Performance of the fusion code GYRO on four generations of Cray computers

Mark R. Fahey
National Institute for Computational Sciences
University of Tennessee Knoxville
Oak Ridge, TN
Email: mfahey@utk.edu

***Abstract*—GYRO is a code used for the direct numerical simulation of plasma microturbulence. It has been ported to a variety of modern MPP platforms including several modern commodity clusters, IBM SPs, and Cray XC, XT, and XE series machines. We briefly describe the mathematical structure of the equations, the data layout, and the redistribution scheme. Also, while the performance and scaling of GYRO on many of these systems has been shown before, here we show the comparative performance and scaling on four generations of Cray supercomputers including the newest addition - the Cray XC30. The more recently added hybrid OpenMP/MPI implementation also shows a great deal of promise on custom HPC systems that utilize fast CPUs and proprietary interconnects. Four machines of varying sizes were used in the experiment, all of which are located at the National Institute for Computational Sciences at the University of Tennessee at Knoxville and Oak Ridge National Laboratory. The advantages, limitations, and performance of using each system are discussed.**

***Keywords*-fusion; microturbulence; GYRO; performance; Cray; XT5; XE6; XC30**

I. Introduction

GYRO is a code used for the direct numerical simulation of plasma microturbulence. It computes the turbulent radial transport of particles and energy in tokamak plasmas. It does this by solving 5-D coupled time-dependent nonlinear gyrokinetic-Maxwell equations with gyrokinetic ions and electrons. It was developed by Jeff Candy and Ron Waltz at General Atomics. It has been ported to a variety of modern MPP platforms including several modern commodity clusters, IBM SPs, and Cray XC, XT, and XE series machines. The mathematical structure of the equations, the data layout, and the redistribution scheme will be described. While the performance and scaling of GYRO on many of these systems has been shown before, instead a comparison of the performance and scaling is shown on four generations of Cray supercomputers including the newest addition - the Cray XC30. The more recently added hybrid OpenMP/MPI implementation is also shown to have a great deal of promise on custom HPC systems that utilize fast CPUs and proprietary interconnects. Four machines of varying sizes were used in the experiment, all of which are located at the National Institute for Computational Sciences at the University of Tennessee at Knoxville and Oak Ridge National Laboratory. The advantages, limitations, and performance of using each system are discussed.

Section II provides an overview of the fusion microturbulence simulation code GYRO, including a brief discussion of the discretization schemes (MPI parallelism) and OpenMP implementation. Section III describes the benchmark problem. Section IV has a description of each platform tested. The test results are shown in Section V including the build and runtime environments as well as short discussion of the results. The paper ends in Section VI with conclusions and future work.

II. GYRO Overview

The most promising and aggressively studied concept for power production by fusion reactions is the tokamak. Advances in the understanding and control of tokamak plasmas are continuously being realized, although uncertainties remain in predicting confinement properties and performance of larger reactor-scale devices. The coupled gyrokinetic-Maxwell (GKM) equations provide a foundation for the first-principles calculation of turbulent tokamak transport.

GYRO is a code that simulates tokamak turbulence by solving the time-dependent, nonlinear gyrokinetic-Maxwell equations for both ions and electrons[1]. It uses a five-dimensional grid and advances the system in time using either a fourth-order explicit Runge-Kutta (RK) integrator or a second-order, implicit-explicit (IMEX) RK integrator. GYRO has both *global* and *electromagnetic* operational capabilities. GYRO has been ported to a variety of modern MPP platforms and has shown good scalability on all these platforms.

The GKM equations couple the *gyrocenter* distribution, f, to the electromagnetic fields, Φ:

$$\frac{\partial f}{\partial t} = \mathcal{L}_a f + \mathcal{L}_b \Phi + \{f, \Phi\}$$

where

$$\mathcal{F}\Phi = \int \int dv_1 \, dv_2 \, f.$$

$\mathcal{L}_b$, $\mathcal{L}_b$ and $\mathcal{F}$ are linear operators. Strictly speaking, f measures the deviation from a Maxwellian background.

[1]Development was partially funded by the *Plasma Microturbulence Project*, a fusion SciDAC project.

For global simulations, a linear adaptive source technique [1] is used to inhibit the evolution of f and Φ on equilibrium scales. Within the standard gyrokinetic ordering, the sole nonlinearity has a Poisson bracket structure. The function $f(\mathbf{r}, v_1, v_2)$ is discretized over a 5-dimensional grid (three spatial and two velocity coordinates), while the 3-dimensional electromagnetic fields $\Phi(\mathbf{r}) = [\phi, A_\parallel, B_\parallel]$ are independent of velocity. Here ϕ, $A_\parallel$ and $B_\parallel$ are the *electrostatic potential*, the *transverse electromagnetic potential*, and the *parallel magnetic field*, respectively. One averages over the fast orbital motion (gyro-orbit) to eliminate the third velocity-space dimension (gyro-angle). However, this so-called *gyro-averaging* operation introduces *nonlocal* spatial operators ($\mathcal{F}$ above, for example) perpendicular to the magnetic field.

A. Discretization schemes

GYRO solves the gyrokinetic Maxwell equations on a fixed grid

$$f(r, \tau, n_{tor}, \lambda, E) \to f(i, j, n, k, e).$$

For different code stages, the distribution of an index across processors is incompatible with the evaluation of operators on that index (see below for more details.) For example, a derivative in r requires all i to be on processor. Therefore, the code has 2- and 3-index transpose operations that must be executed to change how the data is distributed across MPI processes. These transpose operations use MPI_ALL_TO_ALL. The GYRO code base has two distribution libraries, TRANSPOSE and SSUB, for these 3-index and 2-index transposes, respectively. Although these transposes are scaling limiters, sufficiently large problem sizes have been effectively scaled to to 49,152 MPI processes on a Cray XE6 at the Oak Ridge Leadership Computing Facility.

The discretization scheme in each dimension is briefly sketched below. A detailed treatment is beyond the scope of the present paper.

- **toroidal angle:** fully spectral decomposition of the fluctuating quantities $(f, \phi, A_\parallel)$ is made,

$$\phi = \sum_n \phi_n(r, \theta)\, e^{-\imath n[\varphi - q(r)\theta]} \quad . \tag{1}$$

  The integer n labels *toroidal eigenmodes*. Linear studies, still very important for research problems, can be carried out using a single value of n. In Eq. (1), φ is the *toroidal angle*, θ is the *poloidal angle* and q is the *safety factor* (which measures the helicity of the equilibrium magnetic field).
- **radius:** linear advective derivatives on f are treated with an upwind differences, whereas derivatives on fields are treated with centered differences. The gyro-operators $\mathcal{G}$ and $\mathcal{R}$ are approximated using a (banded) pseudo-spectral technique.
- **poloidal angle:** for f, there is no fixed grid in θ. Instead, the transformation

$$\frac{v_{\parallel \mathrm{s}}(r, \lambda, \theta)}{R_0 q(r)} \frac{\partial}{\partial \theta} \to \Omega(r, \lambda) \frac{\partial}{\partial \tau}$$

  is used to eliminate the singularity at bounce points, $v_{\parallel \mathrm{s}} = 0$. Then, an upwind scheme in τ is used to discretize $\partial f / \partial \tau$. Centered differences only are used on τ-derivatives of fields, otherwise numerical instability will result. The use of a τ-grid (leading to a different set of points in θ for every value of λ) for the GK equation dictates that the Maxwell equations are solved by expansion of fields in complex finite-elements: $\phi(r_i, \theta) = \sum_m c_m^i F_m^i(\theta)$. The F_m^i satisfy the phase condition

$$\phi_n(r, \pi) = e^{-2\pi \imath n q(r)} \phi_n(r, -\pi).$$

  where the Fourier coefficients, ϕ_n, are nonperiodic.
- **velocity-space:** The object V above is the 2-dimensional *velocity-integration operator*, defined as

$$\mathrm{V}[f_\mathrm{s}] \doteq \sum_\sigma \frac{1}{2\sqrt{\pi}} \int_0^\infty d\epsilon\, e^{-\epsilon} \sqrt{\epsilon} \int_0^1 \frac{d(\lambda \hat{B})}{\sqrt{1 - \lambda \hat{B}}} f_\mathrm{s}(r, \theta, \varphi; \epsilon, \lambda, \sigma; t) \quad . \tag{2}$$

  A transformation property of Eq. (2) under integration over θ is used to recast the velocity-space integration. Then, in both ϵ and λ, an exact Gauss-Legendre quadrature scheme is numerically generated (by nonlinear root-finding) at run-time. This is different at each radius and for different plasma equilibria.
- **nonlinearity:** The nonlinear Poisson bracket is evaluated with a *conservative* difference-spectral analogue of the Arakawa method. This scheme ensures exact conservation of density and generalized entropy at vanishing time step (independent of grid resolution).
- **collisions:** Collisions are represented by a second-order diffusive-type operator in λ. This operator is split from the collisionless problem and an irregular-grid generalization of the Crank-Nicholson method is used.
- **time-advance:** Either a 2nd-order IMEX RK scheme, with the electron parallel motion ($\partial / \partial \theta$) treated implicitly, or an explicit 4th-order RK scheme is used. The implicit solver is complicated due to the use of a τ-grid, as well as the presence of the field quantity Φ in the advection. For multiscale runs that capture ion and electron scales simultaneously, the explicit integrator is faster and more accurate whereas for typical ion-scale simulations, the implicit integrator is preferred.

B. OpenMP implementation

In December of 2011, preliminary hybrid parallelization (MPI+OpenMP) modifications were made to GYRO. For

a given MPI task, the same number of OpenMP threads can be used, which share memory. The total core count is the product (number of MPI tasks) $\times$ (number of OpenMP threads per task).

The OpenMP code typically targets loops over radius r, which are left undistributed by MPI.These loops tend to have the structure

```
do i=1,n_x
   do ip=-n_band,n_band
      do (distributed stuff)
```

For large radial grids and large gyro bandwidth there's quite a lot of work for OpenMP, and for these problems the OpenMP scaling is effective.

III. Benchmark Problem

For our evaluation, we ran GYRO for one standard benchmark problem: nl02a. The nl02a problem is a 8 toroidal-mode electrostatic (ions and electrons) case on a $8 \times 400 \times 12 \times 8 \times 28 \times 2$ grid. It uses a radial annulus with non-periodic boundary conditions and a flat profile. This test is run for 100 time steps with kinetic electrons and electron collisions and electromagnetic effects. A real simulation would run for at least 250,000 time steps, so the time per time step needs to be as minimal as possible. The 400-point radial domain with 8 torodial modes gives high spatial resolution, typical of production runs and represents roughly the minimum grid size to obtain physically accurate results. Yet the test case is considered small with only 8 modes; multiscale cases use more than 100 modes, which provides much more work for OpenMP loops. As discussed above, communication costs are dominated by calls to MPI_ALL_TO_ALL.

GYRO does automatic logging of the version of the code - for the tests contained here the version was r4-207-g37696de.

IV. Test platforms

Four generations of Cray supercomputers were used in this comparison study:

- a Cray XC30 with Intel® Xeon® E5-2670 CPUs
- a Cray XE6 with AMD Opteron™ Interlagos CPUs
- a Cray Xtreme-X supercomputer with Intel® Xeon® E5-2670 CPUs and Intel® Xeon Phi™ coprocessors
- a Cray XT5 with AMD Opteron™ Istanbul CPUs.

More details on each machine are presented below.

The Cray XC30, named Darter, is comprised of 4 cabinets with 1,496 compute nodes, or 11,968 cores, of Intel E5-2670 processors [2]. The processors have 8 physical cores and can utilize 2 Hypethreads [3], [4]. Each core operates at 2.6 GHz - the aggregate peak of the machine is 248.9 TF. Each node has 32 GB of memory. The machine has the Cray Aries interconnect set up with a Dragonfly topology [5], [6], [7], [8].

Table I
THE ARCHITECTURE, OS, AND COMPILER USED ON EACH MACHINE. THE XC30 IS SHOWN TWICE BECAUSE TEST WERE DONE WITH TWO COMPILERS.

Machine	Architecture	OS	Compiler
Darter	XC30	CLE 5.1	Cray 8.1.9
Darter	XC30	CLE 5.1	Intel 13.1
Beacon	CCS	MPSS 2.1	Intel 13.1
Ares	XE6	CLE 4.2	PGI 12.10
Kraken	XT5	CLE 3.1	PGI 12.10

The Cray XE6, named Ares, is really a hybrid XE6/XK6, but only the XE6 portion was utilized. The XE6 part has 20 nodes each with two 16-core AMD Opteron processors and 16 GB of memory. The XK6 part has 16 nodes each with one 16-core AMD Opteron processor, 16 GB of memory, and one Tesla X2090 with 6 GB of memory. The Opteron processors all run at 2.2 GHz. In total, there are 896 Opteron cores providing an aggregate of 18.5 TF. The machine uses the Cray Gemini [9] interconnect set up as a torus.

The Cray Xtreme-X computer (CCS)[2], named Beacon, is a comprised of two separate acquisitions of Intel Xeon/Xeon Phi compute nodes. The first phase set up 16 development/compute nodes where each node had two Intel Xeon E5-2670 processors and two Intel Xeon Phi 5110P coprocessors (preproduction.) The second phase configured 48 nodes each with two Intel Xeon E5-2670 processors and *four* Intel Xeon Phi 5110P coprocessors. The Xeon E5-2670s are the same processors as seen above in the XC30 machine. In total, 1,204 cores in the Xeon processors.

The fourth machine is also the oldest machine tested - a Cray XT5, named Kraken [10]. Kraken has 100 cabinets with 9,408 nodes each with two 6-core AMD Opterons. Each core operates at 2.6 GHz - the aggregate peak of the machine is 1.17 PF. The machine has the Cray SeaStar2+ [11] interconnect configured as a 3D torus.

V. Test results

The operating systems and compilers used on the test machines are shown in Table I. The choice of compiler is based on machine and its processor architecture and also heavily weighted by the *default* compiler presented to the users. For the XT5 and XE6, they have AMD Opteron processors and the PGI compiler has proven over time to be a good choice. On the CCS and XC30, the processors are Intel Xeons and using Intel makes very good sense. The Cray compiler is the default on the XC30, so it was tested in addition to the Intel compiler.

The original hope was to use the same compiler and version across all machines. The Intel compiler could have been used across all of the machines, but not the same version due to programming incompatibilities and licensing

[2]Now referred to as a Cray Cluster Supercomputer

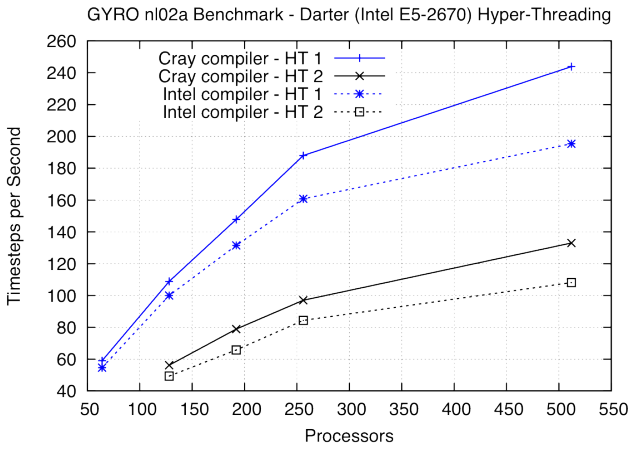

Figure 1. nl02a benchmark on Cray XC30 comparing Cray and Intel compilers and use of Hyperthreading

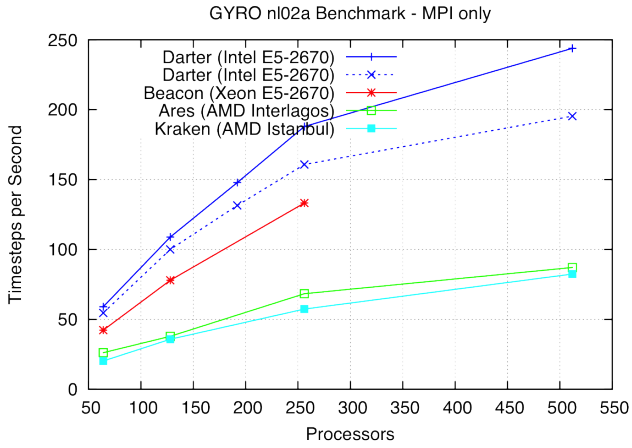

Figure 2. The nl02a benchmark scaling results with MPI only on all four test platforms. See Table I for machine name to architecture mapping. The two blue lines are for the XC30: solid line with Cray compiler and dashed line with Intel compiler.

issues. Further, it is not the default on two of the machines, so the goal of using the same compiler and version was abandoned.

The optimization level "-O3" was used for each compiler except for the Cray compiler, which by default does a good amount of optimizations without the user specifying compiler switches. Certainly, the space of compiler options could have been searched to find better combinations, but that was not the scope of this testing.

Figure 1 shows the performance of the nl02a benchmark problem just on the Cray XC30 comparing the Cray compiler and the Intel compiler. The Cray compiler default optimization produces much better performing code than the Intel compiler at "-O3". Therefore, for the remaining tests, the Cray compiler was used on the XC30. Certainly more testing could be done with both compilers to produce better performing code especially with inlining. Figure 1 also shows the effect of turning on Hyperthreading at runtime, which was also explored during this testing. The testing performed was just the use of Hyperthreading at runtime without doing any source code optimizations. The figure clearly shows that turning on Hyperthreading at runtime does not improve the performance of GYRO, but rather hinders it.

Figure 2 shows the performance of GYRO on all four Cray supercomputers. It shows that the XC30 is roughly 3× faster than the XT5 and XE6 computers; surprisingly, the XE6 is not significantly faster than the XT5. It also shows that the XC30 is 25% faster than the CCS system which has the same processors, and the CCS system has higher memory bandwidth. It is postulated that the XC30 is faster because of the compiler, the custom interconnect, and potentially the highly tuned compute OS (CLE.)

Figure 3 shows the performance of the GYRO in hybrid MPI/OpenMP mode, using 64 MPI processes and varying

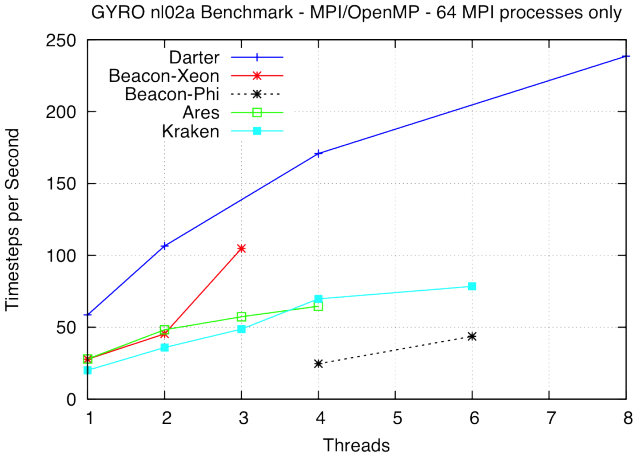

Figure 3. The nl02a benchmark scaling results with hybrid MPI/OpenMP with 64 MPI processes. Some data points are missing due to machine size constraints. A couple data points from the CCS machine using the Xeon Phi's are also included.

the OpenMP thread count. First, the OpenMP implementation scales well regardless of machine. Second, the performance difference between the XC30 and the other machines is even greater, and it shows much better scaling properties on par with the MPI scaling.

Figure 4 shows the performance of the GYRO in hybrid MPI/OpenMP mode, using 128 MPI processes and varying the OpenMP thread count. Very similar conclusions to be made for the this case as well. The XC30 continues to show excellent OpenMP thread scaling, even on a small problem.

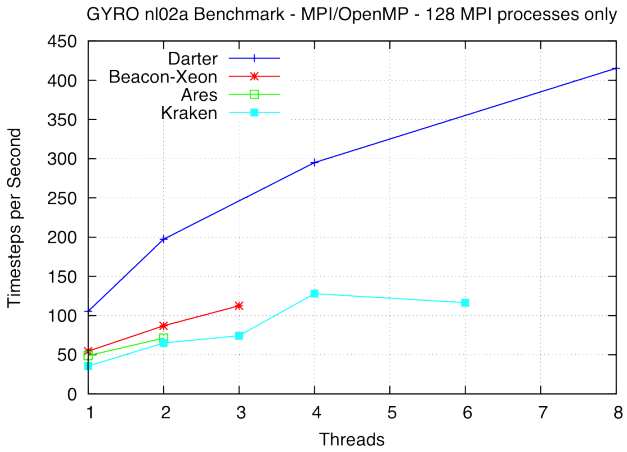

Figure 4. The nl02a benchmark scaling results with hybrid MPI/OpenMP with 128 MPI processes. Some data points are missing due to machine size constraints.

VI. Conclusion

GYRO shows excellent scaling on all the Cray computers shown here. The Cray XC30 shows significant improvements over previous generations. Some conclusions that can be inferred from the results are:

- GYRO scales well in both MPI and OpenMP dimensions though machine dependent
  - The hybrid MPI/OpenMP tests show excellent scaling on the XC30
- There is a large jump in performance from Opterons to Xeons; likely due to the Intel processors being newer
- The Cray compiler does a good job optimizing GYRO by default

There is some continuing work along these lines. Some initial work on porting GYRO to Intel Xeon Phis and to NVIDIA GPGPUs has been done. The performance on both is well beneath that of the scalar processors and so not reported here. Much more work needs to be done before any success can be reported.

Acknowledgment

Vince Betro must be acknowledged for his help in running the tests on Beacon, including some Intel Xeon Phi runs that are not shown here.

This material is based upon work supported by the National Science Foundation under Grant numbers 0711134, 0933959, 1041709, and 1041710 and the University of Tennessee through the use of the Kraken, Ares (now named Mars), and Darter computing resources at the National Institute for Computational Sciences (http://www.nics.tennessee.edu).

References

[1] R. Waltz, J. Candy, and M. Rosenbluth, "Gyrokinetic turbulence simulation of profile shear stabilization and broken gyrobohm scaling," *Phys. Plasmas*, vol. 9, p. 1938, 2002.

[2] M. R. Fahey, R. Budiardja, L. Crosby, and S. McNally, "Deploying Darter - a Cray XC30 system," in *to appear in Proceedings of the 29th International ACM Conference*, ser. Lecture Notes in Computer Science, Leipzig, Germany, June 2014.

[3] T. Leng, R. Ali, J. Hsieh, and C. Stanton, "A Study of Hyper-Threading in High-Performance Computing Clusters," *Dell*, vol. 6, no. 1, pp. 33–36, 2002. [Online]. Available: http://ftp.dell.com/app/4q02-Len.pdf

[4] Z. Zhao, N. J. Wright, and K. Antypas, "Effects of Hyper-Threading on the NERSC workload on Edison Lawrence Berkeley National Laboratory," in *Proceedings of the 2013 CUG Conference*, Napa, CA, May 2013.

[5] J. Kim, W. J. Dally, S. Scott, and D. Abts, "Technology-driven, highly-scalable Dragonfly topology," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 77–88. [Online]. Available: http://dx.doi.org/10.1109/ISCA.2008.19

[6] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, "Cray XC series network," 2012. [Online]. Available: http://www.cray.com/Assets/PDF/products/xc/CrayXC30Networking.pdf

[7] G. Faanes, A. Bataineh, D. Roweth, T. Court, E. Froese, B. Alverson, T. Johnson, J. Kopnick, M. Higgins, and J. Reinhard, "Cray Cascade: A scalable HPC system based on a Dragonfly network," in *Proceedings of the 2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 1–9. [Online]. Available: http://dx.doi.org/10.1109/SC.2012.39

[8] D. T. Marr, F. Binns, D. L. Hill, G. Hinton, D. A. Koufaty, J. A. Miller, and M. Upton, "Hyper-threading technology architecture and microarchitecture," *Intel Technology Journal*, vol. 6, 2002.

[9] Cray Inc., "The Gemini Network," 2010. [Online]. Available: http://wiki.ci.uchicago.edu/pub/Beagle/SystemSpecs/Gemini_whitepaper.pdf

[10] M. R. Fahey, L. Crosby, G. Rogers, and V. Hazlewood, "Kraken: the first academic petaflop computer," in *Contemporary High Performance Computing: From Petascale Toward Exascale*, 1st ed., ser. CRC Computational Science Series, J. S. Vetter, Ed. Boca Raton: Taylor and Francis, 2013, vol. 1, p. 900.

[11] Cray Inc., "Cray XT system overview," Cray Inc., Tech. Rep. S-0010-22, 2009. [Online]. Available: http://docs.cray.com