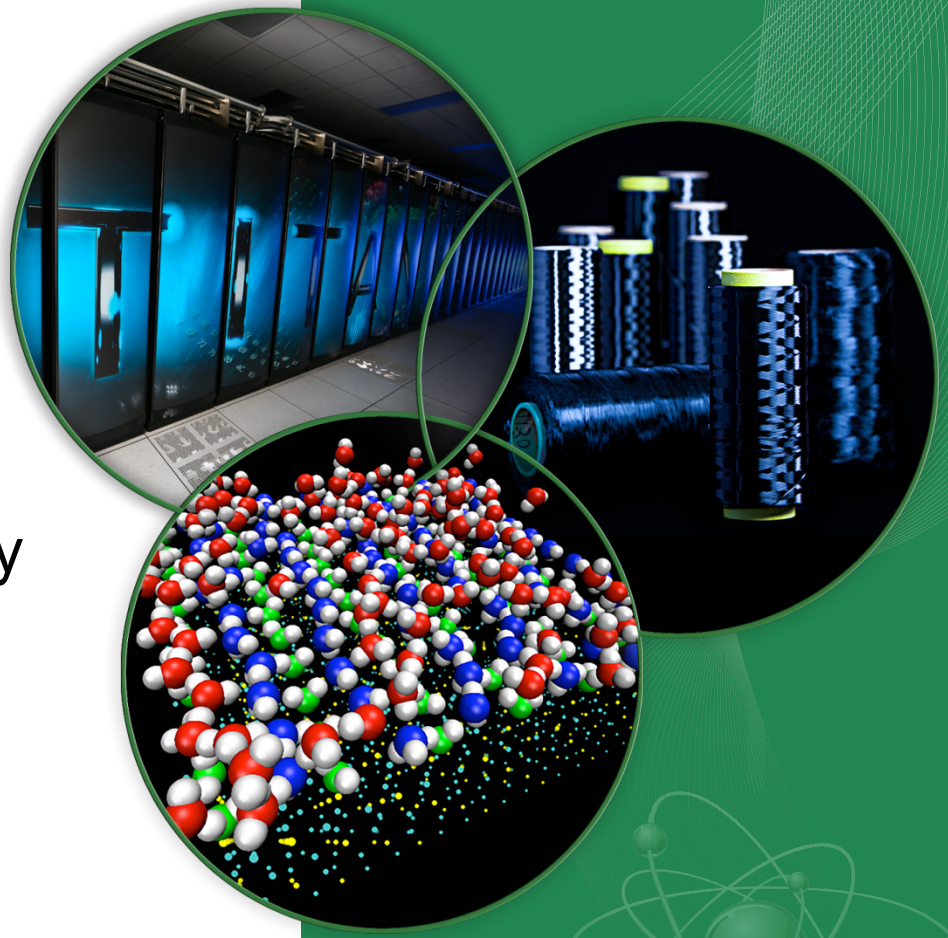
# Enabling Technologies for XGC1 Simulation



E. D'Azevedo (e6d@ornl.gov)

Oak Ridge National Laboratory

OLCF Users Meeting, July 22-24, 2014

ORNL is managed by UT-Battelle
for the US Department of Energy

OAK RIDGE National Laboratory

# EPSI Team

- SciDAC-3 Center for Edge Physics Simulation
- Physics Team:
  - PPPL (C. S. Chang, S-H. Ku, J. Lang, R. Hager, S. Ethier )
  - University of Colorado (S. Parker)
  - Lehigh University (A. Kritz)
  - MIT (M. Greenwald, L. Sugiyama)
  - UC San Diego (G. Tynan)
- Collaboration with SciDAC Institutes:
  - SUPER (P. Worley, E. D'Azevedo),
  - FASTMath (M. Shephard, M. Adams),
  - SDAV (S.  Klasky, M. Parashar),
  - QUEST (R. Moser)

# Outline

- Background on the computational characteristics of XGC1
- Computational challenges in using leadership computing resources
- Future challenges
  - Software and performance portability
  - Balancing computation load



OAK RIDGE National Laboratory

# Particle-in-cell Method

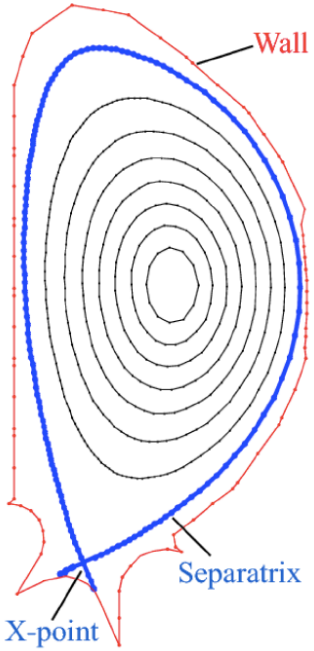
- Particles are used to represent high dimensional distribution function $f(\vec{x},\vec{v},t)$
- Charged particle interact via a potential computed on a background grid
- PIC steps:
  - Deposit charges on grid
  - Solve elliptic equation to obtain electro-magnetic potential
  - Push particle to follow trajectories using forces computed from background potential
  - Account for collision and boundary effects on velocity grid

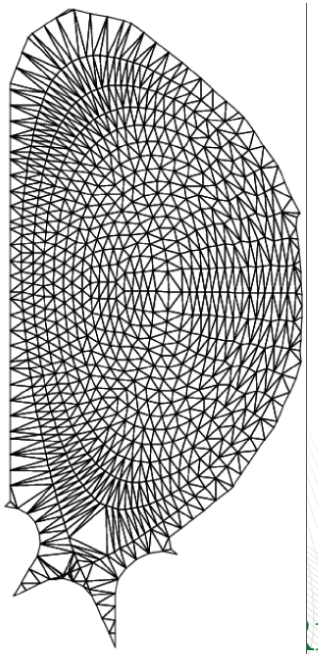


OAK RIDGE National Laboratory

# XGC1

- “Full-f” kinetic simulation to model edge plasma
- Complicated edge geometry and separatrix demands unstructured triangular grid
- Considers ion, electron, and neutral particles
- Nonlinear Fokker-Planck-Landau collision operator
- Multi-scale physics and transport, option to couple to simplified model (XGCa) in whole tokamak volume



# XGC1 – domain decomposition

- Torus divided by computational planes
- Multiple MPI tasks may share a toroidal section
- Identical unstructured triangular mesh in each plane, mesh vertices obtained by following magnetic field lines. Upgraded mesh generator provided by FASTMath greatly reduced the time to generate large meshes (from hours to minutes)
- Mesh used for charge deposition and FEM discretization in Poisson solver
- Sparse linear system solved by PETSc, work increases with grid resolution (independent 2D solves with finite difference across planes)



OAK RIDGE National Laboratory

# XGC1 – charge deposition

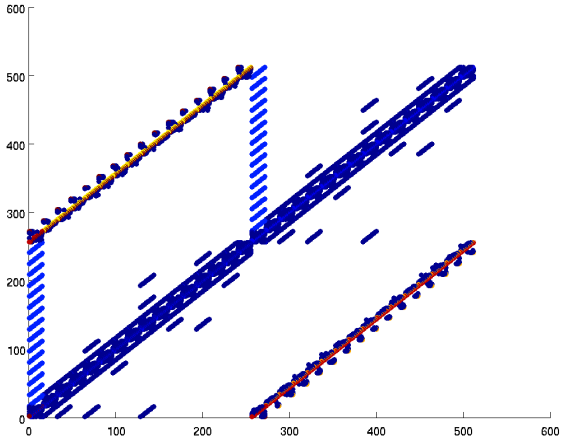
- Need to avoid race condition or memory “collision” in updating grid data when using multiple threads
- Locks may be expensive
- Replicate grid data when using small number of threads, costly in memory use
- GPU device has thousands of threads, use atomic operations on replicated arrays



OAK RIDGE National Laboratory

# XGC1

- Option to periodically sort particles by geometric hashing/bining to improve locality and data reuse. However, sorting and data rearrangement is also expensive.
- Parallel I/O, checkpoint and restart facilitated by ADIOS
- Option of plane major ordering to map MPI tasks to compute nodes. Current work with SUPER to consider communication pattern and network topology in assigning MPI tasks to compute nodes




Matrix of communication volume



OAK RIDGE National Laboratory

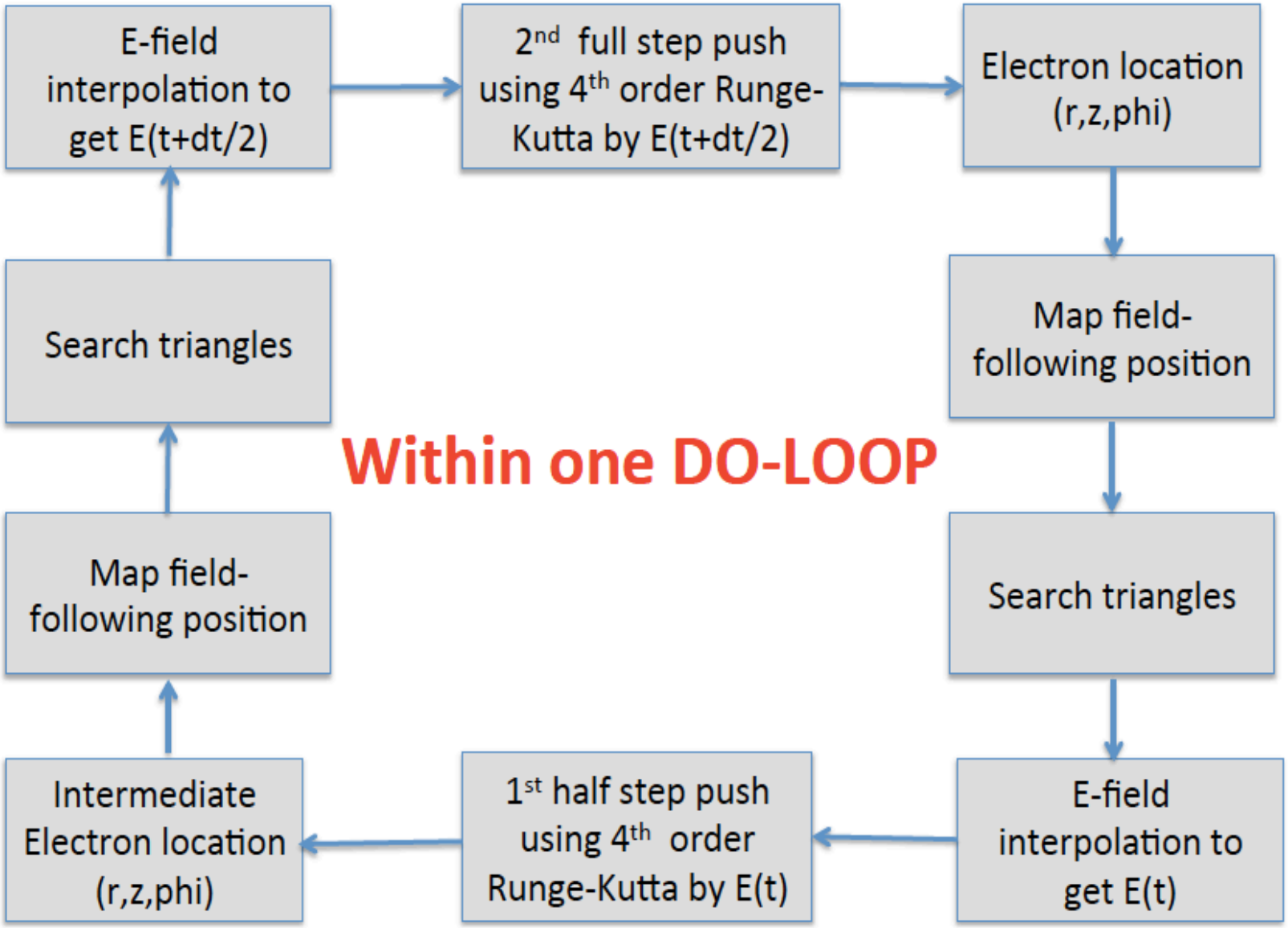
# XGC1 – particle push

- Follow independent trajectories of particles (initial value ODE)
- About 10,000 particles per cell -> tens of billions of particles
- Push kernel accounts for significant (~ 80%) of time, work scales by number of particles
- Pass off work of pushing ion particles  to neighbor domain.
- Electrons have lighter mass than ions – thus has higher velocity – traverse more domains
- Replicate field information on all processors, then push electron particles without frequent communication (on GPU)



OAK RIDGE National Laboratory

# XGC1 – particle push (2)



# XGC1 – particle push (3)

- Electron push performed on *both* GPU and CPU. Ratio determined by input (~70% work to GPU) on Titan, to balance work load between CPU and GPU
- Electron particles may traverse ~1/2 of device in one ion time step (electron subcycling)
- Replicating field data is expensive in communication cost and in device memory
- Option to replicate potential and recompute electric field (gradient of potential) as needed. Trade-off memory vs computation
- Ideally, organize data on host but device can access host data in unified manner



OAK RIDGE National Laboratory

# Software and performance portability

- Variety of hybrid architectures:
  - Effective use of multi-core CPU (64 threads on Mira) using nested OpenMP
  - Thread-safe library and solver (pspline, PETSc)
  - Partition work on both GPU and CPUs
- But:
  - need PGI CUDA Fortran
  - Port separate version of PETSc and pspline
  - Software challenge in maintaining two versions of similar codes (for CPU and GPU)



OAK RIDGE National Laboratory

# PGI CUDA Fortran

- Write CUDA code in Fortran, add attributes(device), attributes(global)
- *Allow deeply nested subroutines (in same module)*
- One way to port Fortran code to GPU
- Access to atomic intrinsic functions
- Support for structures, array notation, limited I/O
- Explicit memory management on device
- Duplication of code and data



OAK RIDGE National Laboratory

# Example of kernel code

```
module mathOps
contains
  attributes(global) subroutine saxpy(x, y, a)
    implicit none
    real :: x(:), y(:)
    real, value :: a
    integer :: i, n
    n = size(x)
    i = blockDim%x * (blockIdx%x - 1) + threadIdx%x
    if (i <= n) y(i) = y(i) + a*x(i)
  end subroutine saxpy
end module mathOps
```

attributes(global) for device kernel

Map thread number to array index

http://devblogs.nvidia.com/parallelforall/easy-introduction-cuda-fortran/



OAK RIDGE National Laboratory

# Example of calling kernel code

```
program testSaxpy
  use mathOps
  use cudafor
  implicit none
  integer, parameter :: N = 40000
  real :: x(N), y(N), a
  real, device :: x_d(N), y_d(N)
  type(dim3) :: grid, tBlock

  tBlock = dim3(256,1,1)
  grid = dim3(ceiling(real(N)/tBlock%x),1,1)

  x = 1.0; y = 2.0; a = 2.0
  x_d = x
  y_d = y
  call saxpy<<<grid, tBlock>>>(x_d, y_d, a)
  y = y_d
  write(*,*) 'Max error: ', maxval(abs(y-4.0))
end program testSaxpy
```

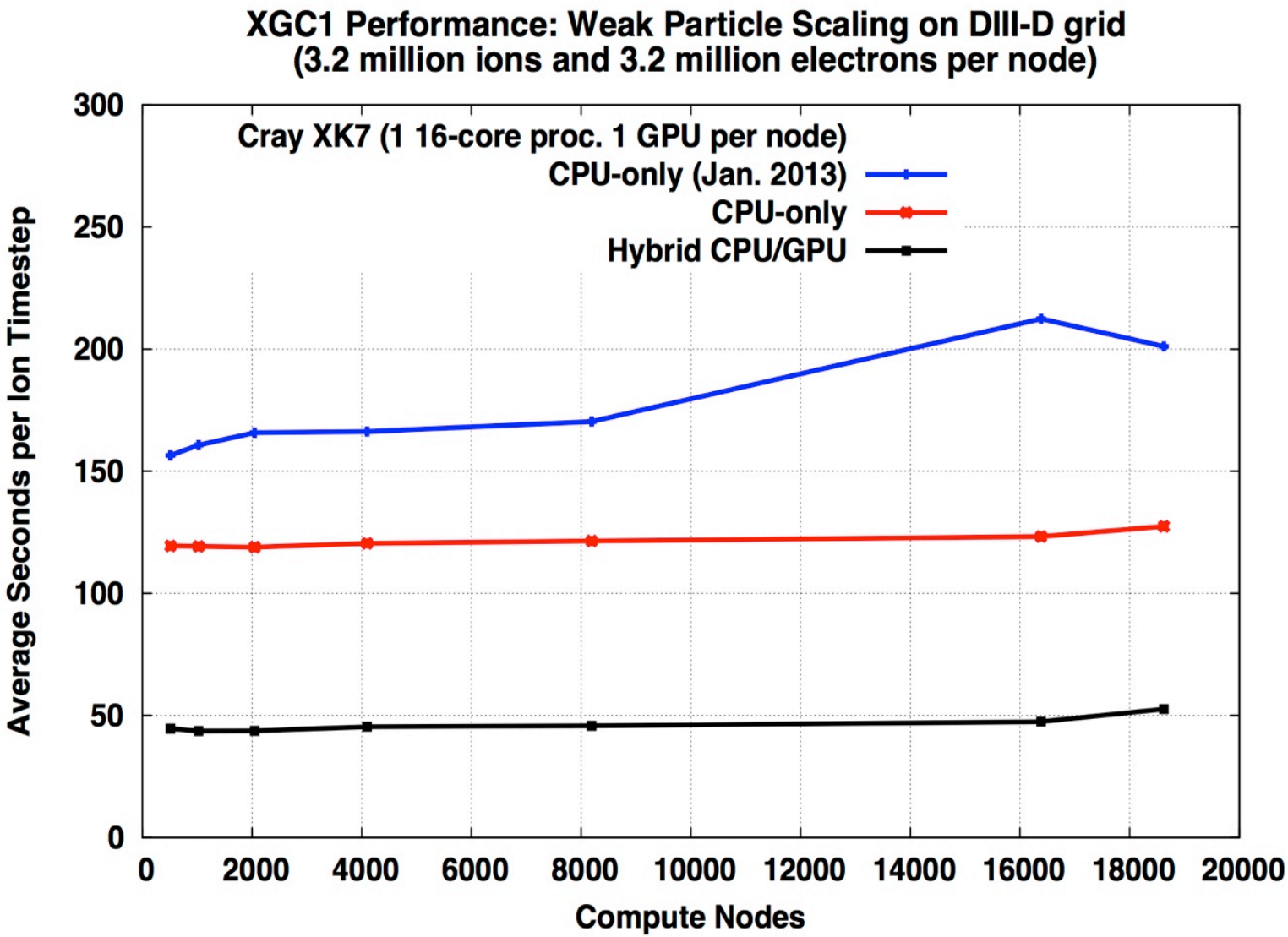
Storage on device

Data transfer by assignment



OAK RIDGE National Laboratory

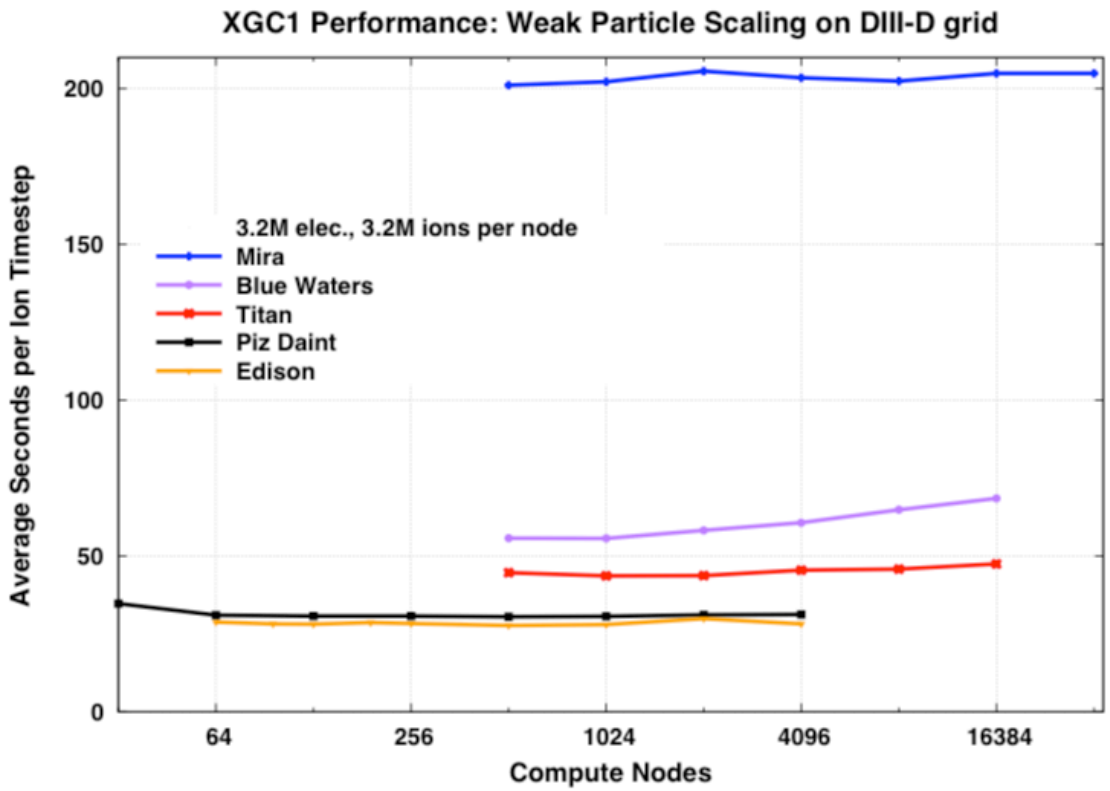
# Effective use of GPU and CPU






OAK RIDGE National Laboratory

# Porting XGC1 to leadership computing resources




- Mira: BG/Q, 16 cores, total 64 hardware threads
- Blue Waters: two 16-core AMD per node (total 32 cores)
- Titan: Kepler GPU + 16-core AMD
- Piz Daint: Kepler GPU + 8-core Intel Sandy Bridge (16 cores with hyper-threading)
- Edison: 12-core Intel Ivy Bridge (24 cores with hyper-threading)



OAK RIDGE National Laboratory

# Nonlinear Collision Operator

| | Main loop time | Collision routine time |
|---|---|---|
| Multi threaded collision | 200 s | 32 s |
| Single threaded collision | 340 s | 171 s |

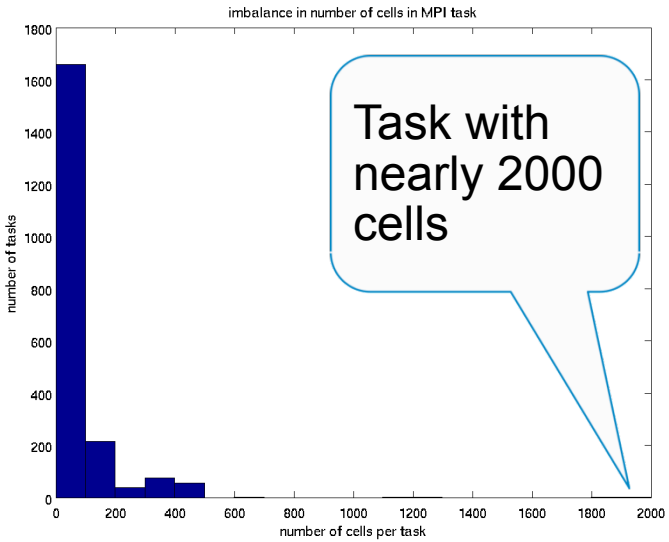
- Initial implementation of new collision operator nearly doubled total compute time
- In each grid cell, recover distribution on 2D velocity phase grid, solve nonlinear system by Picard iteration, calls to elliptic functions to construct matrix, sparse linear solves in each Picard iteration
- Need thread-safe spline and PETSc solver
- Nested OpenMP parallelization across grid cells
- Collision solver accelerated by 5X, XGC1 by 1.7X
- Challenge in balancing work related to particle work and grid work

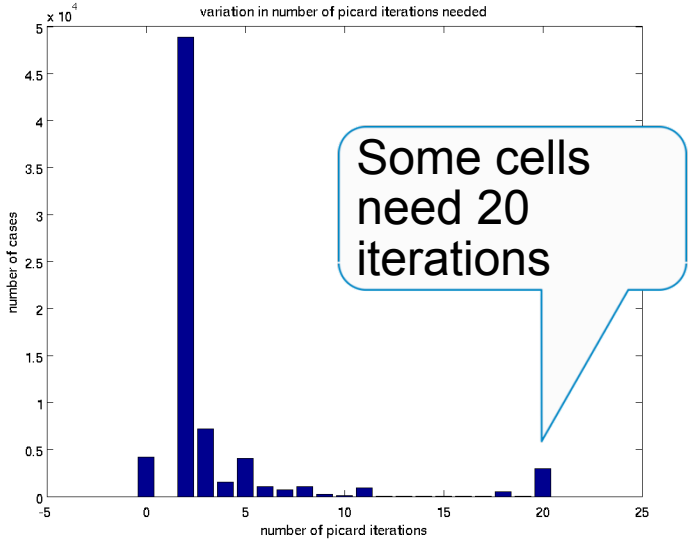


OAK RIDGE National Laboratory

# Imbalance in work load

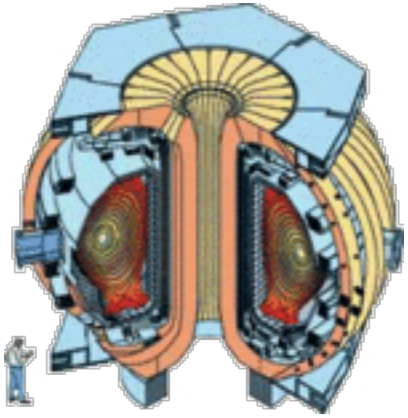
- Partitioning to balance particles
- A few MPI tasks are assigned many grid cells (over 500 cells), but average is around 42
- Work in collision operator related to number of cells
- Most cells convergence within 5 iterations, but some take up to the maximum 20 iterations

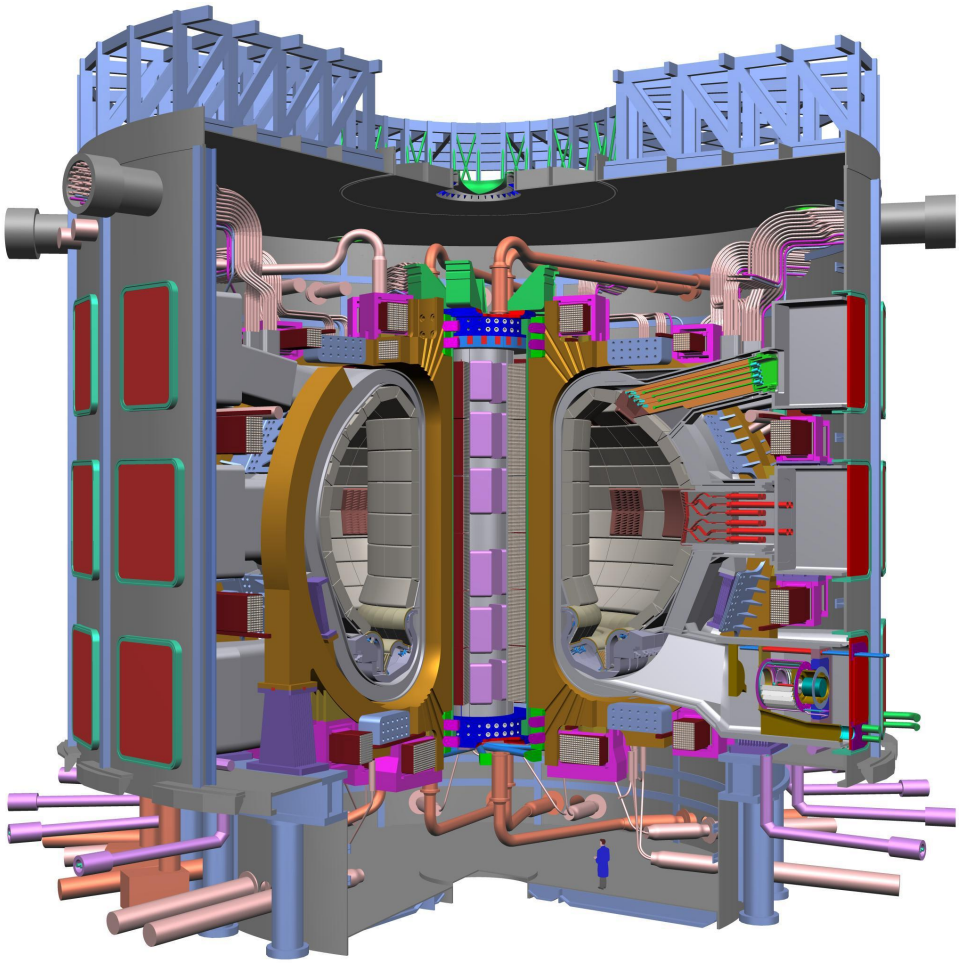


# Simulation of ITER

- ITER is larger than current experiments
- Need advanced simulation to predict performance of ITER







OAK RIDGE National Laboratory

# ITER Challenges

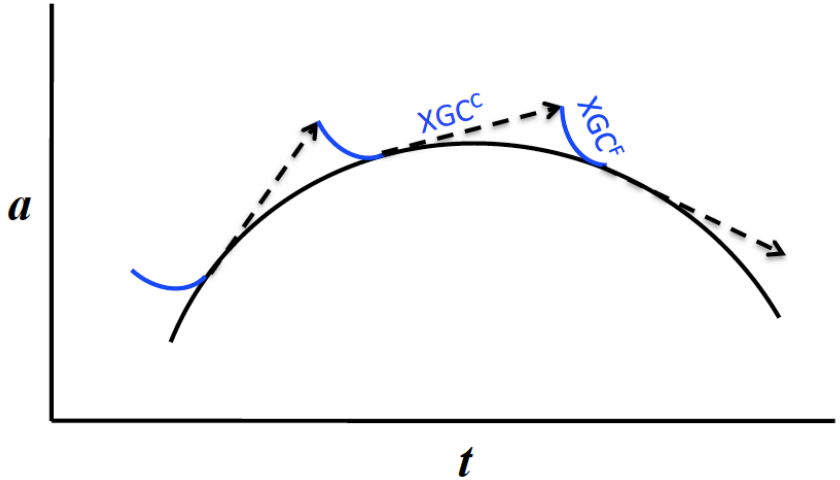
- Higher resolution ITER Grid:
  - 3mm grid for D3D 112,655 elements
  - 4mm grid for ITER will need over *10X* more elements
  - More work in collision operator
  - More total particles (keeping 10,000 particles per cell)
  - Higher cost in communication and memory to replicate electro-magnetic field on all processors
- Higher fidelity in physics:
  - Sub mm grid (in edge region) for electron scale turbulence, can lead to 100X more computation
  - 3D nonlinear EM solver
  - Longer time simulation 10ms -> 10s using multi-scale time integration

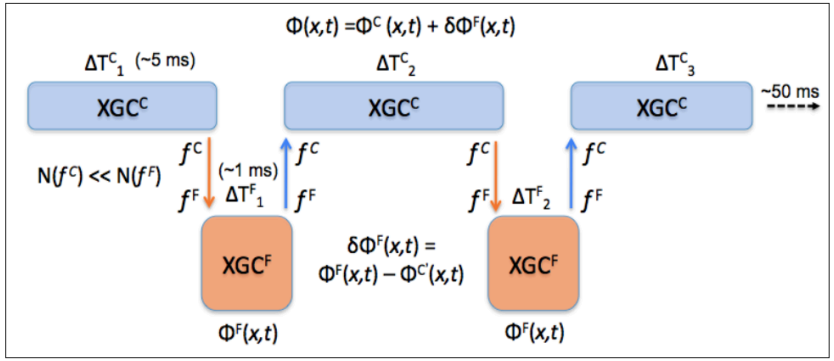


OAK RIDGE National Laboratory

# Multi-scale coupling

- Goal to significantly increase simulation time from 10's milli-seconds to 10's seconds
- Data exchange between high resolution XGC1, faster simplified models XGCa
- DataSpaces (SDAV) to use on-node memory for coupling and RAM in dedicated nodes for data analysis



# Uncertainty Quantification

- Uncertainty Quantification (UQ) at extreme scale is too expensive
- Collaboration with QUEST to reduce number of UQ parameters. Consider the few parameters used in experiments
- “ab initio” code has fewer free parameters, e.g. compute diffusion coefficients directly
- Study at reduced scale then project to full scale



OAK RIDGE National Laboratory

# Software Challenges

- Hybrid architecture:
  - Massively parallel: vectorization and high number of threads (64 threads on Mira)
  - Accelerators: Nvidia Kepler GPU on Titan, future Intel MIC (self-hosted Knights Landing) on NERSC-8 Cori
  - Complicated hierarchy of memory (host memory, device memory, shared cache)
- Different programming models:
  - PGI CUDA Fortran (what works currently)
  - OpenACC (not supported by Intel)
  - Intel offload directives
  - OpenMP target directive for accelerators
- Solver Libraries



OAK RIDGE National Laboratory

# Summary of Challenges

- Software portability and performance portability in using hybrid MPI + OpenMP + directive as programming model
- Simplified memory management between host and accelerator device
- Balancing computation load
- On-memory data management at extreme scale
- UQ at extreme scale
- Thread-safe parallel libraries for extreme scale
- *More computation power -> room for more physics*



OAK RIDGE National Laboratory

# Acknowledgements


- The work was partially performed at ORNL, which is managed by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725; at LBNL under DE-AC02-05CH11231, and at PPPL under DE-AC02-09CH11466
- Awards of computer time was provided by the Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program.
- This work was supported by a grant from the Swiss National Supercomputing Centre (CSCS) under project ID g40




OAK RIDGE National Laboratory

# THANK YOU



OAK RIDGE National Laboratory

# Nvidia GPU

- Massively parallel: blocks of threads (1024 threads per block), grid of blocks ( 64K by 64K)
- Peak 1.3 Tflops/sec for Kepler K20X
- 32 threads per wrap in SIMD, avoid code divergence
- High memory bandwidth (250GB/s) but high latency (100s of cycles): need lots of threads to hide memory latency
- Limited amount of device memory, 6GB on device, 32GB on compute node
- **Slow** data transfers between CPU host & GPU device (effectively ~5GB/s)



OAK RIDGE National Laboratory

# OpenACC

- Compiler directives to perform data transfer and computation, supported in PGI, Cray, CAPS (but not Intel) compilers but version 1.0 was too restrictive
- New capabilities in Version 2.0:
  - device subroutine
  - calling nested device subroutines
  - Flexibility in data allocation and transfer
- Need “deep copy” or “smart copy” data structure
- Exercise in porting collision operator to GPU using OpenACC but compiler tools still need improvement
- Can mix OpenACC with CUDA Fortan using PGI compiler



OAK RIDGE National Laboratory

# Examples of directives

```
! Sum component wise and save result into vector c, perform reduction sum on c
```

```
!$acc kernels !copyin(a(1:n),b(1:n)), copyout(c(1:n))
do i=1,n
    c(i) = a(i) + b(i)
    sum = sum +  c(i)
enddo
!$acc end kernels
```



OAK RIDGE National Laboratory