Application Characteristics and Performance on a Cray XE6

Courtenay T. Vaughan,
Sandia National Laboratories[1]

ABSTRACT: *In this paper, we will explore the performance of two applications on a Cray XE6 and their performance improvement from previous machines, including the XT5 and the XT6. These two applications show different scaling effects as we go from machine to machine and we will explore the differences in the applications to explain these differences. We will use profiling and other tools to better understand resource contention within and between nodes and how that changes with the evolution of the machines with changes in processors and network.*

KEYWORDS: XE6, XT5, XT6, application performance

1. Introduction

The newest capability machine for the Advanced Simulation and Computing (ASC) Campaign is a Cray XE6 called Cielo [1]. The machine currently consists of 6654 compute nodes which are dual-socket oct-core AMD Magny-Cours nodes clocked at 2.4 GHz connected with Cray's Gemini interconnect in a 3D torus, for a total of 106464 cores capable of 1.02 PFLOPS. Each node is connected to 32 GB of 1.333 GHz DDR3 memory. The final system, to be delivered later this year will have 8944 compute nodes and be rated at 1.37 PFLOPS. For this study, we also utilized some smaller XE6 systems which were purchased with Cielo and utilize the same node architecture.

For a short period of time, Cielo and the smaller XE6 machines were configured as XT6 machines which use the XE6 nodes with a SeaStar 2.2 interconnect. Using results collected on the machines in this configuration allows us to isolate the effects of the network and the node architecture. Unfortunately, the machines did not exist for a long enough time period to gather all of the data that we would have liked for this paper.

Sandia also has an XT5 with 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 CLE 2.2.41 and the applications were compiled with PGI version 9.0.2.

The XT5 node consists of two 6 core processors. Each processor is its own NUMA region and is connected to 16 GB of DDR2 memory. The node for the XT6 and XE6 consists of two oct-core processors. Each oct-core processor is divided into two 4 core NUMA regions with each NUMA region connected to 8 GB of 1.333 GHz DDR3 memory. In [2], we examined the effects of memory contention while using varying number of cores per NUMA region on the XT5 and showed that that contention can have a large effect on code performance. Dividing the oct-core processor into two NUMA regions, each with their own path to memory, has the effect of limiting that performance derogation.

The other architectural difference moving from the XT6 to the XE6 is the change from the SeaStar interconnect to the Gemini interconnect. Where the SeaStar interconnect has one NIC per node, each Gemini chip serves two nodes. The network still has the same number of logical connections in each dimension of the machine with every other hop in the Y direction taking place within the Gemini. The effect of the Gemini interconnect is that the bandwidth between nodes is increased some, while the injection bandwidth is increased by an order of magnitude [3].

2. Applications

In this paper, we have chosen to use two Sandia codes, CTH and PRONTO, due to their differences in the way these applications run on the different machines.

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

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 [4]. 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 where there is more activity. CTH uses message aggregation during communication phases which results in a few large messages for each communication phase.

For this study we will be using the code in a flat mesh mode and using two problems. The first is a shaped-charge problem (Figure 1) and the second is a meso-scale impact problem which is better load balanced. Both problems scale with the number of processors used.

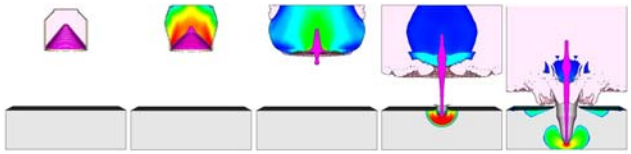

Figure 1. Shaped-charge problem.

Figure 2 shows the communication matrix for CTH on 64 cores for the shaped-charge, on the left, and the meso-scale problem, on the right. The difference between the two diagrams is that the shaped-charge problem is divided onto the processors in a 2 by 8 by 4 grid of processors, while the meso-scale problem is divided onto the processors in a 2 by 4 by 8 grid of processors.

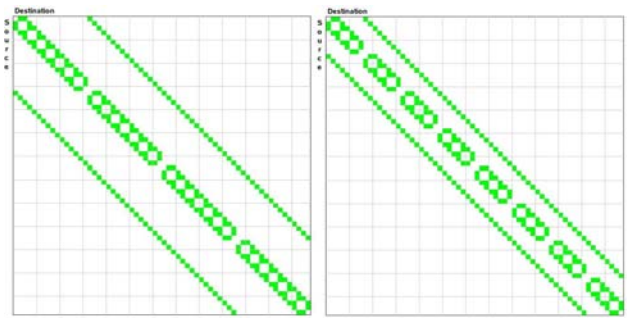

Figure 2. Communication matrices for CTH

Figure 3 shows the communication trace from CrayPat for one timestep for CTH on 64 cores. The top trace is from the shaped-charge problem while the bottom trace is from the meso-scale problem.

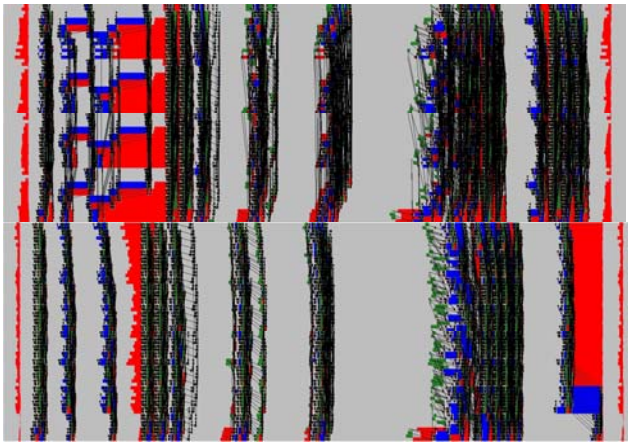

Figure 3. Communication traces for one timestep of CTH

In these figures, black is point to point communication, red is synchronization time waiting for collectives or for communication to finish, green is send, blue is receive, and gray is computation. These figures show that both problems behave similarly overall, but have different amounts of synchronization time which occur at different points in the timestep. CTH has several points in a timestep where boundary information is aggregated and exchanged with up to six neighbors in the grid of processors. For the shaped-charge problem, these messages average 5.2 MB and for the meso-scale problem, these messages average 10.4 MB. Toward the end of the timestep, the shaped charge problem shows some additional communication that is a result of the high explosive calculations.

B. PRONTO

PRONTO is an explicit Lagrangian, finite element code developed for modeling transient solid mechanics problems involving large deformations and contact which was developed at Sandia for parallel and serial computers [5]. The problem is statically partitioned for the finite element portion of the code while the parallel contact algorithm utilizes a second dynamically created decomposition of the surface of the problem to resolve the contact. Both phases of the computation use lots of small to medium sized messages for communication. PRONTO is written in mostly FORTRAN 90 with the parallel contact algorithm being written in C.

For this study, we are using two problems. The first is the “walls” problem in which two sets of two brick

walls each collide. The combination of the brick size and number of bricks is chosen such that none of the bricks is divided when the problem is decomposed onto the processors. This results in a problem in which communication is not required during the finite element portion of the computation, but is during the contact portion of the computation. Figure 4 shows the initial configuration of the problem (the one visible wall obscures the other three walls) with the different colors representing the different processors which own each brick and the portion of the problem that a processor has with the blue bricks in one for the front two walls traveling to the back and the pink bricks in the back traveling to the front. The figure is illustrated with 64 cores, but with a reduced number of blocks per processor so it would be easier to show what is happening. This problem is engineered to exercise the contact algorithm.

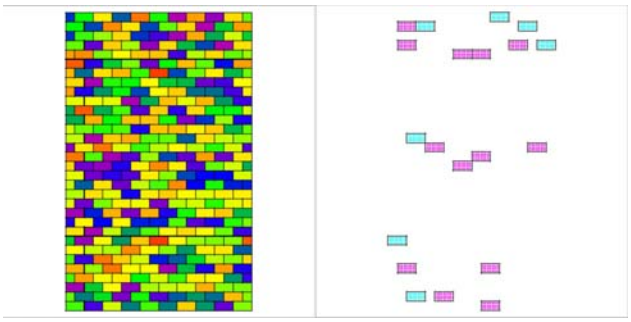

Figure 4. Walls problem and one processor's portion

The second problem is the "can crush" problem in which a cylinder is crushed by a block and is illustrated in Figure 5. This is a more balanced problem in that there is communication for both the finite element and contact phases of the computation and the amount of contact is more representative of typical problems for PRONTO.

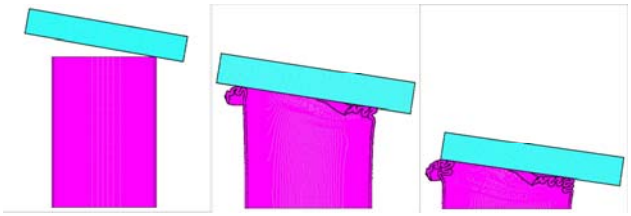

Figure 5. Can crush problem.

Figure 6 shows the communication matrices for PRONTO on 64 cores for these two problems. The matrix on the left is for the walls problem, while the one on the right is for the can crush problem. The matrix for the walls problem illustrates that every processor has to communicate with almost every other processor due to the fact that a processors portion of the problem is spread out throughout the domain. The matrix for the can crush problem is more sparse since the structural analysis portion of the problem is more constrained. This matrix will change slightly during the calculation as the decomposition for the contact algorithm changes.

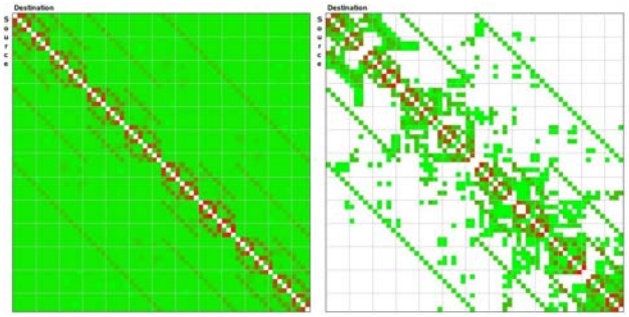

Figure 6. PRONTO communication matrices

Figure 7 shows communication traces from CrayPat for both problems on 64 cores. The top trace shows one timestep for the walls problem, while the bottom trace is one timestep from the can crush problem. These traces show some limitations of CrayPat by the fact that the synchronization points are not lined up for all of the processors. The communication pattern for the contact algorithm is on the left side of both traces, while the communication for the finite element portion of the code is on the right side. That communication is missing for the walls problem since it does no communication for this step.

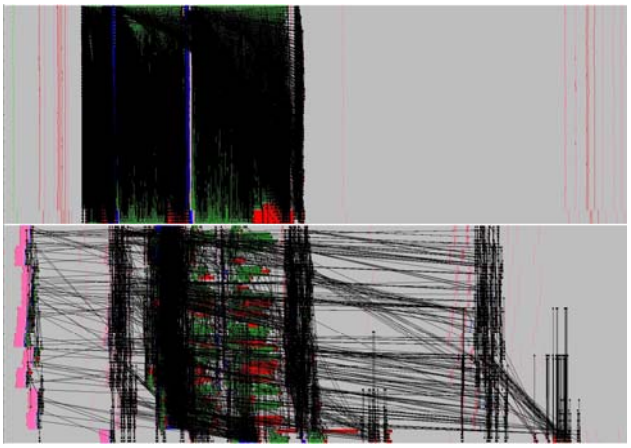

Figure 7. Communication traces for PRONTO

3. Results for XT5, XT6, and XE6

Figure 8 shows a comparison of CTH times running both the shaped-charge (sc) and the meso-scale (meso) problem running for 100 timesteps on the XT5, the XT6, and the XE6.

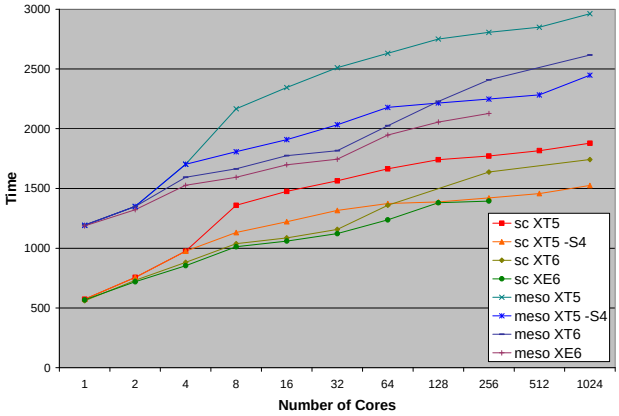

Figure 8. CTH on XT5, XT6, and XE6

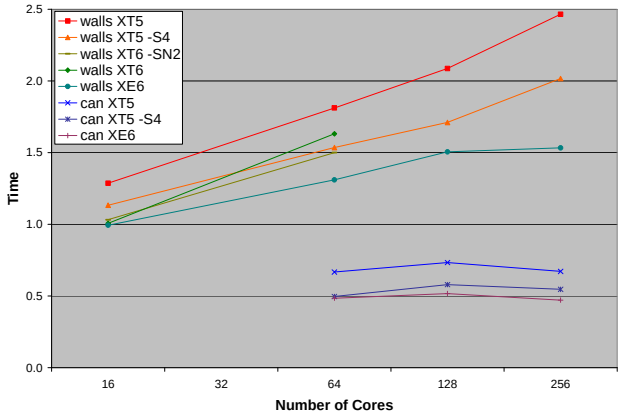

Figure 9. PRONTO on XT5, XT6, and XE6

Overall, CTH shows up to a 30% performance increase (decrease in time) going from the XT5 using all the cores on a node to the XE6 which is also using all of the cores on a node. The performance on the XT6 is similar to the performance on the XT5 using only 4 cores per NUMA region. The XT6 has better performance on both problems up to 64 cores and for 128 or more cores, the XT5 using 4 core per NUMA region has better performance with its performance coming close to that of the XE6. As the number of cores increase, the largest performance difference seems to be between the XT5 and the XT5 using only 4 cores per NUMA region. A large portion of the difference is the memory contention within the processors and a smaller portion of that difference is contention for the NIC. The difference between the XT6 and the XE6 seems to be growing, which shows the difference between the SeaStar and Gemini interconnect. The differences between the XT5 using 4 cores per NUMA region and the XT6 as the number of cores increases shows the effect of contention for the network bandwidth becoming a larger effect than the slightly better performance within a NUMA region due to better memory bandwidth.

Overall, CTH shows some improvement going from the XT5 to the XE6 as a result of better node architecture, fewer cores per NUMA region, and some network bandwidth improvement from moving from the SeaStar to the Gemini network.

Figure 9 shows a comparison of PRONTO running the walls problem (walls) and can crush (can) on the XT5, XT6, and XE6. The timing for the walls problem is for one timestep, while that of the can crush problem is for 5 timesteps. The can crush problem runs faster than the walls problem since the problem is somewhat smaller for a given number of cores, the contact calculation is much simpler, and the communication requirements are less per timestep.

For the walls problem, the XE6 shows up to a 38% performance increase (decrease in run time) from the XT5 to the XE6 and, unlike CTH, shows up to a 25% performance increase from the XT5 using 4 cores per NUMA region. The results for the XT6 show that for 16 cores (1 node) that performance is similar to the XE6, but for 64 cores, the XT6 performance is worse than that of the XT5 using 4 cores per NUMA region, while if we only use two NUMA regions per node with XT6, then its performance is similar to that of the XT5 using 4 cores per NUMA region. The can crush problem shows less but a more consistent performance advantage for the XE6 over the XT5, while the performance of the XT5 using 4 core per NUMA region seems to be getting further from that of the XE6 as the number of cores increases.

While PRONTO shows some improvement from the node architecture changes, it seems to show a greater improvement from the move from the SeaStar to the Gemini network.

4. Interpretation of Results

Both of the codes that we ran on the XT5 showed memory contention when using all 6 cores per NUMA region. Both codes showed about a 20% performance improvement by using 4 cores per NUMA region.

To look at the influence of the NIC and to explore how the behaviors of the codes differ, we compiled figure 10, which captures some of the communication differences between the two codes. The number of messages is messages per minute, while the amount of data transferred is in KB per second to allow both to be plotted on the same graph.

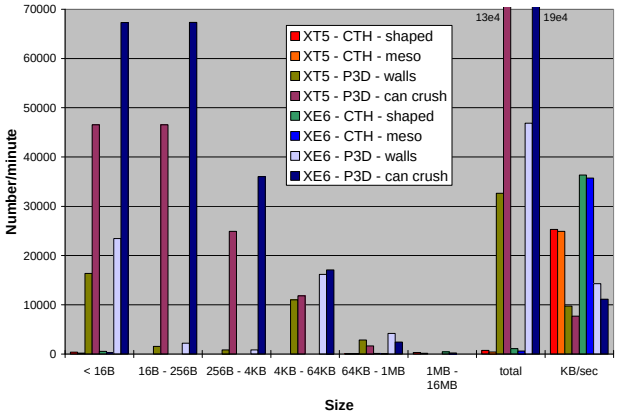

Figure 10. Communication behavior of CTH and PRONTO

CTH sends a few large messages per timestep, which results in the total number of messages being low, but the total amount of information transferred is large. PRONTO, on the other hand, sends quite a few more messages, but the total amount of information transferred is less than half of the amount the CTH transfers. From the run-time communication traces, both codes tend to use network resources in a contentious manner, with CTH making use of the bandwidth by having all of the processors trying to send large messages at the same time, while PRONTO has all of the processors sending lots of messages at the same time.

This helps to explain the performance differences that we see with CTH and PRONTO over these machines. For CTH, bandwidth is important, so as the number of cores gets larger and a larger percentage of message traffic is over the network, the XT5 using 4 cores per NUMA region gets competitive with the XE6 since the XT5 is using more NICs. PRONTO utilizes the increased message injection rate of the Gemini by sending lots of small messages, and as the number of cores increase, the performance difference between the XT5 and the XE6 gets larger.

5. Conclusions and Future Work

We ran CTH and PRONTO on a couple of problems each on an XT5, an XT6, and an XE6. Both codes showed memory contention within a NUMA region on the XT5, but running with 4 cores per NUMA region brought that inline with the contention that we see on the XT6 and XE6. Once we did that, CTH and PRONTO showed different behavior on the machines. With its communication bandwidth requirement, CTH showed little difference between the machines, while PRONTO with its use of lots of small messages showed more improvement moving to the XE6 with the Gemini interconnect.

In the future, we would like to be able to run these codes on larger numbers of cores to see if our observations continue to hold. We are also writing a mini-app to model the communication requirements of CTH and see if we can use the increased message rate to speed up the communications in that code.

About the Author

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.

References

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

2. Courtenay T. Vaughan and Douglas W. Doerfler, “Analyzing Multicore Characteristics for a Suite of Applications on an XT5 System”, Cray User Group Conference, May 2010.

3. Courtenay Vaughan, Mahesh Rajan, Richard Barrett, Douglas Doerfler, and Kevin Pedretti, “Investigating the Impact of the Cielo Cray XE6 Architecture on Scientific Application Codes”, workshop on Large-Scale Parallel Processing at the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS), May 2011.

4. 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).

5. S. W. Attaway, B. A. Hendrickson, S. J. Plimpton, D. R. Gardner, C. T. Vaughan, K. H. Brown, and M. W. Heinstein, “A Parallel Contact Detection Algorithm for Transient Solid Dynamics Simulation Using PRONTO3D”, *Computational Mechanics*, Vol. 22, pp. 143-159 (1998).