# Effect of Rank Placement on Cray XC30 Communication Cost

Reuben D. Budiardja, Lonnie Crosby, Haihang You
National Institute for Computational Sciences
The University of Tennessee, Knoxville, TN 37996
{reubendb,lcrosby1,hyou}@utk.edu

***Abstract*—The newly released Cray XC30 supercomputer boasts the new Aries interconnect that incorporates a Dragonfly network topology. This hierarchical network topology has obvious advantages with respect to local communication. However, as communication patterns extend further down the hierarchy and grow more separated the overall impact of particular bottlenecks and trade-offs between bandwidth and latency become less apparent. In particular, applications may be more or less latency sensitive based on their communication pattern. The dynamic routing options, as a result, may affect some applications more severely than others. In this paper, we investigate the effect of process placement on the communication costs associated with typical communication patterns shared by many scientific applications. Observations concerning the communication performance of benchmarks and selected applications are presented and discussed.**

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

## I. Introduction

The National Institute for Computational Sciences (NICS) at the University of Tennessee, Knoxville recently acquired a Cray XC30 system, which boasts the new Aries interconnect that incorporates a Dragonfly [1] network topology. In this paper, we investigate the effect of process placement on the communication costs associated with typical communication patterns shared by many scientific applications. Selected trends are tested with respect to benchmarks and selected scientific applications which utilize these communication patterns. Observations concerning the performance implications of process placement and/or communication patterns are discussed.

This paper is organized as follows. In Section II we very briefly describe the configuration of the Cray XC30 system. A more thorough description of the XC30 architecture can be found in [2]. Section III details the setup of our benchmarking experiments. In section IV we present the results of these experiments, followed by concluding remarks in Section V.

## II. System Overview

The Cray XC30 is a distributed memory supercomputer equipped with a Dragonfly [1] network topology. The basic building block that is unique to an XC30 system is the compute blade that contains four compute nodes connected to an Aries interconnect ASIC (Application-specific integrated circuit). Aries is a system-on-chip that integrates four Network Interface Controllers (NIC) based on Gemini interconnect [3] with a 48-port high radix router. Thus the Aries NICs on a compute blade provides the network connectivity for all four compute nodes on the blade via independent PCI-Express Gen3 x16 host interfaces.

Sixteen of these compute blades are combined to form a chassis. Each Aries on a blade is connected via a backplane to form an all-to-all connectivity between the blades in the chassis. This connection is referred to as the *green links*, and is the first dimension of the network topology.

A group is formed by connecting two cabinets, consisting of three chassis each, with electrical links from each Aries in a chassis to its peers on each chassis in the two cabinets. There are six total chassis in this group that require the five electrical connectors available on each Aries to form the inter-chassis connections. Another all-to-all network is formed between these Aries ASICs across chassis in the group and is referred to as the *black links*. This is the second dimension of the Dragonfly topology.

Groups are connected with each other via optical links to form a system. These connections, referred as *blue links*, are also all-to-all and provided by each Aries in the system. More information about the possible configuration of this dimension of the Dragonfly topology can be found in a related paper [2]. The optical links may be configured to provide additional bandwidth or to scalably increase the number of groups that may be connected. Figure 1 illustrates the network structures of an XC30 system.

Darter, the Cray XC30 at NICS, is a system consists of two groups with a total of 748 compute nodes and 10 service nodes. Each compute node is dual-socket utilizing 2.6 GHz octa-core Intel Sandy Bridge processors with 32 Gigabytes DDR3 1.6 GHz memory. The system theoretical peak performance is 248.9 TeraFLOPS with. A lustre Sonexion parallel file system is attached to the system providing 360 Terabytes usable disk space at 10.7 Gigabytes/second peak bandwidth.

## III. Experiment Setup

In this section the setup for the benchmark experiments is discussed. A custome code was written in order to provide the most flexible rank placement for these benchmarks. This code creates MPI communicators whose members are nodes

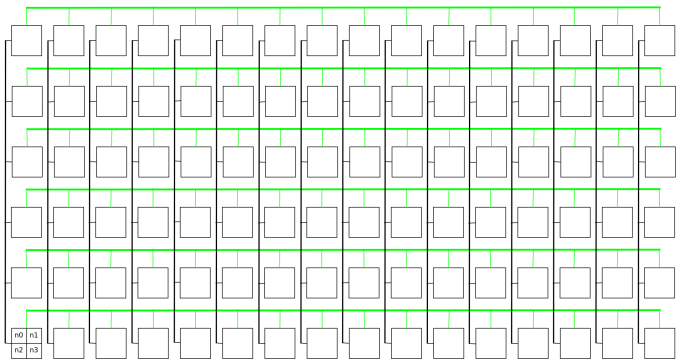


Figure 1: Overview of XC30 network in a group. Each box represents a blade with four compute nodes connected to an Aries ASIC. Thick solid line indicates an all-to-all connections between compute blades. Green links, represented by green lines, are connected via chassis backplane, while black links (black lines) are inter-chassis electrical connections.

on which the benchmarks run. To avoid on-node, SMP specific contributions to the results, only one MPI rank per node (via "`-N 1`" flag to `aprun`) is utilized since off-node network performance is of interest.

The code that creates the MPI communicators utilizes the compute node's canonical name (*cname*) to first determine the location of the node in the system. This *cname* is read from the file `/proc/cray_xt/cname` that exists on every compute node. The name has the format **c**i**-**j**c**k**s**l**n**x where bold letters are literal and *i, j, k, l, x* are numbers to indicate the cabinet coordinate (i, j), chassis number (k) in a cabinet, slot number (l) in a chassis, and node number (x) in a slot. The slot number (l) can be either one digit or two digits (0 - 15) to indicate the slot (i.e. blade) location in the chassis. By parsing this *cname*, one can determine the exact position of a node in the system configuration.

To run the benchmarks, jobs are submitted that require a large number of nodes in order to ensure all nodes needed to form the required communicators are available. By knowing the positions of the nodes in the system, the following sub-communicators from `MPI_COMM_WORLD` are created to represent different hierarchies of the XC30 network:

- *green* communicators that consist of nodes attached to Aries in the same chassis. These Aries are connected by the chassis backplane (the so-called *green links*), which is the first dimension of the all-to-all network For example, the nodes of a communicator such as this have the *cname*: `c3-0c1s[0-15]n[0-3]`, where the square brackets ([]) indicate range of numbers. For this type of communicator, we have the option of using all four nodes attached to each Aries or one node per Aries in a communicator. For the former, we have 64 ranks in a green communicator. For the latter option, the same node number (e.g. node 0) attached to each Aries forms a green communicator with 16 ranks, creating a total of four green communicators per chassis whose members are, for example, nodes with *cname*: `s[0-15]n0, s[0-15]n1, s[0-15]n2, s[0-15]n3`, where here the cabinet and chassis number are denoted by  since they vary depending on the exact identity of the chassis. The benchmarks can then be run simultaneously with the four green communicators per chassis, creating an *ensemble* runs, or with only one green communicator per chassis called the *single* runs.
- *black* communicators that consist of nodes attached to the same Aries number in the two-cabinet group. These Aries are connected by the inter-chassis electrical connection (the so-called *black links*), which is the second dimension of the all-to-all network. Each black communicator has Aries that reside in a column of the two-dimension all-to-all network. For example, the nodes of a communicator such as this have the *cname*: {`c0-0c[0-2]s2n[0-3]`,`c1-0c[0-2]s2n[0-3]`}, where the curly brackets ({}) here indicate union. This makes sure that there is only a single hop between ranks in this type of communicator due to the all-to-all network. As with the green communicators, these may be defined using four nodes per Aries (*single*) or one node per Aries, which allows *ensemble* runs.
- *red* communicators that consist of nodes attached to Aries that are in the diagonal of the two-dimensional all-to-all network in a group. In this configuration none of the Aries included are within the same intra-chassis (row) or inter-chassis networks (column) within the same group. This ensures that there is at least two hops of routing between any two Aries in the communicators. For the experiments using this communicator we used all four nodes attached to each Aries to form a 24-rank MPI communicator.
- *blue* communicators that are red communicators split across two different groups of the XC30 network. That is, three of the Aries of these communicators are in group 0, while the other three are in group 1.

The routine `MPI_Comm_split` is used to create the communicators. The combination of the job submission and creation of these communicators allow us to run benchmarks with specific nodes independent of scheduler placement such as ALPS [4] of our benchmarks jobs.

The Intel MPI Benchmarks (IMB) version 3.2.4 [5] are utilized to measure the performance of these different network hierarchies. IMB was modified to accept custom MPI communicators to run benchmarks by replacing all references to `MPI_COMM_WORLD` with the new communicator. IMB originally writes its output to either `STDOUT` or to a single output file by rank 0. Since multiple different communicators (with different sets of nodes as members)

were passed to IMB with our modification, in a sense multiple instances of IMB ran simultaneously. This necessitated a slight modification to IMB output routine. Rather than a static filename, the IMB output routine was modified such that the output filename is based on the rank from `MPI_COMM_WORLD` of that IMB instance. The many output files from many IMB instances are then reduced using a home-grown PHP script to get the global minimum, maximum, and average of each experiment.

## IV. Results

### A. Default Rank Placement

The 758 compute and service nodes on the Cray XC30 system are identified with node identification (NID) numbers. These are defined in such a way as to ensure that subsequent NIDs reside close to each other in the Dragonfly network topology. Specifically, nodes (0-3) attached to a single Aries ASIC are numbered sequentially, followed by the Aries ASICs (0-15) in a single chassis, followed by the chassis (0-5) in a single group, and cycling through groups (0-1). Service nodes also follow this numbering scheme even though there are only two nodes attached to these Aries ASICs (1 and 2). The NIDs corresponding to the missing nodes (0 and 3) are not utilized within the system, which makes the scheme general and not dependent on the service/compute node breakdown of the system. Each node also has a *cname* designation described previously that identifies the physical location of nodes within the system. Due to the relation between physical and network location, NIDs are analytically calculable from the physical location of nodes.

$$\mathrm{NID} = x + Nl + NSk + NSC(i + Xj), \qquad (1)$$

where $N$ is the maximum number of nodes on a blade (4), $S$ is the maximum number of slots within a cage (16), $C$ is the maximum number of cages within a cabinet (3), and $X$ is the maximum number of cabinets in a row.

The Application Level Placement Service (ALPS) [4] reorders the nodes within a batch jobs reservation according to NID number, which ensures that processes on subsequent nodes reside close to one another in the Dragonfly topology as described above. However, the scheduler is the entity which obtains the original list of nodes for any specific job, which does not necessary have topology information available.

### B. Point-to-point Benchmark Results

Two benchmarks from IMB are used to measure the costs of typical point-to-point MPI communications: *Sendrecv* and *Exchange*. The *Sendrecv* benchmark in IMB is based on the MPI routine `MPI_Sendrecv`. In this benchmark, each process (i.e. MPI rank $r$) sends message to the *next* (i.e. rank $r+1$) process and receives message from the *previous* (i.e. rank $r-1$) one. The *Exchange* benchmark in IMB simulates typical communication pattern that often occurs in domain-decomposed application. In this benchmark, each process sends two messages to the *next* and *previous* rank each. IMB uses `MPI_Isend` (non-blocking send) to send the messages. Each process also receives one message from each of the neighboring processes using `MPI_Recv`. The data type `MPI_BYTE` is used for both benchmarks.

These benchmarks were run on each hierarchy of the Cray XC30 network on a otherwise quiescent system. In particular, for each benchmark we use the MPI communicators as described in Section III. The *green* communicators represent the first hierarchy of the XC30 network. The benchmark results for this first hierarchy in our figures are indicated by green curves and points. The second hierarchy of the network is represented by the *black* communicators. Because of the way these communicators are set up, the only difference between the first (*green* communicators) and the second (*black* communicators) hierarchy of the network reflected in these benchmarks is the chassis backplane versus the electrical connection between chassis. The benchmark results for the *black* communicators in our figures are indicated by black curves and points.

Figures 2 and 3 show the results of these benchmarks. On both figures, the two different point types represent the two benchmarks: square for *Sendrecv* and circle for *Exchange*. The different colors represent the first (green) and second (black) network hierarchy as described above. Figure 2 shows the results using one node per Aries in both *single* and *ensemble* runs while Figure 3 uses all four nodes per Aries. Since IMB runs benchmarks with the number of processes as $[2, 4, 8, \ldots, 2^x < p, p]$ where $p$ is the rank of the communicator, only $4$ processes can be used in both the green and black communicators in Figure 2. For Figure 3 16 processes are used in both the green and black communicators.

From Figure 2 it can be seen that for these benchmarks on a system with more traffic (i.e. in ensemble runs), the intra-chassis latency (*green* link) is slightly lower. This trend persists for all message sizes. This advantage disappears for the single runs and when all the four nodes per Aries is used(Figure 3).

These same benchmarks were run using the *red* and *blue* communicators as described in Section III. The results of these benchmarks are shown on Figure 4 as red and blue curves, respectively. The results of green communicator benchmarks from Figure 3 are re-plotted here for reference. There is virtually no detectable differences in term of the latency of these benchmarks on the green, red, and blue communicators. This is not surprising since router-to-router hops in XC30 only adds approximately 100 ns of latency, well below measurable differences in these benchmarks.

The bandwidth as a function of message size is plotted for these point-to-point benchmarks in Figure 5 for the different hierarchies of the network. Except for the differences at

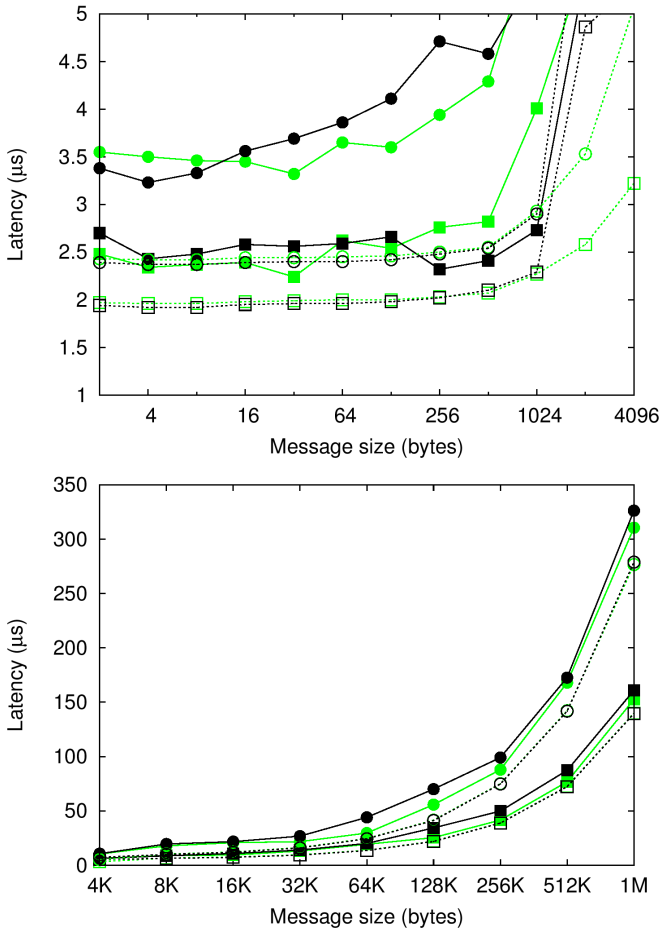


Figure 2: Latency as a function of message size on Darter for *Sendrecv* (square point) and *Exchange* (circle) benchmarks with four MPI ranks. The different colors indicate results on first (green) and second (black) dimension on the all-to-all network in two-cabinet group. The solid lines are for ensemble runs and the dashed lines are for single runs. The upper panel is the result for message size up to 4 Kilobytes and the lower panel is the result for message size up to 1 Megabytes.

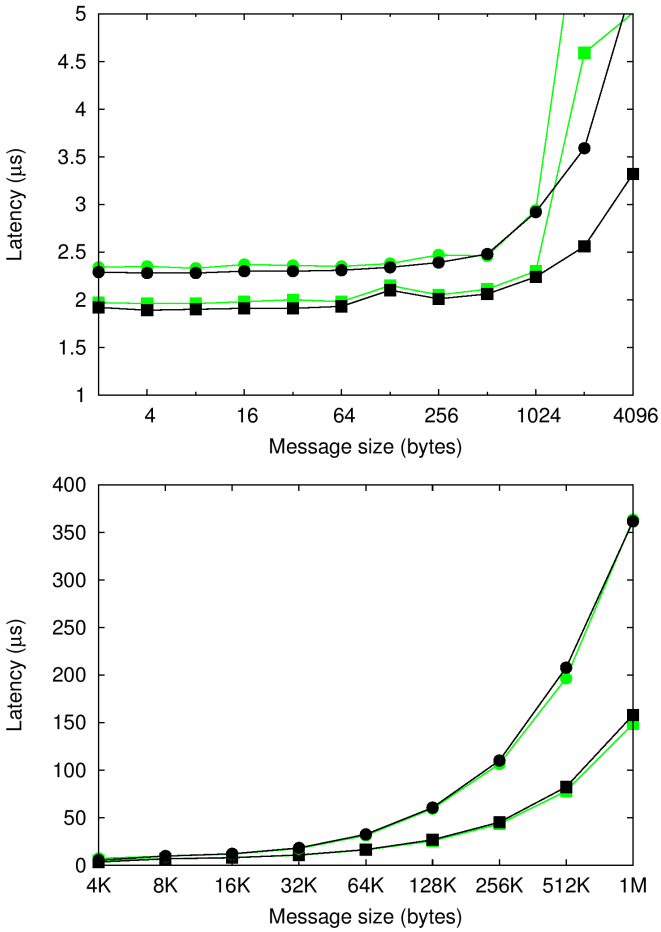


Figure 3: The same benchmarks as Figure 2 except all nodes attached to each Aries are member of the same MPI communicator. The results for 16 MPI rank benchmarks are plotted.

message size 2KB and 4KB, the bandwidths on the green, black, red, and blue communicators for these benchmarks are virtually the same, and increasing with the size of the messages. This tells us that even at the largest message size of these benchmarks, we have not saturated the available bandwidth on any of the network links in the system.

One noticeable trend in these benchmarks is the steep increase in latency for messages size greater than 1024 bytes. A possible explanation for this is that 1024 bytes is the default cut-off point for short message facility for these point-to-point communications. To try to understand this behavior, the *Sendrecv* benchmarks were run with the same configuration as in Figure 3, except the following environmental variable setting change: `MPICH_GNI_MAX_VSHORT_MSG_SIZE=8192`. Figure 6 shows the results of this change. For large messages, we also see a change on the latency curve steepness at around 128 Kilobytes for these benchmarks. Therefore it is advisable for application developers to keep point-to-point messages below these two cut-off points on this system.

### C. Collective Benchmark Results

For the collective operation benchmarks, we used *Allreduce*, *Allgather*, and *Alltoall* from IMB. As can be inferred from their names, these benchmarks are based on the MPI routine `MPI_Allreduce`, `MPI_Allgather`, and `MPI_Alltoall`, respectively. These benchmarks are ordered in increasing complexity in term of the communication patterns and are commonly used in many applications. For *Allreduce* benchmark, the reduction operation is `MPI_SUM` for `MPI_FLOAT` data type. The other two benchmarks uses

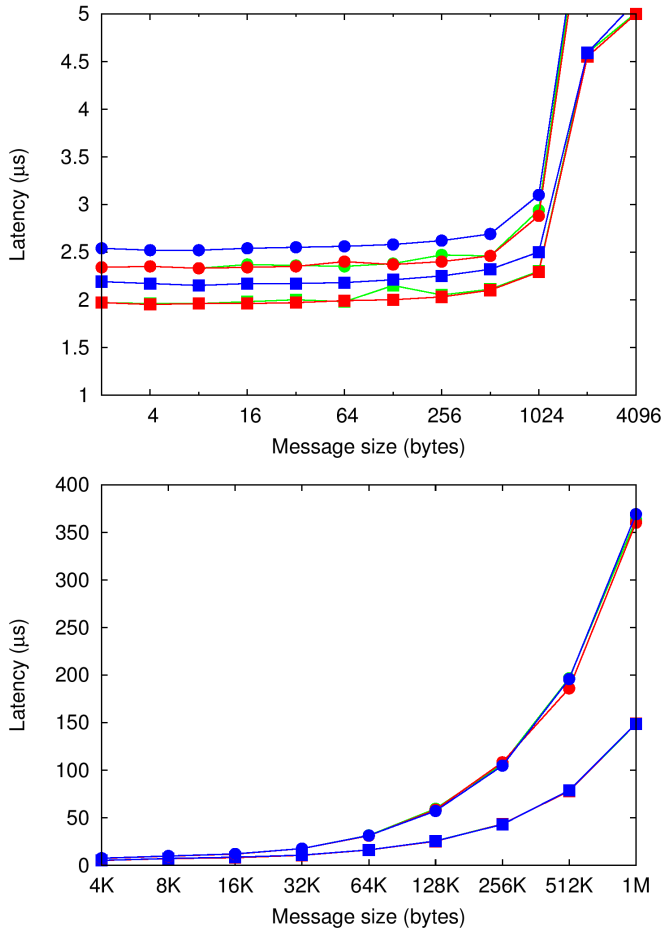


Figure 4: The same benchmarks as Figure 3 except across diagonal Aries in a group is shown in red curves instead of black link results.

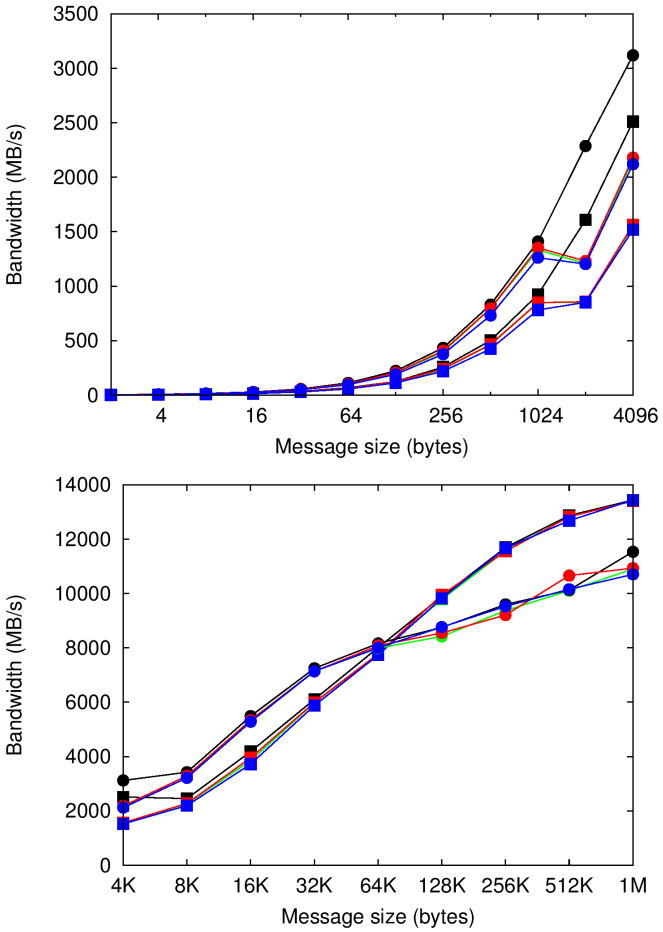


Figure 5: Bandwidth as a function of message size for *Sendrecv* (square point) and *Exchange* (circle) across green, black, red, and blue communicators.

`MPI_BYTE` data type.

Figures 7 and 8 show the results of these benchmarks. As with the point-to-point benchmarks, we ran these on each hierarchy of the XC30 network. The results for these hierarchies are indicated by the different colors in the figures, as previously described. On these figures, three different point types represent the three benchmarks: square for *Allreduce*, circle for *Allgather*, and triangle for *Alltoall*.

Figure 7 shows the latency of these benchmarks as a function of message size with 4 MPI ranks per communicator. The solid lines indicate the result of the ensemble runs while the dashed lines indicate the single runs. From Figure 7, we notice that for *Allreduce* and *Allgather* benchmarks the latency of intra-chassis processes (green link with green communicators) is slightly worse than the latency on the inter-chassis processes (black link with black communicators) until the cut-off point at 1 Kilobytes message size. For messages larger than 1 Kilobyes, the reverse is true: green communicators (intra-chassis) has slightly lower latency than the black communicators (inter-chassis). If the results for these two benchmarks for small messages are slightly surprising, the results for the *Alltoall* benchmark are even more so. For all message sizes, the *Alltoall* benchmark on the green link has anywhere from 30 - 200 percent higher latency than on the black link. For the small-size messages, the latency trend is more erratic than the trend for larger messages. The same trend and erraticism did not occur for the single runs (dashed lines) where only a single node per Aries were used. In this case, the latency on green-link and black-link are nearly the identical. These results seem to indicate that the inter-chassis backplane connection performance is more sensitive to the busy traffics for complicated communication pattern such as *Alltoall*.

On Figure 8 we show the results of the same benchmarks when all nodes attached to the Aries form the same communicator in each network hierarchy. On this figure, we show the the benchmarks results of 16 MPI ranks communicators. The latency for the *Allreduce* and *Allgather* benchmarks are

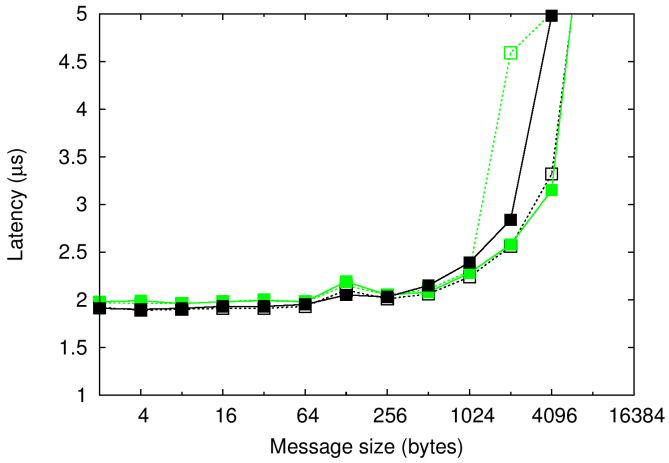


Figure 6: *Sendrecv* benchmark on green and black communicators as in Figure 3 but with `MPICH_GNI_MAX_VSHORT_MSG_SIZE` set to 8192. Data from Figure 3 is re-plotted as dashed line for reference.

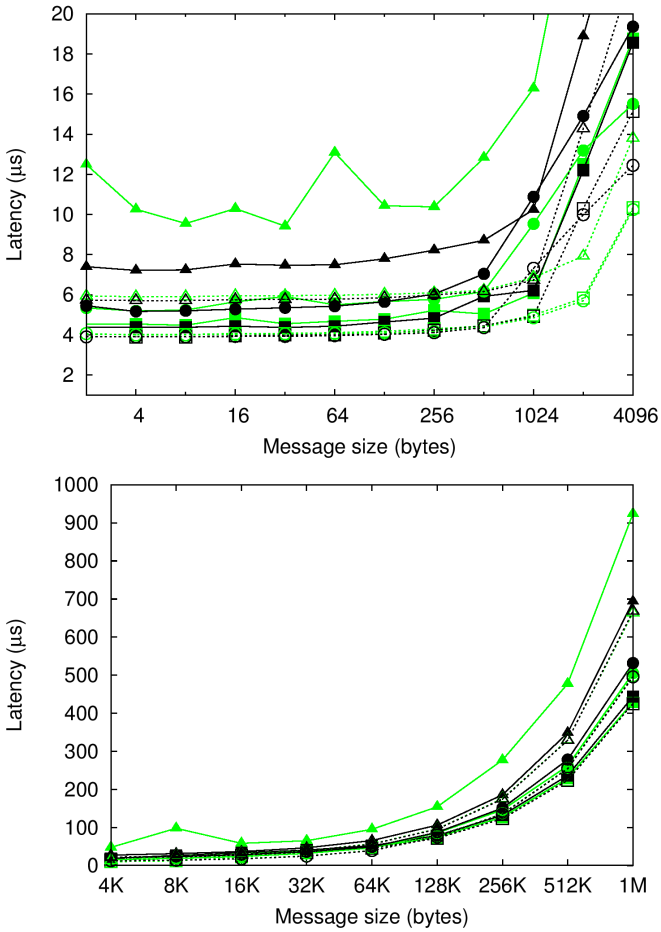


Figure 7: Latency as a function of message size on Darter for IMB collective benchmarks with four MPI ranks. The different point types indicate different benchmarks: square for *Allreduce*, circle for *Allgather*, and triangle for *Alltoall*. The different colors indicate results on first (green) and second (black) dimension on the all-to-all network in two-cabinet group. The solid lines are for ensemble runs and the dashed lines are for single runs. The upper panel is the result for message size up to 4 Kilobytes and the lower panel is the result for message size up to 1 Megabytes.

nearly identical for all four colors of the communicators up to 128 bytes message size. However for the *Alltoall* benchmark, here we again see something rather puzzling. As in the result shown in Figure 7, the latency for the inter-chassis results (green curve) are higher than expected. For message size 128, 256, 512 bytes, the values are especially higher than the latency on the black communicators. The red and blue communicators latency fall in between of the green and black communicators latency for 128, 256, 512 bytes message sizes for the *Alltoall* benchmarks, and follow the green communicators latency for the other collective benchmarks.

On Figure 7 we notice the large changes in latency curve after certain message size. For the *Allreduce* benchmark, this cut-off point is at 1024 bytes message size. This cut-off point seems to happen at smaller message size for the more complicated communication patterns. When more ranks are involved, such as in Figure 8, the cut-off points is at even smaller message sizes. For large messages, the slope of the latency curve is increasing for all benchmarks for message size above 64 Kilobytes on both Figure 7 and 8.

The solid lines on Figure 9 shows *Alltoall* benchmark results with the following environment variable setting: `MPICH_ALLTOALL_SHORT_MSG=8192` to adjust the cut-off points at which algorithm for short messages is used. However, with this setting, the latency on green communicators is even more erratic than before. The latency on black communicators is also slightly worse on all message sizes. Overall, increasing the value of this environmental variable from the default seems to produce worse latency on both green (intra-chassis) and black (inter-chassis) hierarchy of the XC30 network.

### D. Applications Benchmark

The last set of tests we ran to investigate the effect of process placement on communication costs utilized two scientific applications with typical communication patterns shared by many other applications. For these tests, we ran the applications on system when it was both busy and quiescent and measure the performance differences due to different process placement. Major causes of performance variance over multiple number of runs on shared resources are the availability of communication bandwidth, communication

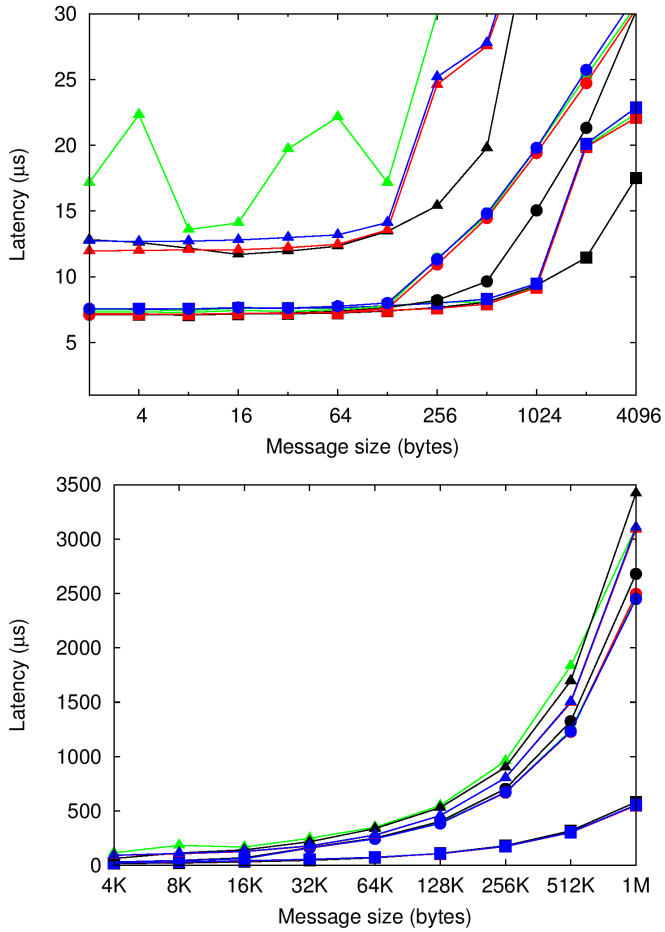


Figure 8: The same benchmarks as Figure 2 except all nodes attached to each Aries are member of the same MPI communicator. The results for 16 MPI rank benchmarks are plotted. Additionally red and blue curves showing the results of these benchmarks with *red* and *blue* communicators are also displayed.

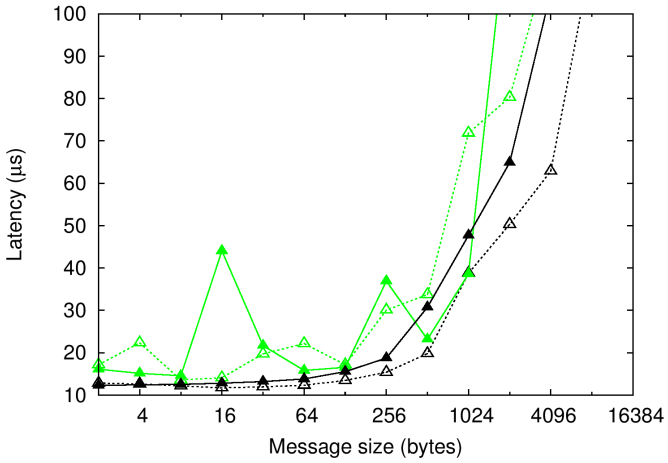


Figure 9: *Alltoall* benchmark on green and black communicators as in Figure 8 but with `MPICH_ALLTOALL_SHORT_MSG` set to 8192. Data from Figure 8 is re-plotted as dashed lines for reference.

latency, I/O bandwidth and latency, and the load on compute resources. Since on Cray system allocated compute nodes are dedicated only for the submitted job, and operating system jitters are minimized by the used of Compute Node Linux [6], these factors can be safely assumed to contribute only minimally to application performance variance. To avoid contribution from the I/O subsystem, other than the minimal informational output to `STDOUT` and parameters file reading at application start-up, on these tests I/O was completely turned off. Therefore it may be assumed that the biggest contribution for the performance variance are from communication subsystem.

A simplified way to quantify the process placement of a job is by a measure of how spread-out the nodes used to run the job are. We use the average distance between node as defined by their NIDs as this measure. This average node-distance $\overline{D}_n$ is simply

$$\overline{D}_n = \frac{\sum_{i=1}^{n-1} \mathrm{NID_{i+1}} - \mathrm{NID_i}}{n-1}, \tag{2}$$

where $n$ is number of nodes allocated to the job. For a job whose nodes are right next to each other, we have $\overline{D}_n = 1$. $\overline{D}_n$ increases as nodes are further apart in physical location as indicated by its NID (as described in Subsection IV-A). There are however two caveats to this simple scheme. The first one is that it ignores any information on network topology and therefore assumes that distance varies linearly with NID. The net effect of this is that $\overline{D}_n$ is independent of network distance and therefore can not be used to infer information regarding network performance such as latency. The second caveat, which is related to the first one, is that $\overline{D}_n$ is not unique. Two completely different arrangement of process placement on the nodes may yield the same $\overline{D}_n$ and yet has different communication characteristic due to the network topology. Despite these limitations, a measure of performance versus the average node-distance number plot may give us insight into the effect of process placement to performance variance of an application if any.

The first application we used is *GenASiS* (General Astrophysical Simulation System). GenASiS is a multiphysics code primarily developed for large-scale simulations of astrophysical phenomena, with initial emphasis on the simulations of core-collapse supernovae [7][8]. It has been run at-scale on the largest Cray supercomputers such as Jaguar and Titan at Oak Ridge Leadership Computing Facility to produce some of the first three-dimensional results on the effect of turbulent to magnetic fields generation in supernova environment [9][10]. For its compressible hydrodynamics

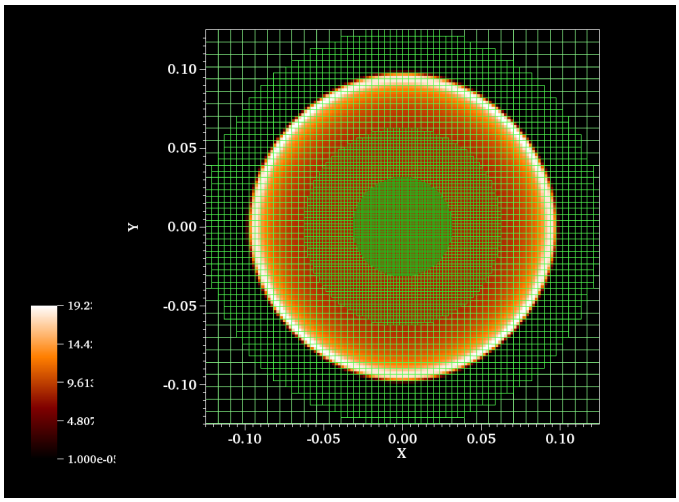


Figure 10: A snapshot of the pressure gradient of Sedov-Taylor blast wave after it evolves for some time.

solver, GenASiS implements finite-volume methods based on approximate Riemann solvers with second-order accuracy using the so-called HLL-type Riemann solver. Second-order temporal accuracy is achieved with Total Variation Diminishing Runge-Kutta method. In GenASiS, the meshes are domain-decomposed for parallel processing. The hydrodynamics solver requires several ghost exchange communication every time step with a global reduction via `MPI_Allreduce` to determine the largest time step that can be taken.

We used two-dimensional Sedov-Taylor blast-wave problem [11][12] to benchmark the effects of different process placement to the performance of GenASiS. Sedov-Taylor blast-wave is a classic test in computational astrophysics in which the self-similar evolution of a strong shock wave expanding into uniform medium (see figure 10). We ran this test problem at two different resolution: $8192 \times 8192$ cells distributed evenly on $4096$ processes with $256$ nodes, and $2048 \times 2048$ cells with $256$ processes with $16$ nodes. For both resolutions, we let the the simulations evolve for $5000$ time step and used the total elapse time of the application as a measure of performance.

Figure 11 shows the results of our tests with Sedov-Taylor blast wave problem. For both problem size, the minimum total elapsed time from multiple runs was picked as a reference point $t_0$. A relative time difference $\delta t_r$ can then be computed using $t_0$ as reference as

$$\delta t_r = \frac{t_i - t_0}{t_0} \tag{3}$$

where $t_i$ is the total elapsed time for run $i$. The values $t_0 = 578.109$ seconds and $t_0 = 571.908$ seconds were used for the $4096$-processes and $256$-processes run, respectively.

From Figure 11 we notice that for this application, for

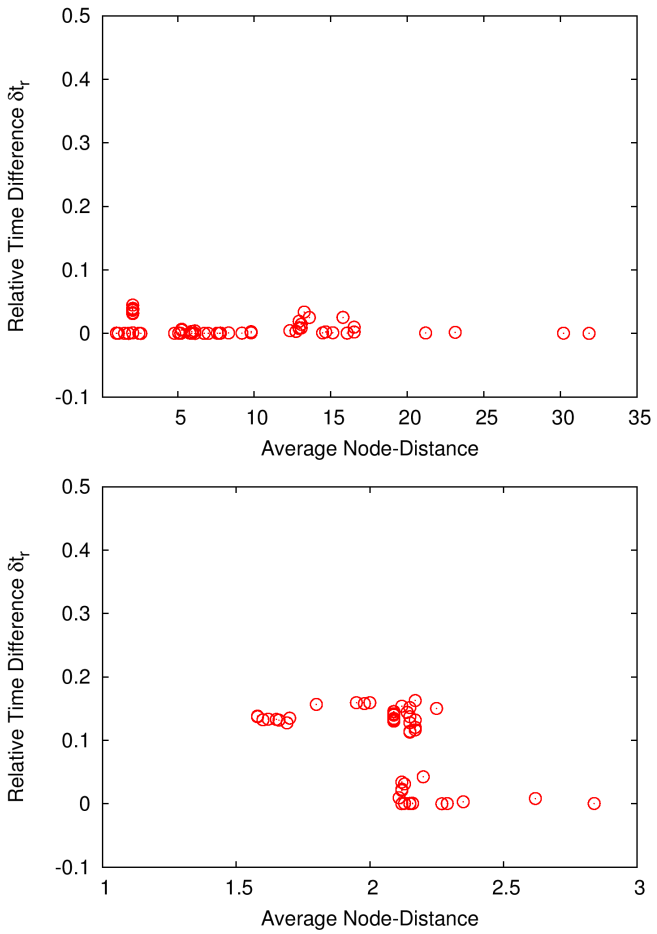


Figure 11: Relative time differences versus average node-distance for Sedov-Taylor blast wave test problem with $4096$ processes (upper panel) and $256$ processes (lower panel).

job with relatively small number of nodes (upper panel), there is essentially no significant performance differences due to job placement. This is consistent with the advocated attribute of the XC30 system with the Dragonfly topology [2]. However, for jobs with relatively large number of nodes on busy system (lower panel on Figure 11), there seems to be some performance benefits when the job occupies nodes with some physical distance between them. This is somewhat counter-intuitive. One may speculates a reasoning for this observation that by using nodes that are less packed together, network traffics have possibilities to take different routes on this topology and thus better able to avoid congestion. Since on this topology the node-to-node network distance is independent of physical distance, the sparseness of the job placement does not contribute very much to the increase of latency.

The second application we used is *PSPFFT*. PSPFFT is an implementation of FFT-based Poisson's equation solver for isolated three-dimensional system on unigrid mesh using

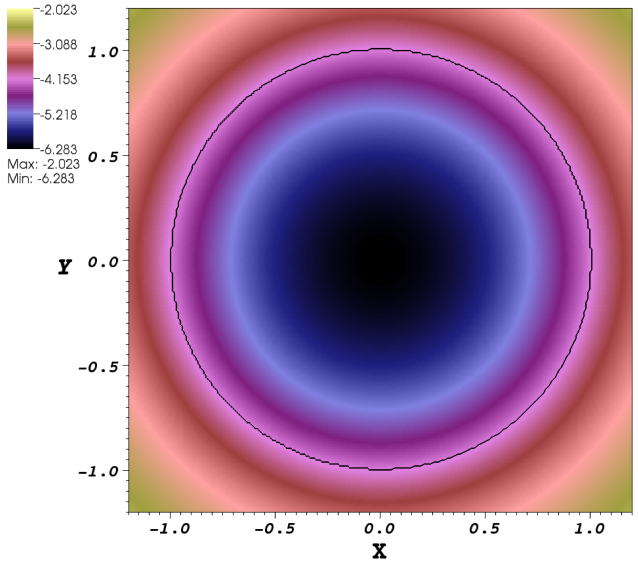


Figure 12: A 2D slice of 3D mesh showing the gravitational potential of a unit sphere with uniform mass density $\rho = 1$. with radius $R = 1$ as indicated by the solid black line.

FFT on a distributed memory parallel computer [13]. The mesh with the source distribution of the Poisson's equation is domain-decomposed and distributed across multiple processes as mesh blocks to be solved in parallel. For multi-dimensional FFT to be performed efficiently, data movement is required to transpose the mesh and its data to arrangements more natural to highly optimized serial FFT libraries such as FFTW. In this application, this is accomplished via `MPI_Alltoall` routine involving different number of processes as members of MPI sub-communicators. Therefore, all-to-all is the major communication pattern used by this application.

For the tests, we ran PSPFFT to solve the gravitational potential of a three-dimensional homogenous sphere at two different resolutions: $768 \times 768 \times 768$ and $384 \times 384 \times 384$ with 4096 processes (256 nodes) and 512 processes (32 nodes), respectively. We measured the total elapsed time for the application to solve the problem 3000 times and 4000 times at the high and low resolution, respectively, and plot it as a function of the average node-distance.

Figure 13 shows the results of our tests with PSPFFT. For these tests, $t_0 = 1508.94$ seconds and $t_0 = 1413.81$ seconds were used as the performance reference points for the 4096-processes and 256-processes run, respectively. As in the case with the Sedov-Taylor test problem, for the small-size job we do not observe performance differences due to job placement. For larger-size job, unfortunately we

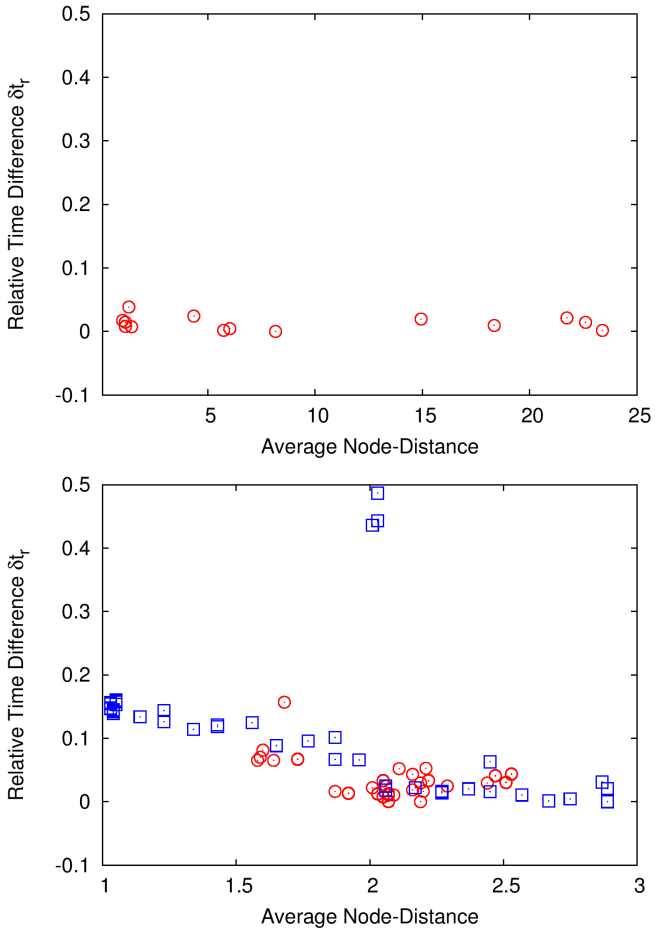


Figure 13: Relative time differences versus average node-distance for PSPFFT application with 4096 processes (upper panel) and 512 processes (lower panel). On the lower panel, blue squares indicate data from runs when the system is completely empty.

do not have enough data points for cases with $\overline{D}_n < 2$, although from what is available the data seems consistent with Figure 11. For this test problem we also show results with a dedicated system, indicated by blue squares. The jobs were then placed using `aprun -l <nodelist>` feature where nodelist file was created in a way such that $\overline{D}_n$ varies from 1 to 3. Here we see a trend that increasing $\overline{D}_n$ gives better performance.

## V. Conclusion

In this paper we have performed benchmarks on the different dimensions and hierarchy of the newly released Cray XC30 with Dragonfly network topology. The motivation for this endeavor was to investigate the effect of rank placement on communication costs of typical communication patterns, if any, since this effect is not immediately apparent across the different dimensions of the all-to-all network. We have

found that in most cases this effect is minimal and below the measurable level from user perspective on many common communication patterns. Although different dimensions and hierarchy of the network topology do in fact performs as well as expected, we have also noticed small peculiarity with regards to the *Alltoall* benchmark performance on the *green* links of the intra-chassis connection. We do not have enough knowledge at this point to draw any conclusion from the results of that particular benchmark.

We have also measured the performance differences due to job placement on two scientific applications with typical communication patterns shared by many other applications. We observed that for relatively large job size, our results show some performance benefits from the increase of average distance between nodes when the system is busy (i.e. more than 80 percent utilized). This results is in contrast to the conventional wisdom that usually advocates to pack the placement of a job as close as possible. Although we speculated a possible explanation for this observation, further work is warranted to corroborate and understand this observation.

The Aries interconnect has the capability of dynamically routing packets via minimal and non-minimal paths [2]. For the experiments using communicators which attempt to solely utilize the inter-chassis links, both intra- and inter-chassis network links, or all three all-to-all network links, non-minimal routing paths would result in higher latencies, improved bandwidth, and reduced impact from contention, if present. This type of routing would have the potential to equalize the performance differences due to rank placement. There may be mechanisms to control this behavior for more rigorous experimentation; however, only the default behavior of the system was measured. There is a future need to identify a mechanism to identify if messages are routed minimally or non-minimally in order to understand this behavior.

## ACKNOWLEDGMENT

This research used resources at the National Institute for Computational Sciences, funded by the National Science Foundation (NSF).

## REFERENCES

[1] John Kim, Wiliam J. Dally, Steve Scott, and Dennis Abts. Technology-Driven, Highly-Scalable Dragonfly Topology. *2008 International Symposium on Computer Architecture*, pages 77–88, June 2008.

[2] Greg Faanes, Abdulla Bataineh, Duncan Roweth, Tom Court, Edwin Froese, Bob Alverson, Tim Johnson, Joe Kopnick, Mike Higgins, and James Reinhard. Cray Cascade: A scalable HPC system based on a Dragonfly network. In *2012 International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–9. IEEE, November 2012.

[3] Robert Alverson, Duncan Roweth, and Larry Kaplan. The Gemini System Interconnect. *2010 18th IEEE Symposium on High Performance Interconnects*, pages 83–87, August 2010.

[4] Michael Karo, Richard Lagerstrom, Marlys Kohnke, and Carl Albing. The application level placement scheduler. *Cray User Group*, pages 1–7, 2006.

[5] Intel. Intel MPI Benchmarks 3.2.4 User Guide.

[6] David Wallace. Compute Node Linux: Overview, progress to date, and roadmap. *Proceedings of the 2007 Cray User Group Annual ...*, pages 1–8, 2007.

[7] RD Budiardja. *Towards Simulations of Binary Neutron Star Mergers and Core-Collapse Supernovae with GenASiS*. PhD thesis, University of Tennessee, 2010.

[8] Christian Y. Cardall, Reuben D. Budiardja, Eirik Endeve, and Anthony Mezzacappa. GenASiS: General Astrophysical Simulation System. II. Nonrelativistic Hydrodynamics. *Submitted to the Astrophysical Journal*, July 2012.

[9] Eirik Endeve, Christian Y. Cardall, Reuben D. Budiardja, and Anthony Mezzacappa. Generation of Magnetic Fields by The Stationary Accretion Shock Instability. *The Astrophysical Journal*, 713(2):1219–1243, April 2010.

[10] E Endeve, C Y Cardall, R D Budiardja, and A Mezzacappa. Magnetic field generation by the stationary accretion shock instability. *Journal of Physics: Conference Series*, 125:012006, July 2008.

[11] L. I. Sedov. *Similarity and Dimensional Methods in Mechanics, Tenth Edition*. CRC Press, 1993.

[12] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. FLASH: An Adaptive Mesh Hydrodynamics Code for Modeling Astrophysical Thermonuclear Flashes. *The Astrophysical Journal Supplement Series*, 131(1):273–334, 2000.

[13] Reuben D. Budiardja and Christian Y. Cardall. Parallel FFT-based Poisson solver for isolated three-dimensional systems. *Computer Physics Communications*, 182(10):2265–2275, October 2011.