XGC1: Performance on the 8-core and 12-core Cray XT5 systems at Oak Ridge National Laboratory [*]

Patrick H. Worley [†]
Mark F. Adams [‡]
Eduardo F. D'Azevedo [§]
C-S Chang [¶]
Seung-Hoe Ku [‖]
Collin McCurdy [**]

Abstract

The XGC1 code is used to model multiscale tokamak plasma turbulence dynamics in realistic diverted magnetic field geometry. In June 2009, XGC1 demonstrated nearly linear weak and strong scaling out to 150,000 cores on a Cray XT5 with 8-core nodes when solving problems of relevance to running experiments on the ITER tokamak. Here we compare performance, and discuss further performance optimizations, when running XGC1 on an XT5 with 12-core nodes on up to 224,000 cores.

1 Introduction

Understanding and predicting the behavior of burning plasma is essential to the development of commercially viable fusion power. XGC1 is a 5D gyrokinetic particle-in-cell (PIC) code designed to model the whole volume plasma dynamics in an experimentally realistic tokamak magnetic confinement fusion device geometry [2, 6].

During May-June of 2009 a performance scalability study of XGC1 was performed on *jaguarpf*, a Cray XT5 at Oak Ridge National Laboratory (ORNL), in preparation for a series of production runs [1]. At this point in time, jaguarpf had 18,722 compute nodes and a three-dimensional torus (25x32x24) interconnect based on the Cray SeaStar2+ communication switch processor. Each compute node contained two 2.3 GHz quad-core AMD Opteron 2356 (*Barcelona*) processors and 16 GB of DDR2-800 memory. Thus each compute node contained 8 processor cores and 2 GB memory per core, and an aggregate of 149,776 processor cores and 299,552 TB of memory for the entire system. Memory access performance was non-uniform (NUMA) at the node level, but uniform memory access could be enforced by pinning a thread of computation and its associated memory to the same socket in the dual socket node architecture

After some performance diagnosis and optimization, excellent performance scalability was observed

[*]This research was 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. The research of D'Azevedo, McCurdy, and Worley was performed at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC 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. Adams was funded under DOE grant DE-FC02-06ER54862.

[†]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5600, Oak Ridge, TN 37831-6016 (worleyph@ornl.gov)

[‡]Department of Applied Physics and Applied Mathematics, Columbia University, 289 Engineering Trace, New York, NY 10027 (mark.adams@columbia.edu)

[§]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367 (dazevedoef@ornl.gov)

[¶]Courant Institute of Mathematical Sciences New York University, 251 Mercer St., New York, NY 10012 (cschang@cims.nyu.edu)

[‖]Courant Institute of Mathematical Sciences New York University, 251 Mercer St., New York, NY 10012 (sku@cims.nyu.edu)

[**]Computer Science and Mathematics Division, Oak Ridge National Laboratory, P.O. Box 2008, Bldg. 5100, Oak Ridge, TN 37831-6137 (cmccurdy@ornl.gov)

out to the largest processor core count attempted, 149,248 cores (18,565 compute nodes). This study used version 7.2.5 of the PGI Fortran compiler and version 3.1.0 of *mpt*, the Cray-supplied version of MPI for the XT system. The results were presented at the 2009 SciDAC Conference [1].

During March-April of 2010 a similar performance study was completed on jaguarpf, in preparation for a new set of production runs. In Fall of 2009 the compute nodes of jaguaprf were upgraded to use two 2.6 GHz hex-core AMD Opteron 2435 (*Istanbul*) processors. The amount of memory per node and the interconnect were unchanged. At the time of this second XGC1 performance study jaguarpf was comprised of 18,688 compute nodes, for an aggregate of 224,256 processing cores. As will be described, XGC1 performance scalability in this second study was measured on as many as 223,488 processor cores (18,264 compute nodes). Versions 9.0.4 and 3.5.1 of the PGI Fortran compiler and the Cray-supplied version of MPI were used, respectively.

In this work we use these two performance studies to investigate changes in XGC1 performance on jaguarpf with the upgrade from the 8-core Barcelona-based compute nodes to the 12-core Istanbul-based compute nodes (and the associated modifications to the software stack). We also discuss performance optimizations that were implemented in XGC1 over this past year and report on current observed performance.

2 XGC1 and Experimental Design

XGC1 solves the gyrokinetic Vlasov equations [4, 7, 8] with marker particles and electric field data on a spatial grid. Each timestep of an XGC1 execution includes, minimally, the following stages:

1. Collect particle charge density on underlying grid (*charge deposition*).
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.

This is a simplified view of the algorithm in that these steps are used within a Runge-Kutta or predictor-corrector time integration method. Depending on the science experiment, the particles may be ions, electrons, or both. Experiments can also include collisions and other physical processes important to full-device simulations.

Parallelization of XGC1 is based on decompositions of both the spatial grid and the particle data across processes and MPI [10] is used to communicate between processes. In particular, the assignment of particles to processes is based on a decomposition of the spatial domain. Minimally, both the spatial grid and the particle decompositions utilize a one-dimensional decomposition in the toroidal direction of the tokamak geometry. OpenMP [3] is also used to parallelize loops over particles and loops over grid nodes or triangles. For more details see [1].

Both performance studies employed a mixed strong/weak scaling benchmark problem. The spatial grid was a 3 mm mesh of the three dimensional global core of ITER [5], an international research/engineering fusion device being contructed in France, and was fixed as the processor core count was varied. Thus the Poisson solve and other work on the grid was likewise fixed (strong scaling). The number of particles was fixed per thread of computation, so the total number of particles increased linearly with the number of processor cores (weak scaling).

A number of changes occurred between June 2009 and March 2010, beyond those in the jaguarpf system, that required careful experimental design to allow meaningful comparisons between the results from the two studies.

1. The spatial grid was modified between the two studies (to improve the simulation in a region not included in the benchmark problem). The grid used in 2009 had 893,884 grid points while the grid used in 2010 had 893,672. For the most part, the two grids are identical, and we do not expect the differences to affect the performance comparisons.
2. The number of particles per thread of computation was 900,000 in the 2009 study. For 2010 this was decreased to 300,000, due primarily to an increase in the number of processor cores being targeted for the production runs. This changes performance significantly. To address this we added experiments using 900,000 particles per thread to the 2010 study. We report 2010 performance data for both 300,000 and 900,000 particles per thread, but only the latter can be compared to the 2009 data.
3. Because of the weak scaling with respect to particle count, the problem size is a function

of the total number of threads of computation. However, it is also interesting to compare performance between the 8-core and 12-core node architectures as a function of the number of compute nodes. To address this, for the 2010 study we ran three different performance experiments:

- Using only 8 cores per node (4 cores per processor), leaving 33% of the cores unused. Here the same total number of particles, same number of particles per compute node, same amount of memory per compute node, and same number of processor cores are used as in the 2009 study when using the same number of compute nodes.
- Using all of the cores in a node, but choosing the node counts so as to reproduce the same numbers of processor cores used in the 2009 study. The number of particles and the memory requirements per node are 50% larger in the 2010 study for a given core count, however.
- Using all cores in the node. Here the total and per node number of particles, the memory requirements per node, and the number of processor cores in the 2010 study are 50% greater than those used in the 2009 study for the same number of compute nodes.

4. The XGC1 code also changed over this period. Because some of the software libraries and input data used in the 2009 study could not be regenerated or recovered for the 2010 study, we were unable to simply run the May-June 2009 version of XGC1.

   Some of the changes in XGC1, described below, were significant performance enhancements, and we ran experiments with these optimizations both enabled and disabled. Other changes, especially with regard to the calculation and output of diagnostics and to the parallel I/O infrastructure, were not feasible to disable. The code contains internal timers and natural synchronization points surrounding I/O-intensive phases. These allow the cost of the I/O and the associated computation to be measured. These timers indicate that, for these benchmark runs, the I/O cost did not contribute significantly to the runtime of the code.

   - An improved search technique for identifying the location of a particle in the spatial grid was implemented prior to the 2009 study, but was not used uniformly throughout the code. This improved algorithm uses a geometric-based hash function to quickly identify a small subset of candidate mesh cells in which the particle may be located. In the current implementation of XGC1 the new search algorithm is used exclusively.
   - A performance analysis in the Fall of 2009 identified that some assignment statements utilizing array syntax within OpenMP-parallelized loops were degrading OpenMP performance [9]. Replacing these statements with equivalent implementations using explicit loops eliminated the problem.
   - Random numbers are used in the charge deposition algorithm. For the 2009 benchmark runs, these random numbers were generated outside of the primary OpenMP-parallelized charge deposition loop, and the length of the loop being parallelized was limited by the number of random numbers precomputed (a compile-time decision, trading off memory for loop length). In the current implementation a thread-safe random number generator is used, the random numbers are computed within the parallel region, and the length of the OpenMP parallelized loop is not restricted.
   - Since June 2009 the spline interpolation algorithm has been reformulated to better reuse common subexpressions. New interpolation routines were also developed that evaluate all higher derivatives in one call, further exploiting common work and reducing the overhead of the subroutine calls (occurring within the innermost computational loops).

For the 2010 study, we measured performance using (A) none of the above optimizations, (B) all of these optimizations except those involving the spline interpolation routines, and (C) all of the above optimizations. Our conjecture is that performance without any of these optimizations is similar to what would have been observed from using the May 2009 version of XGC1 on the current jaguarpf system.

3 Results

All results described below are based on the wallclock time for 10 timesteps of the main computational loop in XGC1. This excludes initialization and final model clean-up activities, and is the relevant metric for predicting performance of long production runs.

In addition to raw performance (wallclock time to execute the 10 timesteps), we are also interested in the parallel scalability exhibited by XGC1 on the Cray XT5. To be "scalable" for a weak scaling benchmark, the runtime should not increase significantly as the processor core count increases. Because of the weak scaling in particle count, the local cost of processing the particles should be reasonably constant. Load imbalances can develop that will be sensitive to the processor core count, but this issue is not significant during the first 10 timesteps. The nonlocal cost of processing the particles includes the cost of local communication, likewise reasonably constant, and of global reductions and synchronizations, likely to grow with process count. Work on the spatial grid becomes communication bound as the process count increases because of the strong scaling in grid size and global communications in the Poisson solver. A slow growth in cost is the best that can be hoped for as the process count becomes large.

3.1 900K particles per thread

8 cores per node, MPI-only. We look first at performance when using 8 MPI processes per node, 4 processes per socket, and no OpenMP parallelism. This wastes 33% of the cores in the current version of jaguarpf, but uses the same amount and percentage of memory in the nodes and the same MPI traffic between the nodes.

		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	-

As mentioned earlier, Experiment A uses the current code but with a number of performance enhancements disabled, hopefully representing the performance of the 2009 version of the code on the current system. Experiment B uses the current code with only the spline interpolation optimizations disabled. Experiment C uses the current code with all optimizations enabled. Missing data that do not represent a code problem are denoted by "-". "OOM" denotes an out-of-memory failure.

From these data we draw the following conclusions.

- XGC1 for this benchmark problem scaled reasonably well on both systems and for all versions of the code. That is, the runtime does not increase significantly as the process count increases from 4096 to 148,992.

- The 2010 Experiment A results are in line with what we would expect from running the 2009 version of the code on the current jaguarpf. Solely from the processor clock change, we might expect the 2010 results to be 13% faster than the 2009 results. For 4096 cores, the improvement is 9%. This decreases as the process count increases, but the ratio of communication to computation also increases, so the trend is reasonable.

- The performance differences between Experiment A and Experiment B are almost entirely in a routine called `shift_decomp` that calculates whether particles should be moved to a different process for the next timestep. The differences arise from the fact that Experiment A uses the old search algorithm in this routine while Experiment B uses the optimized search algorithm. Interestingly, the cost of this routine in the 2009 study is between that of Experiment A and Experiment B, and Experiment B would be an even better model of how we would expect the May 2009 version of XGC1 to run on the 12-core-node version of jaguarpf.

- The optimizations to the spline interpolation routines made a dramatic improvement in XGC1 performance (over 40% compared to not using it). Even with the decreased computational cost, increasing the importance of the interprocess communication performance, the code continues to scale well.

8 cores per node, 4-way OpenMP. We next look at performance when using 8 cores per node, 4 cores per socket, but with two MPI process per node and 4 threads per process. This was the option that achieved the best performance in the 2009 study.

		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

The scalability is even better here than for the MPI-only experiments, except for Experiment A. The smaller total number of MPI processes and the smaller number of MPI processes per node competing for access to the network decrease the communication overhead.

The performance degradation in Experiment A is due to a load imbalance in which some processes spend significantly more time in `shift_decomp`. The performance problem again is associated with using the old search algorithm, which is now called within an OpenMP-parallelized loop. The old search algorithm is more memory intensive than the optimized algorithm, and performance appears to be very sensitive to the memory access patterns occurring within some of the processes. Interestingly, the 2009 study gave little indication of a similar problem. Something about the new node architecture or the newer compiler and runtime system appears to be more sensitive to this issue.

Note that the replacement of the array syntax within OpenMP loops did improve performance when comparing timers in Experiment A and Experiment B, for example, improving the performance of the primary charge deposition loop by 10%.

8 cores per node, 8-way OpenMP. The next comparison uses 1 MPI process per node and 8 threads per process, examining the impact of further increasing OpenMP parallelism. Note that, in contrast to the experiments on the 8-core nodes, the threads are not necessarily divided evenly between the two sockets on the 12-core nodes.

		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

Performance of all experiments is degraded when compared to using 4 OpenMP threads per process. Qualitatively, the results are the same, however, with all but Experiment A demonstrating good scalability.

All cores per node, 4-way OpenMP. Here we compare performance when using all of the cores available in a node and the same total number of cores. The same problem size is being solved on the two systems, but using a different number of compute nodes and with different node memory requirements. This comparison illustrates the relative computational capabilities of the two different systems more directly.

We again look at performance using 4 OpenMP threads per process, so are now using three MPI processes per node for the 12-core nodes with one MPI process having OpenMP threads assigned to two different sockets. (We were unable to run MPI-only experiments because there was not enough memory on the 12-core nodes.)

	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

Qualitatively, the performance is similar to that of the previous 4-way OpenMP experiments. The 2009 data are identical, of course. The small processor core counts are more expensive relative to the previous experiments, possibly due to the relatively coarse decomposition of the spatial grid and the 50% more MPI processes assigned to the node adding to the memory requirements and cost of accessing memory. For large processor core counts, the modest increase in cost can be attributed partly to 3 MPI processes now competing for network access and all 12 cores now contending for access to memory.

All cores per node, two MPI processes per node. We again compare performance when using all of the cores available in a node, but now with the same number of nodes on both systems. In this case, a larger problem is being solved on the 12-core-node system. The per node memory requirements are again different on the two systems as well. Here we compare performance for 2 MPI tasks per node, with 6 OpenMP threads per process on the 12-core-node system and 4 OpenMP threads per process for the 8-core-node system. Earlier studies indicated that these were the most effective ways to run XGC1 on each.

	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

Again, these results are qualitatively the same as for the previous three experiments. Unlike in the previous experiment, the small process count results do not show an anomalously large cost, presumably because 3 MPI processes are not assigned to the node. The cost here is higher than when using 4-way OpenMP and only 8-cores because of the increased memory requirements, increased memory contention from having work assigned to all cores, and, possibly, some non-memory-related loss of OpenMP efficiency when increasing the number of threads from 4 to 6.

3.2 12-core node system

Here we look more closely at the performance characteristics of XGC1 when run on the 12-core node system. In particular, we examine performance for both 300,000 and 900,000 particles per thread, considering only the version of the code with all optimizations enabled. Again, timings are for 10 timesteps of the main computational loop. Data were collected for 20 timesteps also, which were essentially double the 10 timestep results in all cases.

300K particles per process.

		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

While we did not finish the suite of MPI-only experiments for this problem size, it is clear that using 6 OpenMP threads per process is much faster. It is unclear from these limited data whether using only MPI will in fact scale well to large process counts. The earlier experiments, when using only 8 of the 12 cores, indicate that it should scale, unless there is some anomalous behavior when all cores are assigned MPI processes.

Scalability is excellent for both experiments using OpenMP as well as MPI. Restricting the OpenMP threads to the same socket as the associated MPI process, as occurs with the 6 threads per process experiments, appears to enhance performance. However preliminary studies indicate that there are additional reasons that 12-way OpenMP performance is degraded. In any case, the 6-way OpenMP results are approximately 11% faster than the corresponding 12-way results.

900K particles per process.

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

When using MPI only, the 900,000 particles per thread benchmark requires more memory than is available. Again, performance scalability is excellent for both 6-way and 12-way OpenMP parallelism, and using 6-way OpenMP parallelism is faster than using 12-way (by approximately 16%).

3.3 Particle Push Rate

Another metric of performance is the average number of particles processed per timestep per second. This allows us to compare throughput for the different problem sizes and for the different incarnations of jaguarpf.

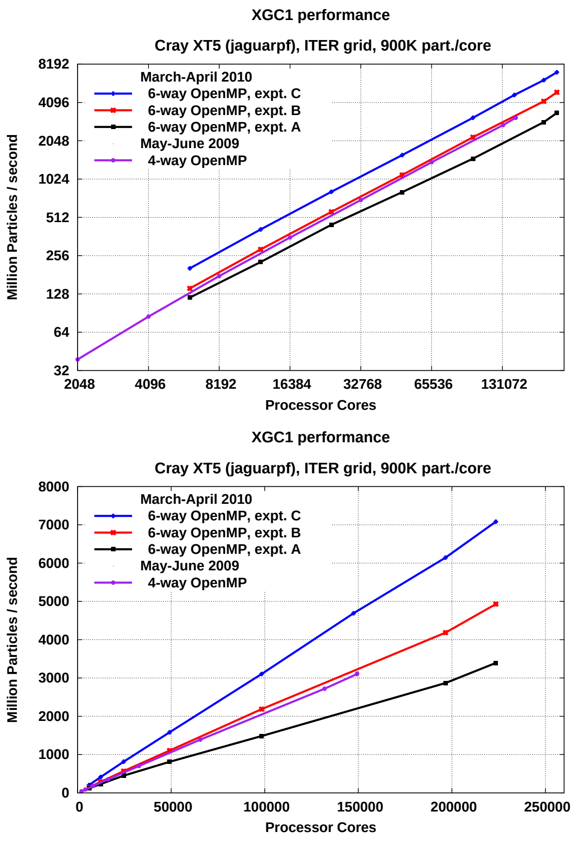

FIGURE 3.1: Performance for 900K part./core (log-log top; lin-lin bottom)

Figure 3.1 contains graphs of performance data for the 900,000 particles per thread benchmark problem, one with logarithmically-scaled axes and one with linear scaling. Here we can see clearly the excellent scalablity of XGC1 out to essentially the whole system, even with the significant reduction in computational cost arising from the optimizations in the interpolation schemes. We also see that the original version of the code, which scaled well on the 8-node system in June of 2009, does not perform as well on the current 12-node system with the current compilers, libraries, and runtime system. However, by making optimizations within the OpenMP-parallelized loops (Experiment B) the earlier performance is recovered as a function of processor core count. Moreover, the 12-core-node system allows much larger problems to be solved with the same number of compute nodes without degradation in performance scalability.

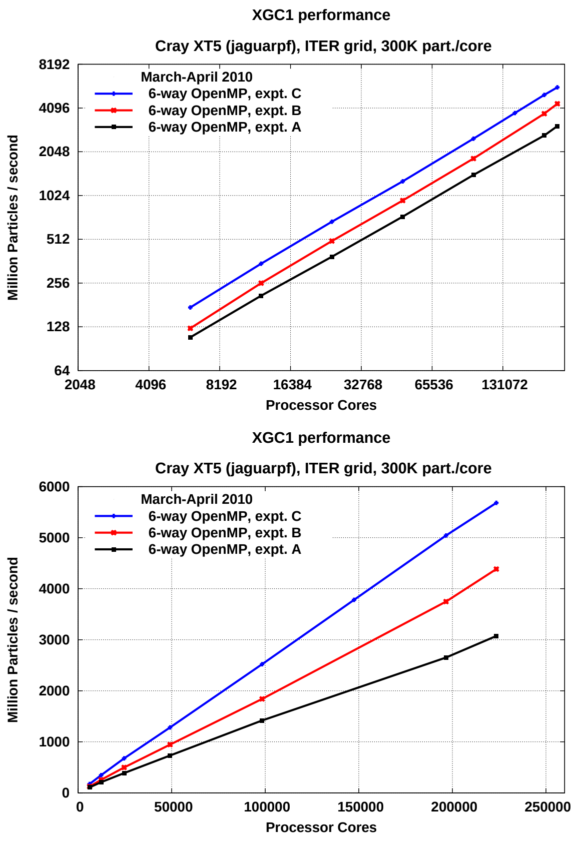

FIGURE 3.2: Performance for 300K part./core (log-log top; lin-lin bottom)

Figure 3.2 also contains graphs of performance data for the 300,000 particles per thread benchmark problem, plotted with both logarithmic and linear scales. Here we again observe the excellent scalablity of XGC1, and the significant impact of the recent performance optimizations.

Figure 3.3 contains graphs comparing the performance of the 300,000 and 900,000 particles per thread benchmarks when using all performance optimizations. From these, the scalability appears to be equally good for both benchmarks, even though the 300,000 particle per thread problem has a higher percentage of time spent in interprocess communication. The impact of this does appear in the absolute comparisons, with the 900,000 particles per thread problem achieving between a 16% and 25% higher particle push rate. Figure 3.4 graphs the relative parallel efficiency, compared to experiments using 6144 processor cores. While the efficiency of the 300K particles per thread problem is somewhat less than for the 900K problem (90% compared to 96%), both are maintaining these relative efficiencies out to the largest processor core counts.

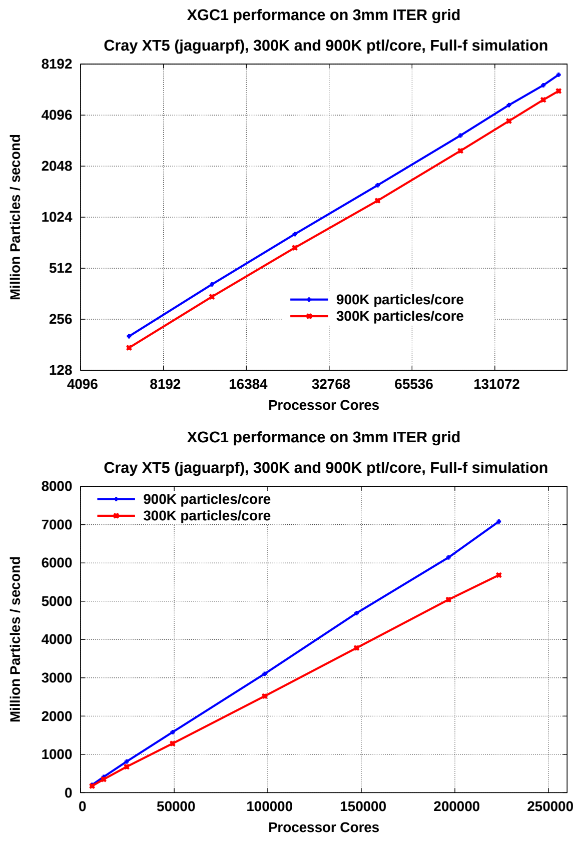

FIGURE 3.3: Comparing 300K and 900K part./core performance (log-log top; lin-lin bottom)

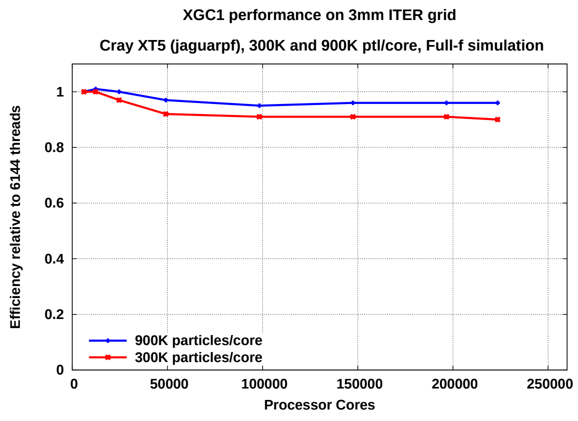

FIGURE 3.4: Comparing 300K and 900K part./core relative efficiency

4 Conclusions

The excellent performance scalability observed in June 2009 in performance studies with XGC1 on the 8-core-node jaguarpf XT5 system was also observed on the current 12-core-node system. This was not a given however. While uncertainties remain because of a number of factors that we could not control in our experiments, it appears that OpenMP performance characteristics and sensitivities have changed over the past year. In consequence the original code would have performed significantly worse on the current system. Whether this is due to the impact of sharing the same amount of memory between 12 cores instead of 8 or whether due to changes in the software stack (or both) can not be determined from these results. We were fortunate to have diagnosed (and, in one case, stumbled upon) optimizations that eliminate these performance issues, recovering the lost performance. Additional optimizations were also applied to the code, further improving XGC1 performance by 40%.

Work is continuing on performance optimizations of XGC1. For example, we have begun looking at alternative compilers as well. One unexpected result of this study is that using 3 MPI processes per node and 4 OpenMP threads per process may perform better than using 2 MPI processes and 6 OpenMP threads per process, even though in the first case one MPI process has threads that are assigned to cores in different sockets. We will be verifying these results and examining them in more detail in the coming months.

5 Acknowledgements

This research used resources (Cray XT5) of the National Center for Computational Sciences at 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.

6 About the Authors

Mark F. Adams is a research scientist in the department of Applied Physics and Applied Mathematics at Columbia University. His research interests are in large-scale numerical simulations, in particular, multigrid equation solvers and parallel, unstructured finite element coupling frameworks in solid mechanics. He currently works with fusion plasma physics applications, scalable solvers for the adaptive mesh refinement and fluid-structure interaction problems. Adams received his Ph.D. in Civil Engineering, from U.C. Berkeley in 1998.

Eduardo F. D'Azevedo is group leader for the Computational Mathematics Group at the Computer Science and Mathematics Division, Oak Ridge National Laboratory. He is co-author of one book and over 20 refereed publications. His current research includes: optimal mesh generation, vectorized iterative solver, out-of-core dense linear solvers, and application specific preconditioners. He received his Ph.D. in 1989 in the Faculty of Mathematics (Department of Computer Science) from the University of Waterloo, Ontario, Canada. He held an ORISE postdoctoral fellowship from 1990-1991, and has been a research staff member at ORNL since 1991. He is a member of the Society for Industrial and Applied Mathematics.

C-S Chang is the head of the SciDAC Prototype Fusion Simulation Project (FSP) Center for Plasma Edge Simulation (CPES). He is a Research Professor at the Courant Institute of Mathematical Sciences, New York University, and, jointly, a Professor of Physics at Korea Advanced Institute of Science and Technology. He is a Fellow of the American Physical Society. He serves in numerous national and international advisory and executive committees, including: the Council of the US Burning Plasma Organization; Executive Committee, US Transport Task Force; Theory Coordinating Committee, US DOE Office of Fusion Energy Sciences; Advisory Committee (Chair), SciDAC Gyrokinetic Plasma Simulation Center; Users' Council Executive Committee, National Center for Computational Sciences (NCCS).

Seung-Hoe Ku is a Research Scientist with the joint title Research Assistant Professor in the Courant Institute of Mathematical Sciences, New York University. He is the primary developer of the XGC1 full-function gyrokinetic code. He has a PhD degree in physics from Korea Advanced Institute of Science and Technology (KAIST).

Collin McCurdy is a Post-Doctoral Research Associate at the University of Tennessee in Knoxville, affiliated with the Future Technologies Group at Oak Ridge National Laboratory. His research focuses on memory system designs in current and future processor architectures and their implications for scientific applications. He has a PhD in Computer Science from the University of Wisconsin–Madison and is a member of the Association for Computing Machinery.

Patrick H. Worley is a senior R&D staff member in the Computer Science and Mathematics Division of Oak Ridge National Laboratory. His research interests include parallel algorithm design and implementation (especially as applied to simulation models used in climate and fusion energy research) and the performance evaluation of parallel applications and computer systems. He is currently a co-chair of the CCSM Software Engineering Working Group, the principal investigator for the Performance Engineering and Analysis Consortium End Station DOE INCITE project, and is an Associate Editor of the journal Parallel Computing. Worley has a PhD in computer science from Stanford University. He is a member of the Association for Computing Machinery and the Society for Industrial and Applied Mathematics.

References

[1] M. Adams, S. Ku, E. D'Azevedo, J. Cummings, and C.-S. Chang, *Scaling to 150k cores: recent algorithm and performance engineering developments enabling XGC1 to run at scale*, Journal of Physics: Conference Series, 180 (2009), pp. 012036–+.

[2] C. S. Chang, S. Ku, P. H. Diamond, Z. Lin, S. Parker, T. S. Hahm, and N. Samatova, *Compressed ITG turbulence in diverted tokamak edge*, Phys. Plasmas, 16 (2009), pp. 056108–+.

[3] L. Dagum and R. Menon, *OpenMP: an industry-standard API for shared-memory programming*, IEEE Computational Science & Engineering, 5 (1998), pp. 46–55.

[4] T. S. Hahm, *Nonlinear gyrokinetic equations for tokamak microturbulence*, Phys. Fluids, 31 (1988), pp. 2670–2673.

[5] ITER. http://www.iter.org/.

[6] S. Ku, C. Chang, and P. Diamond, *Full-f gyrokinetic particle simulation of centrally heated global ITG turbulence from magnetic axis to edge pedestal top in a realistic tokamak geometry*, Nuclear Fusion, 49 (2009), pp. 115021–+.

[7] W. W. Lee, *Gyrokinetic approach in particle simulation*, Phys. Fluids, 26 (1983), pp. 556–562.

[8] ——, *Gyrokinetic particle simulation model*, J. Comput. Phys., 72 (1987), pp. 243–269.

[9] C. McCurdy and J. S. Vetter, *Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms*, in ISPASS, IEEE Computer Society, 2010, pp. 87–96.

[10] MPI Committee, *MPI: a message-passing interface standard*, Internat. J. Supercomputer Applications, 8 (1994), pp. 165–416.