The Effects of System Options on Code Performance

Courtenay T. Vaughan, *Sandia National Laboratories*[1]

ABSTRACT: *There are several options that can be used to run codes on a Cray XT3. In this paper, we will examine the effect of choice of page size, eager or non-eager communication protocol, and choice of malloc has on performance of several codes at different numbers of processors. We will also analyze code characteristics and correlate those to the differences in performance.*

KEYWORDS: Red Storm, XT3, application performance, catamount

1. Introduction

In this paper, we will investigate the effects of three options for running codes on an XT3. The choice of page size and whether or not to use the eager communication protocol are runtime options, while the choice to replace the standard malloc with GNU malloc is a compile time option. We ran several codes with each of these options over a range of number of processors and present those results. We then analyze those results and present some results using the hardware counters gathered through PAPI to explain these results.

Most of these comparisons were run on Sandia's Red Storm computer. Red Storm is a CRAY XT3 with 12960 nodes that are connected in a 27 x 20 x 24 mesh. The mesh is a torus in the z direction. The nodes are dual-core 2.4 GHz AMD Opterons. These runs were run using one core per node on a maximum of 2048 processors. Some of the results were run on our 2 cabinet test system, which is a Cray XT3 with 160 single-core 2.0 GHz AMD Opteron nodes.

We ran each code over a range of processor counts. For each test, we ran the code with all combinations of the options in case the options interacted with each other. For a given code and a given number of nodes, all of the tests were run on the same nodes of the machine, one after another, to minimize variations in the mesh and with possible interference from other jobs on the machine. Most of the runs were run only once since our early experience with our test system indicated that this was sufficient.

2. Available XT3 Options

One of the options that we investigated is the choice of page size. These pages are used by the processor to access memory. For the current AMD Opterons, there is the choice of large or small pages, where large pages are 2 Mbytes and small pages are 4 Kbytes. In order for a memory location to be accessed, that location is first looked up in the TLB (Translation-Lookaside Buffer). If the location is present in the TLB, it improves the speed of virtual address translation. In small page mode, there are 256 entries in the TLB, while in large page mode, there are 8 entries. The default for the XT3 is large pages and to run with small pages, one must specify "-small_pages" on the command line when the job is launched.

Another option is the option to use the eager communication protocol, where messages are sent from one processor to another before there is an acknowledgement that there is space for the message to be received. By default, this option is used for small messages, while for large messages a rendezvous protocol is used where there is an acknowledgement that there is space for the message. When this option is used, all of the messages are sent using the eager protocol and are resent if there is not enough space. To use this option, the environment variable MPI_PTL_EAGER_LONG has to be set.

The third option is the choice of malloc to use in the code. There is a default malloc provided by Catamount (the compute node operating system for XT3), which is optimized for large memory allocations. There is an

[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.

option to replace it with GNU malloc. To use this option, -lgmalloc is specified on the link line when a code is compiled.

3. Applications and Results

In this paper, we have chosen to use some benchmarks from the HPCC benchmark suite to try to get a baseline for these options. We then tested with several applications which are commonly used on the machine. These applications are CTH, PARTISN, SAGE, and Presto.

A. HPCC

The HPC Challenge benchmark suite [1] provides a variety of benchmarks that span the space of processor and network performance for parallel computers. We have chosen to use five of the benchmarks in this study with the hope of capturing application behavior in a simpler framework. The benchmarks that we are using are HPL (factor a large dense matrix) which emphasizes processor performance, PTRANS (matrix transposition) which tests network bisection bandwidth, STREAMS (vector operations) which tests memory performance,

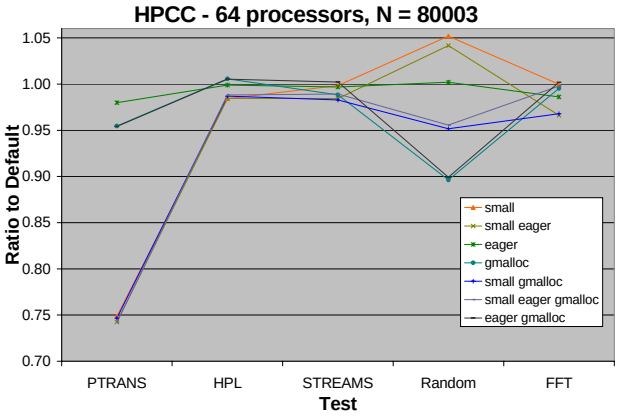

Figure 1. HPCC on 64 processors

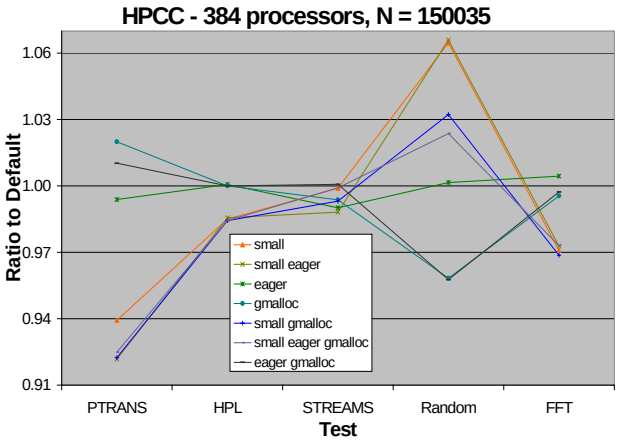

Figure 2. HPCC on 384 processors

RandomAccess (modify random memory locations across the entire machine) which stresses small message network performance, and FFT (a large 1-D Fast Fourier Transform) which is a coupled processor and network test. Results for HPCC are given in figures 1 and 2.

These figures show all of the options compared to the default configuration which is large pages, not to use the eager communications protocol, and catamount malloc. Three of the tests HPL, STREAMS, and FFT seem to indicate that any of the options are either neutral or bad for their performance. Also it seems that overall, the eager communications protocol seems to make little difference. In the case of the Random Access test, small pages seem to make a positive difference while GNU malloc is bad for performance. For PTRANS, small pages are bad for performance, while GNU malloc is slightly better for the 384 processor case. In the 64 processor case, each processor in PTRANS will exchange information with one other processor, while in the 384 processor case, there is several communication phases.

B. 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 where there is more activity. We will be using the code in a flat mesh mode for this study.

For this study, we will be using a shaped-charge problem that scales with the number of processors. The shaped-charge consists of a cylindrical container filled with high explosive capped with a copper liner. When the explosive is detonated from the center of the back of the container, the liner collapses and forms a jet. The problem is run in quarter symmetry. The simulation consists of those three materials and a fourth material that forms a target for the jet.

Results for the shaped charge problem for CTH are shown in figure 3. Lower values are better.

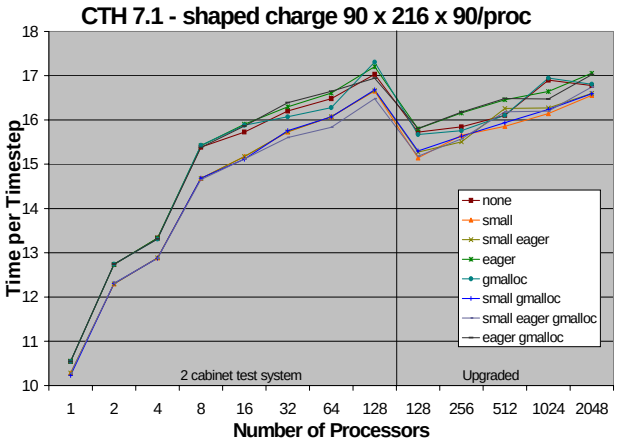

Figure 3. CTH – shaped charge problem

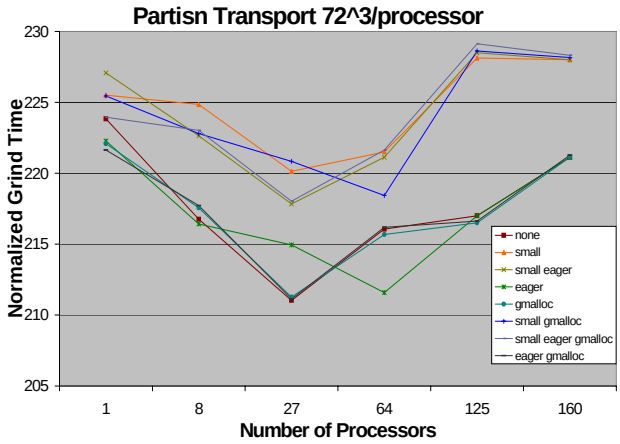

Figure 4. PARTISN transport phase

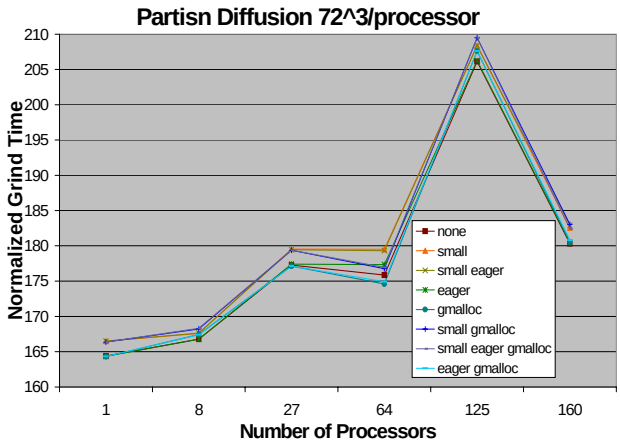

Figure 5. PARTISN diffusion phase

CTH was run both on the two cabinet test system and on Red Storm after Red Storm's upgrade to 2.4 GHz dual core processors. The 128 processor test case was run on both systems and was repeated to confirm that the results from the two systems were consistent. The results indicate that there is a small but consistent benefit for using small pages. The timings seem to have a little more variation on larger numbers of processors, but the trend in consistent.

C. PARTISN

The Parallel, Time-dependent SN (PARTISN) code is designed to solve the time-independent or dependent multigroup discrete ordinates form of the Boltzmann transport equation in several different geometries [3]. PARTISN provides neutron transport solutions on orthogonal meshes with adaptive mesh refinement in 1D, 2D or 3D. Much effort has been devoted to making PARTISN efficient on massively parallel computers. The package can be coupled to nonlinear multiphysics codes that run for weeks on thousands of processors to finish one simulation. The test problem is the Sntiming problem, in which flux and eigenvalue convergence are monitored by PARTISN. The results are shown in figures 4 and 5.

The PARTISN code is mostly FORTRAN and was run only on the two cabinet test system. In these figures, better performance is indicated by a smaller grind time. PARTISN has two phases in the calculation, there is a transport phase and a diffusion phase, and both phases are run in each iteration of the code. With this code, page size again seems to be the dominative difference among the options, with large pages giving better performance than small pages and the difference seems to be bigger for the transport phase of the computation.

D. Presto

Presto is a Lagrangian, three-dimensional explicit, transient dynamics code for the analysis of solids subjected to large, suddenly applied loads [4]. It is built on the SIERRA framework [5], which supplies a data management for a parallel computing environment. Presto is designed for problems with large deformations, nonlinear material behavior, and contact. There is a versatile element library incorporating both continuum and structural elements. The contact algorithm is supplied by ACME [6]. The contact algorithm detects contacts that occur between elements in the deforming mesh and prevents those elements from interpenetrating each other. This is done on a decomposition of just the surface elements of the mesh. The contact algorithm is communication intensive and can change as the problem progresses. The SIERRA framework and Presto are written in C++.

The simulation used in this investigation is the Brick Walls problem consists of two sets of two brick walls

colliding with each other. It is a weak scaling problem where each processor is assigned 80 bricks. Each brick is discretized with 4 x 4 x 8 elements, for a total of 10240 elements per processor. Due to the way the problem is distributed on a parallel computer, each brick is located on one processor so the only communication for the finite element portion of the code is for the determination of the length of the next timestep. The contact portion of the calculation is communication intensive since it is done on a different decomposition which involves only the surface areas of the mesh. The results for Presto are shown in figure 6.

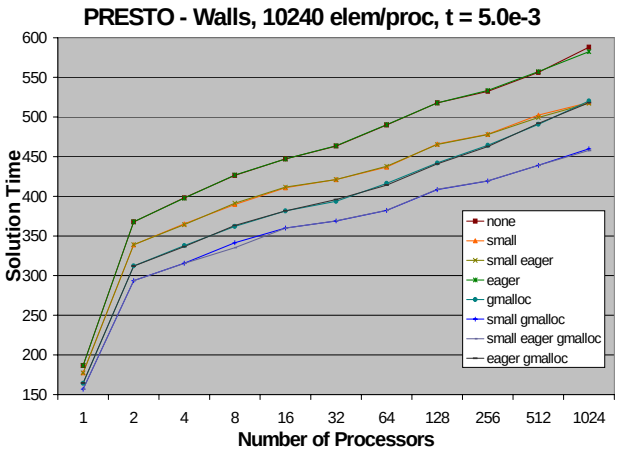

Figure 6. Presto brick walls problem

In this case, there are two options which make a difference in the run time. As we have seen in other codes, small pages benefit Presto. However, in this case, GNU malloc has a larger positive effect on performance. This is probably due to the fact that Presto is a C++ code and makes a larger use of memory allocation during the calculation.

E. 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 [7]. 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. Results for SAGE are shown in figure 7.

For the SAGE results, a larger number of cell-cycles/sec/proc indicates better performance. For this simulation with SAGE, small pages gives a performance improvement from about 45% to 60%, depending on the number of processors. The other options do not seem to make a large difference.

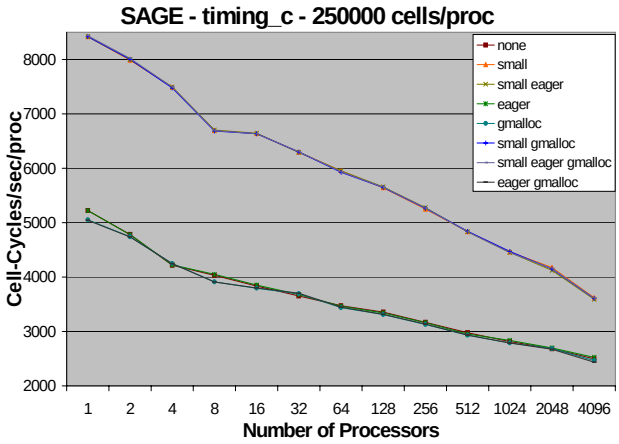

Figure 7. SAGE timing_c problem

4. PAPI Results for Small Pages

Since most of the benchmarks and applications were affected by using small pages, we instrumented several of the applications and looked at the hardware counters using PAPI to see if we could explain how this option affects performance. Table 1 shows the results from processor 0 of these runs. The applications are ordered based on the ratio of performance between the code run with small pages and the code run with large pages, with the applications that benefit most from large pages at the top of the table.

App	Page size	TLB miss	Cache access	Access /miss	Cache hit	FP intensity
PTRANS	large	$3.30e^{7}$	$6.37e^{10}$	1934	98.2%	0.150
64 proc	small	$1.03e^{9}$	$6.54e^{10}$	63.8	98.2%	0.144
HPL	large	$5.57e^{8}$	$3.16e^{12}$	5670	99.1%	1.693
64 proc	small	$2.98e^{8}$	$3.17e^{12}$	10644	99.1%	1.686
PARTISN	large	$1.48e^{9}$	$6.14e^{11}$	414	92.8%	0.558
32 proc	small	$1.38e^{9}$	$6.18e^{11}$	449	92.7%	0.554
STREAM	large	$1.73e^{4}$	$4.25e^{9}$	$2.45e^{5}$	93.5%	0.344
64 proc	small	$6.97e^{6}$	$4.34e^{9}$	623	93.6%	0.333
FFT	large	$2.03e^{7}$	$3.34e^{9}$	164	97.6%	0.466
64 proc	small	$1.52e^{7}$	$3.30e^{9}$	217	97.6%	0.441
CTH	large	$1.08e^{10}$	$1.28e^{12}$	119	96.9%	0.570
32 proc	small	$3.30e^{9}$	$1.32e^{12}$	398	97.0%	0.588
Random	large	$8.50e^{8}$	$4.32e^{10}$	50.8	99.5%	0.0
64 proc	small	$1.01e^{8}$	$4.57e^{10}$	452	99.5%	0.0
SAGE	large	$2.90e^{9}$	$1.29e^{11}$	44.5	98.7%	0.226
32 proc	small	$2.29e^{7}$	$1.02e^{11}$	4457	98.3%	0.231

Table 1. PAPI results for applications

For some of the applications and benchmarks in table 1, the measured statistics may include operations which were not included in the performance measurements. In particular, a couple of the benchmarks, such as STREAMS and PTRANS, have statistics for the entire test, not just for the test for which results are presented.

In these two cases, all of the subtests are consistent in that large pages are better by about the same amount. The statistics for these tests also contain measurements for the portion of the test that checks the results. The statistics for PARTISN contain both phases of the calculation since it is an iterative process and both phases are performed during every iteration.

There are several things that can be seen from table 1. The first is that the L1 cache hit rate is largely independent from the page size and has no effect on which page size is most beneficial. Except for the Random Access benchmark which does no floating point calculations, the floating point intensity indicates which page size is better for performance. That statistic seems to be a result rather than a cause of better performance. There also does not seem to be any correlation between floating point intensity and which page size gives better performance.

What seem to correlate with performance is a combination of the number of L1 data cache accesses and the number of TLB misses. For codes which show the best performance with small pages, the ratio of data cache accesses to TLB misses is larger for small pages than for large pages. However, the reverse does not always apply. In the case of the STREAMS benchmark, that ratio is almost 400 times larger for large pages, but the performance improvement for large pages is less than 1%. The performance does seem to correlate a little better if one looks only at the number of L1 data cache accesses. It looks like there are other factors at work here.

5. Conclusions and Future Work

There are a couple of observations that we can draw from the data presented here. The first is that trends observed from benchmarks do not always translate over to real application codes.

The results indicate that GNU malloc should be tried if one is using a C++ application. This may be the case for some other applications that do a lot of dynamic memory allocation and deallocation. Most of the other applications that we ran will malloc a large amount of memory when the application starts and never change its memory configuration.

The option to use the eager message protocol for all messages seems to have little effect on any of the codes that we ran for this test.

For most codes, using small pages seems to result in better performance, and the difference can be large. In the cases where it does not help, it does not seem to hurt much. The number of TLB entries for large pages would seem to be a limiting factor. With large pages, there is a larger portion of memory that can be accessed without a TLB miss, but it is located in large contiguous blocks. If the memory that is being accessed is spread throughout memory, then small pages may allow it to be accessed with fewer misses. We would like to try this experiment again when the quad-core Opteron processors are available since they will have a larger number of TLB entries in large page mode.

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. P. Luszczek, J. Dongarra, D. Koester, R. Rabensiefner, R. Lucas, J. Kepner, J. McCalpin, D. Baily, and D. Takahasi, "Introduction to the HPC challenge benchmark suite," March 2005, http://icl.cs.utk.edu/hpcc/pubs/index.html.

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. R. E. Alcouffe, R. S. Baker, J. A. Dahl, S. A. Turner, and Robert Wart, "PARTISN: A Time-Dependent, Parallel Neutral Particle Transport Code System," LA-UR-05-3925 (May 2005).

4. J. Richard Koteras and Arne S. Gullerud, *Presto User's Guide Version 1.05*, Sand Report SAND2003-1089, April 2003.

5. Edwards, H. C., and Stewart, J. R., "SIERRA: A Software Environment for Developing Complex Multi-Physics Applications", First MIT Conference on Computational Fluid and Solid Mechanics, Bathe, K.J., editor, Elsevier Scientific, 2001.

6. Kevin H. Brown, Randall M. Summers, Micheal W. Glass, Arne S. Gullerud, Martin W. Heinstein, and Reese E. Jones, "ACME Algorithms for Contact in a Multiphysics Environment API Version 1.0,"Sand Report SAND2001-3318, October 2001.

7. 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.