Fine-Grained Analysis of Communication Similarity between Real and Proxy Applications

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

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

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

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

Jeffery Kuehn
Los Alamos National Laboratory
Los Alamos, NM 87545
Email: jakuehn@lanl.gov

David Richards
Lawrence Livermore National Laboratory
Livermore, CA 94550
Email: richards12@llnl.gov

***Abstract*—In this work we investigate the dynamic communication behavior of parent and proxy applications, and investigate whether or not the dynamic communication behavior of the proxy matches that of its respective parent application. The idea of proxy applications is that they should match their parent well, and should exercise the hardware and perform similarly, so that from them lessons can be learned about how the HPC system and the application can best be utilized. We show here that some proxy/parent pairs do not need the extra detail of dynamic behavior analysis, while others can benefit from it, and through this we also identified a parent/proxy mismatch and improved the proxy application.**

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

I. Introduction

In previous work [1], we explored how the communication behavior of some proxy applications related to their respective real, or parent, application. Proxy applications, sometimes called mini-apps or proxies, are smaller, easier-to-use programs that are used in myriad ways: to evaluate systems, find hardware bottlenecks, and perform algorithmic or system design exploration. In that work, we used two forms of aggregate data: aggregate MPI function data from mpiP [2], and aggregate pairwise communication data from CrayPat. In both cases, the quantitative data represented behaviors that were totaled over the lifetime of the execution. This aggregate data was collected both from the real and proxy applications, and then compared using several novel metrics that we defined and evaluated.

Using aggregate metrics ignores the possibility that the proxy application might only be representing part of the execution of the parent application. Sometimes this is not true—for example, SW4 and SW4lite actually share a common codebase, and our metrics ended up showing a very high similarity between them. On the other hand, HACC, a cosmological code, has a related proxy called SWFFT, which by its name indicates that it is focused only on the FFT portion of the computation; our metrics thus showed less correspondence between these. Proxies and parents could also have similar aggregate communication behavior, but might peak or stress the underlying interconnect in different ways and at different times, thus causing performance differences that are hard to understand.

This leads us to investigate the correspondence of real and proxy applications in their communication behavior *over time*. In this paper we explore an evaluation of various quantitative metrics over the time-varying communication behavior of applications. We use three pairs of real and proxy applications, one of which is different from the ones we used in [1]. The two previously used pairs are LAMMPS/ExaminiMD, and HACC/SWFFT. The new pair is the real application CTH [3], a very large solid mechanics application, and the proxy application miniAMR [4], a proxy application meant to mimic a typical computation using adaptive mesh refinement, and developed with CTH particularly in mind.

The research question that is thus explored here is: Can we find meaningful quantitative ways to analyze the time-varying communication behavior of parent and proxy applications in a way that will give us insight into their dynamic correspondence in terms of the communication they do during execution?

The contributions of this paper are: an exploration and presentation of the dynamic communication behaviors and relationships between three parent/proxy application pairs; an evaluation of several different metrics over the time-varying communication behavior that are potentially useful for comparing real applications to their proxy counterparts; and a resulting improvement to miniAMR that makes it more closely match its parent CTH. During the course of the research described here, it became clear that CTH and miniAMR had far more interesting dynamism in their communication behavior than did the other parent/proxy pairs, and so while we do present results for the other pairs, most of the content of this paper describes CTH and miniAMR, and the results for them.

978-1-7281-5977-5/19/$31.00 ©2019 IEEE
DOI 10.1109/PMBS49563.2019.00016

II. Background

As part of our previous work we formulated and then evaluated metrics that captured aspects of correspondence between the communication behavior of two applications. We specifically were concerned with comparing the communication behavior of a proxy application with the parent application that it intends to represent. For rigorous and detailed descriptions of these metrics, please refer to [1].

Important to this paper are the correlation metrics we devised. For quantitative data, we captured the total number of messages sent from a sending process (rank) to a receiving process (rank) during the execution of the application. This gives us a 2-dimensional matrix of message counts, with the sending and receiving process id's as the row and column indices. When comparing parent and proxy applications, hopefully most nonzero entries are nonzero in both the parent and proxy matrices, but there will be some entries that are nonzero in the parent matrix and zero in the proxy matrix, and some that are zero in the parent and nonzero in the proxy. These, along with different message counts, represent mismatches in communication behavior.

We formed three different data vectors based on three different filters over these matrices, which embody three different views of the data. The vectors end up having the exact same set of (sender,receiver) entries for both the parent and proxy applications, but with their own respective values. The views are:

- **Full view**: Data vector includes all process pairs that have non-zero message counts in *either* the parent or the proxy; where a pair occurs in one but not the other, zero is entered for the other's corresponding message count. This is thus the data for both applications over all pairs that communicate in either the parent or the proxy.
- **Parent view**: Data vector contains data only for those pairs who have nonzero message counts in the parent application; proxy pairs outside of this are discarded, and zero is entered in proxy data for pairs in the parent but not in the proxy. This is thus data for both applications over only the pairs that communicate in the parent.
- **Proxy view**: Data vector contains data only for those pairs who have nonzero message counts in the proxy application; parent pairs outside of this are discarded, and zero is entered in parent data for pairs in the proxy but not in the parent. This is thus data for both applications over only the pairs that communicate in the proxy.

Thus, the first covers the full extent of both behaviors and how they might correspond, the second focuses on how much of the observed parent behavior is covered by the proxy, and the last focuses on how much of the observed proxy behavior actually matches parent behavior.

In the previous work we defined and evaluated descriptive statistics (percent overlap) and correlations (Pearson and Spearman) over these data sets. We saw a perfect match between SW4 and SW4lite, very high similarity between LAMMPS and ExaminiMD, high similarity between HACC and SWFFT, and some serious discrepancies between Nek5000 and Nekbone. We also saw data instances that caused anomalous correlation values.

III. Methodology

In this paper we use some of the basic metrics that we defined and used in the previous work, but then also delve into the dynamic time-varying communication behavior in the applications. For this we capture the same 2-dimensional message count data (for each source and destination pair) at selected intervals during execution lifetime. What intervals are selected is detailed in the results discussion in Section V.

We also investigate the use of another similarity metric, cosine similarity. Cosine similarity calculates the cosine of the angle between the vectors defined by the data, viewing the data as defining a point in the N-dimensional space. This ignores the magnitude of the vector. Thus, unlike the correlation metrics, cosine similarity is not affected by absolute magnitudes of data values, only their relative values. While correlation metrics can also establish a relationship in the presence of different magnitudes (e.g., Pearson will recognize a linear relationship), cosine similarity is more robust in this way. This would seem to be applicable to communication behavior, where we might want to compare executions of different size runs and see if the communication pattern is the same, regardless of how many messages were sent.

IV. Experiment Setup

A. Applications

As noted earlier, the parent and proxy application pairs we used in this work are LAMMPS / ExaminiMD, HACC / SWFFT, and CTH / miniAMR. More application pairs should be subject to analysis similar to what we did in this work, but our available time window precluded adding more pairs, and some of our results below point out part of the difficulty in identifying suitable application pairs.

1) HACC and SWFFT: The Hardware Accelerated Cosmology Code (HACC) [5] 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 [6] 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 meant to be representative of the computation and communication involved.

2) LAMMPS and ExaminiMD: LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [7] 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 [8], 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.

3) CTH and miniAMR: CTH is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories [3]. CTH has models for multi-phase, elastic viscoplastic, porous and explosive materials, using second-order accurate numerical methods to reduce dispersion and dissipation and produce accurate, efficient results.

MiniAMR was developed to study CTH when it is run using adaptive mesh refinement, or AMR [9]. Both CTH and miniAMR use an octree-based AMR scheme, where each processor has a number of blocks, each of which has a few hundreds of cells. When a region needs to be refined, a block is replaced with 8 blocks, each half the size of the original block in each dimension, but with the same number of cells. As the calculation progresses, the number and placement of these blocks in the calculation can change.

In terms of communication, each block has to communicate with its neighboring blocks in the mesh, so each process ends up performing communication within the process as well as to some number of neighboring processes, which can change as the simulation progresses.

In this study, we use two different input simulations for CTH and miniAMR. The first is a simulation with 4 spheres moving through the mesh in such a way that they do not interact and therefore have no distortion. The mesh is refined on the surfaces of the spheres and the refinement of the mesh from each sphere will interact with the refinement from the others. The second problem is a ball hitting a plate. We refine the mesh around the ball as it interacts with the plate as well as on the shock wave moving through the plate. The shock wave is modeled in miniAMR as an expanding hemisphere.

B. MiniAMRZ: a Modified miniAMR

Because this study was focused on time varying communication behaviors, we quickly saw differences between CTH and miniAMR that was due to the ways that they do mesh refinement. An investigation of these differences showed that there are three factors which contribute to the differences, each relating to the implementation of the Recursive Coordinate Bisection (RCB) [10] algorithm in the load balancing phase. For each step of the RCB algorithm, a direction and a number of divisions are chosen. The blocks are sorted in that direction and divided into nearly equal sets based on their position in that direction, and the ranks are also divided. Divisions are based on a prime factorization of the number of MPI ranks. Each of the sets of blocks is assigned to a set of ranks and the process is repeated until each rank has a set of blocks assigned to it.

The first difference is that CTH uses the Zoltan load balancing library [11] which has a generalized version of the algorithm, while the algorithm in miniAMR is more taliored to a rectilinear mesh where block centers are constrained to be discreet values. The effect is that with CTH, when there are several blocks that lie along the cut between groups of blocks that will be assigned to one processor set or another, the blocks are effectively assigned to one set or the other. In miniAMR, those blocks are assigned based on their position in the cutting plane so that blocks that are nearby to each other are more likely to be assigned to the same set.

The second difference is that CTH only allows a certain percentage of blocks to be moved during any refinement step in order to limit the size of the messages that are being sent during block reassignment. This results in random blocks not being moved to the proper processor and has the effect of a processor communicating with more other processors.

The third difference is that the Zoltan implementation of RCB in CTH allows the cut direction for each group of blocks to be determined when the cut is being made, while miniAMR determines the order of cuts once at the beginning. When the cut direction is changed for a group of blocks, this can cause more blocks to be assigned for moving, but due to the limit in CTH, not all of those blocks will be moved.

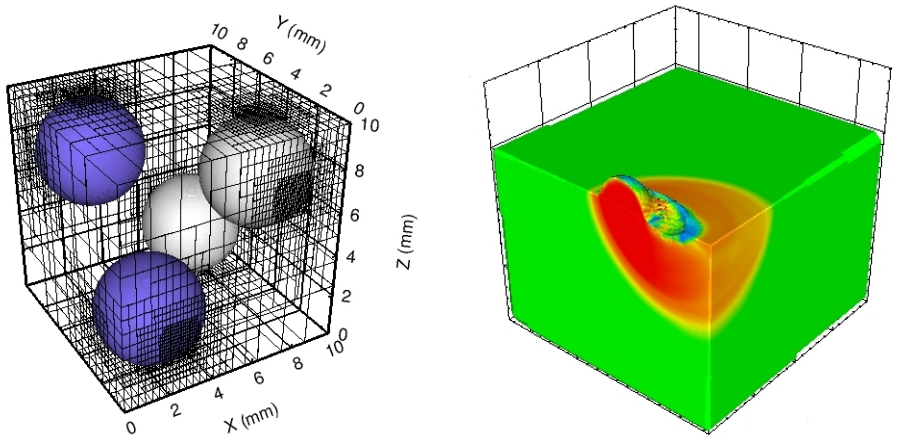
In order to try to make the communication patterns of miniAMR more closely match that of CTH, we modified miniAMR, creating a version we call miniAMRZ, in reference to making it better mimic the Zoltan-based behavior of CTH. Selectable options allow this modified codebase to run as miniAMR or miniAMRZ. MiniAMRZ will do the following: all of the blocks that fall on a cut will be distributed to one side of the cut or the other; a limit can be specified for the maximum number of blocks moved after load balancing; and miniAMRZ allows the RCB algorithm to change the directions of the cuts.

In this study we use both the standard miniAMR, and our miniAMRZ. Uncovering these differences was directly due to looking at the data collected during the course of this study, and so miniAMRZ is a resulting contribution of this work.

C. Instrumentation

In our previous work we collected total aggregate message counts for the lifetime of the program, using the CrayPat tool (v7.0.1). Our interest in this paper is in refining this by collecting data for time-varying communication behavior of an application. Thus, we desired to collect data during the execution rather than just total data at the end.

The CrayPat tool does have the ability for an application to turn it on and off during execution, but not to actually capture consistently sampled values during the execution. For



A) miniAMR Simulation Result for 4 Spheres B) CTH Simulation state for Ball & Plate

Fig. 1. Simulation Visuals for MiniAMR and CTH.

the applications HACC, SWFFT, Lammps, and ExaminiMD, we did instrument the code to turn CrayPat on for one communication step in each execution, and then we ran executions that would capture one unique step each time. This was a very costly process and limited the amount of data we could collect for these applications; however, as will be seen in the results, Section V, this did not matter.

In order to look at the communication differences between CTH and miniAMR, which are much more dynamic, we needed a more efficient method to collect the fine-grained data we wanted from CTH and miniAMR. Ultimately we decided to use by-hand source instrumentation, instrumenting the codes to output the communication matrices at times throughout the execution of the code. For CTH, we counted all of the communications on each rank for the first communication after a refinement step, since once there is a new communication pattern it remains fixed and in effect until the next refinement step. We output that information into a file for each refinement step and then post process the communication patterns for the entire run. For miniAMR, we were able to look at the data structures that are present after each refinement step and determine and output the communication pattern that will be used until the next refinement step. Thus the data from miniAMR corresponded to those from CTH.

D. Input Problem Details

For LAMMPS and ExaminiMD, LAMMPS is set up with a Lennard-Jones atomic interaction scenario, which is what ExaminiMD is meant to correspond to, with a size of 100 in each dimension, a timestep size of 0.005, and a run of 18,000 steps. For HACC, we used a typical example simulation that corresponds with SWFFT, and both used 100 simulation steps and size 1024.

For the 4 spheres problem in CTH and miniAMR, we ran 7819 timesteps for 5e-6 seconds of simulation time with 2607 mesh refinement steps. MiniAMR is complex enough that, given the simple nature of this problem, both codes end up with the spheres in the same position. We estimated, by observation, that number of blocks in the problem differs between CTH and miniAMR by 2.5% at most.

For the ball and plate problem, CTH and miniAMR run for 3642 timesteps and have 1214 mesh refinement steps. This problem is more complex in its behavior, but fairly reproducible in miniAMR. The shock wave is in the plate and does not interact with anything, so there is not too much distortion. Thus the shock wave behavior in CTH is fairly reproducible in miniAMR, but the distortion of the ball and crater will be somewhat different.

Figure 1A shows the resulting simulation state from miniAMR on the 4 spheres problem, while Figure 1B shows a late simulation state from CTH for the ball and plate problem. MiniAMR handles the non-interacting spheres well but is much less accurate on the ball and plate distortion seen in the CTH result. The adaptive mesh can be seen in the miniAMR figure.

E. Platform

All experiments were performed on a partition of Mutrino, a small Cray XC40-based cluster at Sandia National Laboratories, with partition nodes having two 16-core Intel Haswell processors and the cluster having a Cray Aries interconnect. For all six applications, all executions were done with 128 ranks over 4 nodes, and with no OpenMP being used (1 thread per process).

V. Results and Analysis

A. LAMMPS/ExaminiMD and HACC/SWFFT

Table I shows our results for the parent and proxy pairs of LAMMPS and ExaminiMD, and HACC and SWFFT. The rows with full application names are full-execution aggregate results from our previous work (thus not from the same runs as the other data), except that the new cosine similarity metric is computed (using this original data). Rows with abbreviated names are data from this study, each representing an interval of time within the application, roughly about one third of the

TABLE I
RESULTS FOR LAMMPS / EXAMINIMD AND HACC / SWFFT. ROWS WITH FULL NAMES ARE OVERALL RESULTS; THOSE WITH '@' ARE INTERVAL RESULTS.

Parent/Proxy	Parent View #msg	Parent View #pair	Proxy View #msg	Proxy View #pair	Pearson Corr Full View	Pearson Corr Parent View	Pearson Corr Proxy View	Cosine Similarity
LAMMPS/ExaMMD	100	100	100	100	0	0	0	**0.94**
L/M @ 100	100	100	100	100	1	1	1	0.94
L/M @ 500	100	100	100	100	1	1	1	0.94
L/M @ 1200	100	100	100	100	1	1	1	0.94
HACC/SWFFT	68.7	41.1	100	100	0.97	0.92	0.97	**0.93**
H/S @ 10	68.4	41.2	100	100	0.97	0.92	0.97	0.93
H/S @ 25	68.4	41.2	100	100	0.97	0.92	0.97	0.93
H/S @ 40	68.4	41.2	100	100	0.97	0.92	0.97	0.93

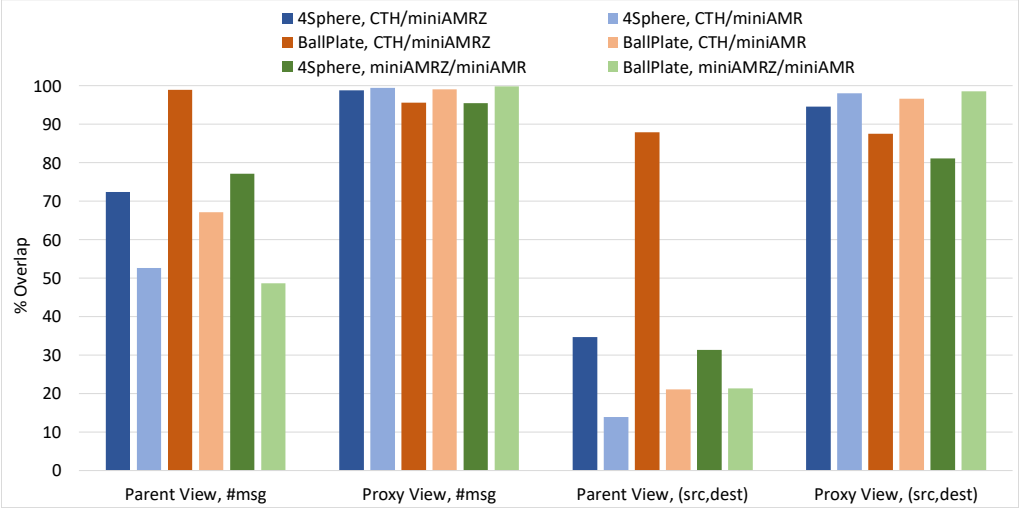


Fig. 2. Communication overlap relations.

execution. Looking at the table, the interval results are virtually identical to the aggregate results, and are equal to each other.

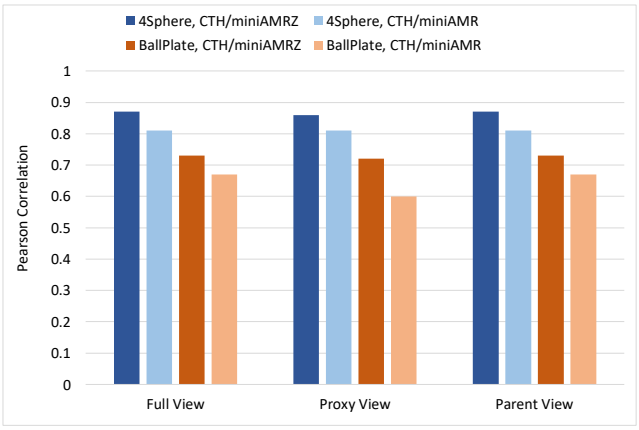
We present this data to show that applications that do not do dynamic communication adapation do not need evaluation over their execution lifetime; aggregate data and analysis is sufficient. All four of these applications simply do the same communication pattern over and over throughout their lifetimes, and so each interval looks exactly like each other, and like the whole execution. Thus, the rest of our analysis focuses on our parent/proxy pair that *does* have dynamic communication adaptation, namely CTH and miniAMR. This static communication pattern is also a reason why we did not include other parent / proxy pairs that we have used in previous studies, and also points to the need to have more identified and labeled proxy applications that are known to have a dynamic communication model.

Note that the one large discrepancy in the table is the correlation values of 0 for the old LAMMPS/ExaminiMD whole-execution data, and 1 for the intervals. The original 0 correlation is due to the proxy being off by one in the message counts, in a way that caused perfect un-correlation; in the intervals the counts are exactly equal, which causes perfect correlation. We believe the off-by-one message happens right at the end, after our interval data collections. Note that the cosine similarity metric over both the old and new data is the same; this result provides some evidence that cosine similarity is a more robust similarity mettric.

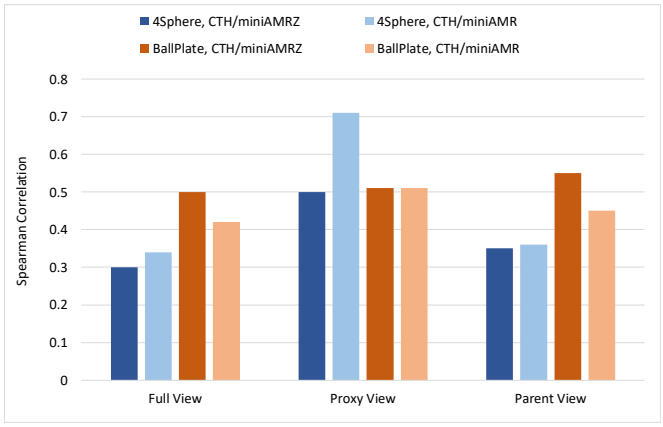
B. CTH and miniAMR

Figure 2 shows the basic overlap relations in the communication of CTH and the two miniAMR versions, and between the two versions of miniAMR themselves (green bars). The left two groups show the percentage of messages that occur in the parent that also occur in the proxy (leftmost) and that occur in the proxy and also occur in parent (left middle). The two right-side groups are similar, except over communicating pairs (boolean, whether they communicated or not) rather than message counts. The percentages in the proxy views are all very high, indicating that almost all message (and pair) communication in the proxy matches communication in the parent. The pairwise bars (far right) are slightly lower than the message-count bars (left middle), indicating that there are a few low-count communicating pairs in the proxies that are not in the parent.

The parent view, however, is very different, indicating that there is parent communication behavior that is not reflected



A) Pearson



B) Spearman

Fig. 3. Correlations for Aggregate Data.

in the proxy. In terms of message count (leftmost), most bars are above 60%, showing that most parent messages have correspondence in the proxy, but when looking at communicating pairs (right middle), most bars are very low (10-30%), indicating that there are large numbers of process pairs that communicate in the parent but not in the proxy (albeit with low message counts). The exception to this is CTH and miniAMRZ for the ball&plate simulation; here miniAMRZ has about 98% correspondence to messages, and about 88% for communicating pairs. MiniAMRz is also significantly more correspondent to CTH than miniAMR for the 4-sphere simulation. The green bars compare miniAMR with miniAMRZ, and if one looks across the groups, the pattern of the green bars is similar to the pattern of the blue and orange bars; thus since miniAMR has a similar relationship to miniAMRZ as it does to CTH, we conclude that this indicates that miniAMRZ is more like CTH than miniAMR.

Figure 3 shows the Pearson and Spearman correlations over different views of parent-proxy relations. *Full view* means data includes all process pairs that have non-zero message counts in *either* the parent or the proxy; where a pair occurs in one but not the other, zero is entered for the corresponding message count. *Proxy view* means that process pair data is kept only for those pairs who have nonzero message counts in the proxy; parent pairs outside of this are discarded, and zero is entered in parent data for pairs that occur in the proxy but not in the parent. *Parent view* means that process pair data is kept only for those pairs who have nonzero message counts in the parent application; proxy pairs outside of this are discarded, and zero is entered in proxy data for pairs in the parent but not in the proxy.

Rather than just the overall percentages as the earlier figure showed, the correlations take into account whether message counts are similar over communicating pairs. Pearson is essentially a linear correlation, while Spearman can handle non-linear correlation effects. Figure 3A shows that the proxy-parent pairs have significant correlations, with the 4-sphere simulations showing distinctly higher correlations, and miniAMRZ showing distinctly higher correlation than miniAMR. However, Figure 3B shows quite lower Spearman correlations across this same data, and shows miniAMR sometimes more correlated than miniAMRZ. In viewing multiple different scatterplot views (too numerous to include), data for miniAMRZ spreads more uniformly on both sides of a linear relationship, while miniAMR data deviates more in one direction, which allows Spearman to curve in that direction and increase the correlation. This is most pronounced in the high correlation spike for miniAMR in the proxy view for the 4-sphere problem in Figure 3B; there are many zero message counts in miniAMR (proxy) that are nonzero in the parent (CTH), so when these are removed for the proxy view, the correlation increases. Given that there is no reason that a non-linear correlation would be a good proxy-parent relationship, what the Spearman correlation shows us is that while there is high deviation between the parents and proxies, miniAMRZ is more balanced in its linear relationship to CTH than miniAMR is.

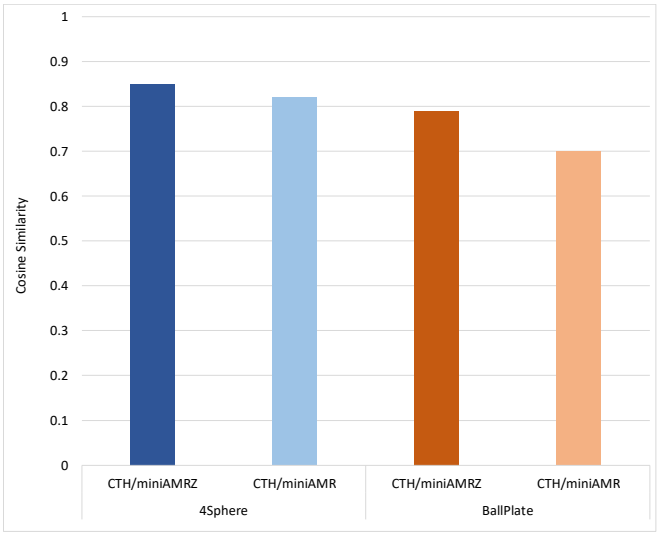
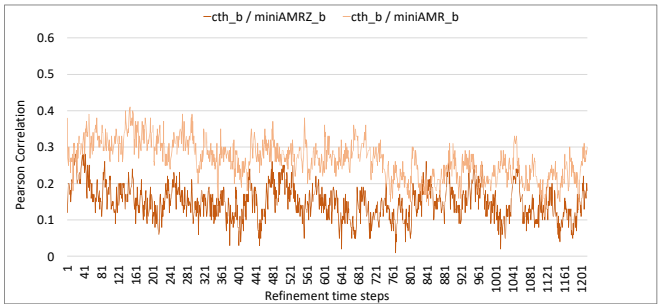
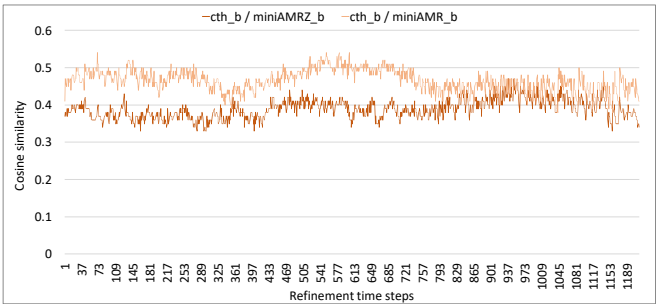


Fig. 4. Cosine Similarity between Proxy and Application.



A) Pearson



B) Cosine Similarity

Fig. 5. Similarity Metrics over Detailed Time Series Data, Ball & Plate.

We also evaluated our data using the *cosine similarity* metric, shown in Figure 4. Cosine similarity ignores the absolute magnitudes of the data values and calculates the angle of difference between the data as vectors, the directions being determined by the relative magnitude of the data values. The figure shows significant similarity between the parent and proxies and, consistent with the Pearson correlations, shows the 4-sphere simulations as more similar, and miniAMRZ more similar to CTH than miniAMR. One can see the same basic figure shape as the Pearson correlation has, indicating aggreement bewteen them.

After these aggregate metrics and comparisons, we now look at the dynamic communication behavior over the execution lifetime of the applications. As noted before, we captured the communication data at every refinement step of the applications, and the analyses below use this data.

Figure 5A shows the Pearson correlation of step data between CTH and the proxies for the ball&plate simulation. Interestingly, the per-step correlation is very low, always much lower than the overall correlation of about 0.66 (miniAMR) and 0.73 (miniAMRZ) from the previous figure. What this means is that even though they are doing the exact same number of timesteps and refinement steps, the communication of the identically offset intervals does not match. Moreover, miniAMRZ, which is in aggregate more correlated to CTH, is lower in per-step correlation than miniAMR.

Even though CTH and the miniAMR proxies perform exactly the same number of timesteps and refinement steps, we verified in the data that their refinement steps are *not* guaranteed to be always in the exact same place—i.e., between the exact same timesteps. Thus not only are their refinements happening over somewhat different methods, they are happening at somewhat different timesteps. Thus we lose correlation when trying to look at similarity at the level of refinement step.

Figure 5B shows the cosine similarity of the same data as Figure 5A. Again, the per-step similarity is significantly lower than the aggregate similarity, and miniAMRZ is less similar, per step, than miniAMR. It was unexpected that miniAMRZ would be less similar, step by step, to CTH than miniAMR. Thus we explore this below.

Figure 6 shows the *cumulative* cosine similarity for the

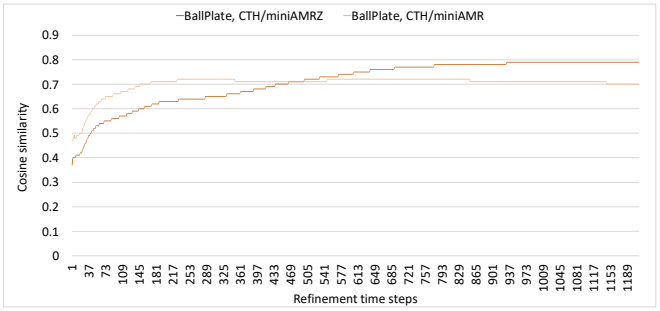


Fig. 6. Cumulative Stepwise Cosine Similarity, Ball & Plate.

ball&plate simulations, where similarity is computed not per step but over the data from the beginning to the current step under consideration. Thus by the end it reachs the aggregate similarity measure. Interestingly, it takes miniAMRZ almost half of the execution to rise above miniAMR in similarity to CTH. Our interpretation of this is that the applications do not quite do refinement in a lock-step similar fashion, but as the simulation proceeds the mesh refinement looks more similar in the proxies and parents than any individual step indicates. MiniAMR rapidly reaches its highest similarity to CTH, while miniAMRZ takes longer but ends up more similar to CTH.

Figure 7 shows the dynamic interval cosine similarity, but aggregated by 10, 100, and 200 datapoints. Thus these are still intervals, not cumulative from the beginning, but just different sizes of intervals. Interestingly, in almost none of the intervals is miniAMRZ more similar than miniAMR to CTH. Moreover, there is a consistent downward trend during execution. Recall that this is for the ball and plate simulation, which has more interaction area than the four sphere simulation. We interpret this as follows: with dynamic AMR, the complex simulations diverge *in their mesh detail* because they simply are not doing the exact same computation (miniAMR's is simplified). Thus communication does indeed slightly diverge, interval by interval. Yet, when accumulated together, since the mesh details are roughly in the same place, the overall aggregate communication is more similar than any sized interval.

Figure 8 shows the dynamic interval cosine similarity over the raw data intervals, and Figure 9 shows it over partial aggregations, both for the four sphere problem. Again, over

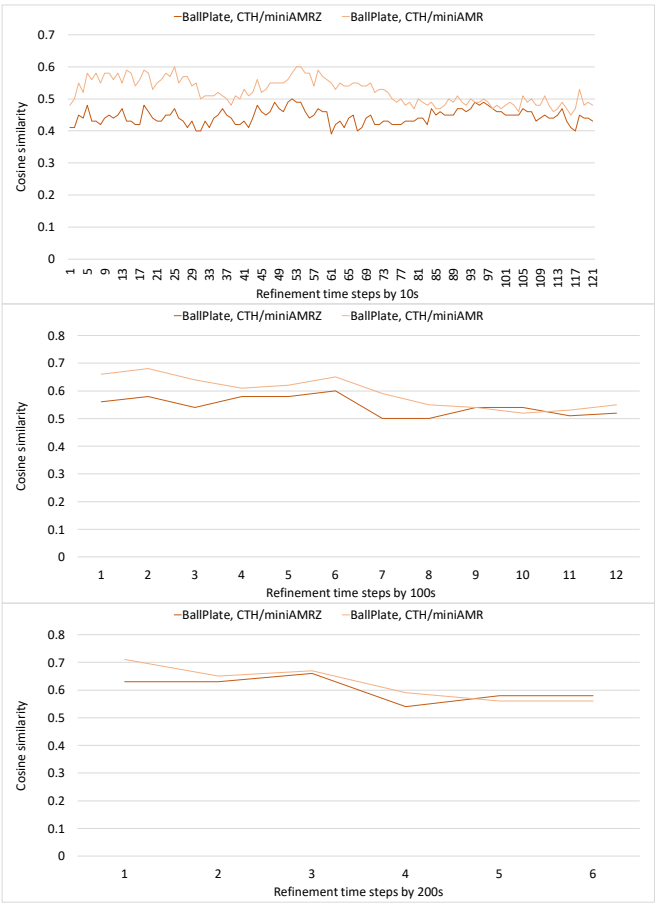


Fig. 7. Cosine Similarity Over Partial Aggregations, Ball & Plate

Fig. 9. Cosine Similarity Over Partial Aggregations, 4 Sphere

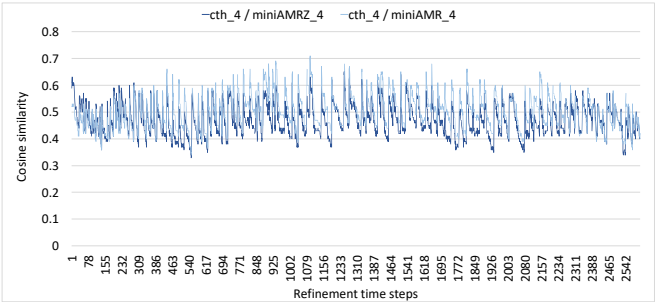


Fig. 8. Cosine Similarity per Each Refinement Step, 4 Sphere.

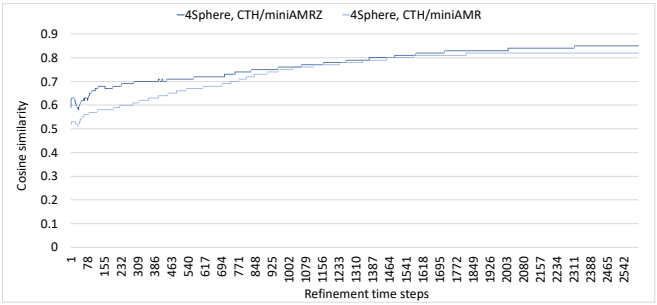


Fig. 10. Cumulative Stepwise Cosine Similarity, 4 Sphere.

none of the interval sizes does the similarity approach the whole-application similarity, and miniAMRZ is almost always lower than miniAMR.

Figure 10 shows the cumulative cosine similarity for the four sphere simulation, which smoothly rises to its aggregate level during the computation, and in which miniAMRZ is consistently higher than miniAMR. The similarities are higher than in the ball and plate simulation, for both miniAMR versions, and we explain this by the fact that since this simulation has less interaction, the miniAMR computation retains a more similar mesh to CTH than it does in the ball and plate simulation.

C. Analysis Summary

In the above we presented many different analyses and views, but what does it all signify? Our main conclusions are that *non-adaptive applications and proxies do not need dynamic communication behavior analysis* and that *adaptive applications and proxies should not necessarily be expected to have fine-grained similarity in their communication behavior*. These two conclusions have some caveats, of course. Both non-adaptive and adaptive communications could still have message sizes and bursts that dynamically exercise or saturate the hardware in different ways, and if an adaptive proxy is

meant to directly follow the parent's adaptive communication, then one should expect it to.

VI. Related Work

In our prior work, we noted at that time that there was little related work done on characterizing the similarity in communication patterns of parallel applications that use MPI. This is still the case and there is only one new piece of work that extends prior work in this area.

Ma et al. [12] present the only other work we have identified on characterizing similarity in MPI communication patterns. 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 [13] and [14] focuses on matching application communication patterns to a library of commonly observed patterns. Their methods are based on pattern matching and they are not focused on understanding pattern similarity (although their method could be applied to this with some extension). The work in [13] has been recently extended [15], and they improved it in [16] by representing the communication matrix (mpiP data–source, destination, number of messages, bytes transferred) as an augmented communication graph and then doing search space pruning based on a library of communication patterns to determine patterns that comprise the particular communication. As noted, this work could be applied to the problem of communication pattern similarity and will be leveraged in our work in the future. In [15] they discuss how to apply deep learning methods in their methodology.

VII. Conclusion and Future Work

In our prior work, we presented an exploration into quantifying a comparison of cumulative communication characteristics between parent and proxy. This work extends that methodology to include comparison of time-varying communication behavior using pairwise communication data. We define metrics that capture how much of one application matches the other and we use correlation metrics over the message counts of communicating pairs to further quantify this relationship. We found that for applications with dynamically driven communication characteristics such as those that use adaptive mesh refinement, the time-varying behavior of the parent and proxy can be quite different, rendering the use of cumulative data potentially misleading.

Although this work reveals the importance of examining the differences in time-varying behavior, it also exposes new questions/issues that need to be addressed. The first pertains to the fidelity of proxy apps. In this team, we have over 30 years of experience with CTH, and an author of miniAMR. miniAMR was originally intended to faithfully model only the communication in CTH. We see from our data that in spite of expertise, we have a proxy that has different communication characteristics with respect to its parent. We believe this is because miniAMR does not do exactly the same computation that is done in CTH, and since the communication is dependent on the dynamic computation, the communication is different. Therefore, for applications that are characterized by dynamic communication, it may be very important to ensure that the same computations are done in both the proxy and the parent. At a minimum, extreme care and caution must go into understanding proxy intent in terms of which specific parent behavior it models and developers must give adequate attention to modeling these behaviors accurately. This implies an iterative development-measurement cycle to ensure accurate representativeness. Intuition is not good enough.

Secondly, although the time-varying communication behavior in CTH and miniAMR do not closely match, the underlying behavior at the network level may be the important characteristic we should be trying to mimic in the proxy. We need to address the question as to how the communication behavior we have observed differs or matches at the hardware level. For example, do these two applications demonstrate the same behavior with respect to network congestion, traffic at the NIC, point-to-point message latency? This is our next step for future work.

The instrumentation used in this work was either expensive to use (e.g., one full execution per data collection interval) or was hand-crafted, which points to an area of possible tool improvement. Efficient tools often aggregate data until the end of execution (e.g., CrayPat, mpiP); an ability to "checkpoint" this data at selectable intervals would enable efficient dynamic data over which to perform analyses.

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, and C. Vaughan, "Exploring and quantifying how communication behaviors in proxies relate to real applications," in *2018 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, Nov 2018, pp. 12–22.

[2] 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

[3] E.S. Hertel, Jr., R. L. Bell, M. G. Elrick, A. V. Farnsworth, G. I. Kerley, J. M. Mcglaun, S. V. Petney, S. A. Silling, P. A. Taylor, and L. Yarrington, "CTH: A Software Family for Multi-Dimensional Shock Physics Analysis," in *Proceedings, 19th International Symposium on Shock Waves*, 1993, pp. 377–382.

[4] C.T. Vaughan and R.F. Barrett., "Enabling Tractable Exploration of the Performance of Adaptive Mesh Refinement," in *Workshop on Representative Applications at IEEE Cluster*, 2015.

[5] 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

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

[7] 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

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

[9] M.J. Berger and J. Oliger, "Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations," *Journal of Computational Physics*, vol. 53, no. 3, pp. 484–512, 1984.

[10] M.J. Berger and S.H. Bokhari, "A Partitioning Strategy for Nonuniform Problems on Multiprocessors," *EEE Trans. Comput.*, vol. 36, pp. 570–580, May 1987.

[11] K. Devine, B. Hendrickson, E. Boman, M. St.John, and C. Vaughan, "Zoltan: A Dynamic Load-Balancing Library for Parallel Applications; User's Guide," Sandia National Laboratories, Tech. Rep. Technical Report SAND99-1377, 1999.

[12] 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.

[13] 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

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

[15] P. C. Roth, "Improved Accuracy for Automated Communication Pattern Characterization Using Communication Graphs and Aggressive Search Space Pruning," *Lecture Notes in Computer Science*, vol. 11027, pp. 38–55, Apr. 2019.

[16] P. C. Roth, K. Huck, G. Gopalakrishnan, and F. Wolf, "Using Deep Learning for Automated Communication Pattern Characterization: Little Steps and Big Challenges," *Lecture Notes in Computer Science*, vol. 11027, pp. 265–272, Apr. 2019.