SAND2019-10583C

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

A1
a@a.com
A1

A1
a@a.com
A1

A1
a@a.com
A1

A1
a@a.com
A1

ABSTRACT

Understanding the HPC applications behavior is necessary to evaluate the HPC system hardware and software for exascale networks and beyond. Most High-Performance Computing (HPC) applications depend heavily on communication to transfer data between running processes. Therefore gaining insight into the application communication performance helps to determine how well set of applications are related to each other. In this paper, we study how well a cumulative method shows representations of the HPC applications. This method ...

KEYWORDS

Workload characterization; Proxy applications; Performance evaluation; Big data

ACM Reference Format:
A1, A1, A1, and A1. 2019. Fine-Grained Analysis of Communication Similarity between Real and Proxy Applications. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 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 [11], 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.

Using aggregate metrics ignores the possibility that the proxy application might only be representing part of the parent application. Sometimes this is 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 and ExaminiMD, and HACC and SWFFT. The new pair is the real application CTH [**?**], a very large ... that is heavily used, and the proxy application is miniAMR [**?**], 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 thus are: an exploration and presentation of the dynamic communication behaviors and relationships between three parent/proxy application pairs; an the 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.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND

In 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 [?].

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. In 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.

We formed three different data vectors from these matrices that defined three different views of the data:

- **Parent view**: keeping all elements in the proxy and parent vectors that are nonzero elements of the parent application, and removing from the proxy any nonzero elements not in the parent set; the proxy vector will have zero elements where the parent has a nonzero element but it did not.
- **Proxy view**: keeping all elements in the two vectors that are nonzero elements of the proxy application; other nonzero parent elements are removed, and zero elements in the parent are kept where the proxy has nonzero elements.
- **Full view**: keeping all elements that are nonzero elements in either the proxy or the parent, and augmenting both with zero elements where needed.

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

In that work we defined and evaluated descriptive statistics (percent overlap) and correlations (Pearson and Spearman) over these data sets. **Note interesting results?...**

3 METHODOLOGY

Write about data analyses used here.

4 EXPERIMENT SETUP

4.1 Applications

4.1.1 HACC and SWFFT. The Hardware Accelerated Cosmology Code (HACC) [2] 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 [3] 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.

4.1.2 LAMMPS and ExaminiMD. LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator) [6] 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 [10], 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.

4.1.3 CTH and miniAMR. CTH is a multi-material, large deformation, strong shock wave, solid mechanics code developed at Sandia National Laboratories [?]. 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 [?]. 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.

4.2 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) [**?**] 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 factor is that CTH uses the Zoltan load balancing library [**?**] 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 factor 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 factor 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 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 closer to 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.

4.3 Instrumentation

In our previous work we collected total aggregate message counts for the lifetime of the program, using the CrayPat [**?**] tool. Our interest in this paper is in refining this into looking at the 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 the applications HACC, SWFFT, Lammps, and ExaminiMD, we did instrument the code to run CrayPat 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.

In order to look at the communication differences between CTH and miniAMR, we instrumented the codes to output the communication matricies 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 in effect until the next refinement step. We outputed that information into a file for each refinement step and were able to post process the communication patterns for the entire run. For miniAMR, we are 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 for the next few timesteps. We likewise were able to output that information into files to compare to those from CTH.

4.4 Input Problem Details

Something for LAMMPS/XMD and HACC/SWFFT??

For the 4 spheres problem in CTH and miniAMR, we ran 7819 timesteps for 5 e-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 1 shows the resulting simulation state from miniAMR on the 4 spheres problem, while Figure 2 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.

4.5 Platforms

Someone read and check this...

All experiments were performed on Mutrino, a small Cray XC40-based cluster at Sandia National Laboratories, with nodes having Intel Haswell processors and the cluster having a Cray Aries interconnect.

For LAMMPS and ExaminiMD, ...

For HACC and SWFFT, ...

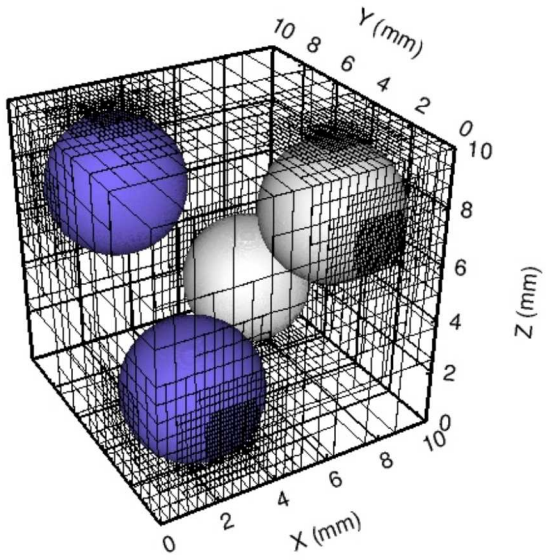

Figure 1: miniAMR Simulation Result for 4 Spheres.

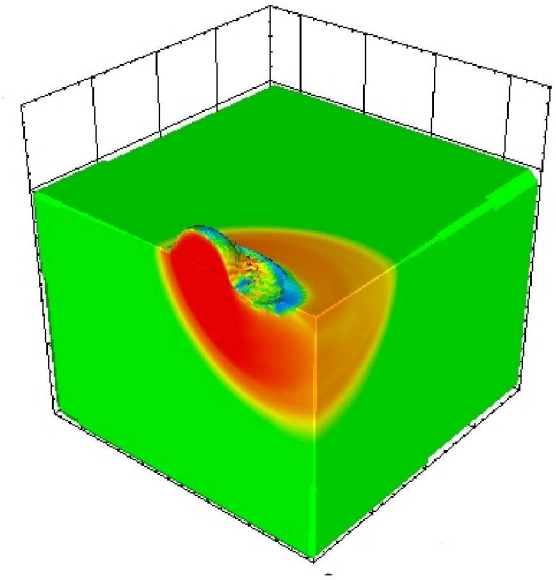

Figure 2: CTH Simulation state for Ball & Plate.

For CTH and miniAMR, all executions were done with 128 ranks over 4 nodes, and with no OpenMP being used.

5 RESULT AND ANALYSIS

Figure 3 shows the basic overlap relations in the communication of CTH and the two miniAMR versions. 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 that also occur in parent (second leftmost). The two right-side groups are similar,

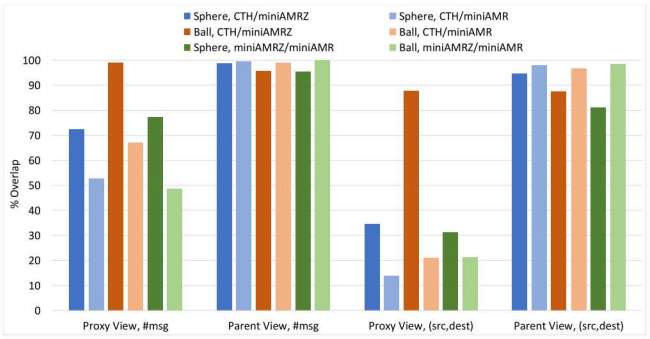

Figure 3: Communication overlap relations.

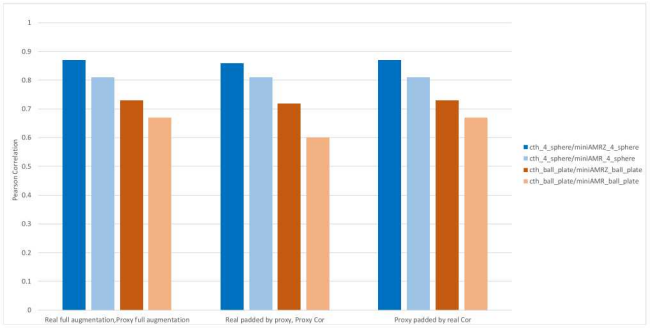

Figure 4: PearsonCorrProxyRealMsgPair.pdf

except over communicating pairs (boolean, whether they communicated or not) rather than message counts. The percentages of proxy in parent are all very high, indicating that almost all messages that the proxy communicates match some communication in the parent. The proxy-in-parent pair bars are slightly lower, indicating that there are a few low-count communicating pairs in the proxies that are not in the parent. However, the parent-in-proxy view is very different, indicating that there is parent communication behavior that is not reflected in the proxy. In terms of message count (leftmost), most bars are above 60%, showing that most 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 that miniAMR for the 4-sphere simulation.

Figure 4 and Figure 5 show the Pearson and Spearman correlations over different views of parent-proxy relations. Full augmentation 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-based 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.

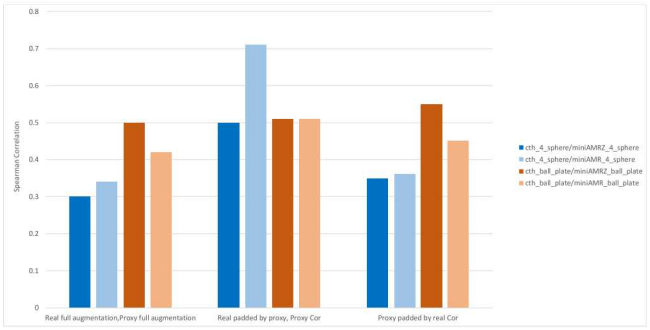

Figure 5: SpearmanCorrProxyRealMsgPair.pdf

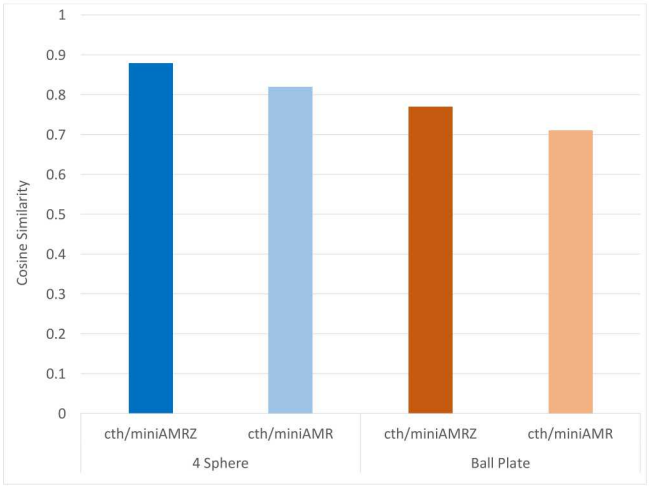

Figure 6: CosineSimilarity.pdf

Real-based view means that process pair data is kept only for those pairs who have nonzero message counts in the real 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 different pairs. Pearson is essentially a linear correlation, while Spearman can handle non-linear correlation effects. *To me it looks like the Pearson figure is wrong – are we sure that there isn't any copied-excel formula problems here???* Figure 4 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, because Figure 5 shows quite lower Spearman correlations across this same data, the simpler Pearson might be be suffering from effects that do not perturb the linear correlation computation. **We should look at the data and visually decide...**

We also evaluated our data using the *cosine similarity* metric, shown in Figure 6. 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

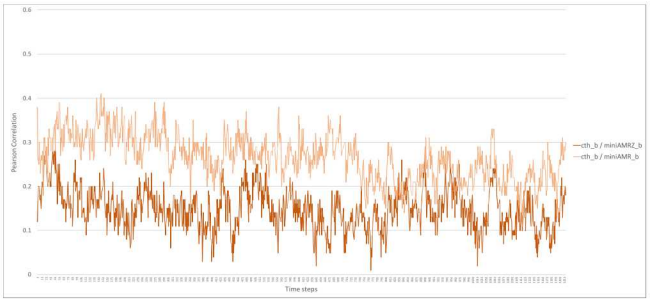

Figure 7: PearsonSimilarityPhasesBall.pdf

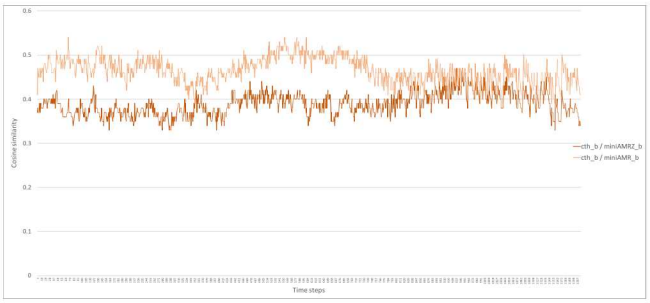

Figure 8: CosineSimilarityPhasesBall.pdf

similarity between the parent and proxies and, consistent with the other correlations, shows the 4-sphere simulations as more similar, and miniAMRZ more similar to CTH than miniAMR.

After these aggregate metrics and comparisons, we now look at the 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 7 shows the Pearson correlation of step data between CTH and the proxies for the ball&plate simulation. **Full augmentation??** 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 need to, and indeed does not, match. **Could we have an off-by-one error?** Also, miniAMRZ, which is in aggregate more correlated to CTH, is lower in per-step correlation than miniAMR.

Figure 8 shows the cosine similarity of the same data as the previous figure (step data for ball&plate simulations). Again, the per-step similarity is significantly lower than the aggregate similarity, and miniAMRZ is less similar, per step, than miniAMR.

Figure **??** shows the *cumulative* cosine similarity for the 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. 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

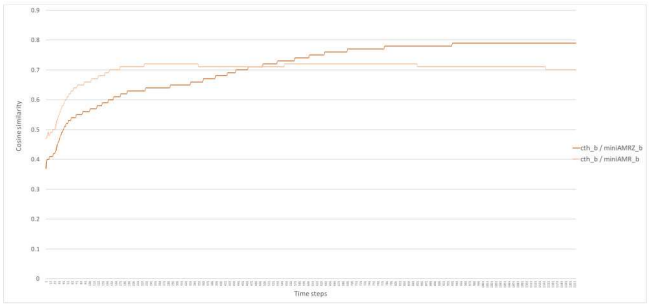

Figure 9: CosinePhasesAccumBall.pdf

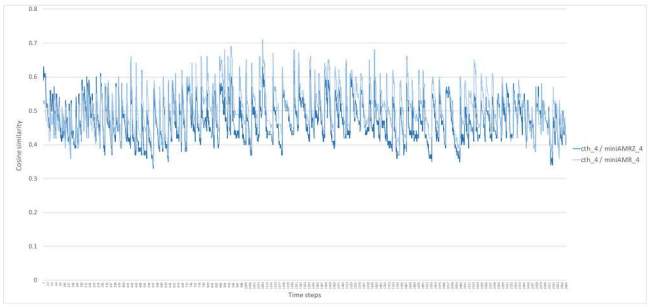

Figure 10: CosineSimilarityPhases4Sph.pdf

than any individual step indicates. MiniAMR rapidly reaches its highest similarity to CTH, while miniAMRZ takes longer but is more similar to CTH within less than half the execution time.

Figure 10 shows the per-step cosine similarity for the 4-shpere simulations. If we are using this we need more figures...

6 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. [5] present the only other work we have identified on characterizing similarity in MMPI 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 [4, 9] 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 [9] has been recently extended [7, 8]. They improved it in [8] by representing the communication matrix (mpiP data–source, destination, number of messages, bytes transferred) as an augmented communication graph 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 [7] they discuss how to apply deep learning methods in their methodology.

7 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 easily collected with mpiP. 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. MAYBE ADD MORE QUANTITATIVE STATEMENTS ONCE I LOOK MORE CLOSELY AT THE DATA.

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 the 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.

8 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)*, pages 12–22, Nov 2018.

[2] Salman Habib, Vitali Morozov, Nicholas Frontiere, Hal Finkel, Adrian Pope, Katrin Heitmann, Kalyan Kumaran, Venkatram Vishwanath, Tom Peterka, Joe Insley, David Daniel, Patricia Fasel, and Zarija Lukić. Hacc: Extreme scaling and performance across diverse architectures. *Commun. ACM*, 60(1):97–104, December 2016.

[3] https://xgitlab.cels.anl.gov/hacc/SWFFT. Swfft (hacc).

[4] Darren J. Kerbyson and Kevin J. Barker. Automatic identification of application communication patterns via templates. In *ISCA PDCS*, 2005.

[5] 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*, pages 1517–1522, Dec 2008.

[6] Steve Plimpton. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.*, 117(1):1–19, March 1995.

[7] Philip C. Roth. Improved Accuracy for Automated Communication Pattern Characterization Using Communication Graphs and Aggressive Search Space Pruning. *Lecture Notes in Computer Science*, 11027:38–55, April 2019.

[8] Philip C. Roth, Kevin Huck, Ganesh Gopalakrishnan, and Felix Wolf. Using Deep Learning for Automated Communication Pattern Characterization: Little Steps and Big Challenges. *Lecture Notes in Computer Science*, 11027:265–272, April 2019.

[9] Philip C. Roth, Jeremy S. Meredith, and Jeffrey S. Vetter. Automated characterization of parallel application communication patterns. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '15, pages 73–84, New York, NY, USA, 2015. ACM.

[10] Aidan P. Thompson and Christian Robert Trott. A brief description of the kokkos implementation of the snap potential in examinimd. 11 2017.

[11] Jeffrey S. Vetter and Michael 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*, PPoPP '01, pages 123–132, New York, NY, USA, 2001. ACM.