal dimension are usually fixed as 32 or 64 due to Landau damping physics, we can choose any combination of process partitioning along the radial and particle dimensions. For scaling with a fixed problem size, we first partition along the radial direction and then switch to particle decomposition for additional scalability. The decompositions were implemented with three individual communicators in MPI (toroidal, radial, and particle communicator), and we further provide tuning options to change the order of MPI rank placement. A gyrokinetic simulation typically has highly anisotropic behavior, with the velocity parallel to the magnetic field being an order of magnitude larger than that in the perpendicular direction. Consequently, the message sizes in the toroidal dimension can be $10\times$ larger than those in the radial dimension at each time step. On Blue Gene systems with explicit process mapping, we group processes to favor the MPI communicator in the toroidal dimension [7]. For other systems, assigning consecutive ranks for processes within each toroidal communicator generally leads to improved performance.

III. Performance Optimization on Modern Computing Platforms

We now discuss highlights of the various strategies employed to optimize performance, maximize parallelism, leverage accelerator technology, and enable portability across one of the broadest sets of evaluated supercomputers in recent literature. Like all PIC methods, GTC-P includes two types of subroutines: grid-based and particle-based. The grid-based subroutines, **poisson, field** and **smooth**, solve the gyrokinetic Poisson equation, compute the electric field, and smooth the charge and potential, respectively. The number of floating-point operations and memory used for these routines roughly scale linearly with the grid size (number of grid points). The particle-based subroutines are **charge**, **push**, **sort**, and **shift**, with **charge** and **push** performing particle-grid interpolation, **sort** used to improve data locality and performance of the particle-grid interpolation, and **shift** moving particles between processes. It is important to highlight that the particle-grid interpolation for electrons follow the approach in standard particle-in-cell methods, where each particle is interpolated only to the the nearest neighboring points. Unlike the gyrokinetic particle-in-cell method for ions, where each particle represents four points on a ring and thus accesses up to 32 grid points, each electron particle only interacts with up to 8 individual grid points in 3D (see Figure 2 for a 2D representation).

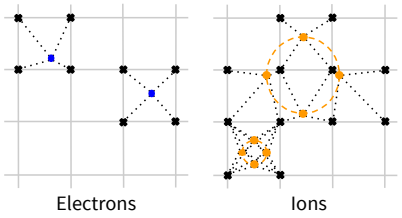

Fig. 2. Particle-Grid Interpolation: An electron is bounded in a cell and interacts with exactly 8 grid points in three dimensions (4 points in two dimensions, as shown in the figure), whereas an ion, modeled as a charged ring and approximated by four points on the ring, may interact with 8-32 grid points (4-16 points in 2D).

All the particle-processing routines have costs linearly proportional to the number of particles. The ratio of the number of particles to the number of grid points is 100 or higher. As such, particle-based routines tend to dominate per-timestep running time. For kinetic electron simulations, since each ion time step includes 10 electron sub-cycling steps, where electron **push** and **shift** are called 60 times, these two subroutines have dominated the total computational time.

A. GTC-P performance model

To prove the scalability of the GTC-P application and guide the optimizations, we have constructed a rigorous performance model. The increasing gap between computation and communication capabilities in current architectures often leads to compute units starving for data. However, analyzing data movement through caches is far more challenging than comparing peak and achieved flop/s performance. Nevertheless, we argue that modeling the former is crucial for performance analysis. In the following, we demonstrate a detailed analysis of the GTC-P code and compute architectures which lead to a precise model for its performance.

Most of GTC-P's execution time is spent detecting, moving and reordering particles. Moreover, particle-grid interactions and conditional statements generate many unavoidable random accesses to the memory, which restrain the effective use of prefetchers and transfers of large blocks. Therefore, to correctly model the performance of the application, we need to model various properties of data movements: size, access

TABLE I
MACHINE PARAMETERS USED IN THE MODEL. THE TOPOLOGIES EXPAND TO FAT TREE (FT), DRAGONFLY (DF) AND 3D/5D TORUS (3DT, 5DT).

machine parameters				
Machine	**Daint**	**Stampede**	**Titan**	**Mira**
CPU	E5-2670	2x E5-2680	AMD 6274	PowerPC A2
accelerator	K20X	KNC	K20X	-
Topology	DF	FT	3DT	5DT
Net BW [GB/s]	14	18	7	12

architecture parameters					
Architecture	**E5-26***	**AMD 6274**	**PowerPCA2**	**KNC**	**K20X**
LLC [MB]	20	16	32	32	-
Vec width[B]	32	32	32	64	64
Peak BW [GB/s]	51.2	51.2	42.7	320	250
Msr BW [GB/s]	41	30	34	121	190
CacheBW[GB/s]	576	450	480	288	-
RandBW[GB/s]	11	12	9	15	10

TABLE II
PROGRAM PARAMETERS INCLUDED IN THE PERFORMANCE MODEL

param.	formula	description
m	$m = \frac{\text{total electron}}{ntoroidal \cdot nradial}$	# part. (electrons) / node
e	$e = 3 \cdot \frac{(mgrid - mpsi)}{nradial} \cdot 16$	# elem. of E field / node
f	$f = 5$	# radial discretization
p	$p = mpsi + 1$	# radial grid
t	t	# threads
s_t	$s_t = \frac{m \cdot ntoroidal}{250}$	# $\approx$ part. sent in toroid. dir.
s_r	$s_r = \frac{m \cdot \sqrt{nradial}}{490000}$	# $\approx$ part. sent in radial dir.
l_1	$l_1 = 240$	# iterations
l_2	$l_2 = 24$	# sorts performed

machine specific parameters								
Machine	c_1	c_2	c_3	c_4	c_5	c_6	c_7	c_8
Piz Daint	6	6	5	16	3	3	6	0.00058 s
Mira	9	6	12	19	6	2	3	0.00002 s
Titan	4	6	1	13	3	3	2	0.0016 s
Stampede	6	6	6	11	3	2	6	0.0048 s

pattern, source, and destination. Our analysis considers both vertical (between the memory hierarchy levels) and horizontal (between the processes) bandwidth. For intra-node accesses, we model the number of cache lines transferred between the memory levels. For inter-node communication, we analyze the amount of data transferred over the network. This systematic approach yields very high accuracy execution time predictions (93% in the worst case). The benefits of our model are threefold: (i) it confirms the high scalability of GTC-P beyond our measurements; (ii) it allows us to guide the optimization of data movement in the system; and (iii) it demonstrates that just modeling the data movement is necessary and sufficient to predict the performance of GTC-P.

1) Architecture: Architectural parameters greatly impact the performance of data movements. We have analyzed each of the machines separately and modeled the cost of local and remote data movement. For local computation, we capture how many cache misses are caused by the code. Another important aspect is the memory bandwidth for random access patterns. For example, accelerator architectures (such as Xeon Phi or GPU) are very sensitive to data alignment and continuity.

We measured the observable bandwidths by evaluating the representative kernels of the GTC-P code. The third loop of the **sort** routine performs a contiguous memory copy, therefore its performance reflects the achievable memory bandwidth. For the random access pattern and cache performance, we evaluated the first and second loop of the **sort** routine, as they exploit both random accesses and memory reuse. Network bandwidth was based on the performance of particle shifts in toroidal and radial directions.

For the inter-node communication, we estimate the cost of point to point and collective data transfer and synchronization. All machine parameters that determine the runtime and scalability of the GTC-P code are listed in Table I.

2) Kernel analysis: We analyze the main two aspects (local and remote memory movement) also on the algorithm level. We first extract the execution *kernels* from the interested subroutines, which determine the granularity at which we analyze the code. For each kernel, we find the modeled execution time based on core program parameters listed in Table II. A static code analysis refined by the runtime profiling was used to find the exact values of those parameters. For example, the machine specific parameters are estimated using statistical curve fitting from the code analysis.

All the heavy kernels (the 7 most time consuming kernels whose net execution time corresponds to over 95% of the total execution time) execute intra-node communication loops over the whole particle array. The efficient vectorization and data locality optimizations are crucial for proper utilization of memory bandwidth. We discuss chosen optimizations applied in §III.

The communication part consists of two stages: toroidal and radial. In the toroidal stage, particles are sent to the neighboring processes in a sweep. As a result, the toroidal stage execution time scales linearly with regard to the size of the toroidal decomposition (with a possible overhead from the network bandwidth saturation). In the radial decomposition, however, the particles are transferred in multiple steps, which implies the use of collective reduction operations. The radial stage scalability is determined by the performance of collective operations on a given topology. Due to large message sizes, the communication cost is bandwidth-limited: the message latency is negligible. Different topologies affect the ratio between the bandwidth in toroidal and radial direction For example, on Mira's high-dimensional 5D torus topology with highly optimized collectives, GTC-P scales almost linearly. On Titan, on the other hand, the lower-dimensional 3D torus causes scalability issues.

B. Data Layout Optimization

GTC-P extensively uses dynamically-allocated arrays for storing particle and mesh information. Most of these arrays are allocated in the initialization phase and do not need to be resized during the actual simulation. Thus, memory management costs are insignificant. The majority of the multidimensional mesh-related arrays are flattened and 0-indexed to simplify indexing, inter-process communication, and also permit memory access locality optimizations. Each MPI task maintains nearly 100 arrays of various dimensions. The majority of these arrays

are read-only during every time step. However, the larger ion and electron arrays are updated frequently.

For storing some of the frequently-accessed particle and grid arrays, we have a choice between using an Array-of-Structures (AoS) layout and a Structure-of-Arrays (SoA) layout. For an illustration of implications of the data layout choice on locality, consider the particle-processing **push** kernel. Here, 12 double-precision values per particle are either read or modified, and 3 double-precision values per grid point (representing the electric field) are accessed. The toroidal followed by radial domain decomposition implies that each MPI task is assigned a poloidal grid partition. The grid points within this poloidal partition are further linearized, and indexing begins from the innermost circle to the outermost one. In order to maximize the data reuse when reading the electric field for grid-to-particle interpolation, we use AoS for storing the electric field. This guarantees that the field read on eight individual grid points (192 bytes) for each particle will be complete by loading up to four cache lines. Further, particles are periodically sorted according to their toroidal and radial position in the sort subroutine, and this improves reuse of the electric field cache lines fetched.

For particle data, we provide both AoS and SoA data layout implementations, as the choice might lead to significant performance variation on diverse architectures. To facilitate code generation and portability, each code version is determined at compilation time by using pre-compiler directives. Generally, SoA is the option for maximizing spatial locality for streaming access and for enabling SIMD instructions, and thus is widely used for GPUs. An AoS particle layout simplifies packing and unpacking of communication buffers in the shift subroutine. We notice that an AoS layout for electrons leads to better overall performance on CPU systems. To ensure data alignment, for AoS layout, we use padding so that each particle takes 16 double-precision values (rather than 12).

C. Improving vectorization

The adiabatic electron (ion dynamics only) model and the first implementation of kinetic electrons of GTC-P use "holes" to represent the invalid particles, i.e., in a distributed environment, at every time step, the particles that are being moved to other processes are marked as holes and considered to be invalid in the local particle array. These invalid particles are then removed from the array periodically to empty memory space for new incoming particles during **sort**. In this implementation, two particles in consecutive memory locations may have different operations in **charge** and **push** depending on if they belong to the same type of particles (valid or invalid) or not, which brings difficulty for automatic vectorization. To maximize the usage of vector units, for kinetic electron simulations, we remove the holes completely for **charge** and **push**, similarly as we do on GPUs. Specifically, the holes are filled at the end of **shift** by the new incoming particles sent from neighboring processors at every time step. If a process has sent more particles than received, then the remaining holes are filled with the last particles in the array.

D. One-Sided Communication

To leverage the capabilities of hardware enabled Remote Direct Memory Access (RDMA), we have implemented the code version that uses MPI-3 One-sided communication [20] for transferring particles between processes. One-sided communication enables direct access to the remote buffers and reduces the communication overhead by avoiding message matching and complex communication protocols. Additionally, explicit separation of communication and synchronization allows better computation overlap.

In the shift phase, we transfer the outgoing particles directly to the neighboring processes particle array using MPI_Put(). To coordinate concurrent access to the particle array at the target process, source processes use MPI_Fetch_and_op() to reserve buffer space in the array. Remote put operations are overlapped with packing the particles locally to the transmission buffer.

While MPI-3 One-sided performance should be better than the traditional message passing model on all architectures supporting RDMA in hardware (all considered machines in our study), not all MPI libraries implement the relatively new specification in a high-performance manner [21]. We observed that on Mira, the One-sided version improved application performance between 5-7% for large runs. On the other architectures, however, our measurements reported a slight performance degradation or no significant difference due to the suboptimal implementation (often one-sided libraries are not implemented using RDMA but instead using normal MPI-1 messages). We expect that future MPI libraries will allow the one-sided communication to utilize the full hardware potential.

E. Leveraging GPU and Xeon Phi Accelerators

To exploit heterogeneous systems accelerated by throughput-oriented processors such as GPUs, we need to identify routines with high compute intensity and low memory and data movement costs, and then refactor the code to exploit the accelerators. We thus migrate the particle-based routines (**charge**, **push**, as well as **sort**) to the accelerator, while leaving the grid-based routines on the CPUs. The **shift** routine is split between the CPU and the GPU.

The GPU implementation benefits from the SoA layout because this layout maximizes spatial locality on streaming accesses. We exploit low-overhead scheduling on the GPUs by using fine-grained parallelism to tackle load imbalance. We leverage the high throughput cores to redundantly compute values rather than precompute and load data from memory. We also use cooperative scheduling [22] to optimally use the limited locality-capturing hardware structure (cache and shared memory) in GPUs. Additionally, we fuse compute loops to further reduce memory usage for auxiliary arrays. Finally, we keep the main data structures persistently allocated in GPU memory to reduce the stress on the PCIe bus transfer time, as this is known to be a major bottleneck for the use of accelerator technologies in HPC. All optimization choices were driven by performance modeling as well as empirical measurements from different design strategies [14, 22].

Intel's Xeon Phi promises ultimate software portability: standard MPI, OpenMP, or hybrid applications can just be recompiled and run in Xeon Phi's native or symmetric mode. In native mode, an application runs on a distributed system consisting of multiple Xeon Phi cards. Furthermore, Xeon Phi offers the option to run an MPI-parallel application on the host CPU and offload parts of it using a pragma-based approach similar to OpenACC or the OpenMP 4.0 accelerator directives.

Xeon Phi's architecture is quite different from standard x86 CPUs and GPUs which requires special care when tuning for highest performance. Cores have been simplified to in-order execution but offer an extended 512-bit vector instruction set that is crucial for application tuning. Thus, special care must be taken for a data alignment, code vectorization, utilization of streaming stores, and occupancy of both the data and instruction pipelines.

We first tune the single-socket performance of GTC-P as this will form the basis for our large-scale parallel performance. GTC-P on a single Xeon Phi with 240 threads in native mode spends most of the execution time in two kernels, **push** and **sort**. We first exploit thread affinity mapping as Xeon Phi implements 4-way hyperthreading which makes thread placement crucial for data locality. Both routines rely heavily on caching and data reuse. Using all four threads per core helps hide the in order execution latency, but forces the threads to share a cache. Changing affinity from scatter to compact results in up to a 40% speedup. To ensure data alignment, each particle is padded two additional elements so that each then uses 128 bytes of memory. The optimized implementation takes full advantage of streaming store instructions.

IV. Experimental Setup

A. Systems

In the last decade, processor architectures have diversified to maximize energy efficiency. Many systems still rely on superscalar out-of-order processors like the Intel Xeon (Sandy Bridge and Ivy Bridge) and AMD's Opteron. Systems such as Argonne National Lab's Mira [23] rely on energy-efficient in-order core to minimize wasted energy. Alternately, systems like the Oak Ridge National Lab's Titan [24], the Swiss National Supercomputing Center's (CSCS) Cray XC30 Piz Daint [25], the Texas Advanced Computing Center's (TACC) Stampede [26], and the Chinese National Supercomputer Center in Guangzhou's (NSCC-GZ) Tianhe-2 [27] rely on accelerators for the bulk of their computing performance. While the former use one NVIDIA Kepler K20x GPU per node, Stampede and TH2 use the Xeon Phi (Knights Corner) coprocessor. In the case of Stampede, there is one per node, while there are 3 per node on TH2.

Concurrent with the development and deployment of novel architectures designed to maximize on-node performance and efficiency, has been the development of high-radix, high-performance networks. Whereas some machines use more conventional 3D torii and fat trees (Titan, Stampede, TH2), other systems have leveraged 5D torii (Mira) and Dragonfly (Piz Daint) topologies. A Dragonfly can be viewed as a hierarchical network in which each rank of the network is fully connected. Thus in rank-0 of Cray's Aries network, 64 nodes are fully connected, while in the rank-1, six groups of 64 nodes are fully connected at the group level [28]. As a result, the Dragonfly can provide very high bisection bandwidth coupled with very low latency.

TABLE III
GTC-P numerical settings for kinetic electron simulations. *mpsi* is the number of grid points in the radial dimension. *mthetamax* is the maximum number of grid points in the poloidal dimension. *mgrid* is the number of grid points per toroidal plane. *ntoroidal* is the total number of toroidal planes. *micell/mecell* is the number of ions or electrons per grid point. *total ion/electron* = *mgrid*×*ntoroidal*×*micell/mecell*. For all simulations, we set *micell* = *mecell* = 100.

	Grid Size			
	A2	**B2**	**C2**	**D2**
mpsi	200	400	800	1600
mthetamax	784	1568	3136	6272
mgrid	157785	629169	2512737	10043073
ntoroidal	32	32	32	32
total ion	504271872	2012060672	8038195200	32132710400
total electron	504271872	2012060672	8038195200	32132710400

B. Programming Models

Unfortunately, one must use three different programming models in order to target the diversity of today's multicore and accelerated supercomputers. MPI is ubiquitous across systems for handling distributed memory parallelism. Conversely, when targeting CPU architectures (including host-only experiments on accelerated supercomputers), NUMA-optimized OpenMP is appropriate. Conversely, when targeting NVIDIA GPUs, CUDA still delivers superior performance to OpenACC. Hence one is required to mix OpenMP for the cpu cores with CUDA for the GPU cores. When attempting to exploit the Xeon Phi coprocessor, Stampede enables both *symmetric* mode, in which one may run one Xeon process (16 threads) and one Xeon Phi process (240 threads) per node (both MPI+OpenMP), as well as *offload mode* in which one must mix MPI, OpenMP, and the Intel-specific offload directives.

C. Problem Specifications and Execution

This paper uses a kinetic electron model in which we simulate the motion of both ions and electrons for four problem sizes (Table III). The electrons are subcycled. Orthogonal to these physical simulation parameters, we perform both weak and strong scaling experiments. For strong scaling experiments, we define a problem of 80.38M grid points with 100 ions and 100 electrons per grid point and strong scale from 512 nodes. For weak scaling experiments, we define a maximum problem size at the full machine of 321.3M grid points with 100 ions and 100 electrons per grid point, and cut this problem down to 80.38M, 20.14M, and 5.024M grid points running on 25%, 6.3%, and 1.6% of each system. We also perform weak scaling experiments with a fixed problem

TABLE IV

PERFORMANCE MODEL OF THE GTC-P APPLICATION. PRESENTED COMPARISON BETWEEN THE BEST FITTING MODEL (PIZ DAINT) AND THE WORST (TITAN). EXAMPLE SCENARIO SHOWN FOR 1024 NODES WEAK SCALING. B_{mc}, B_{mr}, B_c, B_{nt} AND B_{nr} REFER TO MEASURED BANDWIDTH PER CACHE LINE FOR CONTINUOUS MEMORY ACCESS, RANDOM MEMORY ACCESS, CACHE, NETWORK COMMUNICATION IN TOROIDAL AND RADIAL DIRECTION.

			Piz Daint			Titan		
subroutine	**kernel**	**model**	**%time**	**pred.**	**measured**	**%time**	**pred.**	**measured**
intra-node data transfers								
push	loop1	$l_1(\frac{c_1 m}{B_{mc}} + \frac{c_2 e + c_3 f}{B_{mr}} + \frac{c_2(m-e)+c_3(m-f)}{B_c})$	23	3.82	3.77 (1.01)	14	3.33	3.47 (0.95)
	loop2	$l_1(\frac{c_4 m}{B_{mc}} + \frac{c_5 p}{B_{mr}} + \frac{c_5(m-p)}{B_c})$	36	6.13	5.76 (1.06)	28	6.61	6.77 (0.97)
sort	loop1	$l_2(\frac{c_6 m}{B_{mc}} + \frac{c_7 p}{B_{mr}} + \frac{c_7(m-p)}{B_c})$	0.8	0.14	0.13 (1.04)	0.6	0.13	0.12 (1.13)
	loop2	$l_2(\frac{9m}{B_{mc}} + \frac{5m}{B_c})$	2.1	0.35	0.34 (1.03)	2	0.46	0.49 (0.95)
	loop3	$l_2(\frac{8m}{B_{mc}})$	1.9	0.31	0.32 (0.98)	1.8	0.41	0.41 (0.99)
shift	detect part.	$l_1(\frac{3m}{B_{mc}} + \frac{s_t+s_r}{B_{mr}})$	9.1	1.53	1.65 (0.94)	11	2.55	2.53 (1.01)
	pack part.	$l_1(\frac{4(s_t+s_r)}{B_{mc}})$	5.8	0.97	0.93 (1.03)	3.8	0.89	0.88 (1.01)
inter-node data transfers								
shift	toroidal	$l_1(\frac{s_t}{B_{nt}})$	10.6	1.86	2.14 (0.87)	20.3	4.83	5.2 (0.93)
	radial	$l_1(\frac{s_r}{B_{nr}} + c_8 log_2(r))$	7.3	1.04	0.99 (1.05)	9.5	2.27	2.83 (0.8)
total			96.6	16.9	16.8 (1.006)	91	22.4	23.7 (0.94)

size per node and simply scale the number of nodes from 256 to 16K on each system.

V. PERFORMANCE RESULTS

All simulations and scaling experiments were run with shared access to each system in order to highlight their realistic potential, i.e., GTC-P simulations using half or one quarter of the machine run concurrently with other jobs contending for the network. For each data point, we perform multiple experiments, collecting average time for each and reporting the best.

A. Performance and Energy Metrics

Although PIC models are computationally efficient, they embody little data locality. As such, metrics like flop/s or percentage-of-peak are irrelevant on today's architectures which require 10 flops per byte moved from DRAM in order to be compute-limited. Concurrently, energy is becoming a ever larger fraction of total system cost. To that end, we focus on time-to-solution (seconds), energy-to-solution (KWh) as the relevant performance metrics. Our code is executed in a bulk synchronous manner on key phases (**push**, **shift**, etc...) via MPI barriers. We use MPI's `MPI_Wtime()` to time these key phases on process 0. Through interaction with the computing centers, we were able to measure energy consumed on a rack by rack basis (perfectly acceptable for simulations spanning the entire system).

B. Model evaluation

Large runs are hard to reproduce and generate a small number of data points. However, due to the iterative nature of the GTC-P code, we were able to establish a statistically sound model. During each run, most of the functions are called 240 times. The only exception is the sorting procedure, which is executed 24 times. We have instrumented the code with the libLSB library [29] which collects hardware timers and PAPI counters [30] to not only enable a statistically sound analysis of the execution time, but also of issued load and store instructions and cache misses.

We observe that the runtime of all kernels (except for radial communication) depends solely on the local number of particles and is independent of total problem size and number of nodes. This guarantees constant execution time for weak scaling. We also see that the parallel scaling is mainly determined by the network topology. Yet, in all scenarios the radial shift subroutine accounted for up to 30% for the largest number of nodes (30% on Stampede, 20% on Titan, 13% on Piz Daint and 10% on Mira).

In Table IV we illustrate the performance model for Piz Daint and Titan. The differences between the machines apply only to **push** and loop1 of **sort**. The **push** function performs random accesses to 5 auxiliary arrays of varying length. Analysis of PAPI counters confirmed that Piz Daint, having the smallest L3 cache, suffers from the largest number of cache misses. Therefore, this number is adjusted for each architecture. Loop1 of **sort** simply performs a comparison of the particles positions. The efficiency of loop vectorization with conditional assignments depends on the compiler and architecture specification. In Table IV we can observe a different time distribution among the kernels for the best fitting model (Piz Daint) and the worst (Titan).

The measurements show little variation in the execution time. The coefficient of variation is below 0.015 for all heavy kernels. However, as we cannot assume the normal distribution of the data, we use the median of measurements for validating the model. The accuracy of the model varies from 93% for Titan and Mira, 94% for Stampede to 99% for Piz Daint.

C. Model Uses and Predictions

The model helps pinpoint the most critical architectural parameters. Subroutines that rely heavily on streaming, vectorized accesses (**push** and local part of **shift**) benefit greatly from high memory bandwidth. Moreover, large cache size and high cache bandwidth is important to leverage

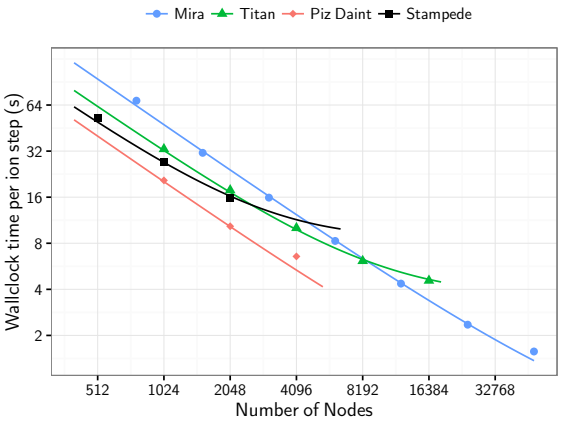

Fig. 3. GTC-P (kinetic electron) strong scaling for the 80 million grid points, 8 billion ion and 8 billion electron case on Titan (GPU), Mira, Piz Daint (GPU) and Stampede (host only). Note the log-log axes. Solid line indicates model-predicted running time.

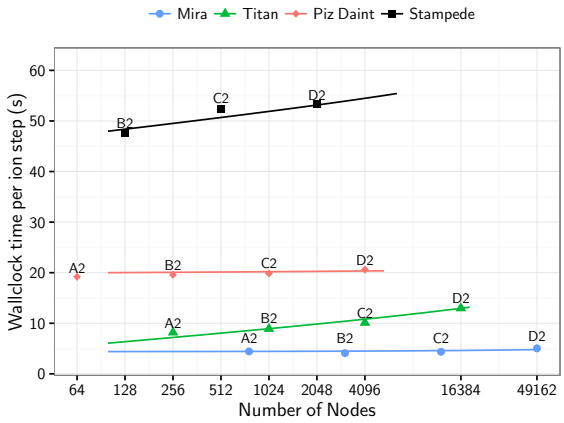

Fig. 4. GTC-P (kinetic electron) weak scaling time-to-solution. On each system, we run four problems (5M, 20M, 80M, and 321M grid points) each using 100 ions and 100 electrons per grid point. These four configurations are run at 1.6%, 6.3%, 25%, and 100% of the maximum nodes we use for each system. Solid line indicates model-predicted running time.

spatial and temporal data locality (**push**). On the other hand, subroutines that perform many random accesses and conditional assignments (**sort**) are limited by random access memory bandwidth. Architecturally, we observe that GPUs deliver the best B_{mc}, as shown in Table I. This suggests that the model could have easily predicted the potential performance benefit of using GPUs before making the development efforts. As we scale, GPUs do not see an advantage for random access B_{mr}. As such, for problem configuration, where the performance is limited by B_{mr}, the model predicts that using GPUs would be less advantageous.

For future architectures, especially with heterogeneous compute engines, the model can predict the profitability or speedup of porting specific routines to a target compute engine. Moreover, optimizations such as reducing energy to solution based on DVFS, discussed in §V-F, relies on the estimating the contribution of communication time to the total execution time. An accurate model for performance obviates the dependence on exhaustive exploration of the optimization space. Another important use of such models is to help procurement decisions by estimating the performance impact on applications of architectural tradeoffs. HPC vendors could offer a wide variety of designs that trades streaming bandwidth with random access bandwidth, especially with the emergence of new memory technologies such as HMC and HBM [31]. Hence, having accurate performance models allows predicting the impact on the applications of making a particular procurement choice.

D. Strong Scaling

Figure 3 presents strong scaling for the kinetic electron version of GTC-P. Initially, the entire 80M grid point, 16B particle (8B ion plus 8B electron) simulation is spread over only 768 nodes on Mira (1K on Titan and Piz Daint due to insufficient GPU memory). Although the computation of ion dynamics is the same as that for the adiabatic electron model, in the kinetic electron configuration, electrons are subcycled (**pushe, shifte**) 60 times per ion time step. We observe the effect of smaller problems and electron subcycling strongly differentiates Titan and Piz Daint performance. Specifically, scalability of Titan is moderately impaired beyond 8K nodes (falling to 46% parallel efficiency at 16k nodes), while scalability is impaired on Mira scaling only beyond 24K nodes (falling to 68% parallel efficiency at 49k nodes). Carrying out a full-scale strong scaling run on Piz Daint (4096 nodes) results in additional bottlenecks that are not observed in any other configuration. This may be due to several factors, including, e.g., additional full system extensive network contention and/or OS noise. Stampede falls to 84% parallel efficiency at 2K nodes. Overall, Mira delivers twice the application performance of Titan despite having less than half of the peak performance.

E. Weak Scaling

Figure 4 presents weak-scaled performance where we maximize the number of nodes used. On each system, we run four problems (5.024M, 20.14M, 80.38M, and 321.3M grid points) each using 100 ions and 100 electrons per grid point. These four configurations are run at 1.6%, 6.3%, 25%, and 100% of the maximum nodes we use for each system. We observe good scalability for Mira (5D Torus) and Piz Daint (Dragonfly) with Titan (3D Torus) showing reduced scalability beyond 1K nodes. For the maximum problem size (common across all machines) at the full concurrency on each machine, we observe that Mira delivers more than 2× the performance of Titan and 4× the performance of Piz Daint — clearly a testament to both efficiency and scale.

In order to fully understand the weak-scaled performance with kinetic electrons, we run GTC-P with a fixed problem size per node (5.024M grid points, 504M ions, 504M electrons),

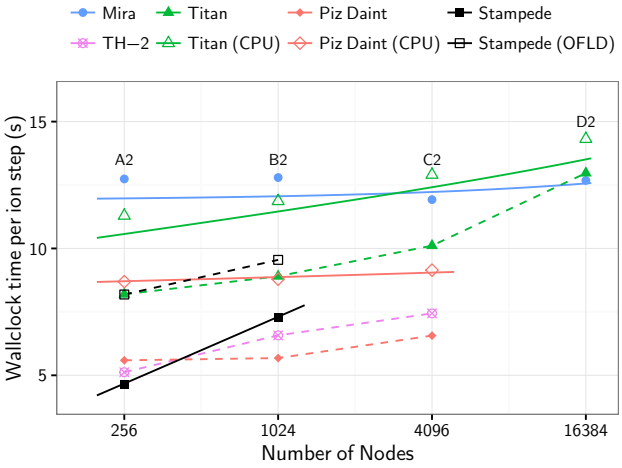

Fig. 5. GTC-P (kinetic electron) weak scaling performance using a fixed problem size per node across all systems allows comparisons of node performance. Solid lines indicate model-predicted running times (shown for Mira, Titan (CPU), Piz Daint (CPU), Stampede), dashed line joins actual running times.

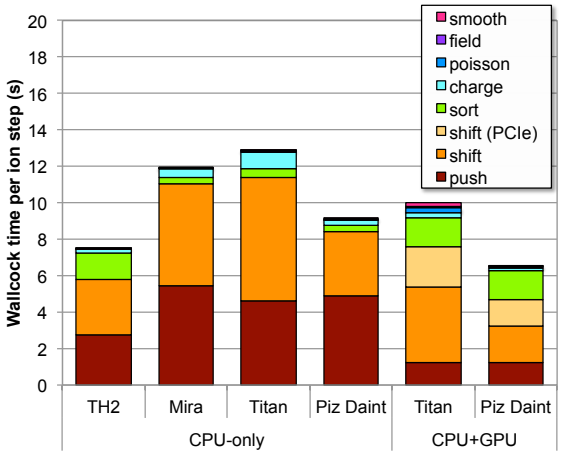

Fig. 6. Operational breakdown of time per step when using 80M grid points, 8B ions, and 8B kinetic electrons on 4K nodes of TH2, Mira, Titan, and Piz Daint.

and simply scale the number of nodes from 256 to 16K on each system by quadrupling grid points and nodes at each stage. Figure 5 highlights the performance differences in the processor and GPU architectures at 256 nodes. We see that performance on Piz Daint is very good, attaining a 60% improvement over Titan despite the same GPU, and more than a 2.2× improvement over BGQ on a node-by-node basis. Whereas BGQ and Piz Daint deliver greater than 85% parallel efficiency, Titan suffers beyond 4K nodes with both CPU and GPU-accelerated implementations falling to roughly 50% parallel efficiency. We see the TH2 (Ivy Bridge-only) and Stampede (Sandy Bridge-only) deliver similar performance (expected given the similarities in processor, memory, and network), and similar performance to the GPU-accelerated Piz Daint — replacing an Ivy Bridge with a GPU provided little net performance gain. Conversely, there was a 1.8× performance penalty when attempting to offload (via directives) work to the Xeon Phi on Stampede. Generally, speaking, the 3D torus, Dragonfly, and Fat Tree deliver similar scalability at small scale (less than 4K nodes). Conversely, networks are differentiated beyond 4K nodes with BGQ's 5D torus continuing to deliver good scalability. We note that Stampede (Sandy Bridge) performs slightly better than TH2 (Ivy Bridge) because the code running on TH2 is an earlier version without a few key optimizations described in §III (due to the time frame we were allowed to access the system). For example, the earlier version will require **sort** to be called more frequently to empty the space for new incoming particles. Future work will include running the new version of the code on TH2 involving multiple Xeon Phis.

A key question arising from this comparison is why the GPU-accelerated systems do not deliver far superior performance compared to the BGQ architecture. Figure 6 shows the breakdown of wall clock time when running a problem of 80M grid points, 8B ions, and 8B electrons on 4K nodes of Mira, Titan, and Piz Daint with or without offloading to the GPUs. Note, **push**, **shift** include 60 calls to the electron variant and 1 call to the ion variant and are hence dominated by operations on electrons. Interestingly, we observe relatively little difference in **push** performance on the three different CPU architectures (Interlagos, BGQ, and Sandy Bridge). Conversely, there is a huge boost in performance when running **push** on the GPUs due to their superior memory bandwidths and flop rates. Given that roughly 14% of the electrons leave each process's domain each time step, coupled with the fact that electron **push** uses a simplified interpolation, the relative cost of **shift** is large. Unfortunately, **shift** is heavily influenced by PCIe and MPI performance. Since the desire for locality demands all particles remain on the GPU, one cannot exploit functional heterogeneity. As such, much of the performance benefits from the GPU are lost and the net performance of the GPU-accelerated Piz Daint is only twice Mira. Note, as described in §III-E, the electron **charge**, **push**, as well as **sort** are running on GPU, while leaving the **poisson**, **smooth** and **field** on the CPUs. The **shift** routine is split between the CPU and the GPU.

F. Time-to-Solution and Energy-to-Solution Comparative Studies

Energy is becoming an increasingly large impediment to supercomputing. The net energy efficiency of large scientific simulations can be particularly non-intuitive as one moves from one processor or network architecture to the next as the interplay between performance and power is highly dependent on algorithm and architecture.

Using power measured under actual load via system instrumentation, Table V shows the energy per time step on 4K nodes of Mira, Titan, and Piz Daint when using 80M grid points, 8B ions, and 8B electrons. We observe that although

TABLE V
ENERGY PER ION TIME STEP (KWh) BY PLATFORM FOR THE WEAK-SCALED, KINETIC ELECTRON CONFIGURATION AT 4096 NODES. POWER IS OBTAINED VIA SYSTEM INSTRUMENTATION INCLUDING COMPUTE NODE, NETWORK, BLADES, AC TO DC CONVERSION.

	CPU-Only			CPU+GPU	
	Mira	**Titan**	**Piz Daint**	**Titan**	**Piz Daint**
Nodes	4096	4096	4096	4096	4096
Power/node (W)	69.7	254.1	204.9	269.4	246.5
Time/step (s)	13.77	15.46	10	10.11	**6.56**
Energy (KWh)	**1.09**	4.47	2.33	3.10	1.84

Mira required the most wall clock time per time step, it also required the least power per node. When combined, this ensured Mira required the least energy per time step of all platforms. Conversely, using the host-only configurations on Titan and Piz Daint required between 2× and 4× the energy with the difference largely attributable to the lack of scalability on Titan. Interestingly, although accelerating the code on these platforms significantly reduces wall clock time per time step, it only slightly increased power. As such the energy required for the GPU-accelerated systems was reduced nearly proportionally with run time.

Optimizing for energy-to-solution could be at odds with time-to-solution when we use technologies such as DVFS [32]. When inter-node communication dominates the execution time, by scaling down the CPU frequency we observed up to 25% reduction in energy consumption for the C problem size with an increase in the execution time by 28%. We conducted such an experiment on an Intel Haswell-based Cray XC40 system. Although supported by many hardware architectures, DVFS control is not enabled on most of the studied systems. As such, we could not explore such optimization on all systems. Another impediment for adopting such technology is the policy adopted for resource allocation by HPC compute facilities, which is based on cpu-hours rather than energy consumption. This policy accordingly makes the "time-to-solution" the best metric from the user perspective.

VI. NEW ITER-SCALE PHYSICS RESULTS ENABLED BY SOFTWARE ADVANCES

Deployment of new extreme scale computing capabilities have produced important new fusion physics findings, including: (i) the rollover trend highlighted in size scaling studies of the ion temperature gradient instability for more than a decade is much more gradual than established previously in far lower resolution, shorter duration studies, with the actual magnitude of the transport level now found to be reduced by a factor of two; and (ii) the more complex size-scaling study up to an ITER-sized plasma for the trapped-electron instability at sufficient phase-space resolution has been carried out for the first time. The successful introduction of drift-kinetic electron dynamics [16] into GTC-P with associated verification and validation of the software has resulted in significant advances from physics perspectives. More specifically,

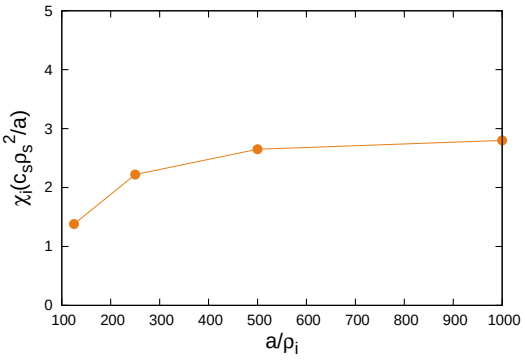

Fig. 7. Global tokamak size-scaling study of trapped-electron-mode turbulence showing the plateauing of the radial electron heat flux (toward the device wall) as the size of the tokamak increases.

we have delivered for the first time new results for the Bohm-to-GyroBohm size scaling investigations with unprecedented resolution and temporal duration (illustrated in Figure 7). Compared to the most prominent earlier publications on this problem, it has now been found that under conditions when the electron dynamics dominate over that of the ions, realistic new results can be obtained at the largest problem size. Specifically, extending the size scaling studies to ITER-scale plasmas have firmly established the plateauing confinement trend, an accomplishment that was previously unachievable for such a large problem size with the associated phase-space requirements to ensure the physics fidelity of the simulations.

VII. FUTURE IMPLICATIONS

A. Demands for Increased Physics Fidelity

At the forefront of outstanding fusion energy computational challenges is the development of higher fidelity kinetic simulations of key instabilities that limit burning plasma performance and threaten device integrity in magnetically-confined fusion energy (MFE) systems. Insights gained from the present studies can be expected to significantly advance such progress, as illustrated by the demonstrated ability to carry out with efficiency and speed the ITER-scale production runs at the spatial resolution and temporal duration demanded, including electron dynamics. Extending these capabilities in the future to encompass electromagnetic physics will introduce new requirements for faster and more portable geometric or algebraic multigrid Poisson solvers, which place much higher demands on interconnect performance than a Jacobi solver. The ultimate goal is to develop a first-principles global simulation encompassing long-time-scale macroscopic dynamics with the fidelity of a microscopic kinetic simulation.

B. Node and Network Architecture

For simulations with kinetic electrons, perhaps the biggest impediments to performance on-node are memory bandwidth (application flop:byte $\approx$ 1) and the inability to maintain a large shared working set in cache. Recently, multicore, many-core, and accelerated architectures have pushed the machine

balance (flops:bytes) to extreme values in order to maximize peak performance. Unfortunately, many science codes cannot exploit these extra flops and can be challenged by rigid SIMD architectures. Today, processor architectures are beginning to integrate multiple types of memory into a single platform and address space, low-capacity fast memory, and high-capacity slow memory. GTC-P can already exploit the smaller fast memories as evidenced by our GPU runs.

The lack of scalability observed today on some platforms is concerning as their networks act as impediments to realizing the full potential of improvements in process and memory technology. Although proprietary high dimensional (5D, 6D) meshes and tori, as well as high-radix networks like the Cray's Aries Dragonfly can be expensive, they provide the requisite scalability for real science codes and can reduce overall system cost per unit of performance substantially.

Heterogeneity promises the best of both worlds, throughput optimized performance on some routines, and latency-optimized performance on others. In this paper, we explored the performance of four heterogeneous systems, Titan, Piz Daint, Stampede, and TH-2. Unfortunately, all of these systems express heterogeneity at the node-level with very limited PCIe bandwidth between host and device. As GTC-P's data cannot be easily partitioned between host and device, exploiting heterogeneity via function specialization is impractical. As a result, some functions (e.g. **push**) were accelerated while others (e.g. **shift**) suffered. Future heterogeneous systems need to be architected to avoid these pitfalls for PIC applications.

C. Energy-efficient Scientific Computing

Today, instrumenting scientific applications to measure energy when running on large supercomputing installations can be cumbersome and obtrusive requiring significant interaction with experts at each center. As such, most applications have little or no information on energy-to-solution across the architecture design space spectrum. In order to affect energy-efficient co-design of supercomputers, energy measurement must be "always-on" by default with, at a minimum, total energy and average power reported to the user at the end of an application. Reporting energy by component (memory, processor, network, storage, etc...) would enable scientists and vendors to co-design their applications and systems to avoid energy hotspots and produce more energy-efficient computing systems.

VIII. Acknowledgments

We gratefully acknowledge the excellent computational and technical support from HPC Centers including ALCF (Mira), OLCF (Titan), CSCS (Piz Daint), TACC (Stampede), and to Professor Yutong Lu and colleagues at the Guangzhou SC Center (TH-2). The efforts from Dr. Chuanfu Xu and Guang Suo of the NUDT/TH-2 team were especially helpful in carrying out runs on TH-2. We would like to acknowledge the assistance of Dr. Lei Huang at TACC, who provided support for the reported offload runs in Stampede. For the key energy to solution measurements, the dedicated efforts and clear explanations of methodology deployed by Venkat Vishwanath of the ALCF, Gilles Fourestey of the CSCS, and Don Maxwell and James Rogers of the OLCF were very much appreciated. We are also grateful to Eliot Feibush of PPPL and Zachary Kaplan for their expert visualization support and to Prof. Zhihong Lin and his colleagues at UC Irvine for the greatly beneficial code verification support in the implementation of the fast electron dynamics feature in GTC-P. The authors express their gratitude for the support provided for our team's research efforts by: the DOE-SC contract DEAC02-06CH11357 at the ANL, the DOE contract DE-AC02-09CH11466 at the PPPL, the DOE-SC contract number DE-AC02-05CH11231 at the LBNL, and the DOE-SC contract DE-AC05-00OR22725 at the ORNL. This work has been supported by the Argonne National Laboratory project "Performance Evaluation for PIC Algorithms on the ALCF BG/Q System" at Princeton University and the Intel Parallel Computing Center project "CliPhi: Towards Optimal Performance for Climate Codes on Intel Xeon Phi" at ETH Zurich.

References

[1] W. Tang and D. Keyes, "Scientific grand challenges: Fusion energy science and the role of computing at the extreme scale," in *PNNL-19404*, 2009, p. 212.

[2] W. Tang, B. Wang, and S. Ethier, "Scientific discovery in fusion plasma turbulence simulations at extreme scale," *Computing in Science Engineering*, vol. 16, no. 5, pp. 44–52, Sept 2014.

[3] W. Lee, "Gyrokinetic particle simulation model," *Journal of Computational Physics*, vol. 72, no. 1, pp. 243–269, 1987.

[4] K. J. Bowers, B. J. Albright, B. Bergen, L. Yin, K. J. Barker, and D. J. Kerbyson, "0.374 pflop/s trillion-particle kinetic modeling of laser plasma interaction on roadrunner," in *2008 SC - International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2008, pp. 1–11.

[5] F. Fiuza and et al., "Record simulations conducted on lawrence livermore supercomputer," https://www.llnl.gov/news/record-simulations-conducted-lawrence-livermore-supercomputer.

[6] M. Bussmann, H. Burau, T. E. Cowan, A. Debus, A. Huebl, G. Juckeland, T. Kluge, W. E. Nagel, R. Pausch, F. Schmitt, U. Schramm, J. Schuchart, and R. Widera, "Radiative signatures of the relativistic kelvin-helmholtz instability," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 5:1–5:12.

[7] B. Wang, S. Ethier, W. Tang, T. Williams, K. Z. Ibrahim, K. Madduri, S. Williams, and L. Oliker, "Kinetic turbulence simulations at extreme scale on leadership-class systems," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage*

and Analysis, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 82:1–82:12.

[8] X. Meng, X. Zhu, P. Wang, Y. Zhao, X. Liu, B. Zhang, Y. Xiao, W. Zhang, and Z. Lin, "Heterogeneous programming and optimization of gyrokinetic toroidal code and large-scale performance test on th-1a," in *Supercomputing*, ser. Lecture Notes in Computer Science, J. Kunkel, T. Ludwig, and H. Meuer, Eds. Springer Berlin Heidelberg, 2013, vol. 7905, pp. 81–96.

[9] E. F. D'Azevedo, J. Lang, P. H. Worley, S. A. Ethier, S.-H. Ku, and C.-S. Chang, "Poster: Hybrid mpi/openmp/gpu parallelization of xgc1 fusion simulation code," in *High Performance Computing, Networking, Storage and Analysis (SCC), 2013 SC Companion:*, 2013, pp. 1441–1441.

[10] Z. Lin, S. Ethier, T. S. Hahm, and W. M. Tang, "Size scaling of turbulent transport in magnetically confined plasmas," *Phys. Rev. Lett.*, vol. 88, p. 195004, Apr 2002.

[11] B. F. McMillan, X. Lapillonne, S. Brunner, L. Villard, S. Jolliet, A. Bottino, T. Görler, and F. Jenko, "System size effects on gyrokinetic turbulence," *Phys. Rev. Lett.*, vol. 105, p. 155001, Oct 2010.

[12] K. Madduri, S. Williams, S. Ethier, L. Oliker, J. Shalf, E. Strohmaier, and K. Yelick, "Memory-efficient optimization of gyrokinetic particle-to-grid interpolation for multicore processors," in *Proc. ACM/IEEE Conf. on Supercomputing (SC 2009)*, Nov. 2009, pp. 48:1–48:12.

[13] K. Madduri, K. Z. Ibrahim, S. Williams, E.-J. Im, S. Ethier, J. Shalf, and L. Oliker, "Gyrokinetic toroidal simulations on leading multi- and manycore HPC systems," in *Proc. Int'l. Conf. for High Performance Computing, Networking, Storage and Analysis (SC '11)*. New York, NY, USA: ACM, 2011, pp. 23:1–23:12.

[14] K. Z. Ibrahim, K. Madduri, S. Williams, B. Wang, S. Ethier, and L. Oliker, "Analysis and optimization of gyrokinetic toroidal simulations on homogenous and heterogenous platforms," *International Journal of High Performance Computing Applications*, 2013.

[15] "GTC-Irvine," http://phoenix.ps.uci.edu/GTC/.

[16] Y. Xiao and Z. Lin, "Turbulent transport of trapped-electron modes in collisionless plasmas," *Phys. Rev. Lett.*, vol. 103, p. 085004, Aug 2009.

[17] S. Ethier, W. M. Tang, and Z. Lin, "Gyrokinetic particle-in-cell simulations of plasma microturbulence on advanced computing platforms," *Journal of Physics: Conference Series*, vol. 16, pp. 1–15, 2005.

[18] M. F. Adams, S. Ethier, and N. Wichmann, "Performance of particle in cell methods on highly concurrent computational architectures," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012001, 2007.

[19] S. Ethier, M. Adams, J. Carter, and L. Oliker, "Petascale parallelization of the Gyrokinetic Toroidal Code," in *Proc. High Performance Computing for Computational Science (VECPAR'10)*, 2010.

[20] T. Hoefler, J. Dinan, R. Thakur, B. Barrett, P. Balaji, W. Gropp, and K. Underwood, "Remote Memory Access Programming in MPI-3," *ACM Transactions on Parallel Computing (TOPC)*, Jan. 2015, accepted for publication on Dec. 4th.

[21] R. Gerstenberger, M. Besta, and T. Hoefler, "Enabling Highly-Scalable Remote Memory Access Programming with MPI-3 One Sided," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. ACM, Nov. 2013, pp. 53:1–53:12.

[22] K. Madduri, E. J. Im, K. Ibrahim, S. Williams, S. Ethier, and L. Oliker, "Gyrokinetic particle-in-cell optimization on emerging multi- and manycore platforms," *Parallel Computing*, vol. 37, no. 9, pp. 501–520, 2011.

[23] "Mira IBM BlueGene/Q," http://www.alcf.anl.gov/mira.

[24] "Titan Cray XK7," https://www.olcf.ornl.gov/support/system-user-guides/titan-user-guide/.

[25] "Piz Daint Cray XC30," http://www.cscs.ch/computers/piz_daint/index.html.

[26] "Stampede," https://portal.tacc.utexas.edu/user-guides/stampede.

[27] "NSCC-GZ," http://www.nscc-gz.cn.

[28] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: a scalable HPC system based on a Dragonfly network," in *Proc. Int'l. Conf. on High Performance Computing, Networking, Storage and Analysis (SC '12)*. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 103:1–103:9.

[29] T. Hoefler and R. Belli, "Scientific benchmarking of parallel computing systems: Twelve ways to tell the masses when reporting performance results," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 73:1–73:12.

[30] P. J. Mucci, S. Browne, C. Deane, and G. Ho, "Papi: A portable interface to hardware performance counters," in *In Proceedings of the Department of Defense HPCMP Users Group Conference*, 1999, pp. 7–10.

[31] A. Suresh, P. Cicotti, and L. Carrington, "Evaluation of emerging memory technologies for HPC, data intensive applications," *The 2014 IEEE International Conference on Cluster Computing (CLUSTER)*, pp. 239–247, Sept 2014.

[32] D. Hackenberg, R. Schne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," *The IEEE International Parallel and Distributed Processing Symposium Workshop (IPDPSW)*, pp. 896–904, May 2015.