The Cray XT4 Quad-core : A First Look *

Sadaf R. Alam, Richard F. Barrett, Markus Eisenbach, Mark R. Fahey,
Rebecca Hartman-Baker, Jeffrey A. Kuehn, Stephen W. Poole,
Ramanan Sankaran, and Patrick H. Worley
Oak Ridge National Laboratory
Oak Ridge, TN 37931

Presented at Cray User Group, Helsinki, Finland, May 5, 2008

Abstract

The Cray XT4 at Oak Ridge National Laboratory (ORNL), named Jaguar, has recently been upgraded, from dual-core to quad-core processors in addition to other significant changes. Although we have had very limited access to the machine and therefore are not presenting definitive performance results, we can share some meaningful and constructive experiences to the user community which could be of assistance as they gain access to Jaguar as well as other multi-core processor based parallel computers. These experiences were gained while porting a broad set of scientific application programs to Jaguar.

1 Introduction

The Cray XT system located at Oak Ridge National Laboratory is the most powerful computing capability for the Department of Energy's (DOE) Office of Science, and in fact represents the largest open science capability machine in the United States. Named Jaguar[1], it is the primary leadership computer for the DOE Innovative and Novel Computational Impact on Theory and Experiment (INCITE) program[2], which supports computationally intensive, large-scale research projects. The 2008 program awarded over 140 million processor hours on Jaguar to groups investigating questions in a broad set of science areas, including global climate dynamics, nuclear fusion, nuclear fission, combustion energy, biology, astrophysics, and materials.

In order to support this required scale of computing, Jaguar has again been upgraded, this time from a 119 TFLOPS capability to 262 TFLOPS. Several fundamental characteristics of the architecture have changed with this upgrade, which motivates this report.

We begin with an overview of the hardware and software of the XT4. Next we present performance results from some key applications running on this platform, with analysis supplemented by some relevant micro-benchmarks. Then we discuss the performance potential for configuring applications using multi-core aware constructs. We conclude with a discussion and summary of this study.

2 System and Software

The current incarnation of Jaguar is based on an evolutionary improvement beginning with the XT3, Cray's third-generation massively parallel processing system, building on the T3D and T3E systems. Based on commodity AMD Opteron processors, a Cray custom interconnect, and a light-weight kernel (LWK) operating system, the XT3 was delivered in the summer of 2005. Each node consisted of an AMD Opteron model 150 (single core) processor, running at 2.4 GHz with 2 GBytes of DDR-400 memory. The nodes were connected by a SeaStar router through HyperTransport, in a 3-dimensional torus topology, and running the Catamount operating system[14]. Each processor was capable of two floating point operations per clock cycle, for a peak

*This research used resources 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.

[1]`http://www.nccs.gov/computing-resources/jaguar`

[2]`http://www.er.doe.gov/ascr/incite`

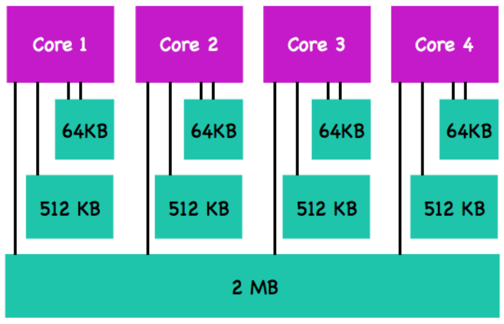
System				Processor Speeds			Memory			Network	
Arch	Date	cores/ node	Num Nodes	GHz	Ops/ clock	Peak TFLOPS	GB/ node	DDR	Mem GB/s	Sea Star	BW GB/s
XT3	6/05	1	5,212	2.4	2	25	2	1	6.4	1	2.2
XT3	7/06	2	5,212	2.6	2	54	4	1	6.4	1	2.2
XT3/4	4/07	2	11,508	2.6	2	119	4	1,2	6.4/10.6	1,2	2.2,4
XT4	5/08	4	7,832	2.1	4	263	8	2	10.6	2	4

Table 1: Jaguar history at ORNL

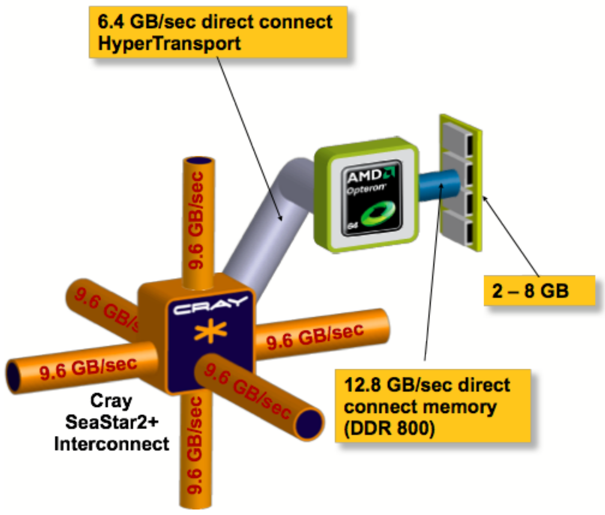
performance of 4.8 GFLOPS. With 5,212 compute nodes, the peak performance of the XT3 was just over 25 TFLOPS. An evaluation of this configuration was presented in [2].

Jaguar processors were swapped out for dual-core Opteron model 100 2.6 GHz processors in July, 2006, with memory per node doubled in order to maintain 2 GBytes per core. It was again upgraded April, 2007, with three major improvements: 6,296 nodes were added; memory on the new nodes was upgraded to DDR2-667, increasing memory bandwidth from 6.4 GBytes per second (GB/s) to 10.6 GB/s; and the SeaStar2 network chip connected the new nodes, increasing network injection bandwidth (of those nodes) from 2.2 GB/s to 4 GB/s and increasing the sustained network performance from 4 GB/s to 6 GB/s. Thus with 23,016 processor cores, this so-called XT3/XT4 hybrid provided a peak performance of 119 TFLOPS. Evaluation of this configuration was presented in [1].

This spring Jaguar was again upgraded: 7,832 quad-core processors (illustrated in Figure 1) replace the 11,508 dual-core, the interconnect is now fully SeaStar2, and the LWK is a customized version of Linux,



(a) Barcelona


6.4 GB/sec direct connect HyperTransport
AMD Opteron
2 – 8 GB
9.6 GB/sec
9.6 GB/sec
9.6 GB/sec
9.6 GB/sec
9.6 GB/sec
9.6 GB/sec
CRAY
Cray SeaStar2+ Interconnect
12.8 GB/sec direct connect memory (DDR 800)

(b) XT4 network

Figure 1: XT4 architecture

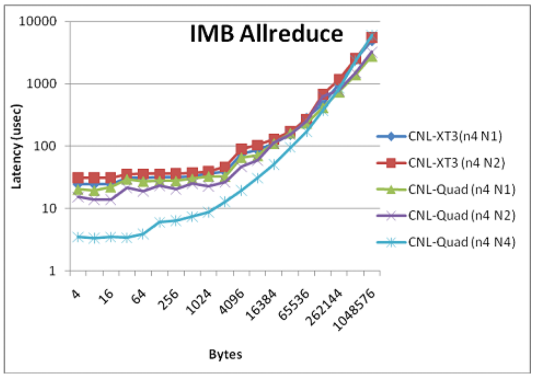
named Compute-Node Linux (CNL)[3]. Each compute node now contains a 2.1 GHz quad-core AMD Opteron processor and 8 GBytes of memory (maintaining the per core memory at 2 GBytes). As before, nodes are connected in a 3-dimensional torus topology, though now fully SeaStar2 router through HyperTransport (see Figure 1(b)). This configuration provides 262 TFLOPS with 60 TBytes of memory. A history of some key specs of the Jaguar upgrades are listed in Table 1.

Third-party developed software is managed by the NCCS, as described in [8]. This strategy presents the user with a well-managed broad set of commonly used software libraries, configured using compilers from three sources (PathScale, PGI, and gnu), consistent across several NCCS computing environments.

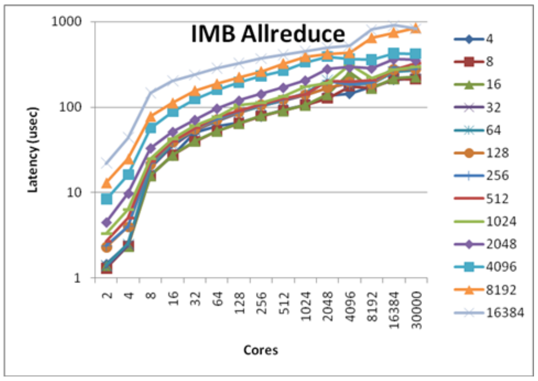
[3]Being re-named Cray Linux Environment (CLE).

2.1 Message passing

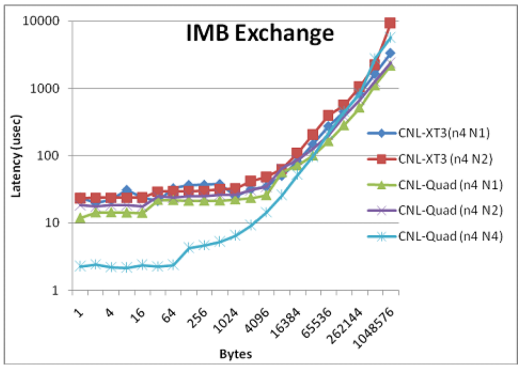
The applications in this report are based MPI functionality. Performance of some key functionality[4] with regard to these applications is shown in Figures 2(a) and 2. These graphs highlight the importance of the



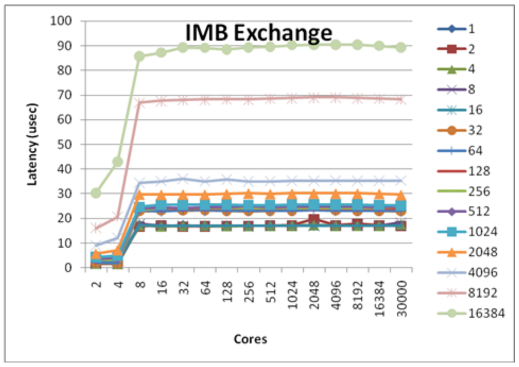
(a) IMB Allreduce on 4 cores



(b) IMB Allreduce on quad core



(c) IMB Exchange on 4 cores



(d) IMB Exchange on quad core

Figure 2: Performance of IMB Allreduce and Exchange.

newly release MPI implementation, contained in module `mpt-3.0.1`. A shared memory device has been incorporated, significantly improving the performance within a quad-core processor. Figures 2(b) and 2(d) show how performance changes as the number of cores increases. As expected, there is a significant increase in the time required for the operations when operating on more than one node.

Unless otherwise stated, performance is reported for code compiled using PGI compilers, and the recently released MPI implementation as contained in module `mpt/3.0`. With more time, however, we intend to explore other compiler options as well as their capabilities (flags).

3 Application case studies

The true test of a machine's ability to deliver performance to the computational scientist is to run the actual application programs using representative experiments of interest. Our focus is on analyzing the behavior of a broad set of representative applications executing problems of interest at the scale for which the architecture is designed and applications are intended. Yet we remind the reader again of the maturity level of the machine and our access to it thus far, and therefore caution strongly against drawing definitive conclusions regarding this architecture's ultimately capabilities. Our intent here is to share our experiences up to this point, which we believe are significant and meaningful when placed in the appropriate context.

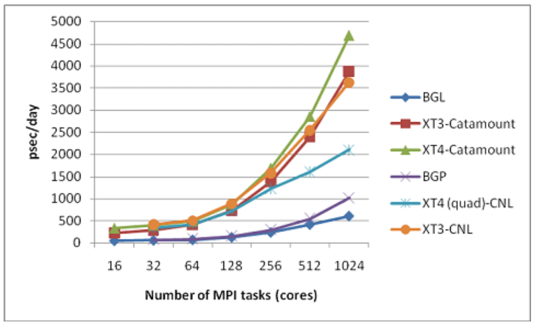
[4]Performance measurements of MPI functionality are provided using the Intel MPI Benchmark (IMB), version 3.1.

3.1 Molecular dynammics: LAMMPS

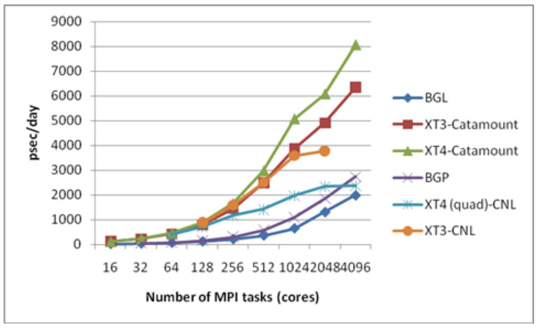
Molecular dynamics (MD) simulations enable the study of complex, dynamic processes that occur in biological systems[13]. The MD related methods are now routinely used to investigate the structure, dynamics, functions, and thermodynamics of biological molecules and their complexes. The types of biological activity that has been investigated using MD simulations include protein folding, enzyme catalysation, conformational changes associated with bimolecular function, and molecular recognition of proteins, DNA, biological membrane complexes. Biological molecules exhibit a wide range of time and length scales over which specific processes occur, hence the computational complexity of an MD simulation depends greatly on the time and length scales considered. With a solvation model, typical system sizes of interest range from 20,000 atoms to more than 1 million atoms; if the solvation is implicit, sizes range from a few thousand atoms to about 100,000. The time period of simulation can range from pico-seconds to the a few micro-seconds or longer. Several commercial and open source software frameworks for MD calculations are in use by a large community of biologists

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator[17]) is a classical molecular dynamics code. It models an ensemble of particles in a liquid, solid, or gaseous state and can be used to model atomic, polymeric, biological, metallic or granular systems. The version used in these experiments is written in C++ and MPI. Inter-process communication in LAMMPS is dominated by global reductions and nearest neighbor communication.

Two experiments are presented herein. In the first, the medium-scale HhaI system is a model for protein-DNA complex (enzyme m5C-methyltransferase M. HhaI with its target DNA sequence), in explicit solvent and counter-ions to allow the system to be charge neutral. This model consists of 61,641 atoms with explicit treatment of solvent using the TIP3P water model. The system was equilibrated before benchmarking runs. The time-step is (10^{15} seconds). The long-range forces are calculated using PME. The performance of a LAMMPS experiment consisting of 64,000 atoms is shown in Figure 3(a). As the number of cores increases,



(a) medium-scale HhaI system



(b) RAQ system

Figure 3: LAMMPS simulation throughput.

throughput, defined as pico-seconds per day, should increase. This was the case for the previous versions of Jaguar as well as on an IBM BlueGene/L. However, beyond 128 cores on Jaguar, performance is starting to degrade, and by 512 cores performance is approximately half that of the dual-core Jaguar.

A larger experiment was run, consisting of 290,220 atoms. Using the RAQ system, this experiment modeled the enzyme RuBisCO with explicit treatment of solvent. Performance results are shown in Figure 3(b). Here, with more work per processor, performance is on par with the previous versions of Jaguar up to 256 cores, though it is starting to degrade. By 512 cores performance is less than half that of previous Jaguars, and even drops below the performance of IBM BlueGene/P by 4,096 cores.

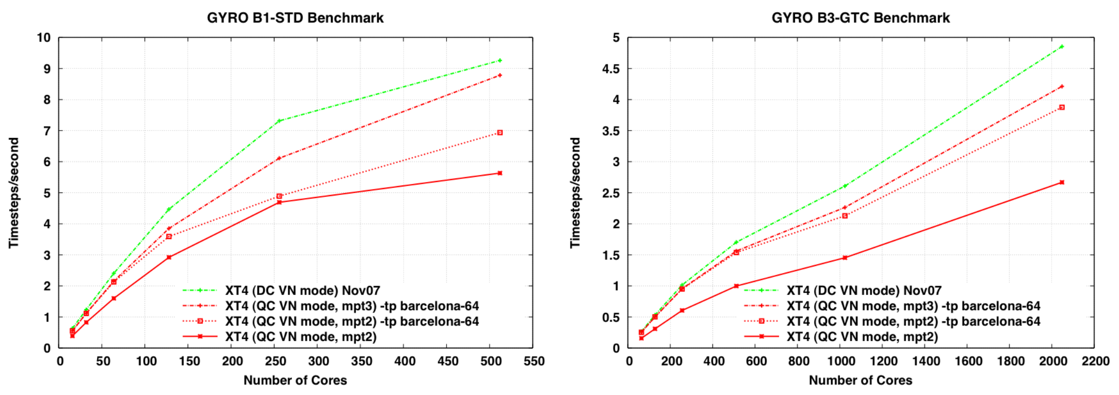
Its important to note that these experiments were run very early during the quad-core upgrade, with no chance to analyze the runtime behavior and make adjustments. In the next section we will see some very simple things that might significantly improve performance.

3.2 GYRO: Fusion Energy

GYRO[5] is a code for the numerical simulation of tokamak microturbulence, solving time-dependent, non-linear gyrokinetic-Maxwell equations with gyrokinetic ions and electrons capable of treating finite electro-magnetic microturbulence. GYRO uses a five-dimensional grid and propagates the system forward in time using a fourth-order, explicit, Eulerian algorithm. GYRO has been ported to a variety of modern HPC platforms including a number of commodity clusters.

Version 6.5 of GYRO was used to run two standard benchmark problems: B1-std and B3-gtc. The two problems differ in size and computational and communication requirements per process. The B1-std problem is smaller but requires more work per grid point than the B3-gtc problem. However, GYRO tends to scale better for the B3-gtc problem than the B1-std problem. The B3-gtc problem can use an FFT-based approach or a non-FFT approach; for our tests we use the FFT-based approach and use the vendors optimized FFT library when available (which it is on Jaguar). The primary communication costs result from calls to `MPI_ALLTOALL`, used to transpose distributed arrays[4] among the dimensions.

The B1-std problem is a 16 toroidal-mode electrostatic (electrons and ions, 1 field) case on a $16 \times 140 \times 8 \times 8 \times 20$ grid. This test simulates kinetic electrons and electron collisions, with no electromagnetic effects, for 500 time steps, and requires multiples of 16 processes. Figure 4(a) demonstrates the strong scaling of GYRO



(a) B1-std (b) B3-gtc

Figure 4: GYRO strong scaling performance

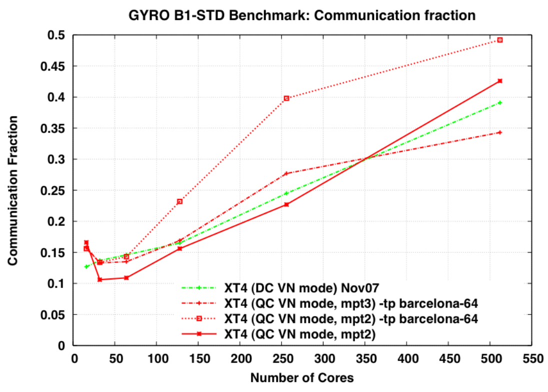
for the B1-std problem achieved. The y-axis represents the number of simulation time steps achieved per second of wall clock. The graph shows the performance advantages of using the `-tp barcelona-64` compiler flag and the recently released shared memory aware MPI implementation contained in `mpt 3`. However, the XT4 quickly runs out of work per process as the process count increases, so performance does not improve much as the number of cores increases beyond 128, and therefore a larger problem is required.

The B3-gtc problem is a 64 toroidal-mode adiabatic (ions only, 1 field) case on a $64 \times 400 \times 8 \times 8 \times 20$ grid. This test runs on multiples of 64 processes, computing 100 timesteps representing 3 simulation seconds. The 400-point radial domain with 64 torodial modes gives high spatial resolution, but electron physics are ignored allowing simple field solves and large timesteps. Figure 4(b) shows the strong scaling of GYRO for the B3-gtc problem. Performance increases up to 2,048 processes without any significant drop in efficiency. For this example, however, inclusion of the quad-core compiler flag `-tp barcelona-64` provides a very significant performance increase, with an additional but smaller boost from the new MPI implementation.

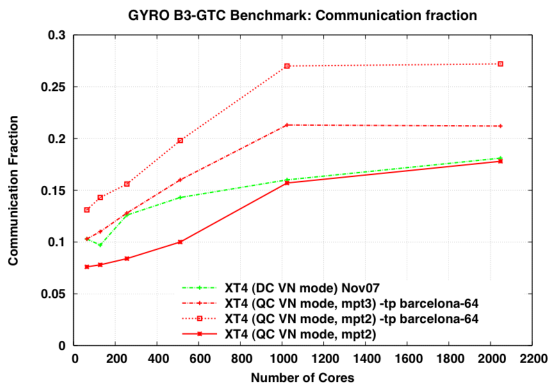
However, its important to recall that this rather small decrease in performance relative to dual-core Jaguar is even more significant since per core capability is twice that for quad-core relative to dual, less the decrease in clock speed. This strongly suggests that vectorization of the code is not taking place, an issue we will explore in the next section.

Figures 5(a) and 5(b) show that communication costs are about the same on dual-core and quad-core XT4. As presented in[7], GYRO is memory bandwidth limited and therefore the quad-core socket memory bandwidth is a bottleneck.

Figure 6 shows the weak scaling characteristics of GYRO for a "modified B3-gtc" problem. The problem was modified to fit the memory of a BlueGene/L for cross-platform comparisons. The code is weakly scaled by



(a) B1-std



(b) B3-gtc

Figure 5: GYRO strong scaling communication costs

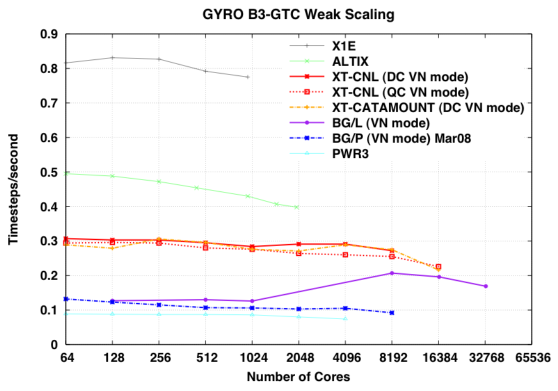


Figure 6: GYRO B3-gtc weak scaling performance

keeping the "ENERGY GRID" size constant as the number of processes increases. In this figure, XT4.Apr08 is the current quad-core XT at ORNL, while XT.Nov07 refers to the previous dual-core XT3/XT4 machine where a job could run across differing numbers of XT3 and XT4 nodes. Since the number of each type of XT node was not tracked for the XT.Nov07 tests, the line is generically labeled XT and could be XT3-only, XT4-only, or a combination and this would explain otherwise anomalous-looking characteristics. Note that although there is a slight drop-off in performance out to 16K cores on the quad-core XT4 (running the CNL operating system), the drop is not near as much as witnessed on the dual-core XT3/XT4 (running Catamount.)

3.3 Combustion: S3D

S3D is a direct numerical simulation solver for the full compressible Navier-Stokes, total energy, species and mass continuity equations coupled with detailed chemistry [9, 19]. It is based on a high-order accurate, non-dissipative numerical scheme. The governing equations are solved on a conventional three-dimensional structured Cartesian mesh. Spatial differentiation is achieved through eighth-order finite differences along with tenth-order filters to damp any spurious oscillations in the solution. The differentiation and filtering require nine and eleven point centered stencils, respectively. Time advancement is achieved through a six-stage, fourth-order explicit Runge-Kutta (R-K) method1. Navier Stokes characteristic boundary condition (NSCBC) treatment [18, 20, 21] is used on the boundaries. Fully coupled mass conservation equations for the different chemical species are solved as part of the simulation to obtain the chemical state of the system. Detailed chemical kinetics and molecular transport models are used. An optimized and fine-tuned library has been developed to compute the chemical reaction and species diffusion rates based on Sandia's Chemkin package. While Chemkin-standard chemistry and transport models are readily usable with S3D, special attention is paid to the efficiency and performance of the chemical models. Reduced chemical and transport models that are fine -tuned to the target problem are developed as a pre-processing step.

S3D is parallelized using a three-dimensional domain decomposition and MPI communication. Each MPI process is responsible for a piece of the three-dimensional domain. All MPI processes have the same number of grid points and the same computational load. Inter-processor communication is only between nearest neighbors in a logical three-dimensional topology. A ghost-zone is constructed at the processor boundaries by non-blocking MPI sends and receives among the nearest neighbors in the three-dimensional processor topology. Performance of this sort of operation is shown in Figures 2(c) and 2(d). Global communications are only required for monitoring and synchronization ahead of I/O.

A comparison of dual-core and quad-core performance is show in Table 2. The initial port showed

		Dual-core		Quad-core			
Problem Size	MPI mode	Time	Cost	Time		Cost	
					vec		vec
$30 \times 30 \times 30$	`-n 1 -N 1`	404	150	415	333	154	123
$60 \times 30 \times 30$	`-n 2 -N 2`	465	172	430	349	159	129
$60 \times 60 \times 30$	`-n 4 -N 4`	n/a	n/a	503	422	186	156

Table 2: S3D single processor performance (weak scaling mode).
The amount of work is constant for each process. "MPI mode" refers to the number of MPI processes and how they are assigned to each node: `-n` *is the total number of processes,* `-N` *is the number of processes assigned to each quad-core processor node. Time is wall clock in units of seconds; "cost" is defined as μsec pergrid point per time step. The "vec" columns show the performance after the code was reorganized for stronger vectorization.*

a decrease in performance, though less than that attributable to only the decrease in clock speed. This suggests that vectorization is occurring, though not as aggressively as desired. Special effort was applied to the computation of reaction rates, which was consuming approximately 60% of overall runtime. The resulting vectorization significantly reduced overall runtime.

Table 3 shows the effects of the compiler when able to vectorize code. Although for each category the

	Before	After
Counters	$\times 10^9$ operations	
Add	182	187
Multiply	204	210
Add + Mult	386	397
Load/Store	179	202
SSE	91	212

Table 3: S3D reaction rate computation counters. (Values from dual-core Jaguar.)

number of operations increases, the proportion of operations occurring in vector mode increased by 233%, resulting in a decreasing in runtime of this computation by over 20%.

3.4 AORSA: Fusion Energy

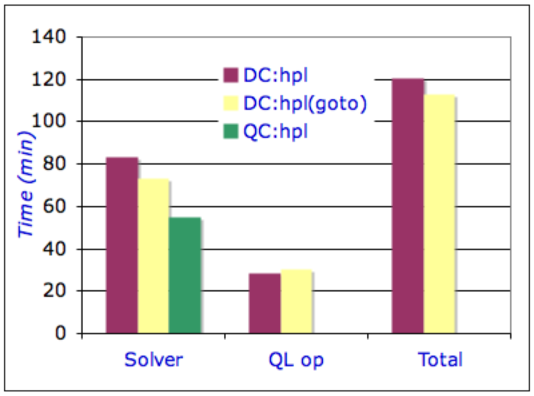
The two- and three-dimensional All-ORders Spectral Algorithm (AORSA[12]) code is a full-wave model for radio frequency heating of plasmas in fusion energy devices such as the International Thermonuclear Experimental Reactor[5] (ITER) and the National Spherical Torus Experiment[6] (NSTX).

AORSA operates on a spatial mesh, with the resulting set of linear equations solved for the Fourier coefficients. A Fast Fourier Transform algorithm converts the problem to a frequency space, resulting in a

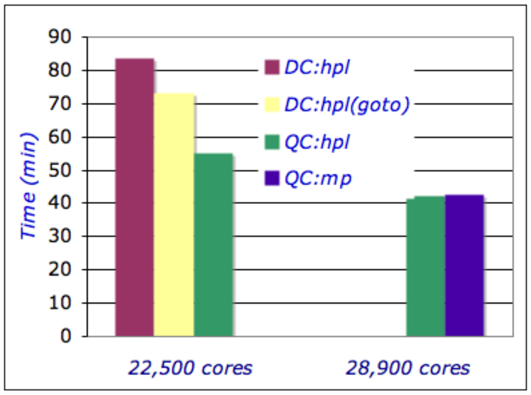
[5]http://www.iter.org
[6]http://nstx.pppl.gov

dense, complex- valued linear system. Parallelism is centered on the solution of the dense linear system, currently accomplished using a locally modified version of HPL[16, 3]. Quasi-linear diffusion coefficients are then computed, which are serve as input to a separate application (Fokker-Plank solver) which models the longer term behavior of the plasma.

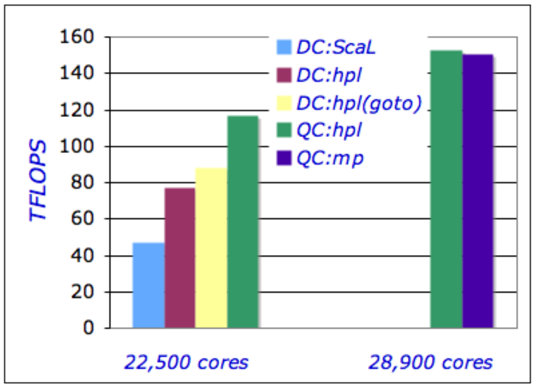
AORSA has a distinguished history running on the XT-series, allowing researchers to conduct experiments at resolutions previously unattainable[2, 1] executing at unprecedented computational scales. For example, the first simulations of mode conversion in ITER were run on the single-core XT3[10] on a 350 × 350 grid. On the dual-core XT3/XT4, this feat was again achieved, at increased resolution (500 × 500 grid), with the linear solver achieving 87.5 TFLOPS[7] (74.8% of peak) on 22,500 cores[11]. This same problem run on the quad-core Jaugar increased this performance to 116.5 TFLOPS, and when run on 28,900 cores performance increased to 152.3 TFLOPS. Performance results for this scale are shown in Figure 7. While impressive,



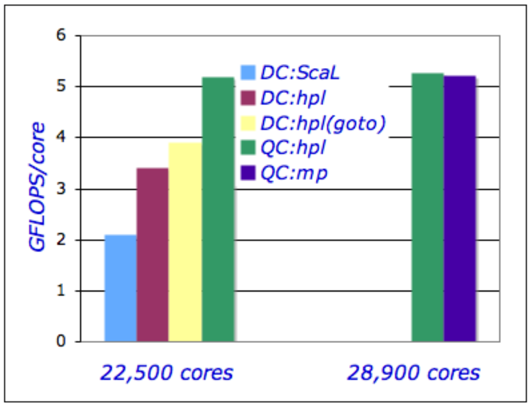
(a) Total time on 22,500 cores



(b) Solver time



(c) Solver TFLOPS



(d) Solver GFLOPS per core

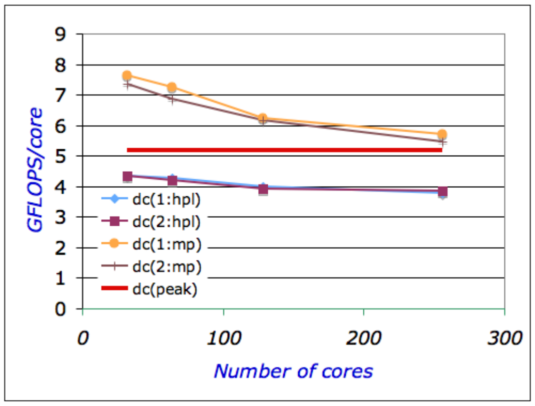
Figure 7: AORSA performance on Jaugar.
Operating on a 500x500 grid, we compare Jaguar dual-core (DC) with the new quad-core (QC), using ScaLAPACK, HPL, then HPL with the GotoBLAS (from then on added the default in `libsci`*) and mixed-precision (MP). Cores counts are 22,500 and 28,900, somewhat dictated by the solver requirement of a 2d logical processor grid.*

relative to the theoretical peak performance has decreased from 74.8% to 61.6%. This is not unexpected due to the decreased clock speed and other issues associated with the increased number of cores per processor, we are pursuing further improvements. However, the time-to-solution (the relevant metric of interest) dropped from 73.2 minutes to 55.0 minutes, a decrease of 33%.

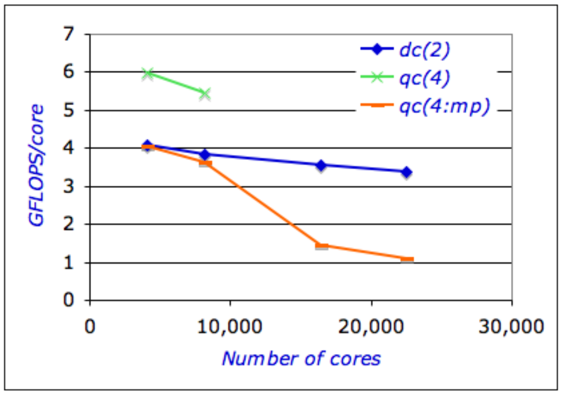
Although we expect performance of the solver phase to increase based on planned improvements to the

[7]Performance of the linear solver is a function of the algorithmic requirements. That is, for matrix dimension N, the number of floating point operations is calculated to be $8/3N^3 + 4/3N^2$ operations, which is divided by wall-clock time to yield the computational rate.

BLAS library and the MPI implementation, we are also experimenting with a mixed-precision approach[15]. This capability is now included in the Cray math library (`-libsci`) as part of the Iterative Refinement Toolkit (IRT). Some early results are shown in Figure 8. While this technique shows promise, it is not



(a) 128x128 grid on dual-core Jaguar.



(b) 350x350 grid on dual- and quad-core Jaguar.

Figure 8: AORSA solver performance : HPL and mixed-precision solver (IRT)

providing an improvement at the relevant problem scales. Although the condition of the matrix increases with resolution, this does not appear to be an issue. More likely is the use of the ScaLAPACK factorization routine within IRT compared with the HPL version: at 22,500 cores on the dual-core Jaguar, ScaLAPACK realized 48 TFLOPS, whereas HPL achieved 87 TFLOPS. Although the magnitude is somewhat surprising, the general distinction is not: ScaLAPACK is a set of reference implementations, and uses computation and communication libraries designed for all of its supported algorithms and data structures. HPL, on the other hand, implements a single algorithm for a single matrix structure using tuned techniques.

The computation of the quasi-linear linear diffusion coefficients consumes the bulk of runtime external of the linear solver. The major task is the solution of a definite integral, involving multiple sweeps across the grid, computing a variety of derivatives. Initial performance was quite disappointing, decreasing somewhat proportionally with the clock speed, strongly hinted that vectorization was not occurring. Reconfiguration of the major (nested) loops involved in this computation resulted in significantly stronger performance, as shown in Figure 9.

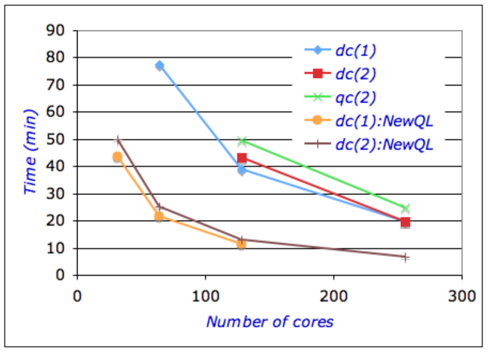


Figure 9: AORSA performance : Computation of the quasi-linear operator on a 128x128 grid, on dual- and quad-core Jaguar.

New experiments are being configured to take advantage of Jaguar's increased capabilities, especially with regard to increased grid resolution. Science and performance results will be presented in future publications.

3.5 DCA++: Materials Modeling

DCA++ is designed to simulate materials where electronic correlations are im- portant using a dynamical cluster approximation or other quantum cluster the- ories. It can use quantum Monte Carlo and also di- agonalisation solvers (such as as Lanczos). The code is based on the psimag toolkit for materials codes, allowing for easy extension, although presently the focus is on solving Hubbard models for the supercon- ducting cuprates. DCA++ is part of our QMOD (quantum models) framework for the study of strongly correlated electrons.

Quantum cluster methods such as the DCA map the problem onto an effective cluster self-consistently embedded in a mean-field. The effective cluster problem is solved with a massively parallel Hirsch-Fye quantum Monte Carlo (QMC) algorithm. Along the QMC Markov chain, measurements of physical quantities such as the single-particle Greens function and two-particle correlation functions are performed. Between the measurements, the Greens function is updated using a Dyson equation. The majority of the CPU time of a typical DCA QMC simulation is spent in the Greens function updates and the measurements. These two inner loops are highly optimized to run efficiently on the NCCS machines. The Greens function updates are given by a vector outer product, This computation is optimized by delaying the update, thus effectively replacing the vector outer product by a slender rectangular matrix-matrix multiply. This allows us to perform the Greens function update very efficiently with the BLAS level 3 subroutine `dgemm`[6]. The other CPU intensive task is the measurement of two- particle correlation functions. In the QMC technique, this reduces to evaluating products of Greens functions which are optimized by transforming from space- time to reciprocal space and frequency. These Fourier transforms are handled using the BLAS level 3 subroutine `zgemm`.

The two inner loops in the Green's function updates have previously been optimized to run efficiently on the NCCS machines: the Green's function updates are given by a vector outer product, $Gc = Gc + a * b$, where the Greens functions Gc and Gc are matrices of size $Nt \times Nt$. To optimize this computation, we delay the update, effectively replacing the vector outer product by a slender rectangular matrix-matrix multiply for matrices of size $32 \times Nt$. This allows us to perform the Greens function update very efficiently with the `dgemm`. Previously, on the Cray X1E installed at ORNL[8] we experienced up to a 5-fold speedup as a result of the delayed updates. The performance of this computation on the quad-core XT4, using the gCC compiler, is shown in Figure 10.

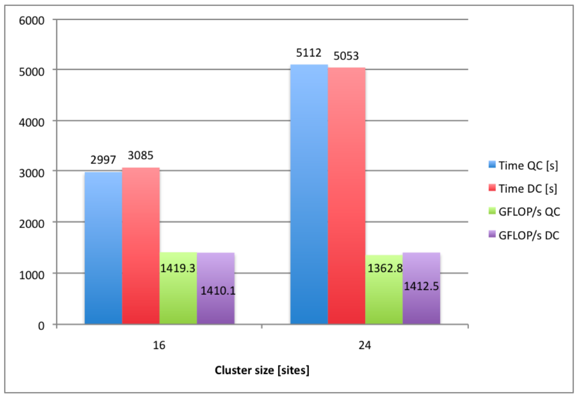


Figure 10: DCA++ performance

The other CPU intensive task is the measurement of two-particle correlation functions. In the QMC technique, this reduces to evaluating products of Greens functions which are optimized by transforming from space-time to reciprocal space and frequency. These Fourier transforms are handled using the `zgemm` and therefore run at speeds near peak performance. The QMC algorithm is parallelized in the standard way for Monte Carlo methods by distributing the Markov chain onto many processors. Several independent, shorter Markov-chain walks on different processors are performed and the final result is obtained by averaging the results of each walk using MPI. Apart from the fraction of the walk required to achieve equilibrium, the result is an almost perfectly parallel speedup with an increasing number of processors. This arises because

[8] `http://www.nccs.gov/computing-resources/phoenix`

no communication is required in the inner loops of the code. As a result, the code scales to several hundreds of processing units with almost ideal speedup.

The DCA++ code is written in the C++ programming language using generic programming models like the C++ Standard Template Library and the Psi-Mag toolkit. BLAS libraries are called for dense linear algebra computa- tions in the inner loops of the code. The MPI library is used for parallelization and communication.

4 Summary

Jaguar, the Cray XT4, located at Oak Ridge National Laboratory has been upgraded in processor, memory, and network. Our "first look" at the performance of a broad set of scientific application programs and micro-benchmarks illustrates that performance increases over the previous Jaguar configuration are possible, though steps must be taken in order to exploit new architecture capabilities.

The compiler flag `-tp barcelona-64` must be included in order to access quad-core specific capabilities when compiling on the Jaguar dual-core login-in nodes. We found the latest MPI implementation, contained in module `mpt3.0`, a significant improvement over the previous version, attributable to the inclusion of a shared memory capability. This should quickly become the default on Jaguar.

The quad-core processor provides the potential for four floating point operations per clock cycle, an increase from the previous two on the dual-core processor. Access to this potential requires that the compiler vectorize the code, a capability hindered by certain coding syntax and semantics. We found it relatively easy to re-organize code so that the compiler can apply its vectorization techniques. Verification of this capability is visible during compilation when generating a loop-mark listing, enabled using compiler flag `-Mlist`. We also recommend examining the information provided by compiler flag `-Minfo`.

We again emphasize that the results reported in this paper must be viewed as preliminary due to our limited access to Jaguar. In some cases the application was compiled and executed once, with no opportunity to analyze performance. (This was the case with LAMMPS, reported in Section 3.1.) In other cases time constraints permitted only rudimentary analysis, which was the case for GYRO, discussed in Section 3.2. When time was spent analyzing performance and making meaningful modifications to the code (as was the case with S3D and AORSA), significant performance gains were made. Its important to note as well that these changes should result in performance improvements on any processor that has vectorization capabilities.

We also illustrated the potential performance improvements of considering alternative algorithm approaches. (AORSA is now configured to use the mixed-precision linear solver technique.) The reality is that clock speeds probably won't increase at the rates seen during the past several years. Instead, performance potential will be provided through processor, memory, and network hierarchies and heterogeneities, so we encourage code developers to explore new algorithms and computational approaches that can exploit these capabilities.

Time constraints prevented us from examining inter-node communication issues. Typically we execute applications using subsets of the cores available on a node, which can expose network contention issues and suggest flow control requirements. The importance of this issue is hinted at in the communication benchmarks shown in Figures 2(a) and 2(b). We intend to examine this issue as soon as possible.

Finally, deeper analysis of the issues discussed in this report, as well as other issues, will be reported in a timely fashion.

Acknowledgements

The application programs cited in this report are developed and maintained by a broad set of projects, representing several science areas, funding by several agencies, including the Department of Energy Office of Science and National Nuclear Security Agency, the Department of Defense, the National Science Foundation, and the National Oceanic and Atmospheric Administration.

John Levesque of Cray, Inc. contributed to the performance improvements reported for S3D. Adrian Tate of Cray configured the mixed-precision solver in AORSA. We had several useful discussions with Brian Waldecker of AMD, Inc. Several members of NCCS system administration provided invaluable assistance.

References

[1] S.R. Alam, R.F. Barrett, M.R. Fahey, J.A. Kuehn, J.M. Larkin, R. Sankaran, and P.H. Worley. Cray XT4: An Early Evaluation for Petascale Scientific Simluation. In *Proceedings of the IEEE/ACM Conference on Supercomputing SC'07*, 2007.

[2] S.R. Alam, R.F. Barrett, M.R. Fahey, J.A. Kuehn, E.O.B. Messer, R.T. Mills, P.C. Roth, J.S. Vetter, and P.H. Worley. An Evaluation of the Oak Ridge National Laboratory Cray XT3. *International Journal of High Performance Computing Applications*, 22(1):52:80, 2008.

[3] R.F. Barrett, T. Chan, E.F. D'Azevedo, E.F. Jaeger, K. Wong, and R. Wong. A complex-variables version of high performance computing LINPACK benchmark (HPL). 2008. In preparation.

[4] J. Candy. A 5-dimensional array distribution algorithm for distributed memory computers. 2002.

[5] J. Candy and R. Waltz. An Eulerian gyrokinetic-Maxwell solver. *Journal of Computational Physics*, 186(545), 2003.

[6] J.J. Dongarra, J. DuCroz, I. Duff, and S. Hammerling. A set of level 3 basic linear algebra subprograms. *ACM Trans.on Math. Soft.*, 16:1–17, 1990.

[7] M. Fahey. Portable performance optimizations based on a performance history of the fusion code GYRO. In *IEEE/ACM Conference on Supercomputing SC'06, Poster*, 2006.

[8] Mark Fahey and Nick Jones. Design, implementation, and experiences of third-party software administration policies at the ORNL NCCS. In *Proceedings of the 50th Cray User Group*, 2008.

[9] E. R. Hawkes, R. Sankaran, J. C. Sutherland, and J. H. Chen. Direct numerical simulation of turbulent combustion: fundamental insights towards predictive models. In *Journal of Physics: Conference Series, vol. 16*, 2005.

[10] E.F. Jaeger, L.A. Berry, S.D. Ahern, R.F. Barrett, D.B. Batchelor, E.F. D'Azevedo, R.D. Moore, R.W. Harvey, J.R. Myra, D.A. A'Ippolito, R.J. Dumont, C.K. Phillips, H. Okuda, D.N. Smithe, P.T. Bonoli, J.C. Wright, and M. Choi. Self-consistent full-wave and Fokker-Planck calculations for ion cyclotron heating in non-Maxwellian plasmas. *Physics of Plasmas*, 13, May 2006.

[11] E.F. Jaeger, L.A. Berry, R.F. Barrett, and E.F. D'Azevedo et al. Simulation of high power ICRF wave heating in the ITER burning plasma. In *Proceedings of the 49th Annual Meeting of the Division of Plasma Physics of the American Physical Society*, volume 52, 2007. Bulletin of the American Physical Society.

[12] E.F. Jaeger, L.A. Berry, E. D'Azevedo, D. B. Batchelor, and M. D. Carter. All-orders spectral calculation of radio-frequency heating in two-dimensional toroidal plasmas. *Phys. Plasmas*, 8(5):1573–1583, 2001.

[13] M. Karplus and G.A. Petsko. Molecular dynamics simulations in biology. *Nature*, 1990.

[14] Suzanne Kelly and Ron Brightwell. Software architecture of the lightweight kernel, catamount. In *Proceedings of the 47th Cray User Group*, 2005.

[15] J. Langou, J. Langou, P. Luszczek, J. Kurzuk, A. Buttari, and J. Dongarra. Tools and techniques for performance—exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems). In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing (SC06)*, 2006.

[16] A. Petitet, R.C. Whaley, J.J. Dongarra, and A. Cleary. HPL: A portable high-performance LINPACK benchmark for distributed-memory computers. `http://www.netlib.org/benchmark/hpl`, January 2004.

[17] S.J. Plimpton. Fast parallel algorithms for short-range molecular dynamics. *Journal of Computational Physics*, 117, 1995.

[18] T. J. Poinsot and S. K. Lele. Boundary-conditions for direct simulations of compressible viscous flows. *Journal of Computational Physics*, 101, 1992.

[19] J. C. Sutherland. *Evaluation of mixing and reaction models for large-eddy simulation of nonpremixed combustion using direct numerical simulation.* PhD thesis, Dept of Chemical and Fuels Engineering, University of Utah, 2004.

[20] J. C. Sutherland and C. A. Kennedy. Improved boundary conditions for viscous, reacting, compressible flows. *Journal of Computational Physics*, 191, 2003.

[21] C. S. Yoo, Y. Wang, A. Trouve, and H. G. Im. Characteristic boundary conditions for direct simulations of turbulent counterflow flames. *Combustion Theory and Modelling*, 9, 2005.