Similarity Measurement for Proxy Application Fidelity

Extended Abstract

Si Chen
Emory University
Atlanta, Georgia
si.chen2@emory.edu

Omar Aaziz
Sandia National Laboratories
Albuquerque, New Mexico
oaaziz@sandia.gov

Jeanine Cook
Sandia National Laboratories
Albuquerque, New Mexico
jeacook@sandia.gov

Avani Wildani
Emory University
Atlanta, Georgia
avani@mathcs.emory.edu

ABSTRACT

Proxy applications are widely used for system co-design and procurement because they are designed to represent some much larger and more complex parent applications with similar behavior or characteristic. However, quantitative validation is missing to prove proxies are faithful representations of their parents. In this work, we applied several machine learning-based methods to quantitatively understand the correspondence of proxy to parent in their underlying node and memory hardware behavior. We show that different similarity measurements have similar results while evaluating the proxy application fidelity, and most proxies are good representations of their parents.

KEYWORDS

Co-design, proxy/parent representativeness, similarity measurement

1 INTRODUCTION

Co-design is crucial for system designs to achieve optimal application performance. While some science applications have thousands of lines of code and complex dependencies, proxy applications are open-source, smaller, more tractable models which represent some characteristics of their parent applications. Many primary computer system vendors use proxy applications for activities like system co-design and procurement. Therefore, we need to ensure that the proxy applications used are high fidelity models of target applications, otherwise, the system would have highly sub-optimal parent application performance. Specifically, our focus is on the quantitative evaluation of proxy fidelity by using an effective and robust method.

2 METHODOLOGY

We present our methodology for collecting data and how we use that data to produce similarity using three different ML techniques. We use two system platforms with different architectures to collect data on 21 applications that span several scientific domains. We collected PAPI [3] hardware performance counters using LDMS [1] to give us a large array of measurements to understand how similar the applications utilize system resources in term of node computation and memory.

To make the performance comparable, we run each application with the same input problem and/or parameters where possible. Data is collected from each application process (i.e., each MPI rank), and we compute an average for each event. In the end, each application on each platform is represented by a vector with 700 dimensions (features), each of which is an average rank value for one hardware event. Moreover, we create several architectural subgroups for these features, including cache, branch prediction, pipeline, instruction mix, memory, virtual memory, and others. We then use distance measurement to evaluate similarity for overall vectors as well as subset vectors.

There are many different distance measurements to choose from. Our analysis tools are cosine similarity and two statistic distance methods. Cosine similarity, which is defined by $\cos\theta = \frac{\sum_{i=1}^{n} x_i y_i}{\|x\| \|y\|}$, compares the angle between two vectors by virtue of properties of the vector inner product, while 0 degree and 90 indicate similar (the same) and dissimilar respectively. Statistic distance methods quantify the distance between two probability distribution. Based on Kullback–Leibler (KL) divergence, which is defined by $D_{\mathrm{KL}}(P\|Q) = \sum_{x\in\mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$, Jensen-Shannon divergence $\mathrm{JSD}(P\|Q) = \frac{1}{2}D(P\|M) + \frac{1}{2}D(Q\|M)$ is know as total divergence to the average. JS divergence has some notable improvement of KL divergence, including symmetric and bounded value from 0 to 1. Wasserstein distance, which is defined by $W_2(p,q) = \sqrt{\min_{P_{XY}} E_{P_{XY}}\left[\|x-y\|_2^2\right]}$ s.t. $P_X \sim p, P_Y \sim q$, is known as the earth mover's distance. It is the minimum "cost" of turning one distribution into the other, which is assumed to be the amount of distribution that needs to be moved times the mean distance it has to be moved.

After calculating the similarity metrics, we use the clustering method to further rank the applications. In our work, we use Hierarchical Clustering to group applications by tree branch height.

3 RESULT

We use a suite of 21 total applications from Exascale Computing Project (ECP). Some are proxy/parent pairs: ExaMiniMD/LAMMPS, miniQMC/QMCPACK, miniVite/Vite, Nekbone/Nek5000, PICSARlite/PICSAR, SW4lite/SW4, SWFFT/HACC, and XSBench/OpenMC. We also used 5 extra applications that don't have designated parents to evaluate our methodology. The applications are: AMG2013, Laghos, PENNANT, SNAP, and Castro.

We set up the experiment on two system platforms with different architecture, Intel Skylake and IBM Power9 system. We run all of our applications in MPI-only mode, using 128 ranks, one rank per core, on four nodes.

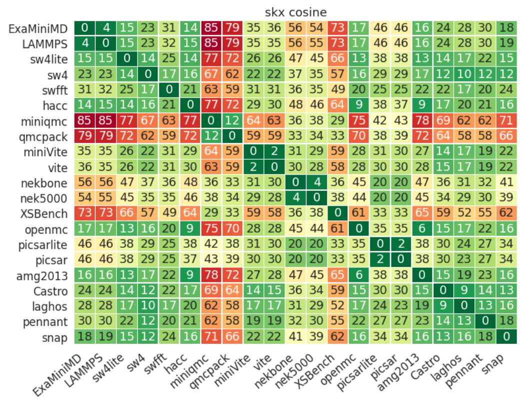

Figure 1: Cosine Similarity, Skylake

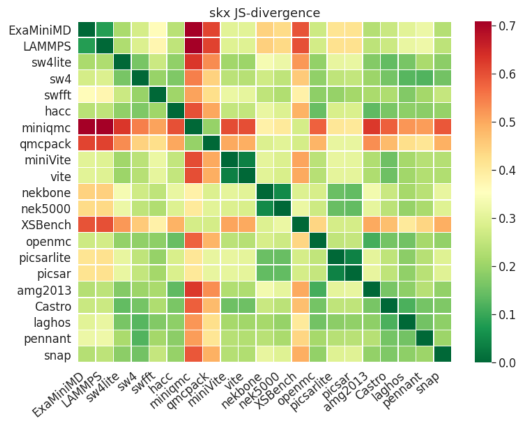

Figure 2: JS distance, Skylake

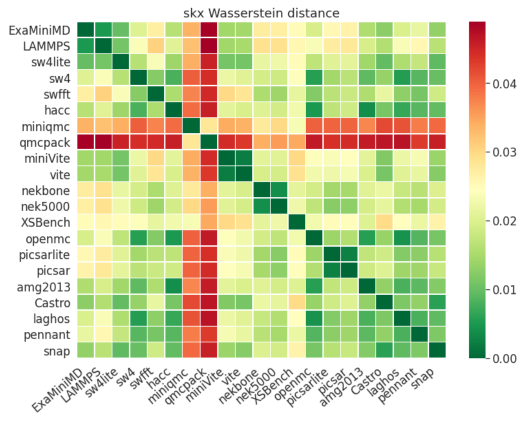

Figure 3: Wasserstein distance, Skylake

Figure 1, Figure 2, and Figure 3 are the overall feature results of three distance methods in the Skylake system. From dark green to dark red means from highly similar to highly dissimilar. We can see that generally, three distance methods look alike. Two applications (QMCPack and MiniQMC) are similar but very different from other applications. Three pairs (miniVite/vite, picsarlite/picsar, ek5000/nekbone) are the most similar proxy/parent pairs. Two pairs (nek5000/nekbone and vite/miniVite) show high similar performance in cosine similarity and relatively high similar performance in the other two methods. The last five unpaired applications show relative similarity.

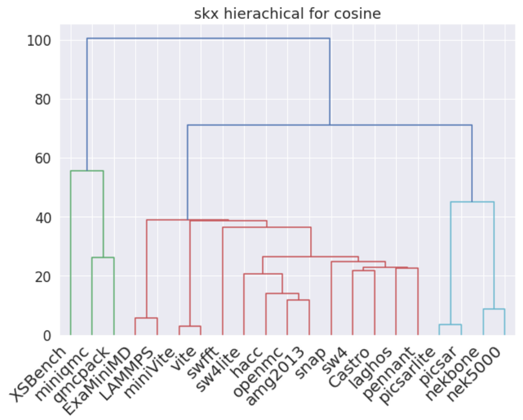

Figure 4: Hierarchical Clustering for cosine similarity, Skylake

We observe that Hierarchical Clustering shows similar ranking results for three distance methods. In Figure 4, three most similar proxy/parents pairs show the shortest branch heights. Cross-platform shows different results because of different memory subsystems and SIMD widths between Intel Skylake and IBM Power9. Overall and subset results do not have much difference, and three subgroups (Cache-related, Instruction mix, and Pipline) contribute most to the overall similarity matrix.

4 RELATED WORK

Related work has been published related to proxy application characterization and comparison of individual proxies to their parent applications (e.g., [2]), and in using proxy applications to project system performance of real applications (e.g., [5]). There are also other cosine similarity metrics proposed by the machine learning community, such as Square Root Cosine Similarity and Improved Square Root Cosine Similarity [4].

5 CONCLUSION

We have explored three similarity measurement methods to evaluate how proxy and parent applications relate, to each other and to other applications. We show that most proxies are good representations of their parents, only with few divergences. Since other distance methods show similar results, cosine similarity is good enough to use for similarity measurement because of its simplicity, quickness, and interpretability by geometric angle. For the next step, we will calculate self-similarity for time series data with sliding windows to further validate the similarity methods. Additional future work will be to validate similarity through measures of latency or some other environmental/ real metric and domain expert evaluation.

REFERENCES

[1] Anthony Agelastos et al. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In *SC'14: Proc. International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '14, pages 154–165, Piscataway, NJ, USA, 2014. IEEE Press.

[2] Richard F Barrett, Paul S Crozier, DW Doerfler, Michael A Heroux, Paul T Lin, HK Thornquist, TG Trucano, and Courtenay T Vaughan. Assessing the role of mini-applications in predicting key performance characteristics of scientific and engineering applications. *Journal of Parallel and Distributed Computing*, 75:107–122, 2015.

[3] Philip J Mucci, Shirley Browne, Christine Deane, and George Ho. PAPI: A Portable Interface to Hardware Performance Counters. In *Proceedings of the department of defense HPCMP users group conference*, volume 710, 1999.

[4] Sahar Sohangir and Dingding Wang. Improved sqrt-cosine similarity measurement. *Journal of Big Data*, 4(1):1–13, 2017.

[5] Miwako Tsuji, William TC Kramer, and Mitsuhisa 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)*, pages 826–833. IEEE, 2017.