Performance Comparison of Scientific Applications on Cray Architectures

Haihang You, Reuben Budiardja, Vince Betro, Bilel Hadri, Pragneshkumar Patel, Jeremy Logan, Mark Fahey, Lonnie Crosby
National Institute for Computational Sciences
The University of Tennessee, Knoxville, TN
hyou,reubendb@utk.edu

***Abstract*—Current HPC architectures are changing drastically and rapidly as compared to mature scientific applications, which usually evolve at a much slower rate. Newly introduced architectures promise to impact the performance of these heavily used scientific applications. Therefore, it is prudent to understand how the supposed performance benefits and improvements of new architectures translate to the applications. In this paper, we attempt to quantify the differences between the theoretical performance improvements (due to changes in architecture) and the "real-world" improvements in applications by gathering performance data of selected applications from the fields of chemistry, climate, weather, materials science, fusion, and astrophysics running on three different Cray architectures: XT5, XE6, and XC30. Of particular interest is the fact that these three Cray platforms spans three different generations of interconnects, namely: SeaStar2+, Gemini, and Aries, respectively and that all three platforms (or its variant) are either in active use or coming online at HPC centers. Therefore, a performance evaluation of these selected applications on these three different architectures gives a user some useful perspective into the benefits of these architectures. These evaluations are done by comparing the improvements of numerical (micro)-benchmarks to the improvements of the selected applications when run across these architectures.**

***Keywords*-Performance; Benchmarking; Computer architecture**

I. Introduction

Twice each year the Top500 [1] releases a list of the fastest machines in the world. The recent Top500 performance lists show continued steady growth towards exascale. While current HPC architectures are changing drastically and rapidly, scientific applications evolve at a much slower pace. A key question is whether a particular application would capitalize on the performance improvement promised by a system upgrade. In this paper, we attempt to quantify the differences between the theoretical performance improvements (due to changes in architecture) and the "real-world" improvements in applications by gathering performance data of selected applications from the fields of chemistry, climate, weather, materials science, fusion, and astrophysics running on three different Cray architectures: XT5, XE6, and XC30.

The High Performance LINPACK (HPL) [2] benchmark is the de-facto standard for performance measurement of a supercomputer. Service Units (SU), usually measured as core-hours, are used as currency on many HPC systems. These SUs are typically consumed by a job according to the node count and total runtime of the job. An exchange rate of SUs between two systems may be calculated using core-count-normalized HPL results using all of the available computing resources on each of the two systems. An SU conversion rule can then be implemented to account for the resource consumption differences between systems. This is especially important for a cyberinfrastructure provider that has diverse computing resources. For example, such a conversion rule is implemented among the resources that XSEDE [3] provides. XSEDE is a leading distributed cyberinfrastructure for open scientific research in the United States. It consists of sixteen systems with different designs and capabilities. In most cases however, these SU conversion rates do not take issues such as application scalability and problem size into consideration. An application's performance ratio between two systems may differ greatly from the *official* one derived from the HPL benchmark.

As a guideline to evaluate application performance differences across these three Cray architectures, this paper utilizes the conversion system described above. First we derive the SU conversion rates for XT5, XE6, and XC30 by comparing the HPL benchmark results on these systems. The conversion rate gives us the expected *ideal* performance improvements. Then we compare these conversion rates to the real-world performance of the selected applications.

The rest of this paper is organized as follows. In section II we describe the three Cray systems, XT5, XE6 and XC30 used in this study. In section III, we describe applications and present benchmarking results and our evaluation. We will discuss SU conversions based on application performance results in section IV. Finally, we conclude our work and discuss future directions in section V.

II. System Overview

National Institute for Computational Sciences located at Oak Ridge National Laboratory hosts three Cray supercomputers: Kraken, a Cray XT5; Ares, a Cray machine composed of XE6 and XK6 (with NVIDIA X2090 accelerator) nodes[1]; and Darter, a Cray XC30 (a.k.a. Cray Cascade). The

[1]In this paper, we will only examine the performance of the XE6 portion of Ares.

	Kraken(XT5)	Ares(XE6)	Darter(XC30)
Processor	6-core 2.6GHz AMD Opteron	16-core 2.2GHz AMD Opteron	8-core 2.6GHz Intel Xeon
Socket/node	2	2	2
Cores/socket	6	16	8
Memory Interface	DDR2-800MHz	DDR3-1.3GHz	DDR3-1.6GHz
Peak GFLOPS/core	10.4	8.8	20.8
Peak GFLOPS/node	124.8	281.6	332.8
HPL GFLOPS/core	8.14	6.61	16.8
HPL GFLOPS/node	97.68	211.5	268.89
L3 Cache (MB)	1	8	20
Memory Size/node (GB)	16	32	32
Memory Bdwth/node (GB/s)	25.6	83.5	102.4

Table I: Compute node specification.

Comparison	Ares/Kraken Ratio	Darter/Kraken Ratio
Peak FLOPS/core	0.85	2
Peak FLOPS/node	2.26	2.67
HPL FLOPS/core	0.81	2.06
HPL FLOPS/node	2.16	2.75
Memory Bandwidth	3.26	4.00

Table II: Per node ratios between Kraken, Ares, and Darter.

compute node specifications of each machine are listed in Table I.

After normalization, Ares and Darter are at least twice as powerful (in terms of FLOPS/node) as Kraken. The `HPL FLOPS/core` ratio in Table II, also serves as the ideal SU conversion rate between those machines.

1) Kraken: Kraken, a Cray XT5 supercomputer, has a total of 9,408 compute nodes, each of which has two 2.6 GHz hexa-core AMD Opteron 2435 processors. There are 112,896 cores in total with a peak performance of 1.17 PetaFLOPS. Each compute node has 16 GB memory and is attached to a Lustre parallel file system. Kraken's high-speed low-latency network is made of Cray SeaStar2+ chips on board plus a scalable 3D torus interconnect fabric.

2) Ares: Ares, a Cray XE6m machine, has 16 XK6 compute nodes, each with 16 GB of memory, a single-socket 16-core AMD 2.2GHz Opteron processor, and an NVIDIA X2090 GPGPU with 6 GB of memory. It also has 20 XE6 nodes with 32 GB of memory and two sockets of 16 core AMD 2.2GHz Opteron processor. All nodes are connected via a Gemini interconnect in a 2D torus configuration. In this paper, we only examine the performance of the dual-socket (XE6) opteron portion of Ares.

3) Darter: Darter, a Cray XC30 supercomputer, has a total of 748 compute nodes, each node has two 2.6GHz octa-core Intel Sandy Bridge processors. The peak performance is 248.9 TeraFLOPS with total of 11,968 cores. There is 32GB of memory on each node. A Cray Sonexion Lustre parallel storage file system is directly attached to this system, providing high-performance I/O. The newly designed Aries interconnect has 500GB/s switching capacity and utilizes the Dragonfly network topology.

III. Application and Experiment Results

In order to provide a fair performance comparison across these three platforms, we chose representative applications from a broad spectrum of research areas. We describe each application and discuss its performance in detail in the following subsections.

A. HOMME

The High Order Method Modeling Environment (HOMME) [4] is a highly scalable, global hydrostatic atmospheric modeling framework based on the spectral element and discontinuous Galerkin methods. It has been integrated into the Community Atmospheric Model (CAM), the atmospheric component of the CCSM. HOMME relies on a cubed-sphere grid, where the planet Earth is tiled with quasi-uniform quadrilateral elements, free from polar singularities. HOMME is the first dynamical core ever that allows for full two-dimensional domain decomposition in CAM. HOMME is linked with netcdf and libsci for using Lapack functions. We compiled with HOMME with Intel Compiler 13.1.0 on XC30, PGI Compiler 12.3.0 on XE6, and PGI Compiler 11.9.0 on XT5 respectively. In Figure 1, we show performance of HOMME in time and total SU consumption. We can see that HOMME exhibit almost linear scalability on the three machines. SU consumption is consistent with different number of nodes on every platform.

B. WRF

The Weather Research and Forecasting (WRF) model is a mesoscale numerical weather predicting system used for both weather research and real-world forecasting needs with a large world-wide user base. WRF was initially developed

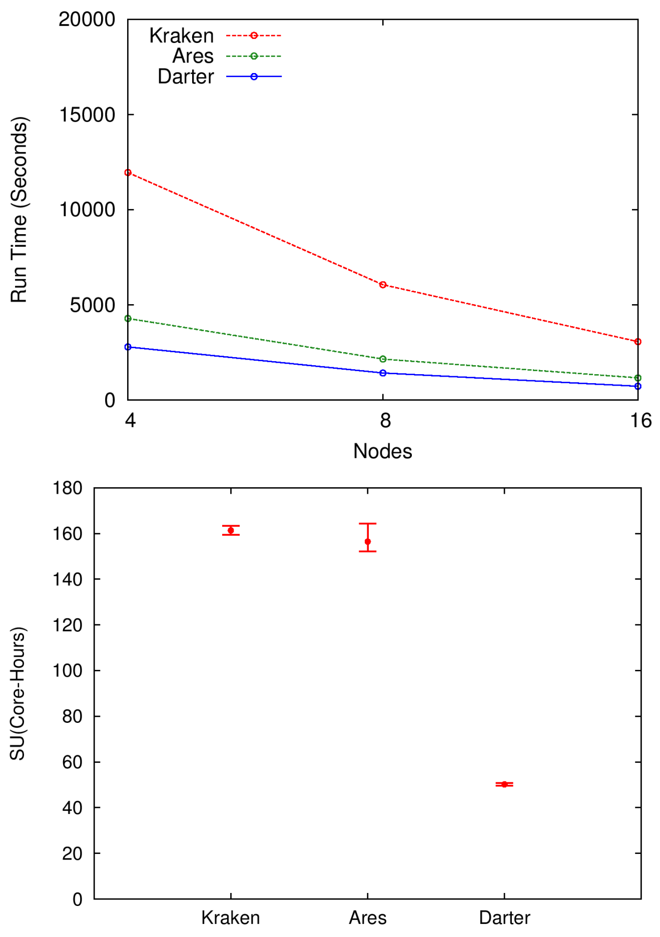

Figure 1: HOMME performance comparison in Time and SU consumption

by partnership among National Center for Atmospheric Research (NCAR), the National Oceanic and Atmospheric Administration (NOAA), the Air Force Weather Agency (AFWA), the Naval Research Laboratory, the University of Oklahoma, and the Federal Aviation Administration (FAA). WRF serves both the forecasting and research needs by its ability to produce simulations based on observational data and idealized atmospheric conditions, while continually incorporating advances in physics and numerics in weather research.

For this paper we utilized WRF version 3.1.1. Although this version is OpenMP capable, only MPI is used for our tests. There are two test cases available: *standard* case and *large* case. These cases are based on a 6025 $\times$ 6025 km domain centered (as close as the resolution will allow) on the Arctic Region Supercomputing Center in Fairbanks, Alaska. The *standard* case has a grid resolution of 675 $\times$ 675 $\times$ 28, while the *large* case has a grid resolution of $3038 \times 3038 \times 28$.

For our experiments, WRF was compiled using PGI compiler version 11.9 on Kraken, 12.3 on Ares, and 12.10 on Darter. WRF requires NetCDF for reading input and writing output data files. We use Cray provided (parallel) NetCDF with HDF5 support version 4.1.2 on Kraken (via module `netcdf-hdf5parallel/4.1.2`), 4.2.1 on Ares (`netcdf-hdf5parallel/4.2.0`), and 4.2.1.1 on Darter (`cray-netcdf-hdf5parallel/4.2.1.1`). WRF was specified to use the serial NetCDF for our tests, although parallel NetCDF I/O is also an option. With serial I/O, for the *standard* case MPI rank 0 is responsible for all I/O. For the *large* case, we used WRF capability of asynchronous I/O called "quilted I/O" by assigning the last 32 MPI ranks exclusively deal with I/O.

Figure 2 shows the strong scaling of WRF *standard* case run on Kraken, Ares, and Darter. The scaling behavior of WRF on Kraken and Ares matches fairly well for job size with 64 cores to 256 cores. On Kraken, the strong scaling trend starts to reverse for job size larger than 256 cores. This eventually results in worse performance of 512-cores as compared to the 256-cores jobs. On the contrary, on Ares the strong-scaling trend continues to 512-cores while maintaining efficiency. Since per-core FLOPS are similar on Kraken and Ares, the differences in this scaling behavior may be due to the network architecture differences. Ares with its Gemini interconnect has higher bandwidth and lower latency than Kraken with its older generation SeaStar+ interconnect. Another possible cause for the differences is the per-node core density. Ares XE nodes has 32 cores per node, while Kraken only has 12 cores per node. Therefore the same number of cores requires less inter-node communication on Ares. On Darter, a similar trend is observed. Strong-scaling is maintained on Darter all the way up to 512 cores for this test case with near-perfect efficiency.

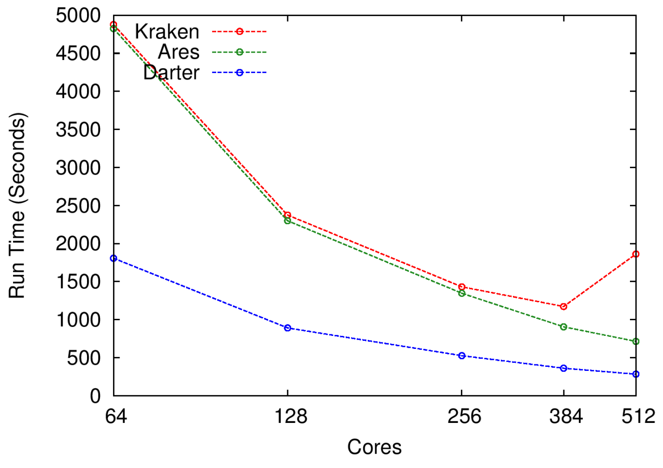

Figure 2: WRF strong-scaling plots on Kraken, Ares and Darter.

One noticeable thing from the plots in Figure 2 is the speed difference between machines. This is more apparent in Figure 3 where average SU usage resulting from different numbers of cores is shown. The minimum and maximum SU measurements are also shown to give some insight on the variation of scaling behavior on different machines. If the code strong-scales well on the machine, the variation (e.g. min to max of SU) should be fairly small. On Kraken, we see large variability of SU. This is caused by the reversing trends of efficiency we have seen on Figure 2. On Ares, the strong scaling is much better, as indicated by the smaller variation on the plot. On Darter, this code strong-scaling is even better than on Ares as indicated by the very small variation of SU.

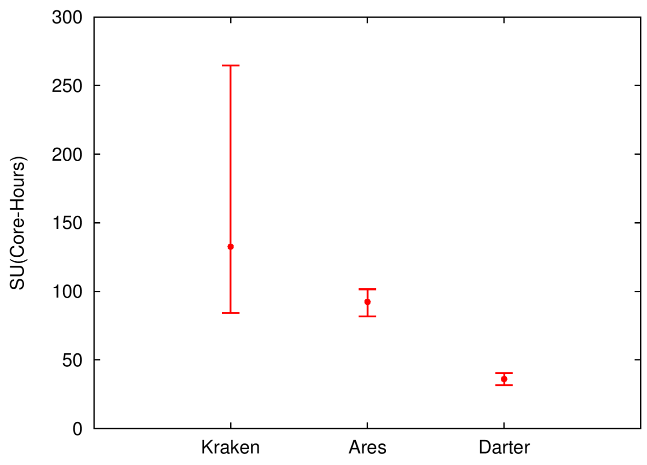

Figure 3: WRF SU usage and its variations on Kraken, Ares, and Darter.

To give better insight of the strong-scalability of WRF with *standard* case on these machines, a plot of SU versus cores is shown in Figure 4. A negative slope indicates increasing efficiency from previous point, while a positive slope indicate a decrease in efficiency. From this figure we can see that Darter is the best machine for this code in maintaining strong-scaling efficiency. On all three machines, there is a slight increase in efficiency in going from 64 cores to 128 cores. Possible explanation for this is that the problem size per core may fit memory hierarchy better with this number of cores. In fact, it is obvious that on all three machines 128 cores is the sweet spot for this test case. As number of cores increases, efficiency decreases on all three machines, although at different rates.

C. LAMMPS

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator") is an open-source molecular dynamics code from Sandia National Laboratories [5]. According to a recent report on the usage of libraries on HPC systems with ALTD tool [6], LAMMPS is the second most used software on Jaguar located at ORNL (with more than 100,000 instances

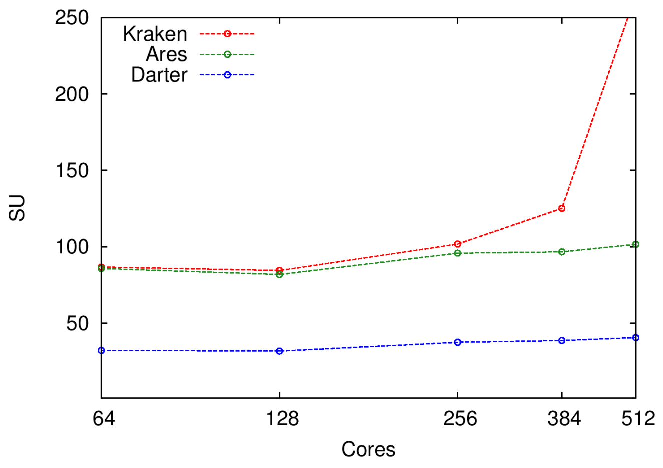

Figure 4: SU usage versus core on Kraken, Ares, and Darter.

of execution) and 8th on Kraken at NICS. In addition, LAMMPS has been executed at the full scale of modern, leadership-class supercomputing systems and it has been selected one of the six critical codes for applications that were to be the vanguards of TITAN at ORNL.

For this benchmark, we used LAMMPS version 25Jan12 compiled using PGI compiler version 12.0 with Cray provided FFTW 3.3.0.0. The LAMMPS atomic fluid with Lennard-Jones (LJ) potential benchmark was used to perform strong scaling studies. LJ is a mathematical model that approximates the interaction between a pair of neutral atoms. The description of the benchmark is as follows. The atomic fluid has:

- 8.4 million atoms for 100 timesteps
- reduced density = 0.8442 (liquid)
- force cutoff = 2.5 σ
- neighbor skin = 0.3 σ
- neighbors/atom = 55 (within force cutoff)
- NVE time integration.

Figure 5 shows the strong-scaling results of LAMMPS on Kraken (XT), Ares (XE), and Darter (XC30). The LAMMPS code outputs timings from the main routines. Using these timings, Figure 6 shows a stacked bar chart with the percentage of time spent in each section of the code depending on the number of cores. This corresponds to the benchmark with a total of 8.4 million of atoms. Most of the time is spent in the "Pair Time" which corresponds to the time spent computing pairwise interactions. Otherwise, the percentage of time spent computing new neighbor lists (neigh time) is quasi similar over the number of cores.

D. GYRO

GYRO (General Atomics GACODE Suite) is a code, developed by General Atomics, which numerically simulates tokamak plasma microturbulence. It computes the turbulent radial transport of particles and energy in tokamak

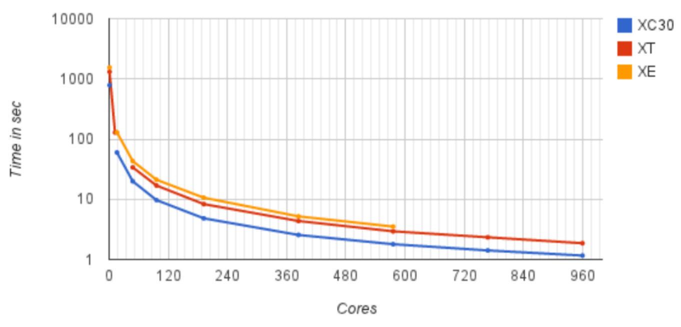

Figure 5: LAMMPS strong-scaling for 8.4 millions atoms on Kraken (XT), Ares (XE), and Darter (XC30)

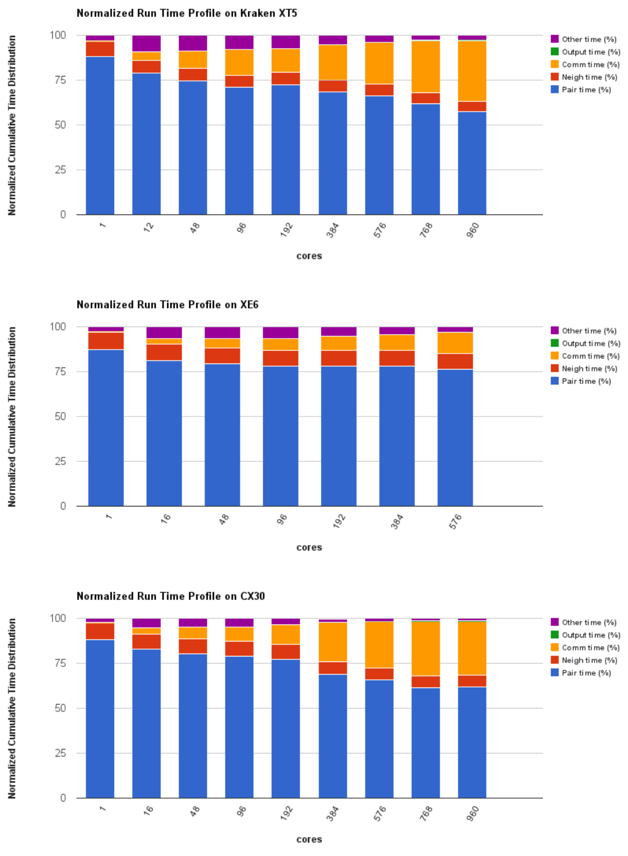

Figure 6: LAMMPS strong-scaling for 8.4 millions atoms on Kraken (XT), Ares (XE), and Darter (XC30)

plasmas and solves 5-D coupled time-dependent nonlinear gyrokinetic Maxwell equations with gyrokinetic ions and electrons. To do so, it utilizes second-order implicit-explicit Runga-Kutta integration with a fourth-order, explicit Eulerian algorithm. It can operate as a flux-tube (local) code, or as a global code, with electrostatic or electromagnetic fluctuations. Libraries utilized by GYRO are netcdf, hdf5, libsci/tpsl and fftw. We compiled GYRO with Cray Compiler 8.1.5 on XC30, PGI Compiler 12.3.0 on XE6, and PGI Compiler 11.9.0 on XT5 respectively.

In Figure 7, we show performance of GYRO in time and total SU consumption. GYRO sees major performance improvements when run on faster networks, such as the Aries interconnects on the Darter XC30, since of the 10% of the execution time spent in communication, 92% is spent in collective communication. Thus, running this code on an all-to-all (Dragonfly) network within a cabinet of the XC30, showed massive performance gains and an order of magnitude less SUs to achieve the same results. The strong scaling was acceptable on all three machines, with the best being on the XC30, not surprisingly.

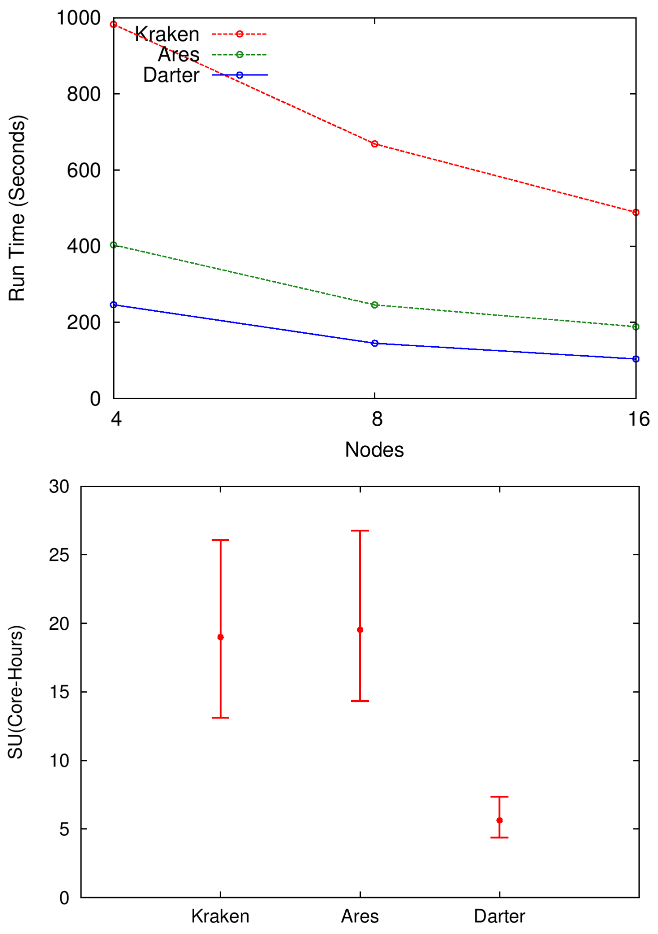

Figure 7: GYRO performance comparison in Time and SU consumption

E. MILC

The MILC code was developed by the MIMD Lattice Computation Collaboration. It consists of several related applications that perform quantum chromodynamics simulations based on four dimensional SU3 lattice gauge theory.

For our testing we have focused on the ks_imp_dyn application, which is a dynamical simulation using the R algorithm and involving a variety of staggered fermion actions. The simulations were each performed using a 32 x 32 x 32 x 36 lattice (x, y, z, t). We performed a strong scaling test using the same inputs in runs involving three different core counts on each of the three platforms.

All of our tests use version 7.6.3 of the MILC code. The code was compiled using the gcc compiler version 4.6.2 on Kraken, 4.6.1 on Ares, and 4.7.2 on Darter. MILC also uses the FFTW library. We built MILC with FFTW version 3.3.0.0 on Kraken, and 3.3.0.1 on Ares and Darter. The results of these tests are shown in Figure 8. MILC exhibited near linear scaling on all three test platforms.

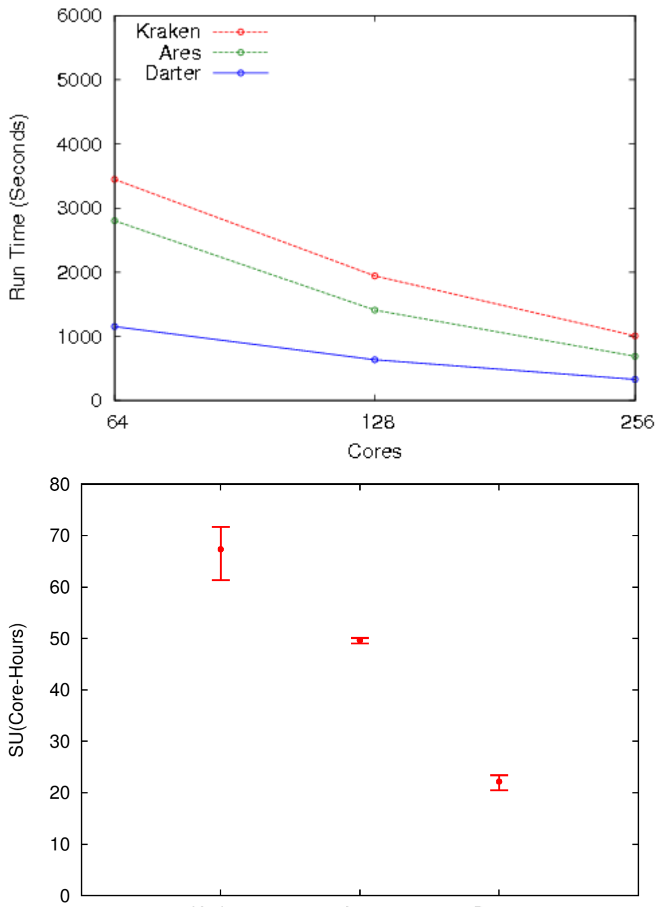

Figure 8: MILC performance comparison in Time and SU consumption

F. NAMD

The molecular dynamics (MD) application called NAMD (Not (just) Another Molecular Dynamics program) [7] is being used to study molecular dynamics, including areas that define life processes in cells such as protein structure, folding and function. NAMD simulates a complex protein with hundreds of millions of atoms over nano to micro (and toward milli) second time frames. NAMD is a parallel molecular dynamics application designed for high performance simulations of large biomolecular systems on parallel computers, scaling up to 300,000 cores for molecular systems of 100 millions atoms or more [8], [9].

The adaptive runtime system Charm++ [10] is utilized in NAMD for parallel computing, which provides scaling to thousands of processors. Charm++'s parallel objects and data-driven execution adaptively overlaps communication and computation and hide communication latency: when an object is waiting for some incoming data, entry functions of other objects with all data ready are free to execute. Many innovative parallel algorithms are implemented in NAMD for petascale simulations.

NAMD is the most utilized application on XSEDE resources such as Kraken [6]. We compiled NAMD-2.9 using GNU compiler version 4.6.3 on Kraken and Ares with Cray provided FFTW 3.3.0.0. We used GNU compiler version 4.7.2 on Darter with Cray provided FFTW 3.3.0.0 to build NAMD.

We used the Satellite Tobacco Mosaic Virus (STMV) dataset for NAMD benchmark. It has the following properties: 1,066,628 atoms, periodic, Particle Mesh Ewald (PME). We used 5000 steps for our benchmark.

In Figure 9, we show the performance of NAMD in time and total SU consumption. NAMD sees less than expected performance improvements when run on the Cray XC30. We need to do further investigation to find out about why NAMD uses fewer SUs on Kraken as compared to Ares.

IV. Discussion

In this paper, we examine Service Units (SU) for each application. For a strong scaling benchmarking case, ideally SU consumption should stay the same when using different number of cores. In each bar plot of application SU consumption, we show variance of SU consumption on each platform. Smaller variance means better scaling. Moreover, for each application we calculate the *observed* SU conversion ratio between two platforms using the following formula:

$$\text{Ares/Kraken Ratio} = \frac{\text{Kraken SU}_{app}}{\text{Ares SU}_{app}}$$

$$\text{Darter/Kraken Ratio} = \frac{\text{Kraken SU}_{app}}{\text{Darter SU}_{app}}$$

In Figure 10, we draw bar plots of the observed SU conversion ratio for each application with its min, max and average. We also mark the *ideal* ratio computed by HPL flops measured on each machine from Table II. We can see that except NAMD, all applications perform better than

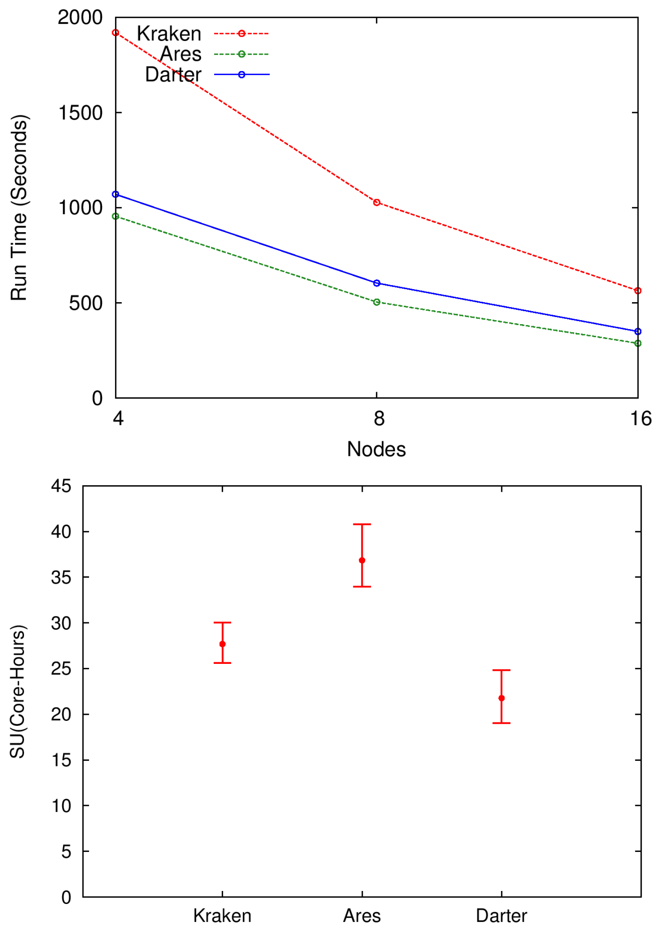

Figure 9: NAMD performance comparison in Time and SU consumption

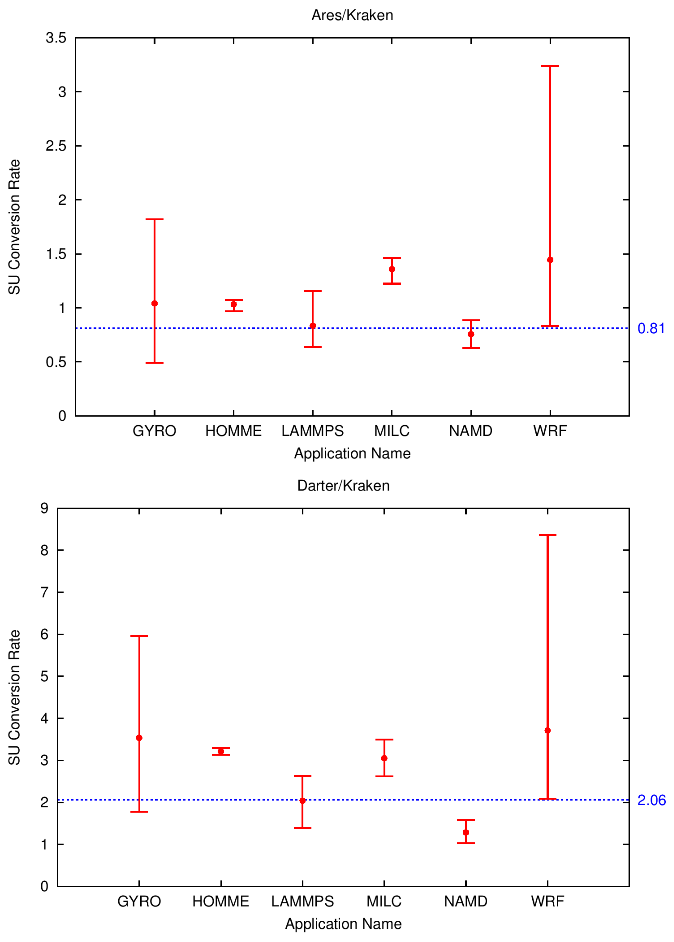

Figure 10: Application SU conversion ratio

expected by HPL ratio and theoretical peak performance ratio.

V. Conclusion

Our benchmarking results show that most applications achieve better performance than expected on CRAY XC30 and XE6 than on XT5 when a direct conversion using HPL is used. As future work, we would like to gather a performance profile for each application on each platform. We plan to investigate whether the observed performance improvements for applications were due to improvements in CPU, network, or other factors.

References

[1] Hans Meuer, Erich Strohmaier, Jack Dongarra, and Horst Simon. TOP500 Supercomputing Sites. http://top500.org.

[2] J. Dongarra A. Cleary A. Petitet, R. C. Whaley. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. http://www.netlib.org/benchmark/hpl/.

[3] Extreme Science and Engineering Discovery Environment. Xsede. http://www.xsede.org.

[4] J. Dennis, A. Fournier, W. F. Spotz, A. St.-Cyr, M. Taylor, S. J. Thomas, and H. Tufo. High resolution mesh convergence properties and parallel efficiency of a spectral element atmospheric dynamical core. In *Int. J. High Perf. Computing Appl.*, volume 19, pages 225–235, 2005.

[5] Lammps. http://lammps.sandia.gov/.

[6] Bilel Hadri, Mark Fahey andTimothy Robinson, and William Renaud. Software usage on cray systems across three centers (nics, ornl and cscs).

[7] Mark Nelson, William Humphrey, Attila Gursoy, Andrew Dalke, Laxmikant Kale, Robert Skeel, and Klaus Schulten. Namd - a parallel, object-oriented molecular dynamics program. In *The International Journal of Supercomputer Applications High Performance Computing*, volume 10, pages 251–268, 1996.

[8] James Phillips, Gengbin Zheng, Sameer Kumar, and Laxmikant Kale. Namd: Biomolecular simulation on thousands of processors. In *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–18, 2002.

[9] James Phillips, Rosemary Braun, Wei Wang, James Gumbart, Emad Tajkhorshid, Elizabeth Villa, Christophe Chipot, Robert Skeel, Laxmikant Kale, and Klaus Schulten. Scalable molecular dynamics with namd. In *JOURNAL OF COMPUTATIONAL CHEMISTRY*, volume 26, pages 1781–1802, 2005.

[10] Laxmikant Kale and Sanjeev Krishnan. Charm++: A portable concurrent object oriented system based on c++. In *Object Oriented Programming Systems, Languages and Applications(OOPSLA)*, 1993.