

Fusion Applications Requirements for Programming Models and Environments in the Exascale Era

Stéphane Ethier

Princeton Plasma Physics Laboratory

With inputs from

Kamesh Madduri (PSU), Sam Williams (LBNL),

Ed D'Azevado (ORNL) and Pat Worley (ORNL)

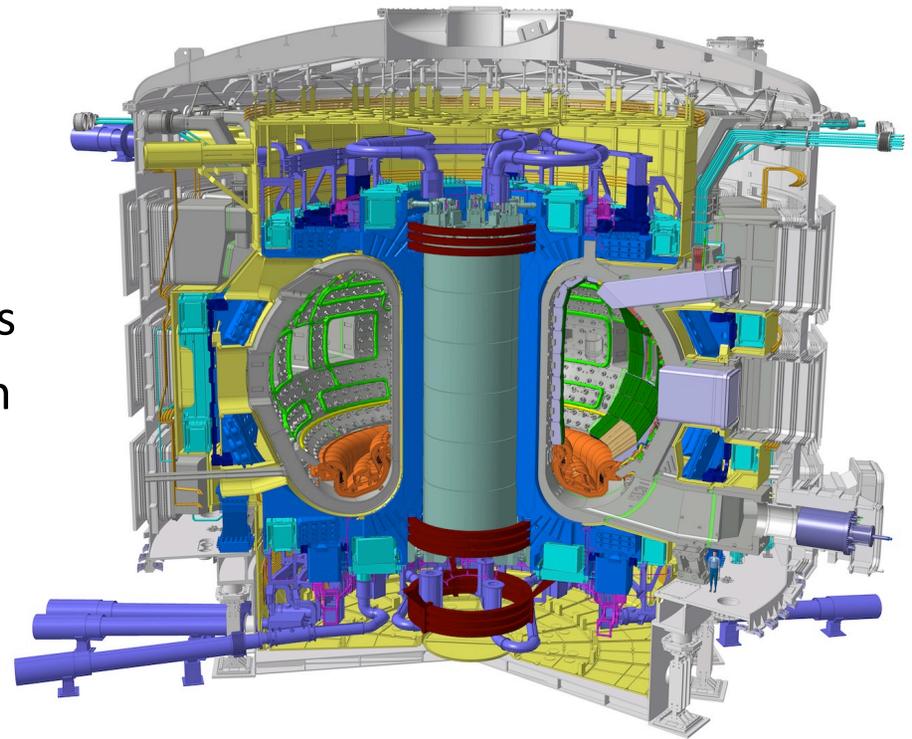
DOE Workshop on Exascale Programming Models and Environments

Mar 9-11, 2015



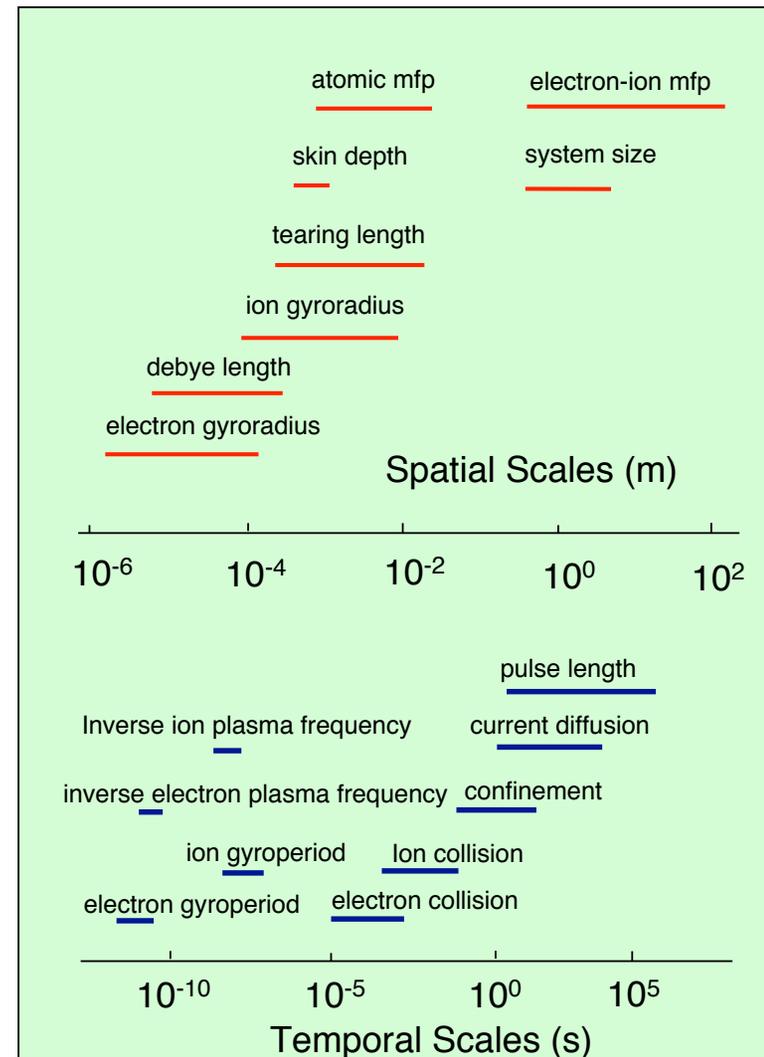
Background

- Simulations of fusion-relevant plasmas confined in a torus-shaped magnetic bottle called “tokamak”
- “Fusion-relevant” means plasma temperatures of 100 million+ degrees
- Biggest challenge: **ITER**
 - Much larger than current tokamaks
 - “Burning plasma” experiments
 - Will require prediction of each tokamak shot
- Ultimate goal is “whole device modeling” (WDM)



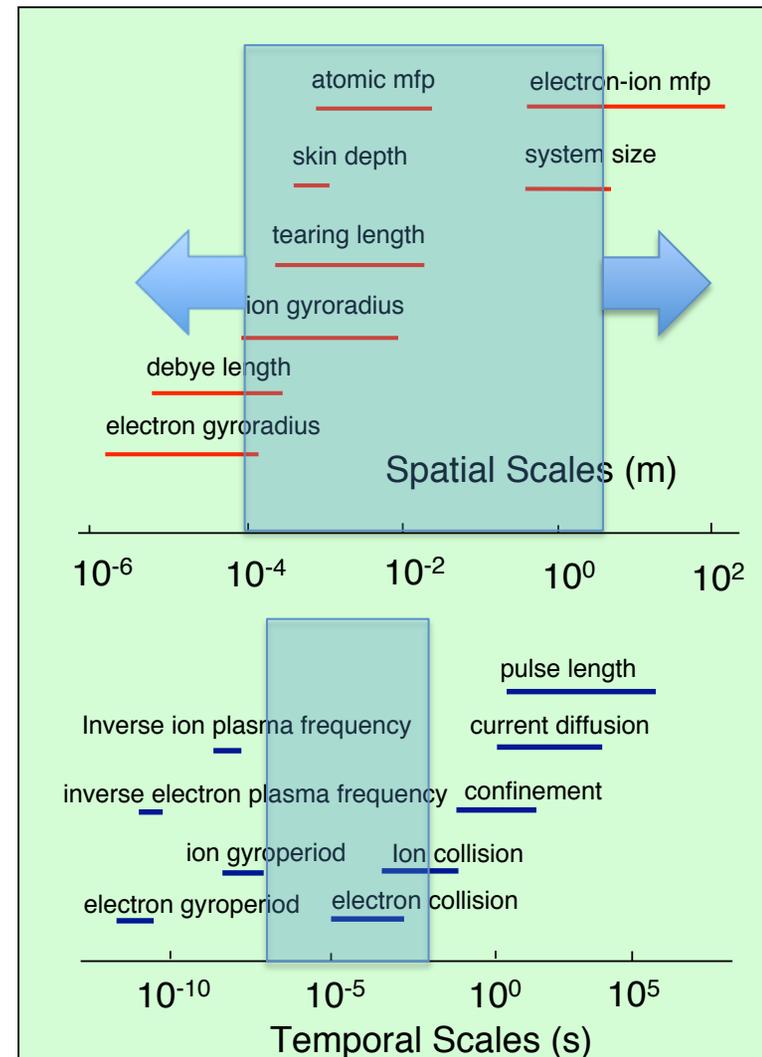
Whole device modeling is extremely challenging

- The important physics spans huge range of spatial and temporal scales
- Overlap in scales often means strong (simplified) ordering not possible
- Transport codes use fluid approach with “reduced” models to simulate full discharge (few seconds)
 - Can miss some important physics hard to capture with models
 - Difficult to find enough parallelism



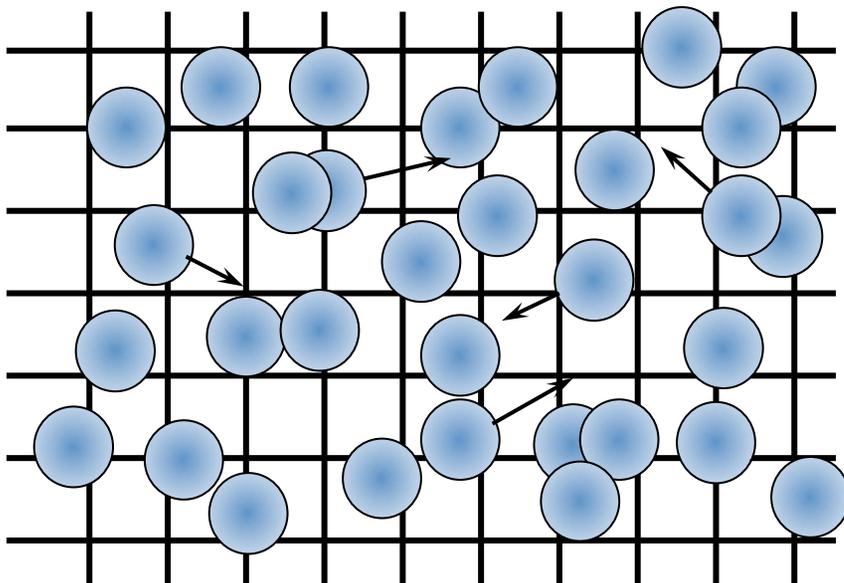
The Gyrokinetic approach

- Start in the middle and do first principles calculation
- Try to expand on “both sides”
- Kinetic approach naturally capture the important physics (turbulence, resonances, etc.)
- The particle-in-cell algorithm is currently the most promising approach to achieve exascale level
 - Lots of parallelism (200 billion particles, 100 million grid points to simulate ITER)



The Particle-in-Cell method in a nutshell

- Particles sample distribution function
- Interactions via the grid, on which the potential is calculated (from deposited charges).
- **100-1000X more particles than grid points**
- Grid resolution dictated by Debye length or gyroradius

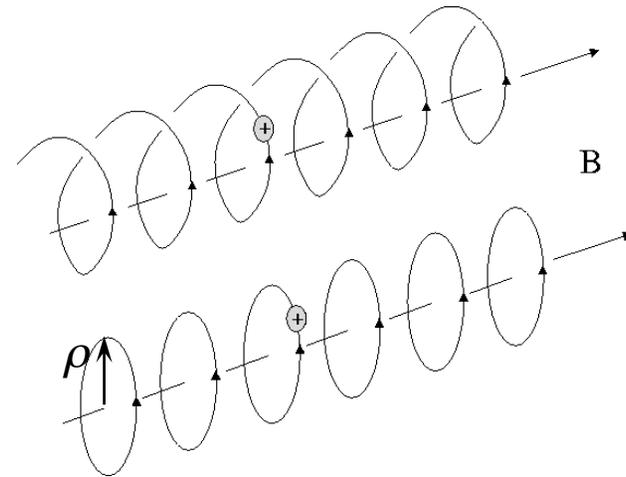


The PIC Steps

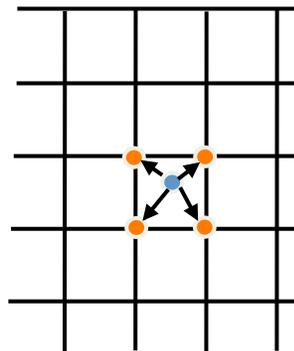
- “**SCATTER**”, or deposit, charges on the grid (nearest neighbors)
- Solve Poisson equation
- “**GATHER**” forces on each particle from potential
- Move particles (**PUSH**)
- Repeat...

Added computational complexity due to “Gyrokinetic” approach

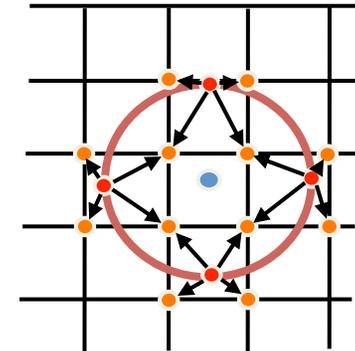
- Fast helical motion of charged particles in strong magnetic field integrated out in gyrokinetic equation
 - Helical motion replaced by moving rings of dynamically changing radius
 - No need to resolve the helical motion = larger time step
- Complicates charge deposition step
 - Random access to memory unless particles are sorted



Charge Deposition Step (SCATTER operation)



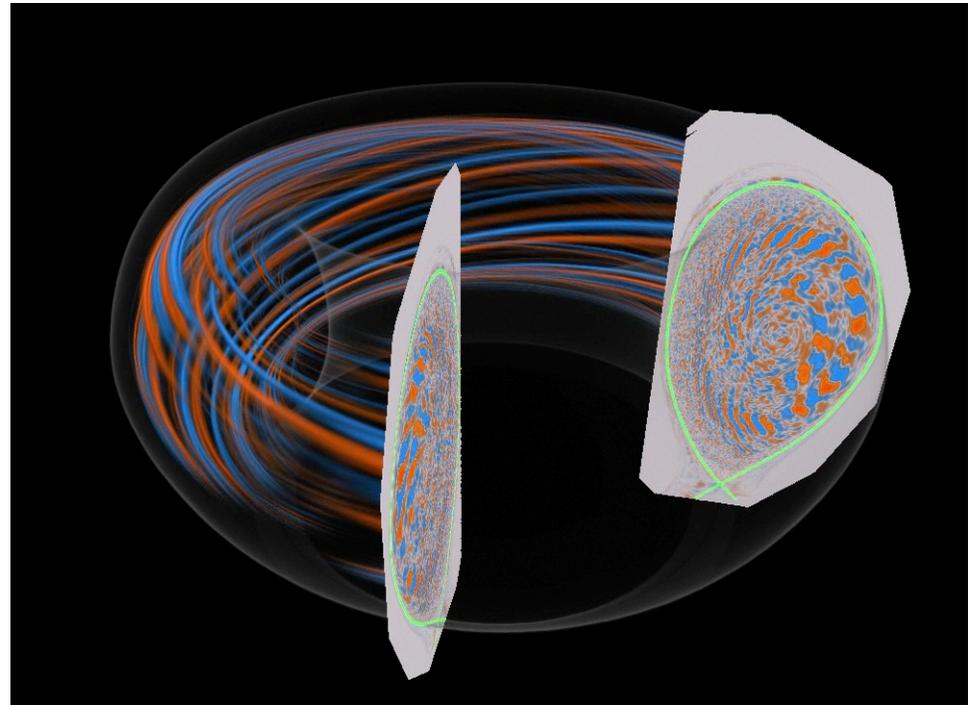
Classic PIC



4-Point Average GK
(W.W. Lee, JCP 1987)

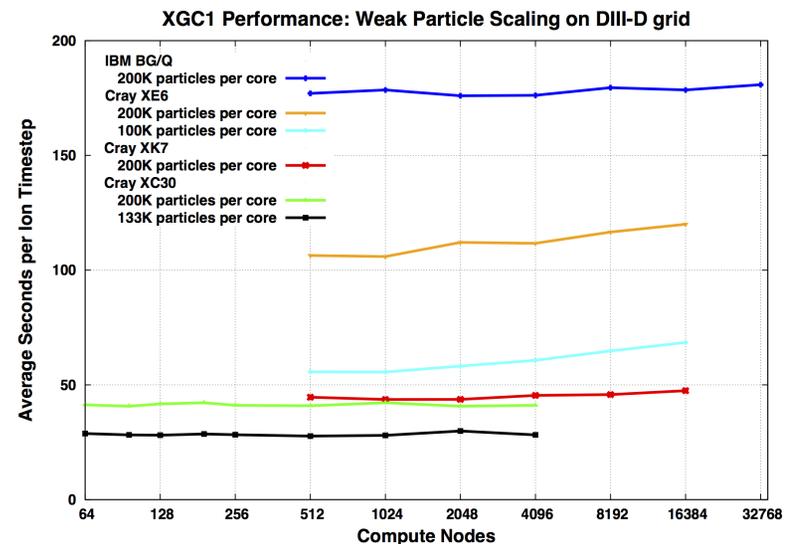
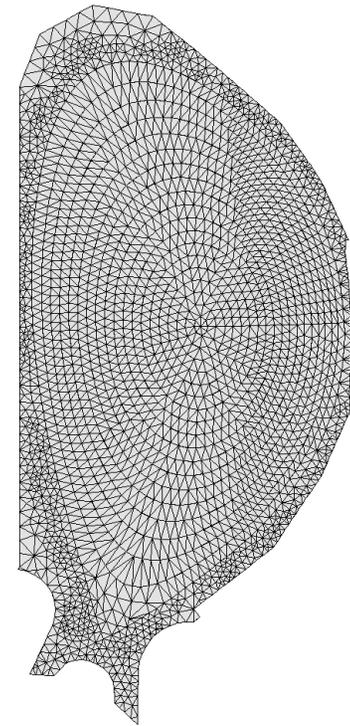
XGC1 code for plasma edge turbulence

- Comprehensive first principles code
- Move particles along the characteristics \rightarrow ODE equation (RK4)
- Solve the self-consistent field \rightarrow PDE equation solved using PETSc (preconditioner Hypre, KSP GMRES)
- Kinetic ions + electrons (real mass ratio)



XGC1's programming models

- MPI + OpenMP + CUDA Fortran for GPU
- Top level = MPI
 - Toroidal domain decomposition
 - Multi-process particle distribution within toroidal domains (require careful load balance)
 - Grid-based solver splits matrices between processes within each toroidal domain (finite element solver), implemented with PETSc library
- Fine-grained OpenMP at loop level
- CUDA Fortran for electron time-advance on GPU



What is an ideal programming model?

- A homogeneous programming model that hides the complexity of multicore hosts, manycore accelerators, and inter-node messaging
 - Nice but... not going to happen, not realistic
- For future systems, non-disruptive programming models with lightweight runtime systems would be preferable
- Some scientists say: Leave it to the application developers to optimize performance. Application developers should not have to fight with or guess the inner workings of programming models in order to write high performance, scalable codes
 - = **TRANSPARENT PROGRAMMING MODEL**
 - Should have the option to access and control low level hardware
- Okay... maybe not entirely. I think that the work on DSLs is very promising for widely-used algorithms (e.g. stencils). Not sure about what they can do for Particle-in-Cell though...

Key new abstractions needed within the programming models to achieve exascale?

- Locality-aware loop partitioning strategies (OpenMP guided not good enough on NUMA nodes)
- Constructs to partition loops and data structures on host and accelerator
- Better thread-processor affinity control
- The ability for programmers to spawn communication threads, computation threads, and helper threads (to help the compute threads with prefetching, etc.)
- Better support for asynchronous IO, asynchronous compute-communicate phases, asynchronous compute-compute phases
- All of the above can be done today with Pthreads but very hard and painful. Source-to-source compilers could take care of implementing this while PM hides the complexity. Don't try to automate everything though. Still want to see the code transformations

Key new abstractions needed within the programming models (continued...)

More specific to global GK PIC code like XGC1:

- Coherent access (or atomic updates) to large working sets
 - All threads within a process may need shared random read-modify-write access to $O(100\text{MB})$, the size of a poloidal plane (PIC “scatter” phase)
 - The biggest gap in existing models is the performance of fine-grained (increment 2 doubles) **atomics or transactions** (the most natural ways of expressing this). This performance deficiency motivates complex decompositions or replications. The former require inter-process communication. The latter require extra memory. Both are unattractive in the manycore limit.
- Dynamic runtime to support computation as directed acyclic graph
 - To tolerate long latencies in data movement, utilize computing resources, overlap data movement, communication, computation (especially important for GPU-like architecture) (Habanero?)

What breakthrough in programming environments is required for exascale?

- Hiding the heterogeneity and memory hierarchy from the programmer as much as possible, while still providing a practical set of abstractions to exploit data locality and key hardware features
- Domain Specific Languages and autotuning are certainly good approaches for dealing with widely used algorithms that are well-known (PDE solvers, FFT spectral solvers, etc.)
 - In the short term we need efficient thread-safe libraries, in particular, MPI library
 - For XGC1, we need robust thread-safe PETSc library for solving multiple independent systems (with profiling turned off please...).

Discuss how your application could utilize task-based and data-driven programming models

- Particle and grid data structure inter-dependencies in XGC1 (and other GK PIC codes, such as GTC and GTS) prevent straightforward use of task-based programming models. Using a task-based/data-driven programming model would probably require re-writing the code from scratch.
- However, some stages of the code could use task-based parallelism, as demonstrated back in 2011 with the particle shifting algorithm in GTS (Preissl et al, SC11)
 - Using a combination of OpenMP tasking and PGAS communications implemented with Fortran co-arrays

How to manage the resiliency challenges in your code?

- Write critical data to local NVRAM (burst buffers) to enable fast restart
- If a code can checkpoint in less than a minute, then it is likely that failure rates of about 1/hr can be endured with little impact
- The programmer needs feedback from the system in order to take action when an error/failure occur
 - Occasional errors affecting a small number of particles is not catastrophic. The code could continue the calculation by simply reinitializing these particles
 - Losing a whole node would certainly require restarting the simulation from a previous time

What to do with periods of idle cores?

- In XGC1 runs we have **>1000 more particles than grid points**
- During computations involving only grid data (e.g. field solver) it is not efficient to engage all the cores. What to do?
 - Power them down to save energy?
 - Give them other tasks to do, such as diagnostics, analysis, rendering for visualization, etc?
 - Do some I/O, such as checkpointing?
- Application developers will need easy ways to implement these possibilities (ability to spawn helper threads)
 - Give hints to OS that now is a good time to power down some cores
 - Easy way to separate groups of threads to work on different tasks
 - Give external applications access to main simulation data (shared space?)

Example: GoldRush (presented at SC13)

GoldRush: Resource Efficient In Situ Scientific Data Analytics Using Fine-Grained Interference Aware Execution

Fang Zheng¹, Hongfeng Yu², Can Hantas¹, Matthew Wolf^{1,3},
Greg Eisenhauer¹, Karsten Schwan¹, Hasan Abbasi³, Scott Klasky³

¹Georgia Institute of Technology

²University of Nebraska Lincoln

³Oak Ridge National Laboratory

ABSTRACT

Severe I/O bottlenecks on High End Computing platforms call for running data analytics in situ. Demonstrating that there exist considerable resources in compute nodes un-used by typical high end scientific simulations, we leverage this fact by creating an agile runtime, termed GoldRush, that can harvest those otherwise wasted, idle resources to efficiently run in situ data analytics. GoldRush uses fine-grained scheduling to “steal” idle resources, in ways that minimize interference between the simulation and in situ analytics. This involves recognizing the potential causes of on-node resource contention and then using scheduling methods that prevent them. Experiments with representative science applications at large scales show that resources harvested on compute nodes can be leveraged to perform useful analytics, significantly improving resource efficiency, reducing data movement costs incurred by alternate solutions, and posing negligible impact on scientific simulations.

generated. Compared to conventional post-processing methods that first write data to storage and then read it back for analysis, in situ analytics can reduce on-machine data movement, disk I/O volume, and deliver faster insights from raw data [2].

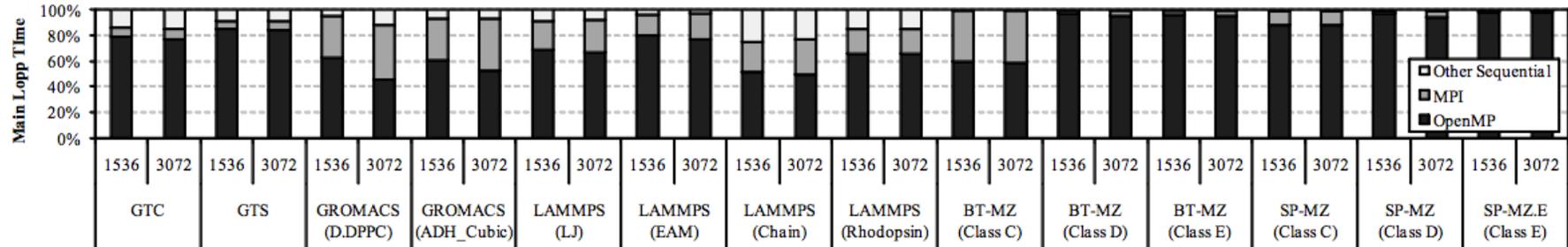
The research presented in this paper has two goals: (1) to improve the resource efficiency of running in situ data analytics, and (2) to do so without perturbing the simulations running on the same nodes. In particular, we seek to over-subscribe compute nodes by co-locating simulation and analytics computations, without affecting the simulation execution, while at the same time, efficiently using compute node resources to run in situ analytics.

Measurements of six representative scientific simulations motivate the argument that node over-subscription can be cost neutral to the core simulation. Specifically, we demonstrate that

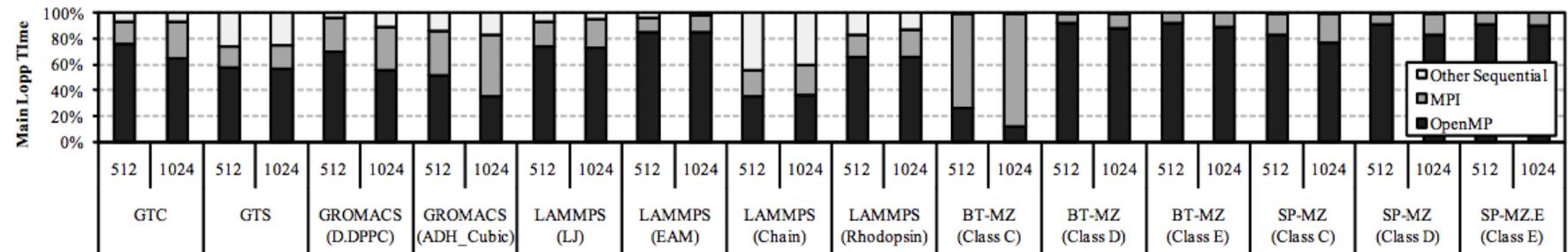
7. ACKNOWLEDGEMENTS

We thank Stéphane Ethier for his help with the GTS application and visual analytics. We also thank the anonymous reviewers and

Plenty of idle cycles in “real” applications, especially with MPI+OpenMP codes (and GPU!)



(a) On Hopper, simulations run on 1056 (256 MPI proc. × 6 OpenMP threads) and 3072 cores (512 MPI proc. × 6 OpenMP threads).

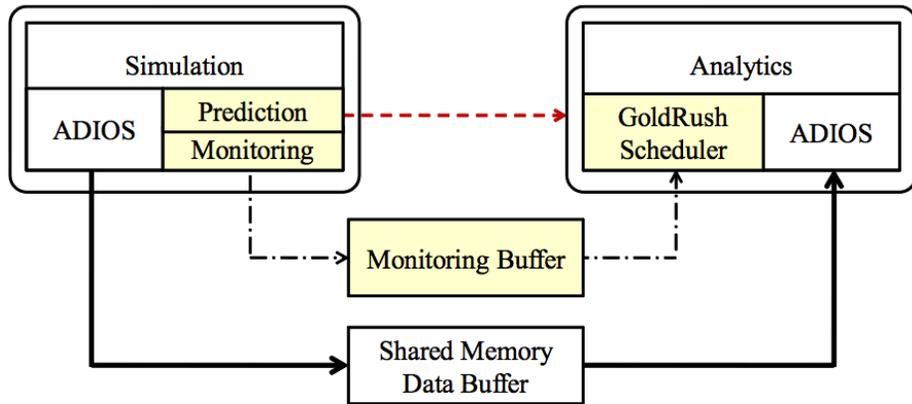


(b) On Smoky, simulations run on 512 (128 MPI proc. × 4 OpenMP threads) and 1024 cores (256 MPI proc. × 4 OpenMP threads).

NOTE: The ratios vary from one system to another!!

- Can we take advantage of this and run analyses on idle cores?
- Is it feasible??

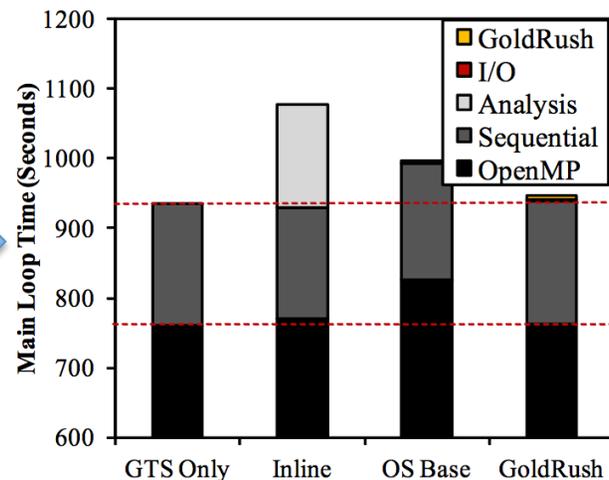
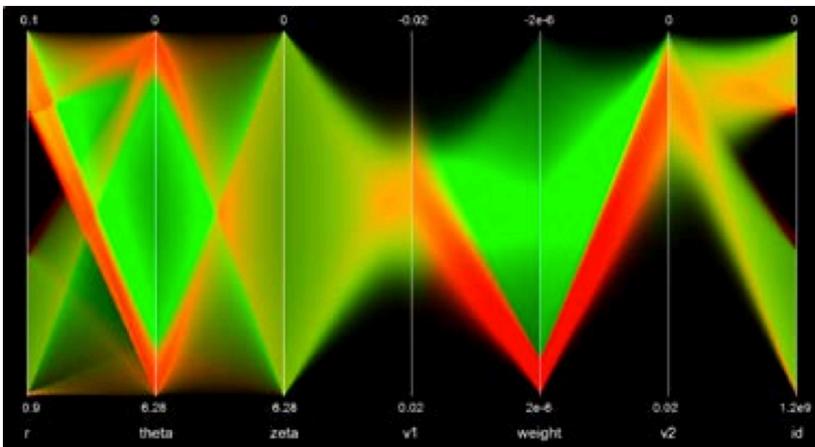
GoldRush: Monitor resources and run analyses on idle OpenMP cores



→ Simulation Output Data -.-> Suspend/Resume Signals -.-> Monitoring Data

- Harvest Idle Resources for In-Situ Analytics
- Dynamically predict idle resource availability
- Reduce interference with execution throttling
- Uses **ADIOS** FlexIO method
- **Overhead of GoldRush < 0.3%**

Particle visualization (parallel coordinates)



Conclusions

- There is a huge amount of parallelism in particle-in-cell codes such as XGC1
- The main challenge is the gather-scatter operations between particles and grid
- Do we need to change the algorithm and get rid of the grid??

Thank you...