XGC1: Performance on the 8-core and 12-core Cray XT5 Systems at ORNL

Patrick Worley	Oak Ridge National Laboratory
Mark Adams	Columbia University
Eduardo D'Azevedo	Oak Ridge National Laboratory
C-S Chang	New York University
Seung-Hoe Ku	New York University
Collin McCurdy	Oak Ridge National Laboratory

CUG 2010

May 27, 2009

Apex Waterloo Place Hotel

Edinburgh, Scotland

OAK RIDGE National Laboratory

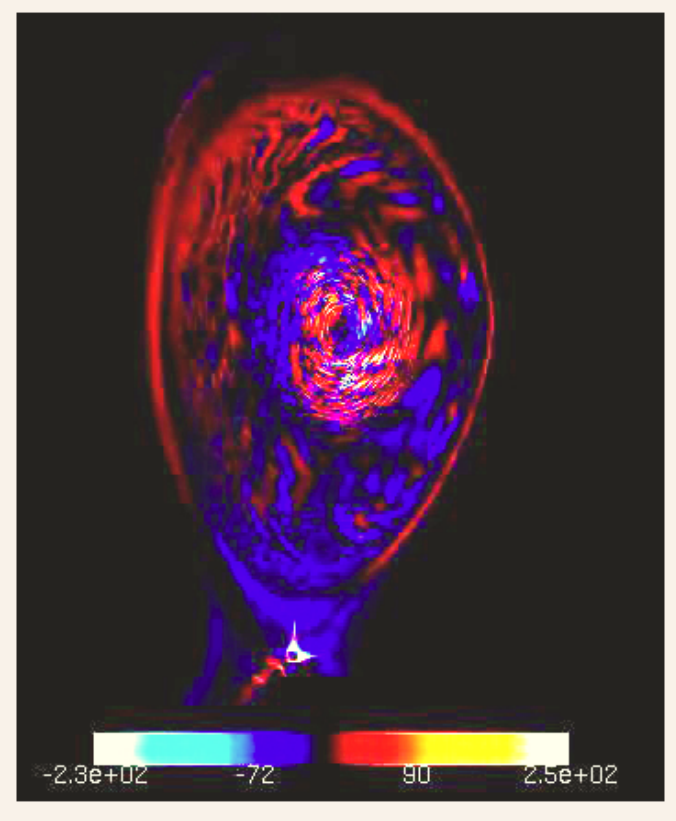
Office of Science U.S. DEPARTMENT OF ENERGY

Acknowledgements

- Research sponsored by the Office of Fusion Energy Sciences and by the Office of Advanced Scientific Computing Research, both in the Office of Science, U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
- This research used resources (Cray XT5) of the National Center for Computational Sciences at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.
- These slides have been authored by a contractor of the U.S. Government under contract No. DE-AC05-00OR22725. Accordingly, the U.S. Government retains a nonexclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.

XGC1: First full-f gyrokinetic simulation of whole device tokamak plasma

XGC-1 is a particle-in-cell code used to study turbulent transport in magnetic confinement fusion plasmas. It uses an unstructured grid, allowing it to treat the edge region in tokamak devices accurately. It uses PETSc to solve an elliptic problem at each timestep. Performance experiments are typically weak scaling in total particle count and strong scaling in grid size.



Turbulent eddies on the whole poloidal cross-sectional plane.

ITER
(*a Tokamak Magnetic Confinement Fusion Device*)

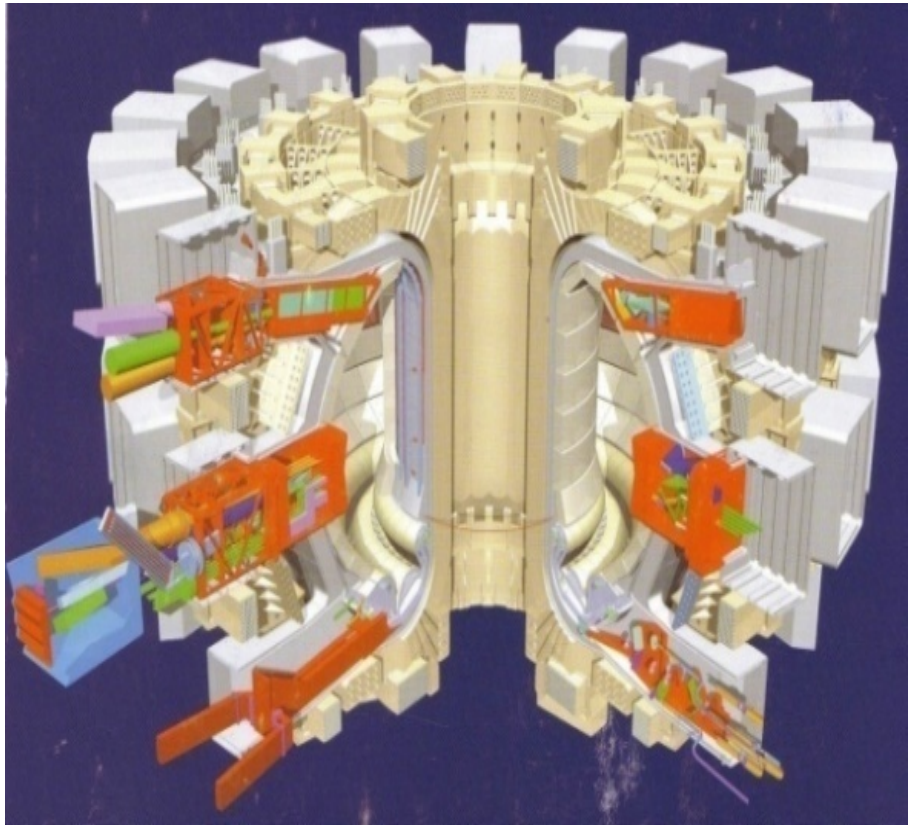


Fig. 8. Schematic of the ITER tokamak, where the first wall of the innermost structure of the device is shown, with the divertor chamber at the bottom. For more information, see www.iter.org.

XGC1: Timestep Logic

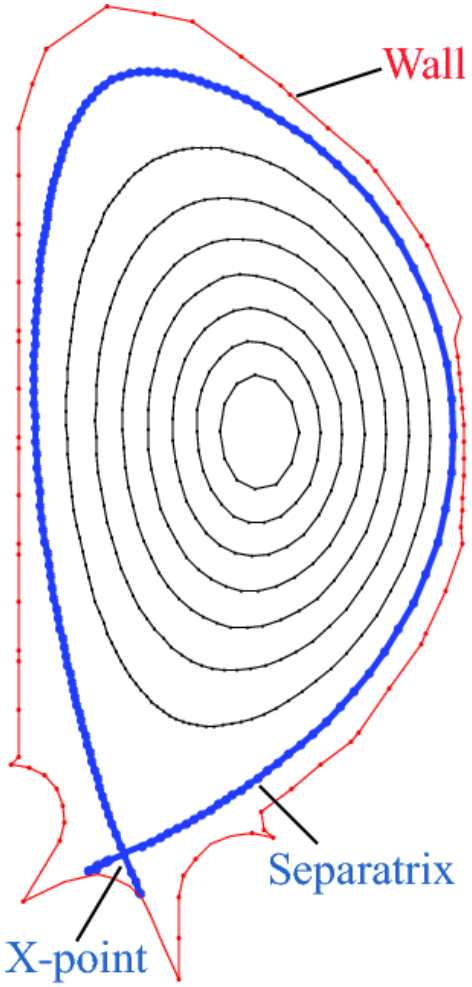
XGC1 solves the gyrokinetic Vlasov equations with marker particles and electric field data on a spatial grid. Each timestep (or each step within a Runge-Kutta or predictor-corrector time-integration method) looks something like:

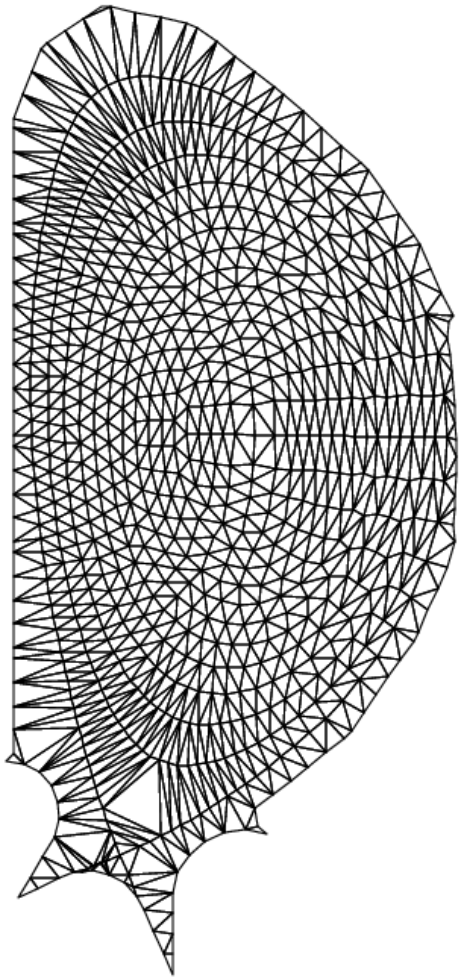
1. Collect particle charge density of underlying grid.
2. Solve gyrokinetic Poisson equation on grid.
3. Compute electric field and any derivatives needed in particle equations of motion.
4. Calculate and output diagnostic quantities.
5. Update particle positions and velocities.

Depending on the experiment, particles can be ions, electrons, or both. Experiments can also include collisions and other physical processes.

Geometry and Sample Mesh

Magnetic flux surfaces of diverted tokamak geometry in a poloidal plane.





A sample unstructured triangular mesh for ITER magnetic field. Actual scale used in simulations is 30 times finer.

OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

XGC1: Parallelization

Parallelization is based on decomposition of both the spatial grid and the particle data across processes. MPI is used to communicate between processes. OpenMP is used to parallelize work within processes.

1. Collect particle charge density of underlying grid.
   - Parallelized loops over particles (MPI and OpenMP).
   - Requires communication between neighboring processes (defined by grid decomposition) and global reductions.
2. Solve gyrokinetic Poisson equation on grid.
   - Parallel conjugate gradient solver accelerated with algebraic multigrid preconditioner (hypre) within PETSc.
   - Other: parallelized loops over grid (MPI and OpenMP).
   - Local and non-local point-to-point and global collective (MPI) communication required.

XGC1: Parallelization

3. Compute electric field and any derivatives needed in particle equations of motion.
   - Parallelized loops over grid (MPI and OpenMP).
   - Requires communication between neighboring processes.
4. Calculate and output diagnostic quantities.
   - Parallelized loops over particles and over grid (MPI and OpenMP).
   - Requires global reductions.
5. Update particle positions and velocities.
   - Parallelized loops over particles (MPI and OpenMP).
   - Requires global collective communication to coordinate moving particles between processes.
   - Requires point-to-point communication between processes to move particles.

Background

1. May-June 2009: Conducted XGC1 performance scalability study in preparation for a series of production runs.
2. March-April 2010: Conducted XGC1 performance scalability study in preparation for a series of production runs.

Both studies

- used a fixed number of particles per thread of computation (weak scaling)
- used a fixed grid independent of number of threads (strong scaling)

where the grid is a 3 mm mesh of the 3-D core of ITER.

May-June 2009

Cray XT5 at ORNL (*JaguarPF*)

- 18,722 compute nodes, 8 processor cores per node, 2 GB memory per core:
  - 149,776 processor cores and 299,552 TB memory
- Compute node contains two 2.3 GHz quad-core Opteron processors (AMD 2356 “Barcelona”) linked with dual HyperTransport connections and DDR2-800 NUMA memory
- 3D Torus (25x32x24) with Cray SeaStar2+ NIC (9.6 GB/s peak bidirectional BW in each of 6 directions; 6 GB/s sustained)
- Version 2.1 of the Cray Linux Environment (CLE) operating system
- Version 3.1.0 of the Cray MPI Library (MPT)
- Version 7.2.5 of PGI Fortran compiler

March-April 2010

Cray XT5 at ORNL (*JaguarPF*)

- 18,688 compute nodes, 12 processor cores per node, 2 GB memory per core:
  - 224,256 processor cores and 299,008 TB memory
- Compute node contains two 2.6 GHz hex-core Opteron processors (AMD 2356 "Istanbul") linked with dual HyperTransport connections and DDR2-800 NUMA memory
- 3D Torus (25x32x24) with Cray SeaStar2+ NIC (9.6 GB/s peak bidirectional BW in each of 6 directions; 6 GB/s sustained)
- Version 2.2 of the Cray Linux Environment (CLE) operating system
- Version 3.5.1 of the Cray MPI Library (MPT)
- Version 9.0.4 of PGI Fortran compiler

"Let's use the two expts. to compare the 8-core and 12-core systems"

Issues:

1. Number of particles per thread was 900,000 in 2009 and only 300,000 in 2010. This is a big difference. Solution:
   - Use both 300,000 and 900,000 in 2010 experiments.
2. Problem size (in total number of particles) is a function of the number of threads, introducing ambiguity in comparison. Approach:
   - Use only 8 cores per node and same node counts. Wasting 33% of cores in 2010 expts., but problem sizes, core counts, and memory requirements per node are the same.
   - Use all cores in node but adjust number of nodes in 2010 expts. so that same problem sizes and core counts are examined. Memory requirements per node are 50% larger in 2010 expts.
   - Use all cores in node and same node counts. Problem sizes, core counts, and memory requirements per node are 50% larger in 2010 expts.

"Let's use the two expts. to compare the 8-core and 12-core systems"

Issues:

3. XGC1 code has continued to evolve, with some changes affecting performance. Approach:
   - Back out as many performance-sensitive changes as possible, including: a more efficient search technique for locating particle position in grid; removing array syntax within OpenMP-parallelized loops that was degrading OpenMP performance; thread-safe random number generation; spline interpolation optimizations (experiment **A**)
   - Back out only the spline interpolation optimizations (experiment **B**)
   - Run with the current version (experiment **C**)

   Then attempt to verify that experiment **A** does represent how the June 2009 version of XGC1 would have run on the 2010 version of *JaguarPF*.

Results: Wallclock Time

900K particles/thread, 8 cores per node, MPI-only

		Seconds for 10 timesteps			
		2009	2010		
Nodes	Cores		A	B	C
512	4096	448	410	396	267
1024	8192	460	418	402	275
2048	16384	463	442	425	290
4096	32768	465	441	432	-
8192	65536	464	455	423	294
16384	131072	-	453	423	-
18624	148992	OOM	448	403	-

- Reasonable scaling for all experiments.
- 2010(A) vs. 2010(B): most of difference due to improved search algorithm
- 2.6/2.3=1.13; 2009 vs. 2010(A): 9%=>2%; vs. 2010(B): 13%=>10%

Results: Wallclock Time

900K particles/thread, 8 cores per node, 4-way OpenMP

		Seconds for 10 timesteps			
		2009	2010		
Nodes	Cores		A	B	C
512	4096	435	416	373	252
1024	8192	417	428	372	252
2048	16384	415	422	372	251
4096	32768	421	472	378	252
8192	65536	425	494	385	263
16384	131072	435	516	387	264
18624	148992	433	524	382	263

- Scaling and performance improved, except for 2010(A).
- 2010(A): “old” search alg. in OpenMP loop introduced significant load imb.
- 2009 experiments do not show behavior similar to 2010(A).

Results: Wallclock Time

900K particles/thread, 8 cores per node, 8-way OpenMP

		Seconds for 10 timesteps			
		2009	2010		
Nodes	Cores		A	B	C
512	4096	540	545	431	320
1024	8192	499	524	406	289
2048	16384	477	530	401	281
4096	32768	473	535	401	283
8192	65536	473	581	403	288
16384	131072	482	606	412	293
18624	148992	484	616	411	289

- Scaling good, except for 2010(A). Performance inferior to 4-way OpenMP.
- 2010(A): same problem, though relative degradation not as great.

Results: Wallclock Time

900K particles/thread, all cores per node, 4-way OpenMP

	Seconds for 10 timesteps			
	2009	2010		
Cores		A	B	C
4096	435	480	410	300
8192	417	471	390	276
16384	415	457	390	272
32768	421	491	399	276
65536	425	504	401	-
131072	435	532	401	273

- 2 MPI processes per node for 2009; 3 MPI processes per node for 2010.
- Relative to 8-core expts., 12-core expts. include increased contention for memory and for access to network, and possible inefficiencies in cross-socket OpenMP thread placement. Performance is degraded < 10%.

Results: Wallclock Time

900K particles/thread, all cores per node, 2 MPI processes per node

	Seconds for 10 timesteps			
	2009	2010		
Nodes		A	B	C
512	435	462	391	272
1024	417	485	386	269
2048	415	494	391	272
4096	421	547	401	280
8192	425	597	405	285
16384	435	617	423	288
18624	433	593	408	284

- 4-way OpenMP for 2009; 6-way OpenMP for 2010.
- 2010 all-core 4-way OpenMP results appear to be better than the 6-way performance (*an unexpected result*)

Results: Wallclock Time

300K particles/thread, 12 cores per node, 2010(C) expts. only

		Seconds for 10 timesteps (threads per process)		
Nodes	Cores	1	6	12
512	6144	186	106	138
1024	12288	204	106	123
2048	24576	232	109	120
4096	49152	-	115	121
8192	98304	-	117	126
12288	147456	-	117	140
16384	196608	-	117	134
18624	223488	-	118	131

- MPI-only not scaling well; 6-way OpenMP best performer.
- Did not try 4-way OpenMP in these experiments.

Results: Wallclock Time

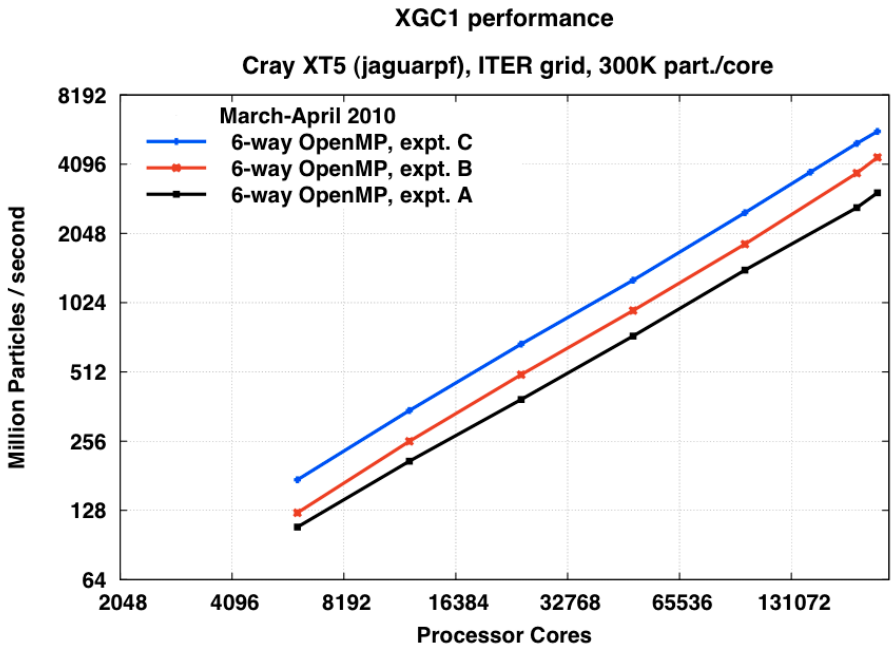
900K particles/thread, 12 cores per node, 2010(C) expts. only

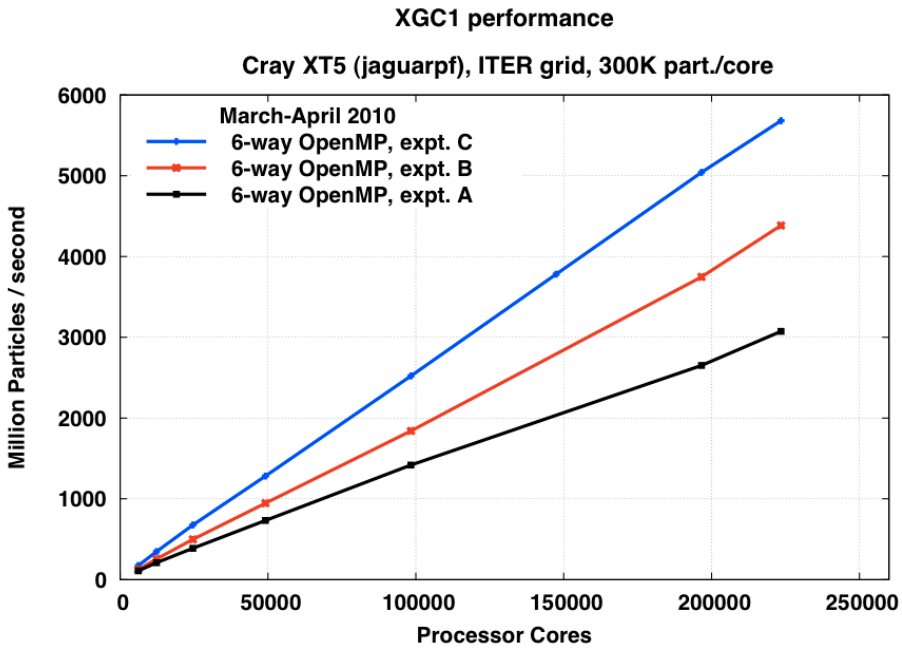
		Seconds for 10 timesteps (threads per process)			
Nodes	Cores	1	6	12	4
512	6144	OOM	272	352	
1024	12288	OOM	269	324	
1366 16384					272
2048	24576	OOM	272	318	
2731 32768					276
4096	49152	OOM	280	321	
8192	98304	OOM	285	326	
10923 131072					273
12288	147456	OOM	283	344	
16384	196608	OOM	288	338	
18624	223488	OOM	284	330	

- MPI-only could not run; 6-way OpenMP better than 12-way.
- Performance of 4-way OpenMP appears to be competitive with 6-way.

Results: Particle Push Rate

300K particles/thread, 12 cores per node, 2 MPI processes per node

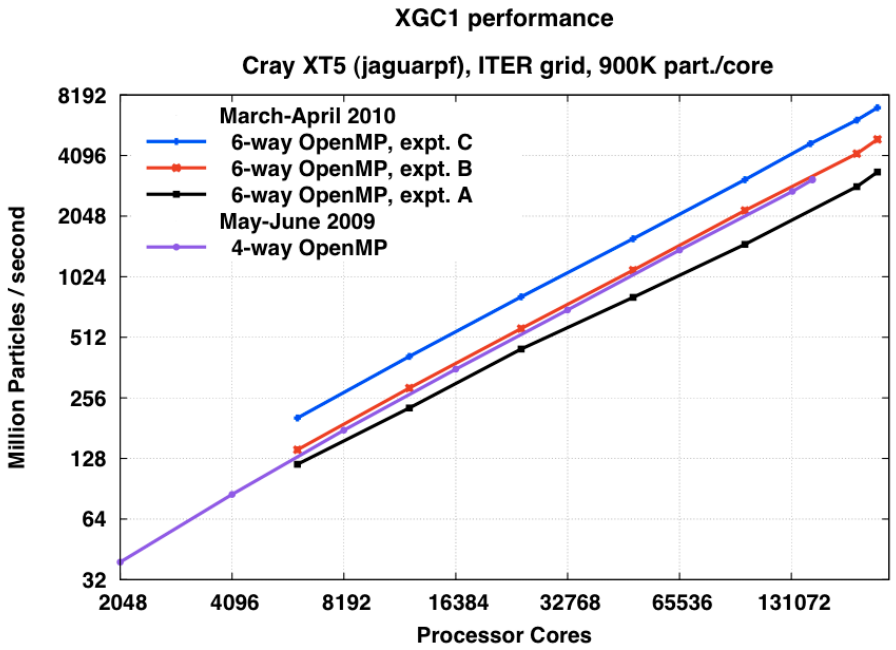


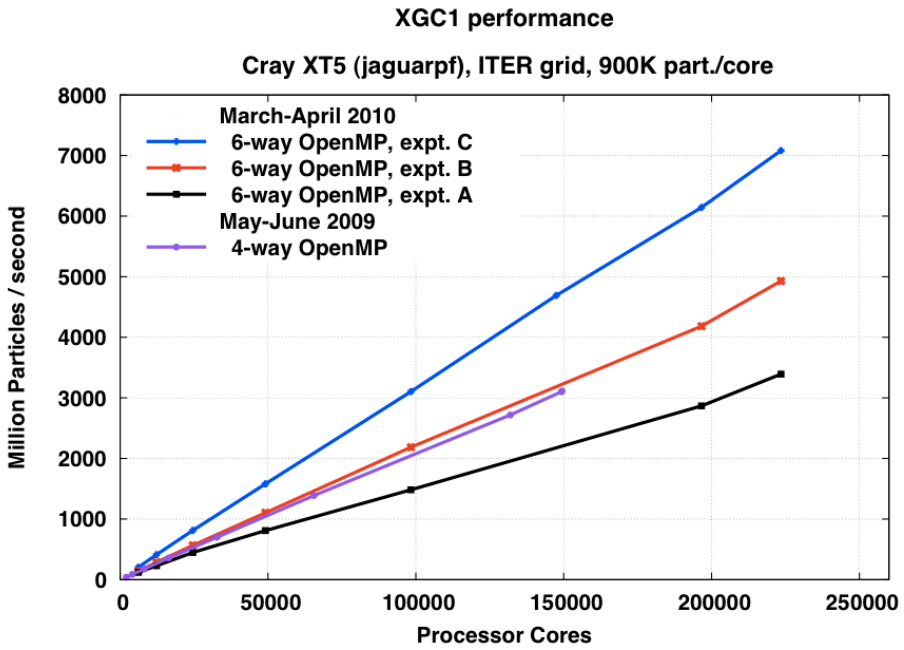


- Plotting number of particles processed per timestep per wallclock second.
- Impact of performance optimizations over last year has been significant, nearly doubling performance.
- Performance scalability also appears to excellent (more on that later).

Results: Particle Push Rate

900K particles/thread, all cores per node, 2 MPI processes per node





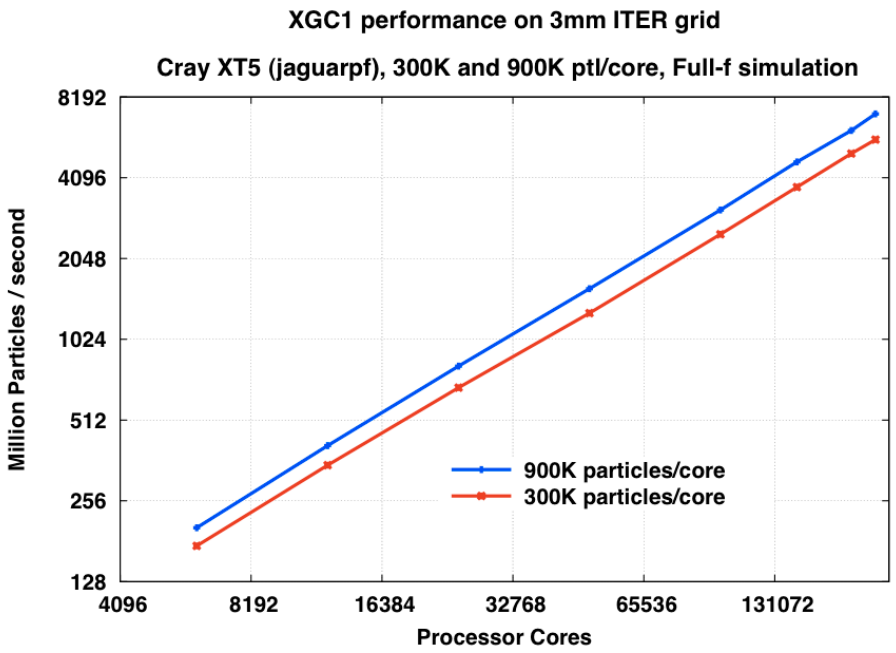
- Impact of performance optimizations and performance scaling for 900K particles per thread problem are both similar to that for 300K.
- Performance of 2009 version of code is degraded when run on current system with current compilers? Performance is almost exactly recovered by using “OpenMP-sensitive” optimizations?

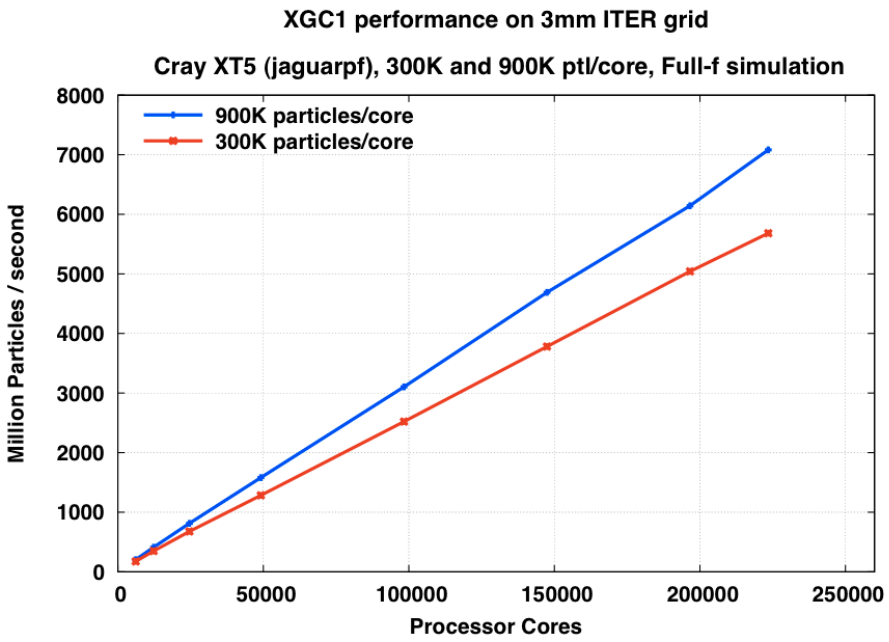
OAK RIDGE National Laboratory

Office of Science U.S. DEPARTMENT OF ENERGY

Results: Particle Push Rate

12 cores per node, 2 MPI processes per node

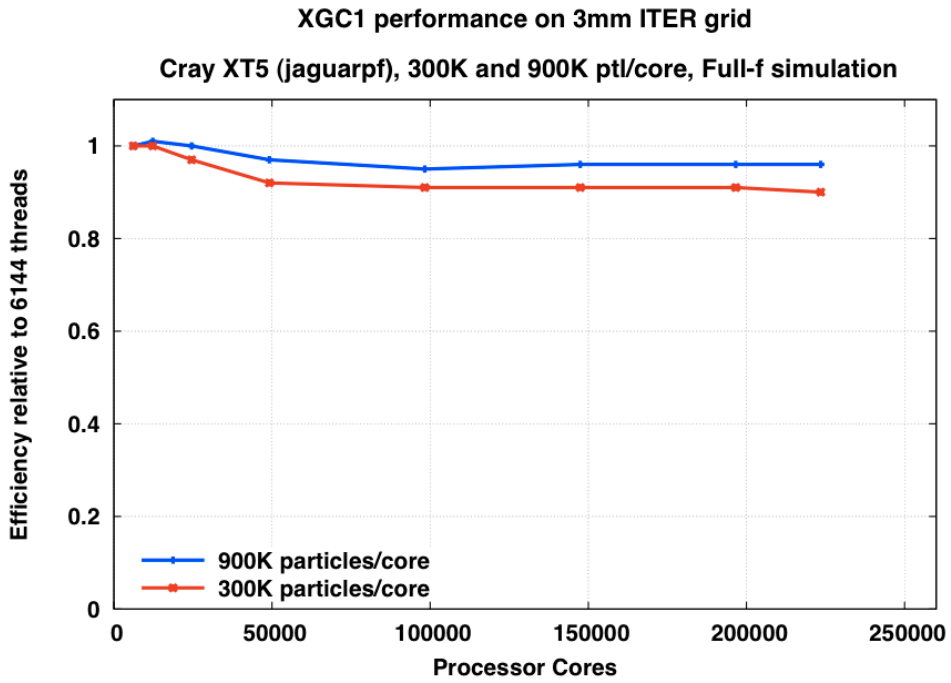




- 900K particles per thread problem is more computationally intensive than 300K problem, which leads to a somewhat higher particle push rate (approx. 20%).
- Performance scaling is excellent for both problems.

Results: Parallel Efficiency

12 cores per node, 2 MPI processes per node



- 900K particles per thread problem achieves higher relative efficiency than 300K problem, but this is sensitive to the definition of the baseline processor core count.
- Both problems retain relatively constant parallel efficiencies out to largest processor core count.

Conclusions

1. Performance of the June 2009 version of XGC1 on the 12-core XT5 is significantly worse than that on the 8-core XT5. Problems appear to arise in OpenMP-parallelized loops, so is perhaps due to a change in the compilers or runtime environment?
2. Recent code optimizations eliminate the performance problems, and more. The 2010(B) experiments show performance similar to the 2009 experiments on the same number of processor cores, but the 12-core XT5 allows the 2010(B) version of the code to run on 50% more cores and solve a 50% larger problem than was possible on the 8-core system with the same number of compute nodes.
3. The current XGC1 code demonstrates excellent performance scalability, and on the 12-core XT5 is achieving a particle push rate 2.3 times greater than that achieved in June 2009 on the same number of compute nodes.