Exploring and Quantifying How Communication Behaviors in Proxies Relate to Real Applications

Omar Aaziz
Sandia National Laboratories
Albuquerque, NM 87123
Email: oaaziz@sandia.gov

Jeanine Cook
Sandia National Laboratories
Albuquerque, NM 87123
Email: jeacook@sandia.gov

Jonathan Cook
New Mexico State University
Las Cruces, NM 88003
Email: joncook@nmsu.edu

Courtenay Vaughan
Sandia National Laboratories
Albuquerque, NM 87123
Email: ctvaugh@sandia.gov

***Abstract*—Proxy applications, or proxies, are simple applications meant to exercise systems in a way that mimics real applications (their parents). However, characterizing the relationship between the behavior of parent and proxy applications is not an easy task. In prior work [1], we presented a data-driven methodology to characterize the relationship between parent and proxy applications based on collecting runtime data from both and then using data analytics to find their correspondence or divergence. We showed that it worked well for hardware counter data, but our initial attempt using MPI function data was less satisfactory. In this paper, we present an exploratory effort at making an improved quantification of the correspondence of communication behavior for proxies and their respective parent applications. We present experimental evidence of positive results using four proxy applications from the current ECP Proxy Application Suite and their corresponding parent applications (in the ECP application portfolio). Results show that each proxy analyzed is representative of its parent with respect to communication data. In conjunction with our method presented in [1] (correspondence between computation and memory behavior), we get a strong understanding of how well a proxy predicts the comprehensive performance of its parent.**

***Index Terms*—Workload characterization; Proxy applications; Performance evaluation; Big data**

I. Introduction

Proxy applications, sometimes called mini-apps or representative applications, are relatively small programs that attempt to capture some fundamental aspects of a real, and much larger, application or class of applications. Being much smaller, proxy apps are designed to be easily built, installed, and executed. They typically have few build constraints and dependencies, simple input specifications, and are thus usable with little human overhead or time commitment. Their purpose is to provide an easy-to-use-mechanism to evaluate system performance, find hardware bottlenecks, perform algorithmic, parallel, or system design exploration, and in general gain an understanding of how a particular class of applications might perform on an HPC system, and how to best design and configure both the system and the application to maximize performance.

Underlying all this is the assumption that the proxy app does indeed capture the essence(s) of the real application. Recent work has attempted to address this. One group looked at a specific proxy and parent, e.g., [2], and also explored generalizing their work [3]. Others investigated extracting kernels from the real apps instead of comparing them to proxy apps [4], and others first compared proxy apps to benchmarks as an intermediate step towards comparing to real applications [5]. Overall, though, this is an area that needs more work.

This paper presents an exploration of quantifying the correspondence between parents and proxies specifically in the communication domain. This work builds on our work of creating a methodology for measuring parent/proxy correspondence in general. The results presented here show the potential of our communication comparison methodology and in conjunction with the comparison method presented in prior work (computation and memory behavior) we potentially have a comprehensive, quantitative way of understanding the representativeness of proxy applications.

Section II discusses our prior work and the context of this work, Section III presents the methodology we used, Section V presents our results and analysis, and Sections VI and VII presents related work and conclusions.

II. Background

Proxy applications often keep track of their own time and computation and output some performance data, but this is generally *domain specific*. This can be used to compare various runs of the proxy itself on different platforms or with different build and run configurations, but given that the proxy does not embody the full parent's computation, it is not self-evident that, say, the "atoms per second" processing rate in a molecular dynamics proxy should be the same as that in the parent. Thus the domain level seems like the wrong level at which to compare parents and proxies.

In our prior work [1], we proposed a method that evaluates the performance of both the real and proxy applications at a level below that of their domain-specific performance: that of how they exercise the hardware. This avoids the issue of domain-specific metrics and targets the real question of performance and correspondence, which is: do they utilize and exercise the HPC system in similar ways?

In that work we divided the resources used into four domains: basic node (processors and memory), accelerator (e.g., GPUs), communication, and storage I/O. We then proposed a methodology whereby a vector of measurements is collected in each domain for each run of the parents and proxies

TABLE I
MOST SIGNIFICANT MPIP PER-ROUTINE RAW METRICS

Metric type	Description	Routines for
Apptime_%	% of total application time	send, isend, recv, irecv, sendrecv, allreduce, bcast, wait, waitall, barrier
MPI_%	% of total MPI time	send, isend, recv, irecv, sendrecv, allreduce, bcast, wait, waitall, barrier
Count/Time	total calls / application time	send, isend, recv, irecv, sendrecv, allreduce, bcast, wait, waitall, barrier
AvgByte/Time	avg bytes per call / application time	send, isend, recv, irecv, sendrecv, allreduce, bcast
Simplified/Reduced mpiP Metrics		
all_send = send + isend all_recv = recv + irecv all_multi = bcast + sendrecv + allreduce all_wait = wait + waitall + barrier		

under investigation; the vector is reduced in size by principal components analysis (PCA); and then the resulting values are clustered using a hierarchical clustering algorithm. In this prior work, we used hardware performance counter data to show correspondence between parent and proxy with respect to computation and memory behavior, we used mpiP data for communication correspondence, and we did not investigate the accelerator or storage I/O domains.

Although our method worked very well for demonstrating proxy/parent correspondence for computation and memory behavior, the mpiP data we chose to represent communication behavior did not adequately show correspondence or divergence. A quick summary of that result is given here.

Using mpiP, we extracted the Aggregate Time and Aggregate Sent Message Size from the instrumented application and, for the 20 most used call sites, aggregated the data per MPI function, then computed the metrics shown in Table I. The functions listed are the only MPI functions with significant usage across our suite of applications.

Since different applications use different MPI functions, many of the collected metrics in Table I are zero. Further, some proxy/parent applications do not use the same MPI functions, and even use different types of communication primitives (collectives versus peer-to-peer). To correct for this disparity, we reduced and simplified the mpiP data by combining data from categorically similar MPI functions into a generic category, shown at the bottom of the table. Our four categories are: send, receive, multi-way communication, and wait. These categories reduce the mpiP metrics into a more consistent set of category metrics across all of the applications. Our communication domain data vector, then, was sixteen values—the four metrics for each of the four categories.

We ran our experiments on an Intel Broadwell system using four proxy/parent pairs (will be described further in Section V):

- SW4lite and SW4 (seismic modeling)
- Nekbone and Nek5000 (thermal transport)
- SWFFT and HACC (cosmology/FFT)

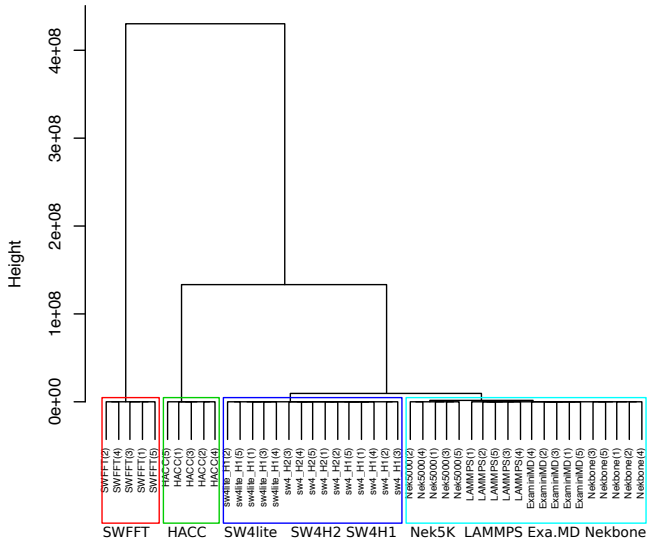


Fig. 1. Communication Similarity, Broadwell

- ExaMiniMD and LAMMPS (molecular dynamics)

Although we ran each application, both parent and proxy, over a variety of configurations (e.g., number of nodes; number of cores/node), we chose one configuration (128 MPI ranks distributed across 8 nodes, using 16 cores per node) and input size (Table IV) for data analysis since we are trying to understand similarity under equivalent conditions for both proxy and parent application. The input used was intended to represent an Exascale challenge problem. For each configuration and for each application, we collected data for five distinct runs. Each run's data is kept independent of other runs.

Figure 1 shows the clustering that results from the communication (MPI) data, for the Broadwell platform only. Note first the scale of the height axis, and how far away the final two clusterings are from the lower clusterings. This indicates very little similarity at these levels (hierarchical clustering will always connect everything, eventually). Even the connecting of the SW4* cluster and the Nek*/LAMMPS/ExaMiniMD cluster is, relatively, quite high on the axis. In looking at application and proxy implementations, some do use the same MPI communication primitives and style, and these cluster well together. Others, because they were written separately, may try to implement a similar *model* of communication, but they do so with different MPI primitives, and end up having very different aggregate statistics over the mpiP profile data.

We viewed this result as indicating that our basic attempt at abstracting the mpiP data away from specific MPI routines was not enough to actually capture an abstract-enough model of communication where different applications might end up with similar data. This motivated the approach presented in this paper that attempts to characterize communication patterns and other detailed communication characteristics to show correspondence in communication behavior between parent and proxy applications.

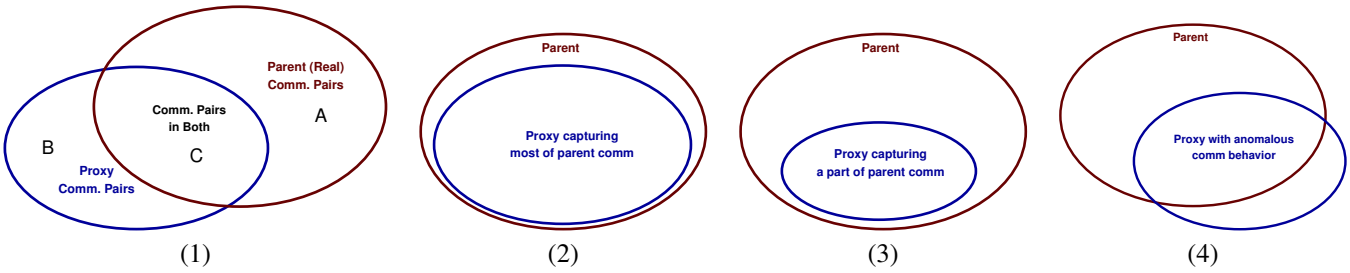


Fig. 2. Relationships Between Pairwise Communication in a Parent (Real) and Proxy Application.

III. Methodology

The basis of our methodology for characterizing the correspondence of proxy/parent communication comprises similarity characterization of: (1) communication patterns, and (2) a communication vector of values gathered from mpiP output. We apply different similarity measures to communication patterns and mpiP metric vectors.

A. Pairwise Pattern Data

We generate a communication pattern for each application using the CrayPat toolset. These communication patterns are generated from the peer-to-peer statistics which capture point-to-point communicating MPI processes and the total number of calls between these points. We then use the CrayPat Apprentice2 tool to visualize the data and generate a plot that clearly shows frequency of communication between MPI processes, which realizes a communication pattern. Apprentice2 is also able to dump the data from the plot into a CSV file, which allows us to do additional analysis on the data.

This point-to-point data is recorded as the number of messages sent from a specific source process (rank) to a specific destination process (rank). We use this data in the formally defined metrics below.

Figure 2 shows the relationships that can occur between the set of communicating process pairs in a parent application and its proxy. Figure 2.1 defines the relationships, and we label the unique areas A, B, and C for future reference. Figures 2.2–2.4 show three possible relationships: a proxy capturing most of a parent's communication; a proxy capturing only a portion of the parent communication (possibly intentional); and a proxy having communication that does not match anything in the parent. This is only an abstract view of pairwise communication and does not address the volume of communication between pairs, nor the time-varying behavior of the communication. Typically, the expectation is that area B is small or nonexistant, area C is large, and area A is small, or well-defined if the proxy is purposely meant to not model it.

Let N be the set of processes in the parallel computation, where $n \in N$ is an identifying integer of the process (e.g., its MPI rank). We assume that the parent and proxy applications are executed with the same N.

Let R be the set of pairwise non-zero message counts in the parent (real) application, where $m_{s,v} \in R$ is the number of messages sent from process s to process v, $s \in N$ and $v \in N$. The shorthand m_p will be used as well, where p is the tuple (s, v). R does not contain zero-valued elements, so for any (s, v) that has no messages, there is no element in R. Similarly, let P be the set of pairwise non-zero message counts in the proxy application. In Figure 2.1, R is areas A and C, and P is areas B and C, though each with their own data.

We also form sets of each of the parent and proxy application data *filtered and zero-padded* by the other. The first is defined as:

$$R_P = \begin{cases} m_p \in R \; if \; m_p^{'} \in P \\ 0_p \; if \; p \notin R \wedge p \in P \end{cases}$$

That is, R_P is R with elements dropped that do not occur in P, and augmented with zero elements that occur in P but not in R. We call it R filtered by P. The reverse, P filtered by R, is also defined:

$$P_R = \begin{cases} m_p \in P \; if \; m_p^{'} \in R \\ 0_p \; if \; p \notin P \wedge p \in R \end{cases}$$

Note that $|R_P| = |P|$ and $|P_R| = |R|$. In Figure 2.1, R_P is parent data in area C and 0's in area B, and P_R is proxy data in area C and 0's in area A.

We also define the full augmentation of both sets, adding zero elements where the element exists in the other but not in it:

$$R_A = \begin{cases} m_p \in R \\ 0_p \; if \; p \notin R \wedge p \in P \end{cases}$$

$$P_A = \begin{cases} m_p \in P \\ 0_p \; if \; p \notin P \wedge p \in R \end{cases}$$

Note that $|R_A| = |P_A|$, which is greater than or equal to both $|R|$ and $|P|$. In Figure 2.1, R_A is parent data in areas A and C, and 0's in area B; and P_A is proxy data in areas B and C, and 0's in area A.

With these defined and constructed sets, we can then define our metrics that attempt to measure the similarity between the parent and proxy communication data. These metrics are:

1) the percentage of parent communication that is covered by the proxy, by number of messages and by number of process pairs;

$$\#msg = \frac{\sum m_p \in R_P}{\sum m_p \in R} * 100$$

$$\#pair = \frac{|\{m_p \in R_P \ s.t. \ m_p \neq 0\}|}{|R|} * 100$$

2) the percentage of proxy communication that is covered by the parent, by number of messages and by number of process pairs;

$$\#msg = \frac{\sum m_p \in P_R}{\sum m_p \in P} * 100$$

$$\#pair = \frac{|\{m_p \in P_R \ s.t. \ m_p \neq 0\}|}{|P|} * 100$$

3) statistical/correlation metrics comparing R_A and P_A;
4) statistical/correlation metrics comparing R_P and P;
5) statistical/correlation metrics comparing R and P_R;

The percentages in 1 and 2 are capturing how much communication in one is represented, or "covered", by the other—i.e., the pairs communicate in both. In relation to Figure 2.1, the formulae in 1 capture the amount of parent communication in C in relation to its total communication in A and C. The first computes this over the number of messages, and the second computes this over the count of communicating pairs in each area. The formulae in 2 are analogous, over B and C.

In items 3-5, the tests in 3 compare the full sets of pairwise communication data in the parent and proxy, the tests in 4 compare that part of the parent's communication that is "matches" the proxy, and the tests in 5 compare that part of the proxy's communication that matches the real. These are different views on the way the parent and proxy might be similar and different.

B. Vector Data and Clustering

The second part of our similarity characterization involves gathering a vector of data from mpiP for each proxy and parent application. We use a modified version (Section IV) of the *mpiP* lightweight profiling library [6] to collect basic communication metrics for our applications. MpiP collects statistical information about MPI functions (per call site) and produces a report at the end of the application's execution. From the mpiP output, we form a communication data vector that is then used as input to a clustering algorithm. Specifically, our vector for each application comprises data that represents a histogram of message size and respective frequency. We also include three summary metrics:

1) total number of messages
2) KB/sec - total size of data transferred (KB) divided by total execution time (sec)
3) MPI KB/sec - total size of data transferred (KB) divided by total time spent in MPI (sec)

Clustering algorithms are sensitive to large dimensionality data; principal component analysis (PCA) reduces the dimensionality of the data input. Because our communication vectors comprise only 13 values (10 message size buckets; 3 summary metrics), PCA is not necessary. We use hierarchical clustering [7] to understand similarity between proxy/parent pairs. We use the elbow method [8] for hierarchical clustering to determine the optimal number of clusters for the hierarchical clustering algorithm. Our hypotheses expect that proxy apps cluster with the real apps that they represent, and that the real apps do not cluster together, particularly if they are not of the same dwarf classification [9].

IV. Experimental Platform

For this work, we use an Intel Broadwell (Xeon E5-2695 v4) platform. Communication patterns are not hardware-dependent so we use only a single platform here. Table II presents the architecture details of the system used.

We use the Intel 18 compiler for all proxy/parent pairs; compiler flags are kept as consistent as possible across each proxy/parent pair and we replicate the compiler flags that are present in the distribution build files as close as possible.

As mentioned previously, we execute each proxy and application in several different scaling configurations, but always pin only one process per core. We report data only for a single configuration per application, each of which uses 128 MPI ranks distributed across 8 nodes, using 16 cores per node.

A. Data Collection Tools

For this study, we are using a modified version of mpiP. The standard version of mpiP collects information on point to point and collective communications that are binned by the size of the messages. For example, one such bin would be messages that are between 256 and 511 bytes. For each bin, mpiP calculates the volume of communication in that bin by adding the length of each message that falls within that bin. At the end of the simulation, the top twenty bins are displayed ranked by volume. In our modified version, we display all of the bins and also keep track of the number of messages that are in each bin.

B. Proxy and Parent Applications

This work is done as part of the DOE Exascale Computing Project (ECP) [10]. Therefore, we use applications that are being used in ECP Application Development [11] projects and use proxy applications that are in the current ECP Proxy App Suite 1.0 [12]. For this work, we chose the following four ECP proxy/parent application pairs:

- SW4lite and SW4 (seismic modeling)
- Nekbone and Nek5000 (thermal transport)
- SWFFT and HACC (cosmology/FFT)
- ExaMiniMD and LAMMPS (molecular dynamics)

SW4 [13], [14] is a geodynamics code that enables 3D modeling of surface topologies to understand the physics and impacts of earthquakes and other seismological events. The seismic wave equations are solved on locations that are specified either by Cartesian or geographic coordinates. The finite difference wave equations numerically simulate wave propagation to fourth-order, which is very accurate for calculating surface waves. Cartesian local mesh refinement is used to improve accuracy in regions near the free surface, where more resolution is needed to solve short wavelengths and maintain accuracy. SW4lite [15] is a scaled-down version of SW4 that has limited seismic modeling capabilities, but

TABLE II
HARDWARE CHARACTERISTICS OF BROADWELL PLATFORM

Component	Details
L1 data cache (per core)	32 KB, 8 way, 64 sets, 64B line size
L1 instr. cache (per core)	32 KB, 8 way, 64 sets, 64B line size
L2 cache (per core)	256 kB, 8 way, 512 sets, 64B line size
L3 cache (shared)	45 MB, 16–20way, 64B line size
Memory (per node)	128 GB DDR4-2400 MHz
Cores/threads	18/36
Sockets/node	2
Total nodes	1122
Interconnect	Intel OPA and Mellanox ConnectX4
Max Memory BW	76.8 GB/sec

does solve the elastic wave equation and uses some of the same numerical kernels as those implemented in SW4. Its limited modeling capability limits the types of surfaces and areas that can be used in the simulations. SW4lite is a close enough representation of SW4 that it is used to explore performance optimization, particularly with respect to memory layout and threading, but it is also representative of the computation and communication present in SW4.

Nek5000 [16], [17] is a spectral element code designed for large eddy simulation (LES) and direct numerical simulation (DNS) of turbulence in complex domains. It simulates thermal transport on a full range of scales set by the geometry encountered within a reactor. Nek5000 has a broad range of applications including vascular flow, ocean modeling, combustion, heat transfer enhancement, stability analysis and MHD (magnetohydrodynamic) flows. Nekbone [18] implements the computationally intensive linear solvers that account for a large percentage of the Nek5000 run time, as well as the communication costs required for nearest-neighbor data exchanges and vector reductions. The Nekbone kernel is embedded in a conjugate gradient iteration to solve the 3D Poisson equation. Preconditioning is either a simple diagonal scaling (simpler than Nek5000) or a spectral element multigrid on a block or linear geometry which is more similar to the multigrid structure found in Nek5000. In this paper, we run Nek5000 and Nekbone using 3D and 2D distributions. Since Nekbone designed to simulate 3D problems, we changed the processors breakdown to reflect the 2D processor map by setting the z-axis to 1.

The Hardware Accelerated Cosmology Code (HACC) [19] is an N-body framework that simulates the evolution of mass in the universe and its structure within the context of dark matter and dark energy. It uses particle mesh techniques, splitting the force calculation into a grid-based spectral particle mesh component for medium to long-range interactions and direct particle-to-particle solvers for short-range interactions. The long-range solvers implement an underlying 3D FFT that is domain-decomposed to 2D. SWFFT [20] is the 3D FFT that is implemented in HACC. Since this FFT accounts for a large portion of the HACC execution time, SWFFT serves as a proxy for HACC. SWFFT replicates the transform and is representative of the computation and communication involved.

TABLE III
PROXY/PARENT VERSION INFORMATION

Proxy	Version	Parent	Version
SW4lite	2.0	SW4	2.0
Nekbone	3.1	Nek5000	17
SWFFT	1.0	HACC	1.0
ExaMiniMD	1.0	LAMMPS	17 Aug 2017

LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [21] is a classical molecular dynamics code that models particles in solid, liquid, and gas states. A particle can range from a single atom to a large composition of material. LAMMPS integrates Newton's equations of motion to model particle interaction, using lists to track neighboring particles. It implements mostly short-range solvers, but does include some methods for long-range particle interactions. Like LAMMPS, ExaMiniMD [22], which is a proxy for LAMMPS, uses spatial domain decomposition. But compared to LAMMPS, ExaMiniMD's feature set is extremely limited, and only three types of interactions (Lennard-Jones/ EAM/SNAP) are available. The SNAP interaction is a much more complicated and computationally expensive potential that attempts to approach quantum chemistry accuracy when modeling metals and other materials. ExaMiniMD and LAMMPS both use neighbor lists for the force calculation. ExaMiniMD is intended to represent both the computation (including memory behavior) and communication that is implemented in LAMMPS.

The problems and input sizes we use for data collection are shown in Table IV and proxy/parent application versions that we use in this work are in Table III. We define problems and input sizes based either on conversations with developers or from The problems and input sizes we use for data collection are shown in Table IV and proxy/parent application versions that we use in this work are in Table III. We define problems and input sizes based either on conversations with developers or from development team performance reports. In all cases, we attempt to define problems that are pertinent in the exascale timeframe and we map application problems to their respective proxies to be as consistent as possible. While we realize that the characterization and subsequent clustering results and communication patterns may be sensitive to input, a study of this sensitivity is beyond the scope of this paper and will be done in future work.

C. Statistical Analyses

For our statistical analyses, we use the principal component analysis and clustering algorithms provided by the R Statistical Computing Tool [23]. We use hierarchical clustering, which is an unsupervised machine-learning technique, and we use the elbow method to select the optimal number of clusters to use as a parameter to the clustering algorithm. The hierarchical clustering algorithm is agglomerative and uses the Ward [7] cluster criterion.

TABLE IV
PROXY/PARENT PROBLEMS/INPUT SIZES

Proxy / Parent	Problem/Input size
SW4lite / SW4	LOH.1-h50.in, LOH.1-h50, time=9
Nekbone 2D / Nek5000 2D	2D decompisition; polynomial order=8; spectral multigrid=off max local elements per MPI rank=300 / eddy_uv, with Dim=2; polynomial order=8 max local elements per MPI rank=300
Nekbone 3D / Nek5000 3D	Dim=3; polynomial order=8; spectral multigrid=off max local elements per MPI rank=300 / 3dbox, with Dim=3; polynomial order=8 max local elements per MPI rank=300
SWFFT / HACC	n_repetitions=100; ngx=1024 / steps=100; ngx=1024
ExaMiniMD / LAMMPS	units=lj; nx, ny, nz=100; Timestep=0.005; Run=18000 (single- and multinode)

V. RESULTS

A. Pairwise Communication Data Results

Figures 3 to 7 show side-by-side heatmaps of the pairwise communication for our parent / proxy application pairs. Similarities and differences show up readily. Note that the color scale is auto-generated and spans whatever the value range is for the particular image; in some cases this is thousands, while for others (e.g., exaMiniMD) it is only one. These figures are for visualizing the parent and proxy communication patterns, similarities, and difference, but most of the discussion below is over the metrics.

Table V shows the results from the defined metrics for the pairwise communication data. We first discuss the first four metrics, which are the percentages of communication not found in the other related application (parent or proxy), and then discuss the correlation metrics.

The pairs of LAMMPS / ExaminiMD and SW4 / SW4lite both have exactly the same set of process pairs communicating for the parents and proxies, and so in relation to Figure 2.1, there are no areas A and B for them, meaning that nothing is filtered out in creating R_P and P_R. Thus, the four percentage metrics are all 100, even though the message counts are not equal between the parent and proxy.

The first two metrics for Nek5000 2D/Nekbone 2D show that although only about 57% of communicating pairs in Nek5000 do indeed communicate in Nekbone, this is the true core of the parent communication, since it captures 99.9% of the messages. On the other hand, as shown in the next two metrics, Nekbone has only 63% of its communicating pairs being represented in Nek5000, and this accounts for only about 37% of the communication of Nekbone. So while Nekbone does capture the major portion of Nek5000's communication, it has a large amount of communication itself that does not represent Nek5000, its parent. This can be seen in the heatmap figure as well, Figure 7.

The results for Nek5000 3D/Nekbone 3D are somewhat similar to the 2D case, but different in some of the correlation metrics. in Nek5000 3D, 51% of the communicating pairs are found in Nekbone 3D and this accounts for about 99.9% of the messages. This indicates that the extra communication in Nek5000 have low counts. From the third and fourth metrics, we see that about 68% of Nekbone 3D's communicating pairs are in Nek5000 and this accounts for 58% of the Nekbone communication. As in the 2D case, Nekbone does capture a major portion of Nek5000's communication, but it does a large amount of communication that is not in its parent. The heatmap figure shows this as well (Figure 7.

Looking at the first four metrics for the pair HACC / SWFFT, we see a high degree of difference between the two. Only 29% of communicating pairs in HACC do communicate in SWFFT, with this accounting for about half (52%) of all HACC messages. On the other side, about 71% of SWFFT communication is representative of HACC. SWFFT has almost exactly equal numbers of messages between all of its communicating process pairs, and so the percentage is the same for both the pair count and the message count metrics. Since SWFFT is meant to only capture one core portion of HACC (the FFT), it makes sense that it would miss much of HACC's communication; however, the fairly high percentage (29%) of SWFFT that is not in HACC may be indicating that it is not doing as good of job even on that core that it should be representing.

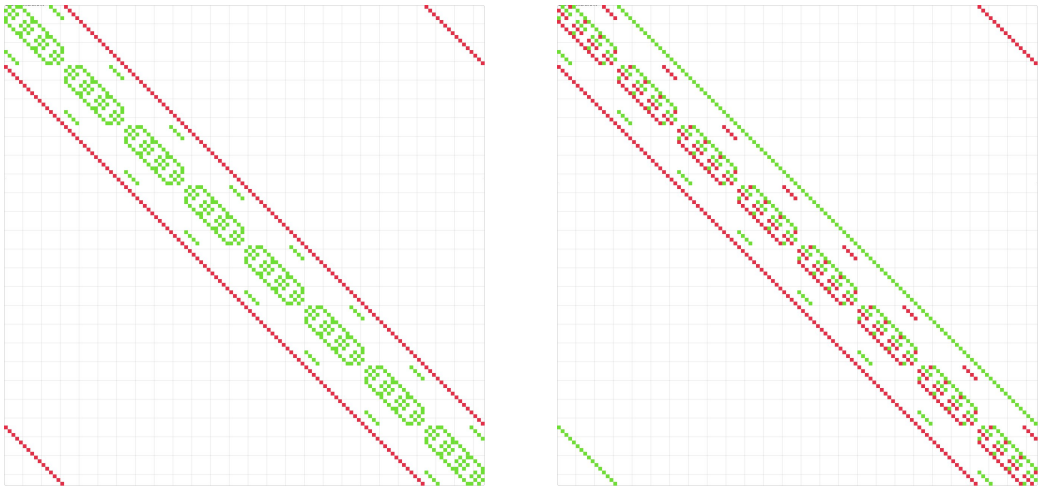
We now discuss the statistical and correlation test metrics for the parent / proxy pairs. Although not shown in the table, for all three set comparisons (R_A, P_A), (R, P_R), and (R_P, P), we performed the Wilcoxon Signed-Rank test and a paired T-test as statistical tests of similarity. For all of the proxy / parent pairs, both of the tests give a p-value of essentially zero (e.g., $\sim 1e^{-20}$), meaning that they always detect a very significant difference between the populations. This is because the message counts *are* significantly different. Even for SW4 and SW4lite, which are almost identical, we observed that, e.g., SW4 sent exactly and always 3,347 messages between each pair of communicating processes, while SW4lite sent exactly and always 3,345 messages. With zero variance, these two populations are very significantly different, statistically, even though we would want to say that they are essentially the same.

This leads us to conclude that this type of statistical test is inappropriate for these comparisons, and so we now focus on the Pearson and Spearman correlation results over our data sets. Pearson is a linear correlation of the values in the data, while Spearman is a correlation over the ranks and thus can capture a non-linear correlation.

For LAMMPS / ExaMiniMD and SW4 / SW4lite, which both had percentage metrics of 100, the two pairs have exactly opposite correlation metrics; LAMMPS and ExaMiniMD have zero correlation, while SW4 and SW4lite have perfect correlation. As explained above, SW4 and SW4lite have perfectly constant numbers of messages over exactly the same communicating pairs, and so they obviously correlate perfectly. LAMMPS has a very bimodal distribution of message counts over its communicating pairs, while ExaMiniMD

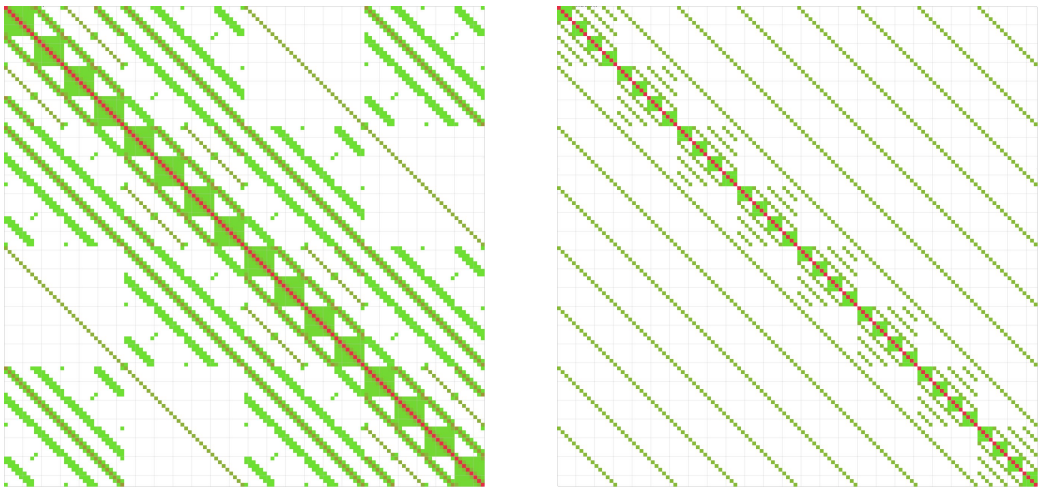
TABLE V
METRICS FOR ALL PARENT / PROXY PAIRS. PCOR IS PEARSON CORRELATION; SCOR IS SPEARMAN CORRELATION.

	Parent in Proxy		Proxy in Parent		R_A, P_A		R, P_R		R_P, P	
Parent/Proxy	#msg	#pair	#msg	#pair	PCor	SCor	PCor	SCor	PCor	SCor
LAMMPS/ExaMMD	100	100	100	100	0	0	0	0	0	0
Nek5K 2D/Nekbone 2D	99.9	57.4	37.5	62.8	0	0.06	-0.47	-0.05	0.55	0.93
Nek5K 3D/Nekbone 3D	99.9	51.4	58.0	68.4	-0.1	-0.05	-0.65	-0.23	0.04	0.49
SW4/SW4lite	100	100	100	100	1	1	1	1	1	1
HACC/SWFFT	51.7	29.4	71.4	71.4	0.58	0.31	0.61	0.28	0.87	0.81



LAMMPS exaMiniMD

Fig. 3. LAMMPS and exaMiniMD, Point to point message count.



HACC SWFFT

Fig. 4. SWFFT and HACC, Point to point message count.

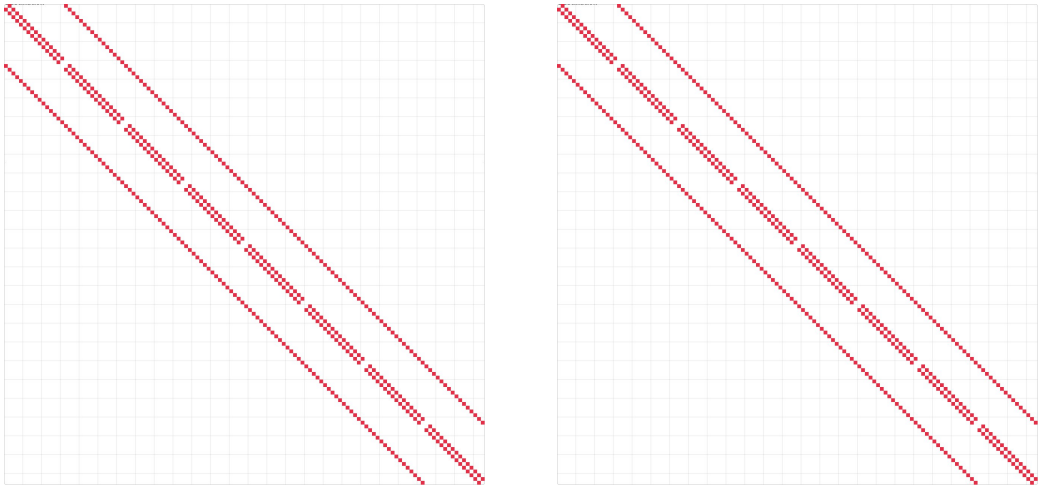


Fig. 5. SW4 and SW4lite, Point to point message count.

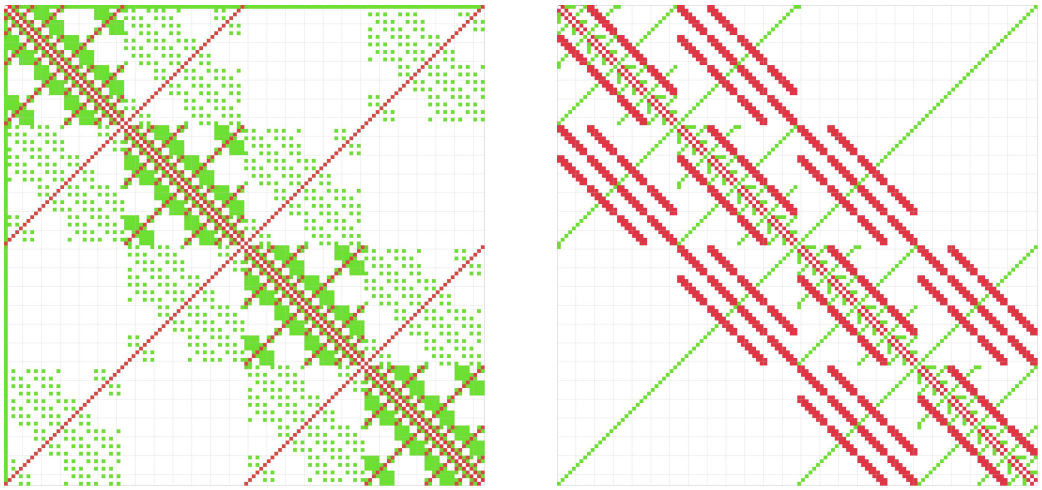


Fig. 6. Nek5000 and Nekbone, Point to point message count, 3D configuration.

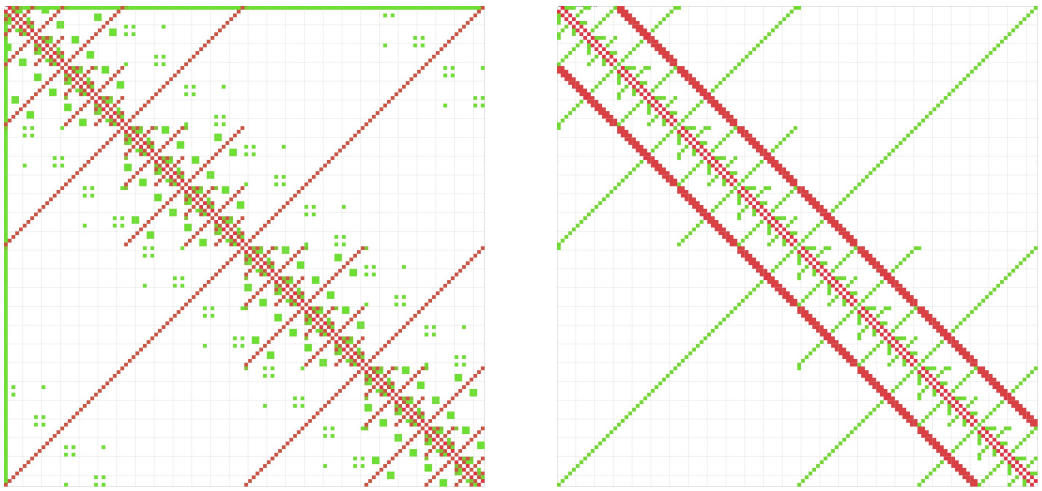


Fig. 7. Nek5000 and Nekbone, Point to point message count, 2D configuration.

always has message counts of either 20,702 or 20,703, which is essentially constant. This causes the correlation (between a bimodal function and a constant function) to be essentially 0. This shows the value in both the percentage metrics and the correlation metrics together; using only one can give the wrong understanding.

For Nek5000 / Nekbone we compare both 2D and 3D executions, though recall that Nekbone 2D was approximated by minimizing the z-axis. The percentage metrics are similar in both cases, with almost 100% of the messages in the parent covered by the proxy even though the proxy does not cover many of the low-frequency communicating pairs. In both cases the proxy has significant communication that is not in the parent, and this is higher in the 2D case (63.5% of messages), as might be expected given that Nekbone does not inherently implement 2D. The correlation metrics across both cases show no correlation for the R_A, P_A sets, but exhibit an interesting "flip" in the two filtered scenarios, with R, P_R showing somewhat negative correlation and R_P, P showing some positive correlations, including a very high Spearman correlation for the 2D case. In looking at Figures 6 and 7, this flip is probably related to the heavy red bands of communication in Nekbone along the diagonal, which somewhat overlap some green communication patterns in Nek5000. Filtering these two only by the parent (Nek5000) results in some opposing-value (negative) correlation, while filtering these by the proxy (Nekbone) results in some positive correlation. We have not yet further investigated the exact cause and so this is a supposition based on the visual evidence.

For HACC / SWFFT, again the highest correlations are for R_P, P, which make sense, since SWFFT is only meant to capture a portion of HACC. With the Pearson (linear) correlation being highest (0.87), this indicates that SWFFT does a pretty good job of capturing this portion, such that a simpler linear correlation holds up. The strong relation between this portion is also probably why the correlations for R_A, P_A and R, P_R are still reasonably positive.

B. Communication Vector Data Clustering

Figure 8 shows the clustering result. Note that this data is collected from a single run of each application since mpiP implements software-based counting that remains constant across runs. In a dendrogram, the y-axis is the connection height, which is a measure of similarity–the lower the connection height, the higher the similarity of the runs below it. First note that at the lowest level (highest similarity), LAMMPS and ExaMiniMD cluster together, as do HACC and SWFFT and SW4 and SW4lite (although this is difficult to see given the scale). Nekbone does not cluster with Nek5000, rather Nekbone 2D and 3D cluster together. But notice that Nekbone 2D and 3D are not as similar as LAMMPS/ExaMiniMD, HACC/SWFFT, SW4/SW4lite as its cluster is at about the same height as the cluster that contains all of these proxy/parent pairs. Nek5000 2D is very much an outlier and does not cluster with any applications except at a very high level. Nek5000 3D is similar in that it clusters with all of the other applications (except

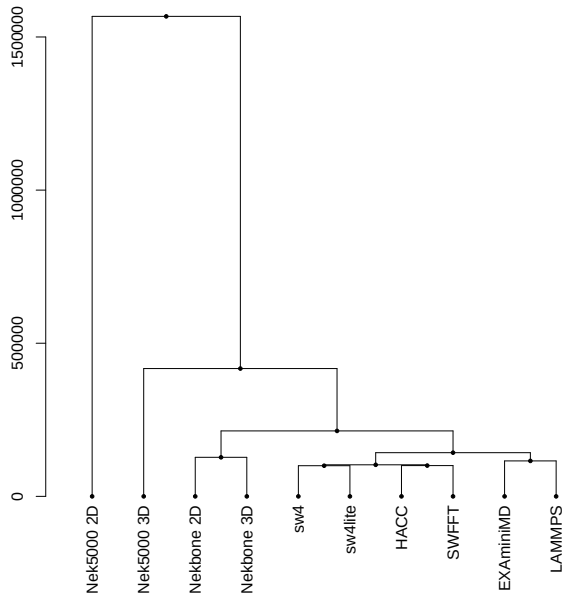


Fig. 8. Communication Similarity, Clustering Message Size and Frequency

Nek5000 2D at a relatively high cluster height. Both Nekbone and particularly Nek5000 appear to have communication (communicating pairs and message frequency) that is very different than the other applications.

The idea behind doing clustering is to determine if a simple method such as this could accurately detect communication similarity. The vector used for clustering does not contain any information about communication pattern, only message size and frequency for each size histogram bin. Looking at the data in Table V, the clusters do seem to somewhat agree for all proxy/parent pairs except Nek5000 and Nekbone, which requires more exploration to adequately explain. We will continue to further examine communication vectors in conjunction with our statistical measures to really determine if simple clustering of key metrics can accurately quantify similarity.

VI. Related Work

Although there is a lot of older work done in characterization of application communication [24]–[30], relatively little work has been done in characterizing the similarity of communication patterns [31]. We outline this work and its distinction from ours below.

In [31], Ma et al. present a method for characterizing similarity in communication patterns of parallel applications that use MPI, then they apply this method to four of the NAS parallel benchmarks. Their method uses a linear correlation coefficient on ranked metric values in conjunction with a graph isomorphism metric. They construct a graph based on communicating pairs (source, destination), then use graph isomorphic degree to determine the similarity between graphs.

The metrics they use for correlation are temporal, which reflects message rates, volume for representing message size, and spatial that captures communication locality in terms of communicating pairs. Their results are mixed with three out of six benchmark comparisons showing strong similarity and three out of six showing weak, but some similarity. In contrast, our proposed method is much simpler, using data directly gathered from mpiP. We do non-linear correlation, which we believe is key, and use real applications in addition to proxies (similar to benchmarks).

The work presented in [32], [33] focuses on matching application communication patterns to a library of commonly observed patterns. Their methods are based on pattern matching; they do not examine pattern similarity. Therefore, this work is peripherally related, but not applicable to our task of interest, which is determining similarity in communication patterns.

VII. Conclusion and Future Work

We presented an exploration into quantifying how the communication of a proxy application relates to its parent application. This was done in two ways.

One, quantifying over pairwise communication data, with metrics capturing how much of one application matches the other, and with correlation metrics over the message counts of communicating pairs. Variations of the correlations were performed over parent-centric and proxy-centric views, with the intent of capturing how much of the parent is represented in the proxy, and if the proxy has communication behavior outside that of the parent. A wide variety of relationships was observed.

Two, quantifying over message characteristics, by creating a application characteristic vector with values representing a histogram of message size and with values capturing overall message rates. These vectors were the input into a hierarchical clustering algorithm to produce clustering-based relatedness measures of the parent applications and their proxies. Most parents and proxies clustered closely together in this view, with Nek5000 and Nekbone being the one exception.

This work needs to be expanded in several ways. One is to incorporate the work on the communication patterns known as the seven dwarves [9], and relate both parents and proxies to the dwarf patterns. Two is to provide a way to make easier sense of the metrics and combine them into higher level concepts. Many aspects of communication were not captured in our metrics and should be explored as to whether they are important or not; one prominent one is the time-varying behavior—parents and proxies could look similar in terms of the metrics we present and yet have radically different time-varying behavior over the execution period. An important aspect of this is whether or not the communication reaches a bottleneck that affects performance, and if this is captured in the proxy.

VIII. Acknowledgement

This research was supported by the Exascale Computing Project (ECP), Project Number 17-SC-20-SC, a collaborative effort of two DOE organizations, the Office of Science and the National Nuclear Security Administration, responsible for the planning and preparation of a capable exascale ecosystem including software, applications, hardware, advanced system engineering, and early testbed platforms, to support the nation's exascale computing imperative.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE- NA0003525.

References

[1] O. Aaziz, J. Cook, J. Cook, T. Juedeman, D. Richards, and C. Vaughan, "A Methodology for Characterizing the Correspondence Between Real and Proxy Applications," in *To appear in 2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018.

[2] P. T. Lin, M. A. Heroux, R. F. Barrett, and A. B. Williams, "Assessing a mini-application as a performance proxy for a finite element method engineering application," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 17, pp. 5374–5389, 2015, cpe.3587. [Online]. Available: http://dx.doi.org/10.1002/cpe.3587

[3] R. Barrett, P. Crozier, D. Doerfler, M. Heroux, P. Lin, H. Thornquist, T. Trucano, and C. Vaughan, "Assessing the role of mini-applications in predicting key performance characteristics of scientific and engineering applications," *Journal of Parallel and Distributed Computing*, vol. 75, no. Supplement C, pp. 107 – 122, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514001695

[4] Y. Kim, J. M. Dennis, and C. Kerr, "Assessing representativeness of kernels using descriptive statistics," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 818–825.

[5] M. Tsuji, W. T. C. Kramer, and M. Sato, "A performance projection of mini-applications onto benchmarks toward the performance projection of real-applications," in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sept 2017, pp. 826–833.

[6] J. S. Vetter and M. O. McCracken, "Statistical scalability analysis of communication operations in distributed applications," in *Proceedings of the Eighth ACM SIGPLAN Symposium on Principles and Practices of Parallel Programming*, ser. PPoPP '01. New York, NY, USA: ACM, 2001, pp. 123–132. [Online]. Available: http://doi.acm.org/10.1145/379539.379590

[7] D. Müllner, "Modern hierarchical, agglomerative clustering algorithms," *CoRR*, vol. abs/1109.2378, 2011.

[8] A. E. Zambelli, "A data-driven approach to estimating the number of clusters in hierarchical clustering," *F1000Research*, vol. 5, pp. ISCB Comm J–2809, 2016. [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC5373427/

[9] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," Electrical Engineering and Computer Sciences University of California at Berkeley, Tech. Rep. UCB/EECS-2006-183, 2006.

[10] https://www.exascaleproject.org, "Exascale computing project." [Online]. Available: https://www.exascaleproject.org

[11] https://www.exascaleproject.org/focus_area/application-development, "Ecp application development." [Online]. Available: https://www.exascaleproject.org/focus_area/application-development

[12] https://proxyapps.exascaleproject.org, "Exascale proxy applications." [Online]. Available: https://proxyapps.exascaleproject.org

[13] B. Sjrogreen and N. A. Petersson, "A fourth order accurate finite difference scheme for the elastic wave equation in second order formulation," *Journal Of Scientific Computing*, vol. 52, no. 1, 2011.

[14] N. Petersson and B. Sjrogreen, "Sw4 v2.0. computational infrastructure of geodynamics," 2017.

[15] https://github.com/geodynamics/sw4lite, "Sw4lite." [Online]. Available: https://github.com/geodynamics/sw4lite

[16] https://nek5000.mcs.anl.gov, "Nek5000." [Online]. Available: https://nek5000.mcs.anl.gov

[17] H. M. Tufo and P. F. Fischer, "Terascale spectral element algorithms and implementations," in *Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*, ser. SC '99. New York, NY, USA: ACM, 1999. [Online]. Available: http://doi.acm.org/10.1145/331532.331599

[18] https://asc.llnl.gov/CORAL benchmarks/Summaries/Nekbone_Summary_v2.3.4.1.pdf, "Nekbone." [Online]. Available: https://asc.llnl.gov/CORAL-benchmarks/Summaries/Nekbone_Summary_v2.3.4.1.pdf

[19] S. Habib, V. Morozov, N. Frontiere, H. Finkel, A. Pope, K. Heitmann, K. Kumaran, V. Vishwanath, T. Peterka, J. Insley, D. Daniel, P. Fasel, and Z. Lukić, "Hacc: Extreme scaling and performance across diverse architectures," *Commun. ACM*, vol. 60, no. 1, pp. 97–104, Dec. 2016. [Online]. Available: http://doi.acm.org/10.1145/3015569

[20] https://xgitlab.cels.anl.gov/hacc/SWFFT, "Swfft (hacc)." [Online]. Available: https://xgitlab.cels.anl.gov/hacc/SWFFT

[21] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *J. Comput. Phys.*, vol. 117, no. 1, pp. 1–19, Mar. 1995. [Online]. Available: http://dx.doi.org/10.1006/jcph.1995.1039

[22] A. P. Thompson and C. R. Trott, "A brief description of the kokkos implementation of the snap potential in examinimd." 11 2017.

[23] https://www.r-project.org, "The r project for statistical computing." [Online]. Available: https://www.r-project.org

[24] S. Chodnekar, V. Srinivasan, A. S. Vaidya, A. Sivasubramaniam, and C. R. Das, "Towards a communication characterization methodology for parallel applications," in *HPCA*, 1997.

[25] R. Zamani and A. Afsahi, "Communication characteristics of message-passing scientific and engineering applciations," in *Proceedings of the International Conference on Parallel and Distributed Computing Systems (PDCS)*, ser. PDCS '05, Phoenix, AZ, USA, 2005, pp. 644–649.

[26] J. Shalf, S. Kamil, L. Oliker, and D. Skinner, "Analyzing ultra-scale application communication requirements for a reconfigurable hybrid interconnect," in *SC '05: Proceedings of the 2005 ACM/IEEE Conference on Supercomputing*, Nov 2005, pp. 17–17.

[27] J. Kim and D. J. Lilja, "Characterization of communication patterns in message-passing parallel scientific application programs," in *Proceedings of the Second International Workshop on Network-Based Parallel Computing: Communication, Architecture, and Applications*, ser. CANPC '98. London, UK, UK: Springer-Verlag, 1998, pp. 202–216. [Online]. Available: http://dl.acm.org/citation.cfm?id=646092.680542

[28] J. S. Vetter and F. Mueller, "Communication characteristics of large-scale scientific applications for contemporary cluster architectures," in *Proceedings 16th International Parallel and Distributed Processing Symposium*, April 2002, pp. 10 pp–.

[29] S. Karlsson and M. Brorsson, "A comparative characterization of communication patterns in applications using mpi and shared memory on an ibm sp2," in *Proceedings of the Second International Workshop on Network-Based Parallel Computing: Communication, Architecture, and Applications*, ser. CANPC '98. London, UK, UK: Springer-Verlag, 1998, pp. 189–201. [Online]. Available: http://dl.acm.org/citation.cfm?id=646092.680546

[30] P. G. Raponi, F. Petrini, R. Walkup, and F. Checconi, "Characterization of the communication patterns of scientific applications on blue gene/p," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, May 2011, pp. 1017–1024.

[31] C. Ma, Y. He, and N. Xiong, "Mpacp: An approach for automatic matching of parallel application communication patterns," in *2008 IEEE Asia-Pacific Services Computing Conference*, Dec 2008, pp. 1517–1522.

[32] P. C. Roth, J. S. Meredith, and J. S. Vetter, "Automated characterization of parallel application communication patterns," in *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '15. New York, NY, USA: ACM, 2015, pp. 73–84. [Online]. Available: http://doi.acm.org/10.1145/2749246.2749278

[33] D. J. Kerbyson and K. J. Barker, "Automatic identification of application communication patterns via templates," in *ISCA PDCS*, 2005.