Porting and Optimizing GTC-P Code to NVIDIA GPU

Bei Wang

Princeton University

March 19, 2015

GTC Technology Conference, San Jose

PRINCETON UNIVERSITY

Collaborators

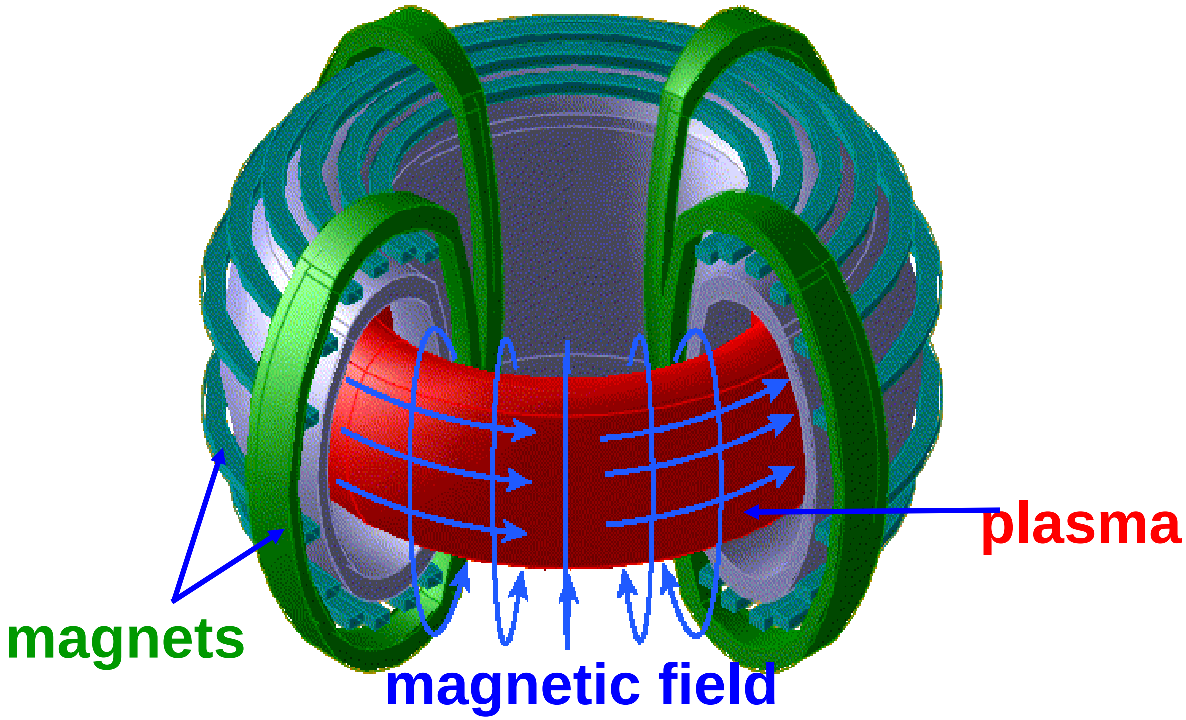
- Computer Science Department of Computational Research Division at LBL: Khaled Ibrahim, Sam Williams, Lenny Oliker

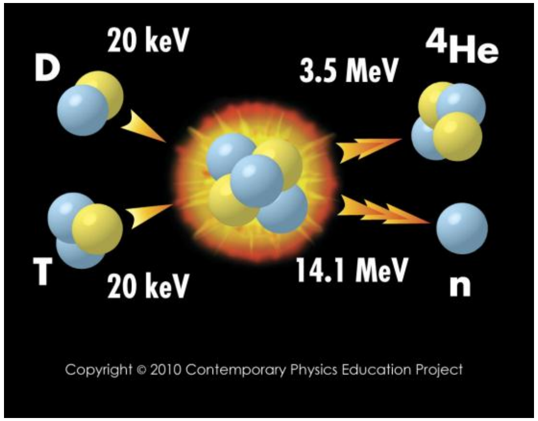
- Penn State University: Kemash Madduri

- NVIDIA: Peng Wang etc.

What we simulate?

TOKAMAK

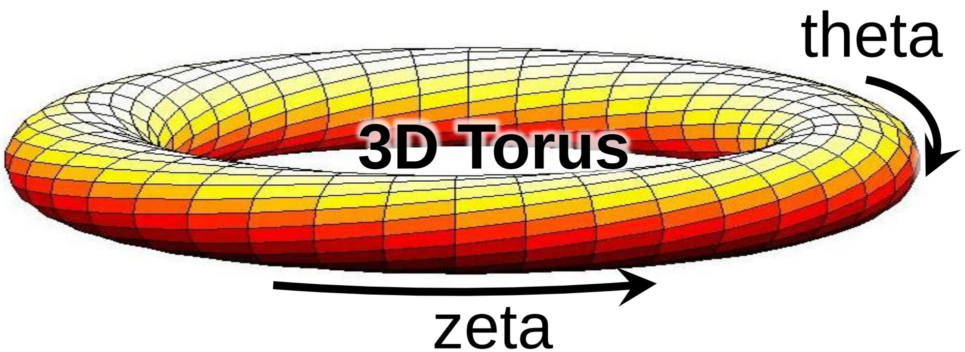


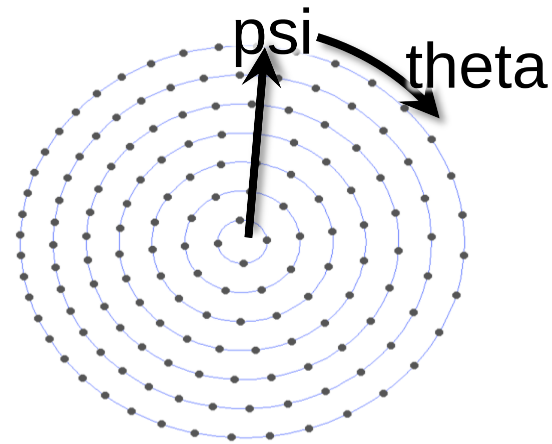


source: cpepweb.org

- Extremely hot plasma (several hundred million degree) confined by very strong magnetic field
- **Turbulence:** What cause the leakage of energy in the system?

- Mathematics: 5D gyrokinetic Vlasov Poisson equation
- Numerical method: Gyrokinetic Particle-in-cell (PIC) method





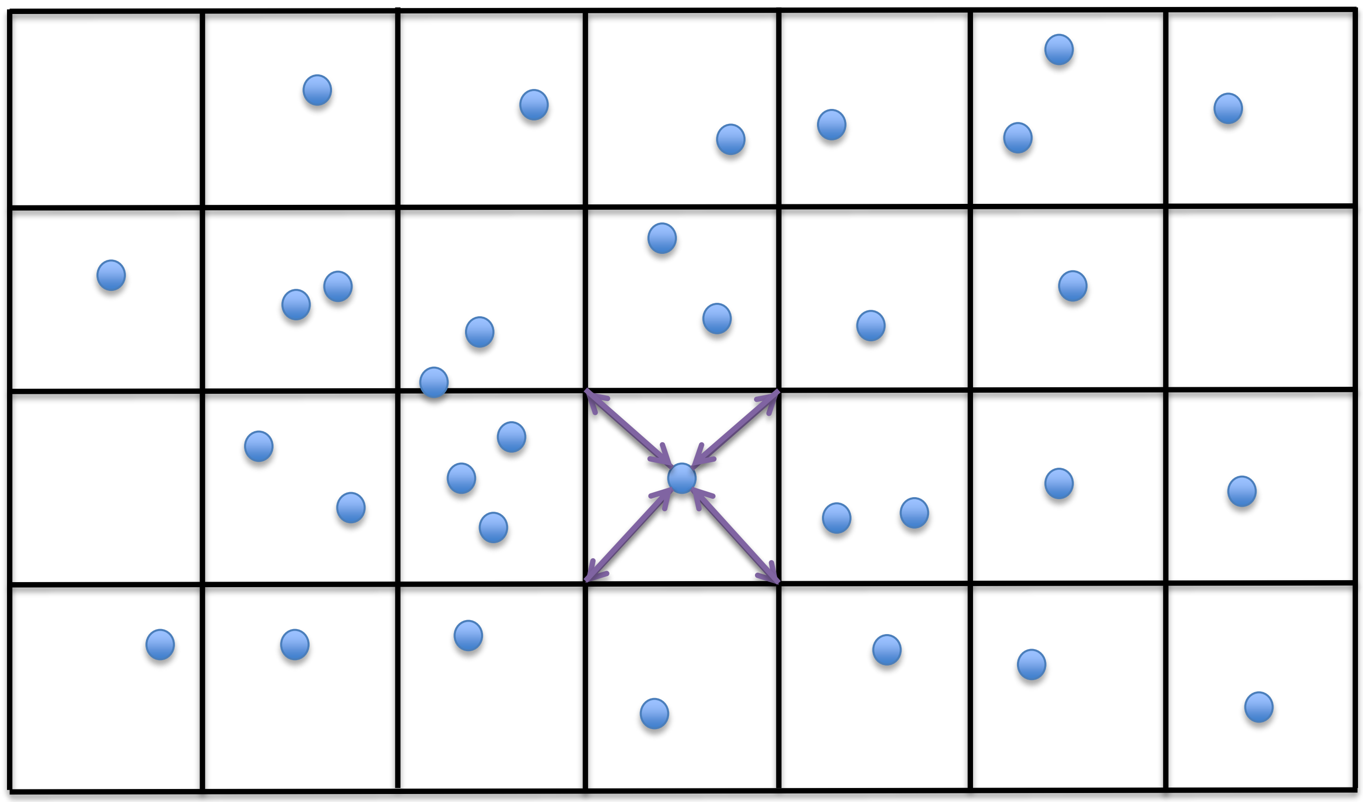
131 million grid points, 30 billion particles, 10 thousand time steps

- Objective: Develop an efficient numerical tool to reproduce and predict turbulence and transport in tokamak using high end supercomputers

PIC method

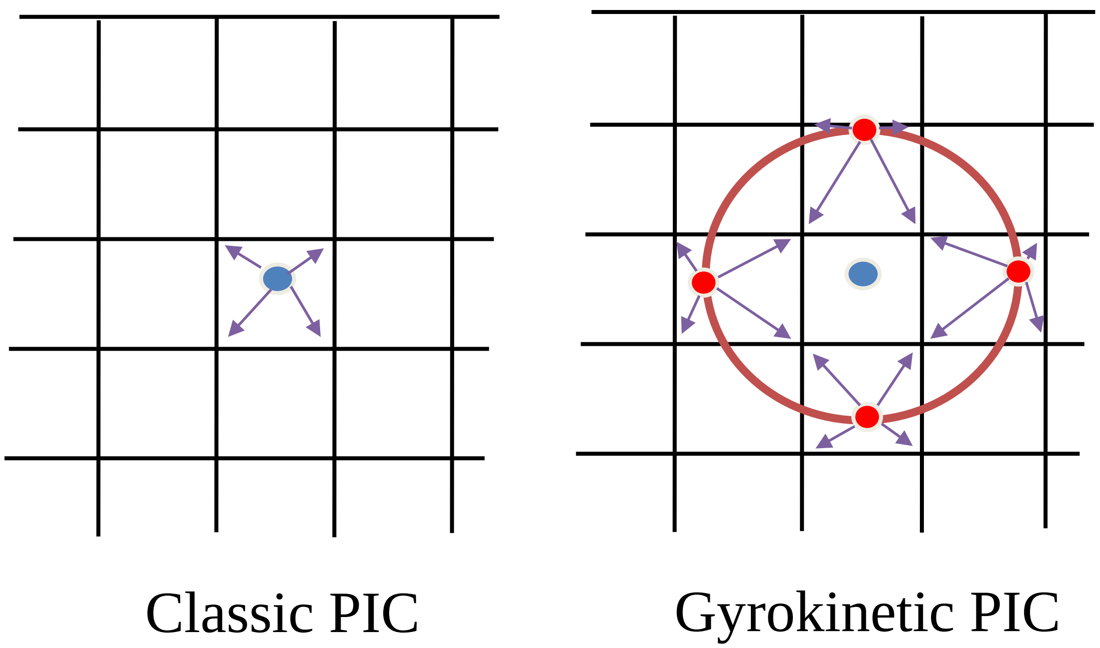
- The system is represented by a set of particles
- Each particle carries several components: position, velocity and weight ($\mathbf{x}$, $\mathbf{v}$, w)
- Particles interact with each other through long range electromagnetic forces
- Naively, forces are calculated pairwise ~ $O(N^2)$ computational complexity
  - Intractable with million or billion number of particles
  - Fast Multiple Method (FMM) ~ O(NlogN)
- Alternatively, forces are evaluated on a grid and then interpolated to the particle ~ O(N+MlogM) (N/M=100-10,000)
- PIC approach involves two different data structures and two types of operations
  - **Charge**: Particle to grid interpolation (**SCATTER**)
  - **Poisson/Field**: Poisson solve and field calculation
  - **Push**: Grid to particle interpolation (**GATHER**)

Particle-grid interpolation

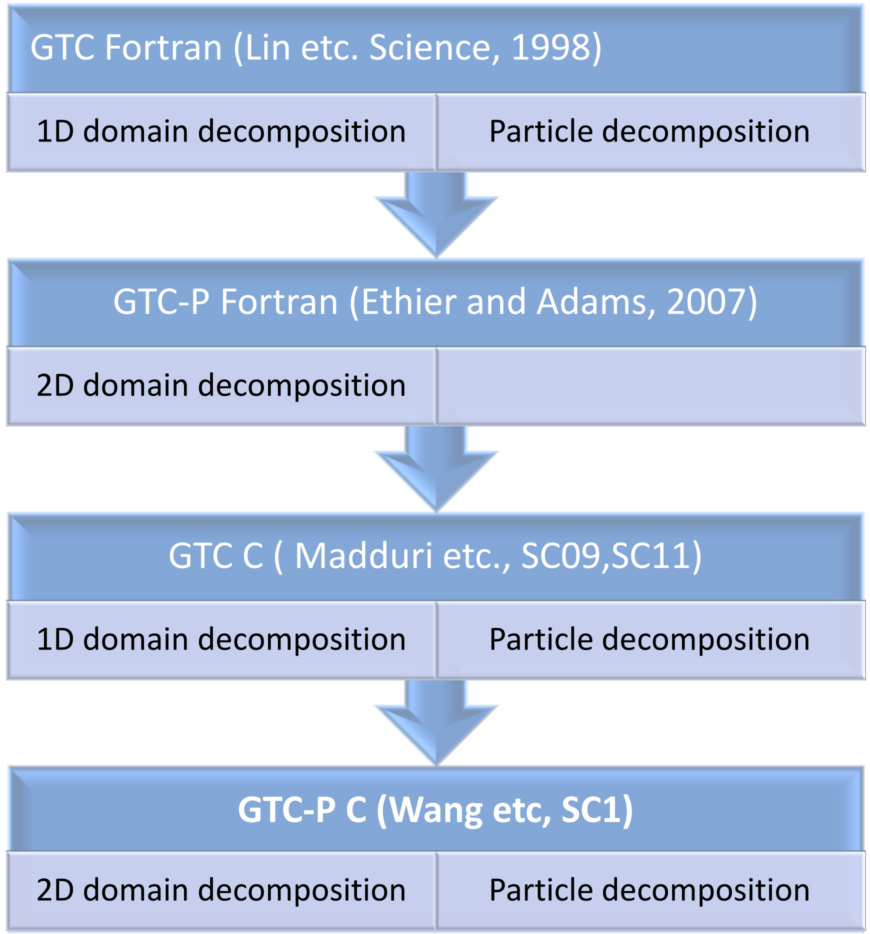


Gyrokinetic PIC method

- Each particle is a charge ring that varies in size
- Particle-grid interpolation is through 4 points on the ring
- Each particle accesses up to 16 unique grid memory locations in 2D plane (increase cache working set)

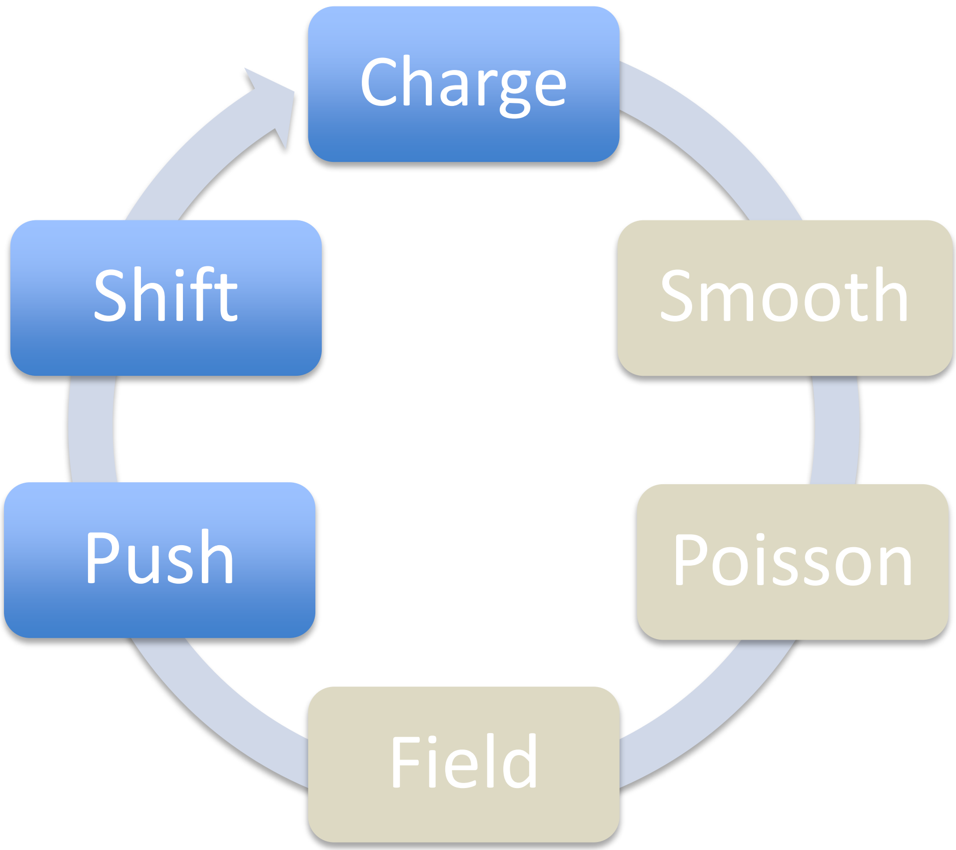


The history of GTC-P code



All codes share the exact same physics model, e.g., electrostatic, circular cross-section

GTC-P: six major subroutines



- **Charge**: particle to grid interpolation (SCATTER)
- **Smooth/Poisson/Field**: grid work (local stencil)
- **Push**:
  - grid to particle interpolation (GATHER)
  - update position and velocity
- **Shift**: in distributed memory environment, exchange particles among processors

GPU Implementations - Charge

- Challenges
  - High memory bandwidth to stream data requires changing the data layout
  - Small memory to core ratio restricts the use of memory replication to avoid data hazards
  - Random memory access -- small cache capacity makes it difficulty to exploit locality to avoid expensive memory access

- First Attempt
  - Use SoA data structure – stream data access
  - Use global atomics – non coalesced

- Second Attempt
  - Use cooperative computation to capture locality for co-scheduled threads – use global atomics, but in a coalesced way (by transposing in shared memory)
  - Leads to relatively poor performance on Fermi, but great performance on Kepler

- Third Attempt
  - Explored different techniques for explicit management of the GPU shared memory – shared memory atomics
  - Leads to good performance on Fermi (where DP atomics operation is expensive), but relative poor performance on Kepler (because it enabled fast DP atomic increment while preserves the same amount of shared memory as Fermi) – extensive shared memory usage

GPU Implementations - Push

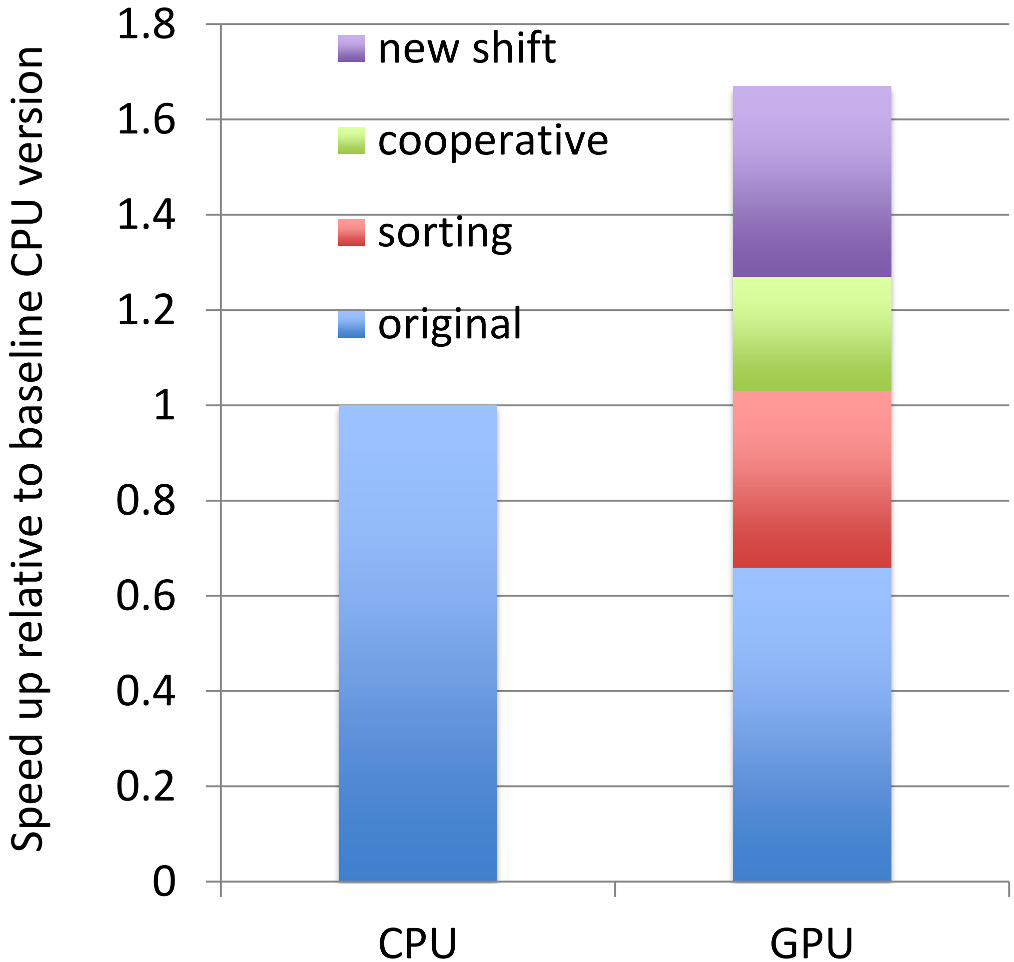
- Challenge
  - Random memory access
- Optimizations Attempts
  - Redundantly re-compute values rather than load from memory
  - Use loop/computation fusion to further reduce memory usage for auxiliary arrays
  - Use GPU texture memory for storing electric field data

GPU Implementations - Shift

- First Attempt
  - Maintain small shared buffer that are filled as it traversed its subset of the particle array – extensive shared memory usage
  - Particle are sorted into three buffers for left, right shift and keep buffer
  - Whenever the local buffer is exhausted, the thread automatically reserve a space in a pre-allocated global buffer
  - Transpose data while flush to global memory – normal shift algorithm on CPU use AoS data structure - data transpose

- Second Attempt
  - Shift and sort are simultaneously executed
  - Particles are sorted into three buffers, left, right shift and keep buffer, relying on fast Thrust lib – no shared memory usage
  - Modify the normal iterative shift algorithm to pass message with SoA data structure – no data transpose

Single Node Results



- Double Precision
- ECC enables
- Intel Xeon E5-2670 vs. NVIDIA Tesla K20x (Piz Daint)
- Speed up ~1.7x

Experimental Platforms

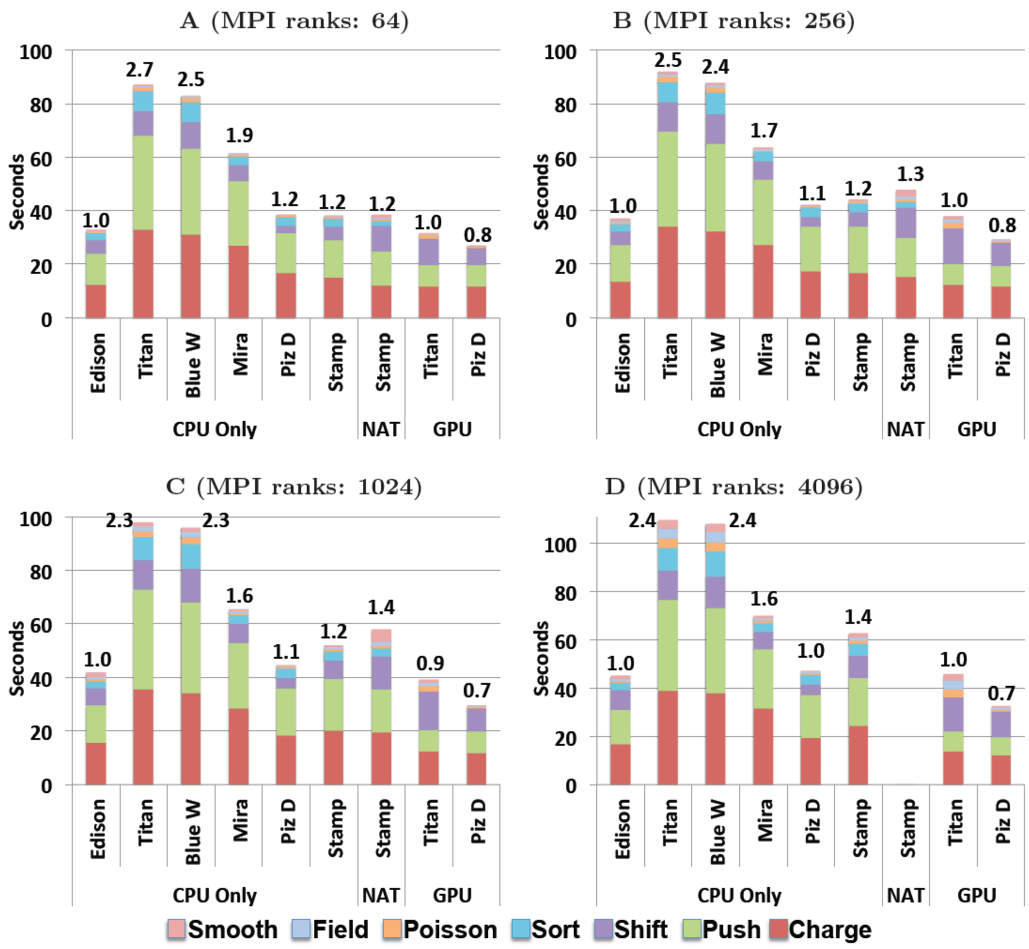
	Edison XC30	Titan XK7		Blue Waters	Mira BGQ	Piz Daint XC30		Stampede MIC Cluster	
Processor	Intel Xeon	AMD Opteron	NVIDIA Kepler	AMD Opteron	IBM BGQ	Intel Xeon	NVIDIA Kepler	Intel Xeon	Intel Xeon Phi
Freq. (GHz)	2.4	2.2	0.733	2.3	1.6	2.6	0.733	2.7	1.1
D$ per core (KB)	32+256	16+2048†	64	16+2048†	32	32+256	64	32+256	32+512
Cores per CPU	12	8	14	8	16	8	14	8	60
D$ per CPU (MB)	30	8	1.5	8	32	16	1.5	20	—
CPUs/Node	2	2	1	4	1	1	1	2	1
DP GFlop/s	460.8	140.8	1314	294.4	204.8	166.4	1314	345.6	1056
STREAM (GB/s)	78*	31*	171	62*	26	38	171	78*	160
STREAM/NUMA	39	15.5	171	15.5	26	38	171	39	160
Memory (GB)	64*	32*	6	64*	16	32	6	32*	8
Network and Topology	Aries Dragonfly	Gemini 3D torus		Gemini 3D torus	5D Torus	Aries Dragonfly		Infiniband Fat Tree	
Compiler	Cray	Cray	NVCC	Cray	XLC	Cray	NVCC	ICC	ICC

TABLE 1

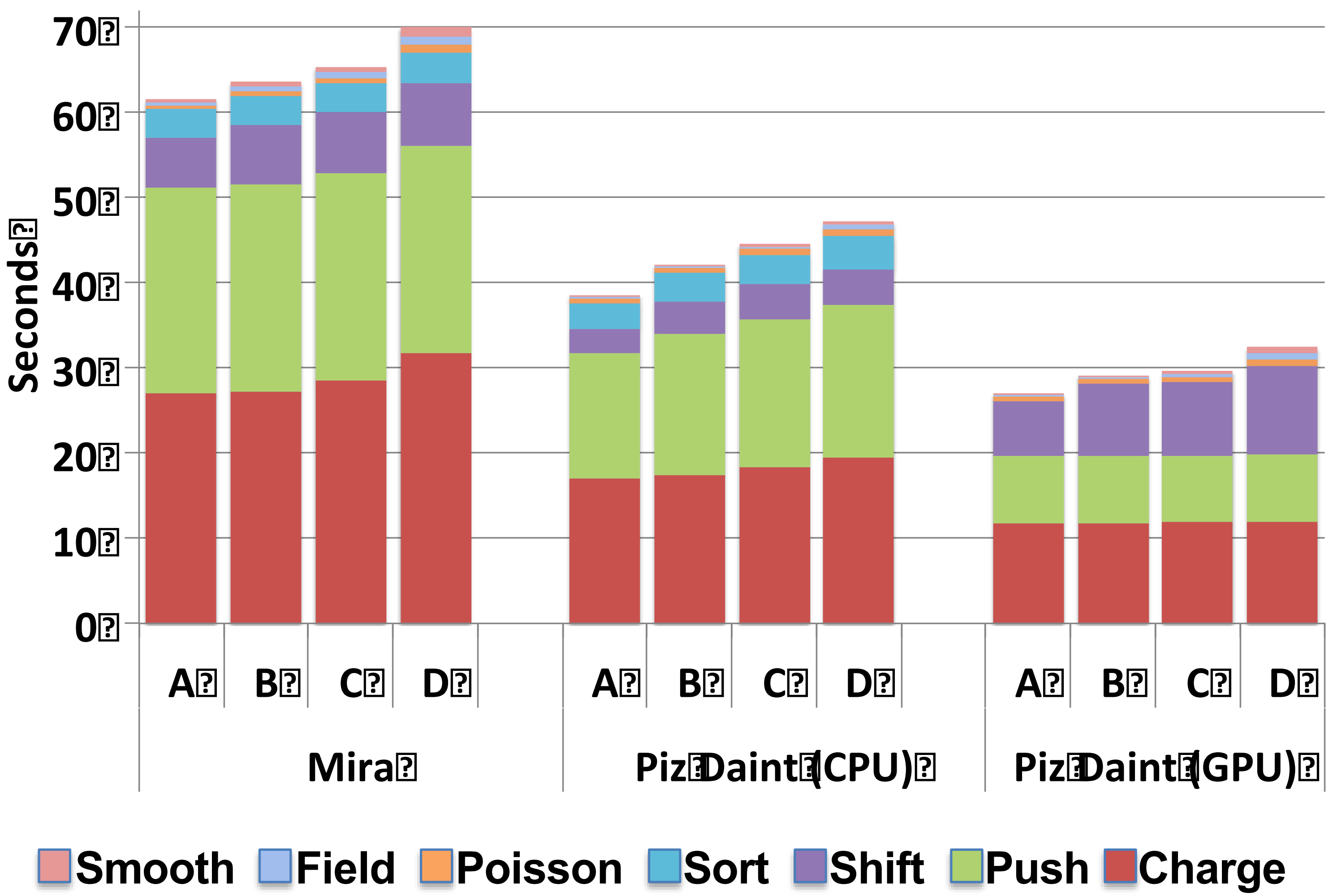
Systems evaluated in this paper. †*Each pair of cores shares a 2MB L2 cache.* **NUMA.*

Performance Evaluation (Weak Scaling) – One Process Per NUMA Node

	— CPU-only Configurations —						MIC-only	CPU+GPU	
	Edison	Titan	Blue W.	Mira	Piz D.	Stamp	Stamp	Titan	Piz D.
MPI Procs	2	2	4	1	1	2	1	1	1
OMP Threads	12	8	8	64	8	8	240	8	8
Accelerators	—	—	—	—	—	—	—	+K20x	+K20x



Performance Evaluation – One Process Per NUMA Node



Conclusion

- Performance of the CPU-based architectures is generally correlated with the DRAM STREAM bandwidth per NUMA node; Large size plasma simulations on CPU suffer from cache misses issue.
- On GPU, substantial speed up on compute intensive “push” despite its data locality challenges (2.29x Kepler vs Intel Xeon E5-2670), moderate speed up on “charge” due to synchronization (1.6x), and no speed up on “shift” (0.78x) due to PCIe challenges; Cache misses issue on large size plasma simulations is no longer significant on GPU as it relies on massive number of threads to hide latency

References

B. Wang, S. Ethier, W. Tang, K. Madduri, S. Williams, L. Oliker, “Modern Gyrokinetic Particle-In-Cell Simulation for Fusion Plasmas on Top Supercomputers”, submitted to SIAM Journal in Scientific Computing, 2015

B. Wang, S. Ethier, W. Tang, T. Williams, K. Madduri, S. Williams, L. Oliker, “Kinetic Turbulence Simulations at Extreme Scale on Leadership-Class Systems”, Supercomputing (SC), 2013.

K. Ibrahim, K. Madduri, S. Williams, B. Wang, S. Ethier, L. Oliker “Analysis and optimzation of gyrokinetic toroidal simulations on homogenous and hetergoenous platforms”, International Journal of High Performance Computing Applications, 2013.

K. Madduri, K. Ibrahim, S. Williams, E.J. Im, S. Ethier, J. Shalf, L. Oliker, "Gyrokinetic Toroidal Simulations on Leading Mult- and Manycore HPC Systems", Supercomputing (SC), 2011.

K. Madduri, E.J. Im, K. Ibrahim, S. Williams, S. Ethier, L. Oliker, "Gyrokinetic Particle-in-Cell Optimization on Emerging Multi- and Manycore Platforms", Parallel Computing, 2011.

S. Ethier, M. Adams, J. Carter, L. Oliker, “Petascale Parallelization of the Gyrokinetic Toroidal Code”, In Proc. High Performance Computing for Computational Science, 2010.

K. Madduri, S. Williams, S. Ethier, L. Oliker, J. Shalf, E. Strohmaier, K. Yelick, "Memory-Efficient Optimization of Gyrokinetic Particle-to-Grid Interpolation for Multicore Processors", Supercomputing (SC), 2009.

M. Adams, S. Ethier and N. Wichmann, “Performance of Particle-in-Cell methods on highly concurrent computational architectures”, Journal of Physics: Conference Series, 2007.

Thank you

Questions?