Analyzing Multicore Characteristics for a Suite of Applications on an XT5 System

Courtenay T. Vaughan and Douglas W. Doerfler,
Sandia National Laboratories[1]

ABSTRACT: *In this paper, we will explore the performance of applications important to Sandia on an XT5 system with dual socket AMD 6 core Istanbul nodes. We will explore scaling as a function of the number of cores used on each node and determine the effective core utilization as core count increases. We will then analyze these results using profiling to better understand resource contention within and between nodes.*

KEYWORDS: XT5, application performance

1. Introduction

The next capability computing machine being acquired by the National Nuclear Security Administration's (NNSA) Advanced Scientific Computing Program (ASC), code named Cielo, will be based on Cray's "Baker" architecture [1]. Baker will be a dual-processor node, like the XT5, and the processor (also referred to as a socket in this paper) will be AMD's recently announced 6100 series processor, and in particular the 8-core model 6136, providing 16 cores per node. Sandia's current large machine, Red Storm, which is a mixture of CRAY XT3 and XT4 boards, has a single socket per node with a mixture of dual-core and quad-core processors. In preparation for Cielo, it is desirable to better understand the effects of higher numbers of cores on application performance, Cielo will not become available until later in 2010, but Sandia has recently acquired a CRAY XT5 that does provide higher levels of node parallelism than Red Storm and was used for this early investigation into higher processor core counts. The XT5 has 160 compute nodes. These nodes are dual-socket with 6 core AMD Istanbul processors clocked at 2.4 GHz with 32 GB of 800 MHz DDR2 memory per node. The XT5 is configured as a 6 x 4 x 8 3D torus and uses SeaStar 2.2 for the interconnect. The XT5 is running CNL 2.2.41 and the applications were compiled with PGI version 9.0.2.

We will use a suite of applications to study the effects of memory and high-speed network contention due to the larger number of cores per socket. The analysis will look at both the effect of a dual-socket vs single-socket node and the effect of a larger number of cores per socket.

2. Applications

In this paper, we have chosen to use the suite of ASC applications that were chosen to benchmark Cielo performance. These applications include CTH, Charon, AMG2006, UMT2006, SAGE, and xNOBEL. The descriptions of these codes follow. These applications are representative of the Cielo workload and are composed of applications from all three of ASC's national laboratories, Sandia, Los Alamos, and Lawrence Livermore.

A. CTH

CTH is an explicit, three-dimensional, multimaterial shock hydrodynamics code which has been developed at Sandia for serial and parallel computers. It is designed to model a large variety of two- and three-dimensional problems involving high-speed hydrodynamic flow and the dynamic deformation of solid materials, and includes several equations of state and material strength models [2]. CTH is written mostly in FORTRAN 77 with a little bit of C code.

The numerical algorithms used in CTH solve the equations of mass, momentum, and energy in an Eulerian finite difference formulation on a three-dimensional Cartesian mesh. CTH can be used in either a flat mesh mode where the faces of adjacent cells are coincident or in a mode with Automatic Mesh Refinement (AMR) where the mesh can be finer in areas of the problem

[1] This research was sponsored by Sandia National Laboratories, Albuquerque, New Mexico 87185, and Livermore, California 94550. Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

where there is more activity. For this study we will be using the code in a flat mesh mode and a shaped-charge problem that scales with the number of processors.

B. Charon

Charon is a transport reaction code that was developed to simulate the performance of stockpile semiconductor devices under irradiation [3]. Finite element discretization of the drift-diffusion equations produces a large sparse, strongly coupled nonlinear system which is solved via a fully-coupled Newton-Krylov algorithm. The time to solution is strongly dependent on how efficiently the potentially very large linear systems can be solved. The particular model investigated for this study involves the steady-state solution of the drift-diffusion equations for a silicon NPN bipolar junction transistor. Charon is written in mostly C++, but uses a variety of libraries that are written in C, C++, and FORTRAN.

C. xNOBEL

xNOBEL is a Continuous Adaptive Mesh Refinement (CAMR) code that models hydrodynamics with adaption and high-explosive burn modeling. This code is based on the radiation hydrodynamics code RAGE and on the HE burn code NOBEL [4]. This benchmark is a 3D simulation of a 105 mm shaped charge calculation. xNOBEL is written in a combination of FORTRAN and C.

D. SAGE

SAGE is SAIC's Adaptive Grid Eulerian hydrocode, a multidimensional, multimaterial hydrodynamics code with adaptive mesh refinement that uses second-order accurate numerical methods [5]. We used a standard problem called timing_c, which uses adaptation and heat conduction, with 250000 cells per processor. SAGE is mostly written in FORTRAN 90.

E. UMT2006

UMT is a 3D, deterministic, multigroup, photon transport code for unstructured meshes. The transport code solves the first-order form of the steady-state neutral-particle Boltzmann transport equation. The benchmark for Cielo is written in a combination of Python and C++, but for this study we are using the C++ version of the code. The code also is capable of using both MPI and OpenMP based parallelism, but for this study, we use only the MPI based parallelism.

F. AMG2006

AMG is a parallel algebraic multigrid solver for linear systems arising from problems on unstructured grids [6]. The simulation is the solution of the Laplace equations on the unit cube. It is discretized with standard finite differences to yielding 7 point stencils in 3D. It is written in C using MPI for parallelization. It also has the option to use OpenMP directives, but we will not be using those for this study. AMG is written in C.

3. Results for Red Storm and XT5

Figure 1 shows a comparison of CTH running on the quad-core nodes of Red Storm and on the XT5, with both utilizing all of the cores available on a socket. The quad-core processors on Red Storm are 2.2 GHz AMD Budapest processors with 800 MHz DDR2 memory and the nodes are connected by a SeaStar 2.2 interconnect. Red Storm is running Catamount CNW 2.1.56.1. Note that the performance for Red Storm and the XT5 using one socket per node is very close despite the XT5 running on two-thirds the number of nodes. We do see a difference in the performance on the XT5 when using one socket per node versus two sockets per node, which seems to get larger as the number of cores used gets larger and gets to about 7% at 512 cores. When most of the communication is on-node or between a few nodes, the difference is small but gets larger as the communication moves off node.

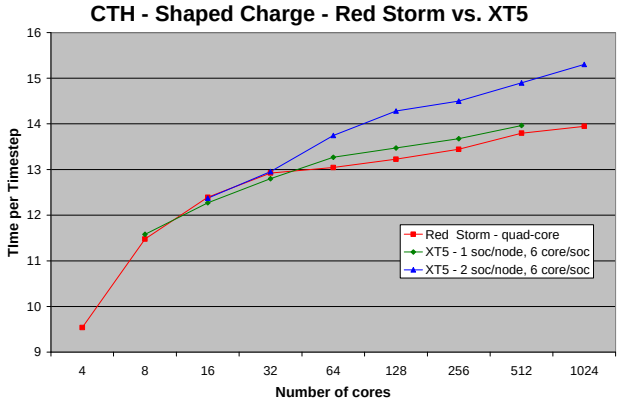

Figure 1. CTH on Red Storm and XT5

Figure 2 shows a comparison of CTH running on Red Storm and the XT5 while using only 4 cores per socket on the XT5. The Red Storm and XT5 single-socket results have a similar shape to their curves with the difference between the results being fairly consistent with Red Storm being about 18% slower than the XT5. Since the XT5 processor is clocked 9% faster than those on Red Storm, it appears other improvements to the processors were made which may explain the difference in performance. We again see a difference in the XT5

single-socket versus dual-socket results that starts small, but then seems to be fairly constant as scale increases.

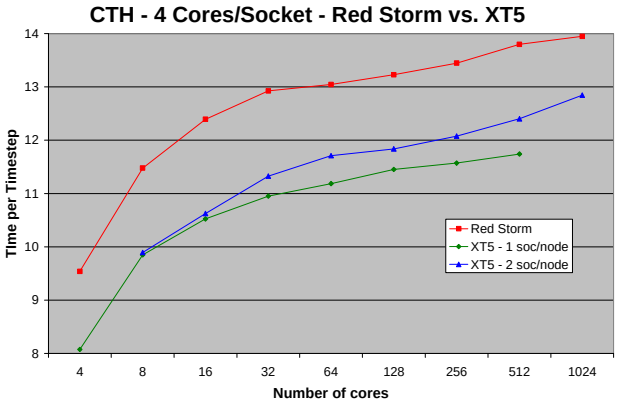

Figure 2. CTH on Red Storm and XT5 using 4 cores/socket

In Figure 3, we compare the two machines running CTH on 128 cores and varying the number of cores per socket used. In this case, we find that the difference between Red Storm and the XT5 while running with one core per node is about 8%, which is fairly close to the difference in clock speeds between the two machines. As the number of cores per socket used increases, the difference in run time between the two machines also increases, getting to about 16% when running with 4 cores per socket and using only one socket per node on the XT5. Since the communication network is the same on both machines, this seems to indicate that the Istanbul processors in the XT5 are better than the Budapest processors in Red Storm at pulling data from main memory when several cores are contending for access to memory. The performance difference between single-socket and dual-socket XT5 performance grows from about 2% at one core per socket to 6.5% at six cores per socket.

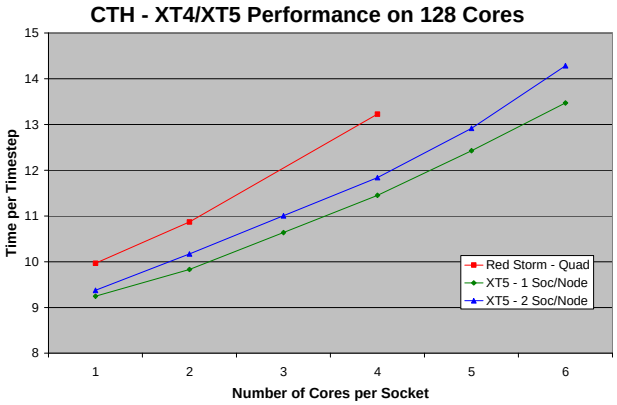

Figure 3. CTH on 128 cores of Red Storm and XT5

4. Application Results on XT5

We ran all six of the codes on 128 cores of the XT5 and varied the number of cores used per socket and the number of sockets per node. The results are shown in Figure 4.

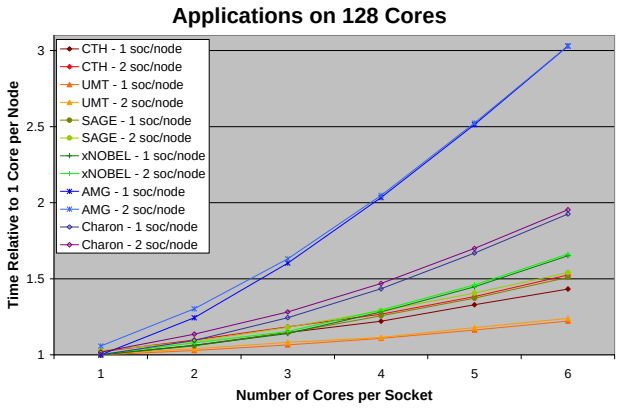

Figure 4. Applications onXT5

The times for all of the applications in Figure 4 were normalized by dividing by the application time using one core per node. Most of the codes show a modest increase in run time as more cores are used per socket. The average for the six applications running on all of the cores on a node is a runtime of 1.83 times the runtime of using one core per node. This represents an effective use of about 6.6 cores per node. Most of this time is a result of increased contention for memory access on the sockets. As each additional core is added per socket, the time differential grows nonlinearly. This would indicate that contention increases as core count increases.

AMG performance is the most sensitive to increasing core count. We profiled the code using CrayPat and the amount of computation time goes up by a factor of three while the MPI time decreases slightly as the number of cores used per socket goes from 1 to 6. While the L1 cache hit rate (98.8%) is similar to CTH (98.9%), the L2 cache hit rate is 11.9% while for CTH it is 45.1%. This seems to reflect the nature of an algorithm like multigrid using memory in a more unpredictable fashion and requiring more main memory accesses and creating more contention for the processor's main memory controller.

The time difference for each application when using one socket per core and two sockets per core represents the overhead from having both sockets competing for the NIC. For most codes, the difference remains fairly steady as a percentage of the application time. The two exceptions for this are AMG and CTH. In the case of AMG, the time difference between running one socket

per node and two sockets per node decreases as the number of cores increases to the point where the time for running on 6 cores per socket using both sockets per node is slightly less than that while using only one socket per node. We ran AMG using CrayPat to profile the code and found that the vast majority of the MPI time is in synchronization for MPI_ALLREDUCE operations. This is consistent with the code developers' observation that on a large number of processors, more than 90% of the time can be spent in these operations. Except for running with one core per socket, the time is less when running with two sockets per node than running with one socket per node and the time decreases as the number of cores used per socket increases. This would indicate that the system is performing MPI_ALLREDUCE operations more efficiently on node than between nodes. The amount of time for MPI decreases from about 36% for one core per socket to about 15% for six cores per socket.

We also profiled CTH running the shaped charge problem on 128 cores with CrayPat and the results are shown in Figure 5. We have run CTH with another problem which is more load balanced and the overall results are similar.

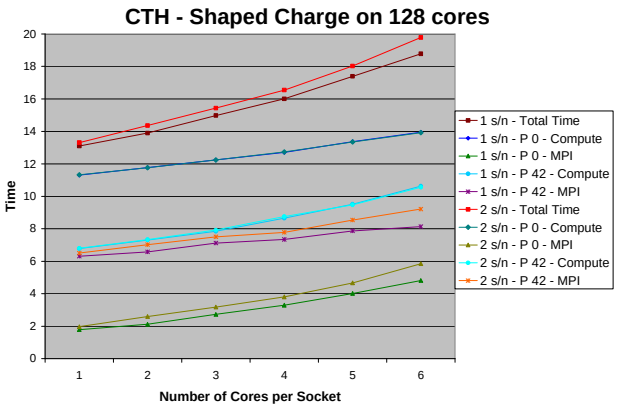

Figure 5. Profile of CTH on 128 cores

In this figure, we plotted the times for the computational portion of CTH and the time spent in MPI for two different ranks within the calculation, while running on a varying number of cores per socket and both one and two sockets per node. We chose rank 0, which has the computational hotspot for this calculation but is on the spatial edge of the calculation and gets updates from three of its neighbors, and rank 42, which is an internal rank in the calculation and thus gets updates from six of its neighbors. The total problem time agreed for rank 0 and rank 42 and is thus plotted as simply total time. Notice that the computational time for both ranks does not vary with the number of sockets used per node. This validates our earlier observation that the difference between the line for one socket per node and two sockets per node is due to the difference in the MPI time.

The computational time goes up by 23% for rank 0 while going up by 40% for rank 42. This is explained by the fact that rank 0 is the computational hotspot and is sharing a socket with ranks which have a lower computational load. Therefore, as the number of cores per socket goes up, it can use some of the memory bandwidth that the ranks that it shares the node with are not using. Rank 42, on the other hand, is sharing a socket with ranks that have a similar memory bandwidth need and has a bit more of a slowdown as a result.

For this problem on 128 cores, CTH sends large (about 5 MB) messages several times per iteration (from 54 to 108 depending on rank). If we compare the MPI times for rank 0 and rank 42, the largest difference that we find is that the time for operations such as MPI_RECV and MPI_WAIT are much higher for rank 42 than for rank 0. This time is in excess of the time expected since these operations were called more often on rank 42 since it does more communication than rank 0. In CTH, these operations have a synchronizing effect on the ranks, since a rank sends and receives from its neighbors and can not continue calculations until it has finished the communications. The MPI times on rank 0 go up by a factor of 2.7 when running on one socket per node and by a factor of 3 when running on two sockets per core. These factors in MPI time on rank 0 represent contention for the NIC and for the memory on the socket. If we look at the MPI time for a given number of cores on node, we find that the time is longer if we look at the case where those nodes share a socket. For example, we can run with 6 cores on a node either by running with 3 cores per socket and using both sockets or by running with 6 cores on one socket. In the latter case, the MPI time is 51% larger than for the former case. On the other hand, on rank 42, the MPI time goes up by a factor of 1.29 while running on one socket per node and by a factor of 1.42 while running on two cores per node.

In order to study the communication time for CTH, we have a program that closely simulates the communication pattern of CTH for this problem CTH_Comm. The comparison of the MPI time from rank 0 and CTH_Comm is shown in Figure 6.

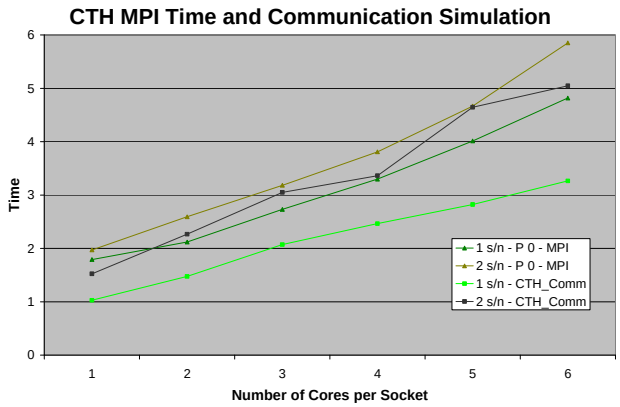

Figure 6. CTH MPI time and communication simulation

The simulator uses the same communication pattern CTH uses with similar message sizes. This figure shows that we have captured the overall effect of the communication in CTH with the simulator. The time per timestep is a little smaller for the simulator than for the code, which seems to indicate that there is more of an interaction with the code for CTH than we capture in the simulator. However, the trends are similar to those that occur with the code. The simulator time is not as smooth when running on two sockets per node. Part of that seems to occur with the time for using four cores per socket being smaller than expected, which would correspond with communications in some directions being contained on a node and not having to go out onto the network. The results for the communications simulator also seem to indicate that some of the MPI time above for rank 42 would be waiting for operations to complete.

5. Conclusions and Future Work

We ran a suite of applications on a Cray XT5 and Red Storm and examined their performance. We have found that performance of CTH on the XT5 is similar to that on Red Storm. On average on 128 cores, the suite of applications effectively used about 6.6 cores out of the 12 available per node on the XT5. Most of the slowdown is a result of contention on a socket for memory and access to the communications network. Most of the codes showed a 1% to 3% additional slowdown on 128 cores due to the network contention when using both of the sockets on an XT5 node.

Building on this work, we plan to model the behavior of these applications in order to predict the behavior of the applications on machines with higher levels of node parallelism, include Cielo.

About the Authors

Courtenay Vaughan is a Senior Member of Technical Staff at Sandia National Laboratories. He can be reached at Sandia National Laboratories, P. O. Box 5800, MS 1319, Albuquerque, New Mexico 87185, E-Mail: ctvaugh@sandia.gov.

Douglas Doerfler is a Principle Member of Technical Staff at Sandia National Laboratories. He can be reached at Sandia National Laboratories, P. O. Box 5800, MS 1319, Albuquerque, New Mexico 87185, E-Mail: dwdoerf@sandia.gov.

References

1. Sudip Dosanjh and John Morrison, “An Alliance for Computing at the Extreme Scale”, Cray Users Group Conference, May 2010.

2. E. S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. McGlaun, S. V. Petney, S. A. Silling, P. A. Taylor, L. Yarrington, “CTH: A Software Family for Multi-Dimensional Shock Physics Analysis,” *Proceedings, 19th International Symposium on Shock Waves* 1, 274ff (Université de Provence, Provence, France) (1993).

3. P. Lin, J. Shadid, M. Sala, R. Tuminaro, G. Hennigan, and R. Hoekstra, “Performance of a Parallel Algebraic Multilevel Preconditioner for Stabilized Finite Element Semiconductor Device Modeling,” to be published by the Journal of Computational Physics. Available online at http://dx.doi.org/10.1016/j.jcp.2009.05.024

4. Gittings, M. L., Weaver, R. P., et. al., “The RAGE radiation-hydrodynamic Code”, in Computational Science and Discovery, Vol. 1, (2008)

5. D. J. Kerbyson, H. J. Alme, A. Hoise, F. Petrini, H. J. Wasserman, and M. Gittings, “Predictive Performance and Scalability Modeling of a Large-Scale Application”, in *Proceedings of the ACM/IEEE International Conference on High-Performance Compution and Networking (SC 2001)*, November 2001.

6. Van Emden Henson and Ulrike Meier Yang, “BoomerAMG: A Parallel Algebraic Multigrid Solver and Preconditioner”, Appl. Num. Math. 41 (2002), pp. 155-177.