Hierchical Clustering and K-means Analysis of HPC Application Kernels Performance Characteristics

M.L. Grodowitz and Sarat Sreepathi
Oak Ridge National Lab
Oak Ridge, Tennessee
Email: {grodowitzml, sarat}@ornl.gov

***Abstract*—In this work, we present the characterization of a set of scientific kernels which are representative of the behavior of fundamental and applied physics applications across a wide range of fields. We collect performance attributes in the form of micro-operation mix and off-chip memory bandwidth measurements for these kernels. Using these measurements, we use two clustering methodologies to show which applications behave similarly and to identify unexpected behaviors, without the need to examine individual numeric results for all application runs. We define a methodology to combine metrics from various tools into a single clustering visualization. We show that some kernels experience significant changes in behavior at varying thread counts due to system features, and that these behavioral changes appear in the clustering analysis. We further show that application phases can be analyzed using clustering to determine which section of an application is the largest contributor to behavioral differences.**

I. Introduction

Parallel scientific applications present a unique set of problems to system and architecture design. Large scientific computational clusters and data centers are created to produce experimental results for a wide variety of fields, resulting in highly varied workloads. It is therefore an ongoing problem to determine which potential system improvements would benefit which computations.

Existing classifications of parallel application behaviors attempt to group applications either by algorithm type [1] or by scientific sub-domain [2]. However, these classifications do not take into account the variability of algorithms due to input sets, architectural features, levels of parallelism, or other transient effects experienced on real systems. To study a richer characterization of application behaviors, we have built an infrastructure to support the collection of large numbers of experimental results of applications in various configurations [3].

However, with large numbers of experiments, traditional techniques [4] of comparing numeric performance numbers for each experiment becomes overwhelming. In this study, we apply big-data analytic methods to the exploration of performance data of applications. In particular, we are interested in using clustering and grouping methods to classify applications based on runtime conditions and system requirements.

Recently, a set of representative test applications was compiled for the purpose of large scale system design and acquisition [5]. This CORAL collaboration between several large scientific laboratories collected a set of test applications to provide a comprehensive characteristic scientific workload. Though that particular system selection process has now ended, there is value in continued study of this collection of test applications. In this work, we present the results of characterizing the complete set of microkernels from the CORAL collaboration, listed in table I.

The remainder of this paper is organized as follows. Section II describes the methodology of application characteristic collection and clustering methods. Section III provides descriptions of the computational kernels. Section IV presents the results of the study, followed by a discussion of conclusions and future directions in section V.

II. Analytic Methodology and Framework

The results presented here are not a stand-alone project. This work is part of a larger ongoing project to provide an infrastructure for application analysis and the storage of performance experiments. This section briefly summarizes previous work on the tools used to collect application data, then explains the new analytic methodology and visualizations used to extend these tools in this study.

A. Oxbow Toolkit and PADS Storage Infrastructure

The Oxbow project [3] provides a set of tools to characterize application performance beyond traditional methods of timing sections. One of the primary drivers of this project is the analysis of co-design proxy application and HPC computational kernels in order to project how future architectural features will benefit these applications.

In contrast to timing experiments, which ask "How well does this program run on this machine", the Oxbow project asks "On what kind of machine will run these programs well"? By using real codes on current architectures, along

TABLE I. Collection of Representative Scientific Kernels

Application	Language	Description
AMGmk	C	Algebraic multigrid solver
GFMCmk	Fortran	Solving Schrödinger equation with Monte Carlo methods
HACCmk	C	N-body cosmology simulation short force evaluation step
MILCmk	C	Heavily used kernels from the quantum chromodynamaics (QCD) linear algebra library (QLA)
Nekbone-kernel	Fortran	Poisson equation solver (incompressible Navier-Stokes solver)
UMTmk	Fortran	Deterministic radiation transport angular flux discretization step

TABLE II. INSTRUCTION MICRO-OPERATION DESCRIPTIONS.

Category	Description
BrOps	Conditional/unconditional branches; direct and indirect jumps
FpOps	Scalar floating-point arithmetic
FpSIMD	Vector floating-point arithmetic
IntOps	Scalar integer arithmetic
IntSIMD	Vector integer arithmetic
MemOps	Scalar load and store operations
MemSIMD	Vector load and store operations
Moves	Integer and floating-point register copies; data type and precision conversions
Misc	Other miscellaneous operations, including pop count, memory fence, atomic operations, privileged operations

with a variety of lightweight profiling tools, this project can quickly collect characteristic information about a large variety of applications targeting extreme scale systems.

For this study, we use two tools from Oxbow. The first tool collects instruction mixes. This tool is built on top of the Intel PIN dynamic instrumentation toolkit [6]. Since a given CISC x86 instruction can perform many hardware operations, each x86 instruction is analyzed and broken down into the micro-operations listed in table II. These correlate more closely to generic RISC instructions.

The second tool uses PAPI counters to collect the total number of off-chip accesses in a given program section. These counters are used in combination with PAPI counters for cycle count, and the known cache line size, to generate main memory bandwidth results in terms of bytes per cycle.

All applications are instrumented with calipers around the computational sections of interest. The Oxbow code base allows for using the same calipers for multiple runs with different tools. This is important because tools using PIN would interfere with hardware counters, so each application must be run twice, once to collect instruction mix and again for memory bandwidth. Using the same calipers for multiple tools makes it possible to combine multiple runs without the chance of mismatched sections.

The PADS infrastructure is the data collection backend for Oxbow. It consists of a MongoDB database for flexible results collection and easy addition of new tools. This feeds data to a web front end which visualizes the stored experiments. To view the current collection of Oxbow experimental data, visit https://oxbow.ornl.gov, and follow the link to the "Portal".

B. Hierarchical/Agglomerative Clustering

The first analytic technique used to process the output from the instruction mix and memory bandwidth measurement tools is hierarchical/agglomerative clustering.

This type of clustering uses a distance metric to measure the difference between each experiment. In general, a distance metric for this type of clustering can be any function. We defined distance as euclidean distance for three different types of space:

- Type I: 9D space between points at the coordinates defined by the percentage-wise micro-op mix of an experiment.
- Type M: 2D space between points at the coordinates defined by the bytes per cycle consumed for read and write bandwidth.
- Type M-I: 9D space between points at the coordinates defined by instructions per cycle of all non-memory operation types (7D for micro-op mix), and cache lines per cycle consumed for read and write bandwidth (2D for memory bandwidth).

The first two types of analysis use direct output from the tools. The final analysis type integrates tool outputs in a unique way. Since micro-op mix is based on PIN tool output, the timing of calculations is very slow due to instrumentation overhead, but counts are known. The second tool has much lower overhead, and so we get more accurate cycle counts and bandwidths. By combining two experiment runs, we obtain information that would not be possible with either tool alone.

The following equation defines the distance metric between two experiments, α and β. Each experiment has a set of coordinates, X, which are defined by the type of analysis being performed.

$$\sqrt{\sum^{x_\alpha \in X_\alpha, x_\beta \in X_\beta} (x_\alpha - x_\beta)^2}$$

Using the distance equation, a tree is built iteratively. At every iteration, the geometric center is calculated for each cluster, which is then used for distance calculations between clusters. Clusters are joined together with the nearest neighboring cluster, resulting in the final relational hierarchy.

Leaves under a common branch are more similar to each other than leaves under a different branch. Vertical branching points represent a relative degree of difference between branches. Drout and Smith [7] provide a good introduction to dendrogram interpretation. Section IV will describe in more detail how to read the dendrograms presented here.

C. Kmeans Clustering

The hierarchy created by agglomerative clustering provides a good deal of information about relative similarity. However, as the number of experiments increases, it becomes desirable to have a more simplistic analytic output, such as a set of clusters, each of which is a list of similar experiments.

To produce this output, we use k-means clustering. This method partitions n observations into k clusters, where each observation belongs to the cluster with the nearest mean.

Formally: Given a set of experimental results, $X_1, X_2, ...X_n$, where each result is a d-dimensional vector, e.g. 9D vector for Type-I analysis, k-means clustering will partition the n experiments into k sets $S = \{s_1, s_2, ..., s_k\}$ so as to minimize the within cluster sum of squares. So, k-means will attempt to find:

$$min \sum_{i=1}^{k} \sum_{x \in s_i} ||x - \mu_i||$$

where μ_i is the mean of points in s_i

The results of our k-means clustering are presented as lists of 6 clusters along with a simple projection of the clusters

onto 2 dimensional space, using the primary two dimensions as the projection. We chose $k = 6$ by examining the curve of decreasing distortion with increasing k, where distortion is a metric of how good of a fit could be made for all points into clusters. For this analysis, distortion did not decrease significantly past $k = 6$.

III. Application Descriptions

This section provides descriptions of the kernels used for this study. All of these kernels were extracted from some larger proxy application. All but one of these kernels (UMTmk) uses OpenMP parallelism, and none of them use MPI. These kernels are therefore all meant for studies of single node performance issues, such as processing and memory. In all cases, there is no input set, as the tests are essentially hardcoded. The only parameter to be varied between runs is the number of threads being used.

A. AMGmk

AMGmk was derived from the larger benchmark application AMG, a parallel algebraic multigrid solver for linear systems arising from unstructured grids. In AMG, linear systems are built for 4 test problems: 2 stencil solutions to Laplace type problems, and 2 PDE solvers with Dirichlet boundary conditions. In AMKmk, these problems are abstracted into three computational kernels: a compressed sparse row matrix-vector multiply, an AMG mesh relaxation, and a vector operation to solve $aX + Y$.

AMG is written in C and uses OpemMP parallelism. Runtime of AMGmk is not expected to relate linearly to the runtime of the full AMG benchmark, but improvements to AMGmk runtimes can be expected to improve runtimes of AMG by some amount.

B. GFMCmk

GFMCmk extracts the primary computational kernel from the GFMC (Green's Function Monte Carlo). In GFMC, the nuclear quantum-mechanical wave function is expressed as a vector of components describing the state of each nucleon. GFMC starts with an approximate wave-function, then improves it iteratively. Each improvement step requires a new 3A-dimensional integral, where A is the number of nucleons. This calculation is an integral of many thousands of dimensions, done by Monte Carlo method.

GFMCmk is a Fortran code that uses OpenMP parallelism.

C. HACCmk

HACCmk is a kernel of the HACC (Hardware Accelerated Cosmology Code) framework. HACC was designed to simulate the evolution of the Universe using N-body methods. HACCmk calculates short force evaluation by looping over a list of particles and the particle neighborhood (i.e. particles close enough to be influenced by short forces).

Each particle force is independently calculated, and the loop over all particles is parallelized with OpenMP.

D. UMTmk

UMTmk is the most computationally intense single threaded kernel from the UMT (Unstructured Mesh Transport) proxy application. The purpose of the full proxy app is to explore solutions to the problem of obtaining accurate transport solutions to three dimensional problems modeling the transfer of thermal protons. This type of solution requires the use of unstructured grids.

This Fortran microkernel measures performance of a set of loops in a single function (SNSWP3D) extracted from UMT. SNSWP3D performs the discretization step for angular fluxes in a single direction. In the full proxy app, this function is called within OpenMP loops, so SNSWP3D, and therefore UMTmk, is completely serial.

E. Nekbone-kernel

The Nekbone-kernel solves $A \times u = w$ over some number of 3D elements using a series of matrix-matrix multiplications. This computation forms the core of the Nekbone proxy app, which solves a standard Poisson equation using a spectral element method conjugate gradient solver with a simple preconditioner. Whereas Nekbone has a setup phase and solution phase, Nekbone-kernel is only the solution phase.

Nekbone-kernel is a Fortran microkernel that is parallelized with OpenMP.

F. MILCmk

MILCmk is a set of seven kernels from the MIMD Lattice Computation (MILC) collaboration code. These are some of the more heavily used routines extracted from the QCD Linear Algebra library developed under SciDAC. The purpose of MILC is to simulate the fundamental theory of strong nuclear force, quantum chromodynamics (QCD).

This microkernel is written in C, and uses OpenMP for parallelism. It can be run in either single or double precision mode. For this study, we ran only in double precision mode.

IV. Results of Kernel Analysis

We ran all experiments on a server running Linux CentOS 6, with 16 cores provided by 2, 8-core 3.3 GHz Intel Xeon Processors with a Sandybridge-EP architecture. Cache size per processor is 1MB shared L3, 256K L2 per core, and 32K+32K L1 per core. Total system memory is 24 GB provided by 8x4GB 1600 MHz DDR3 registered DIMMs.

All clustering was done using the SciPy and NumPy python libraries, along with matplotlib for visualization.

Each of the kernels, with the exception of UMTmk, had a single parameter to vary: number of OpenMP threads. So, the total set of all experiments is each kernel run with 1, 2, 4, 8, and 16 threads. In addition, some kernels had subsections or phases indicated by the software designer. Each phase was isolated using calipers, and totals for an application are the sums of all individually measured phases.

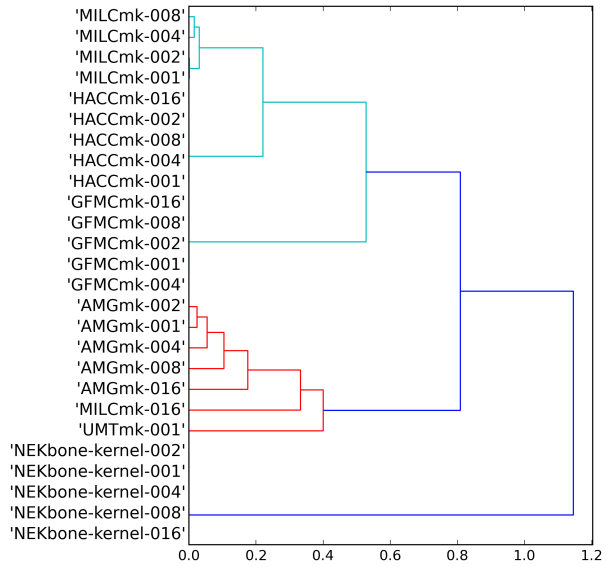

Fig. 1. Hierarchical Clustering: Type I Analysis of Micro-operation Mix

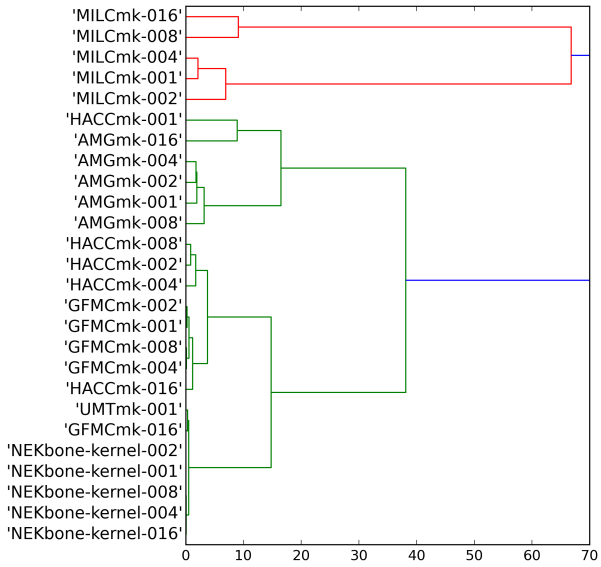

Fig. 2. Hierichal Clustering: Type M Analysis of DRAM Access

A. Kernel analysis using micro-operation mix, DRAM bandwidth, and a combination of both

Figure 1 shows the results of hierarchical clustering on the ratios of micro-operations from table II. Each experiment is titled with the name of the kernel and the number of threads used.

To read the dendrogram, note that experiments in the same subtree are more similar, and the height of each subtree indicates the level of difference. The subtree containing HACCmk at various threading levels has height of zero. This indicates that HACCmk instruction mix does not vary at all between threading levels. The same is true for GFMCmk and Nekbone-kernel. By contrast, the behavior for AMGmk changes at a regular rate, with each thread increase behaving most like the most similar thread count.

The notable result of Figure 1 is the placement of MILCmk with 16 threads. It is clearly very different in behavior from MILCmk with lower thread counts. When we examined the results from this experiment in more detail, we found that when MILCmk uses 2 physical processors, the threading behavior changes significantly, resulting in a much larger percentage of branch operations. By using clustering, we were able to avoid looking at the large number of numeric results of all experiments, in which we might have missed this change in behavior altogether.

Moving onto memory behavior, Figure 2 shows the results of clustering based on memory bandwidth. This figure is truncated for readability. The actual height of the root of all subtrees is located at distance 160. This means that MILCmk has significantly difference memory behavior than all other application. Further, the height of the topmost subtree indicates that MILCmk has more difference in memory behavior at 8 and 16 threads than all other applications. However, we do not

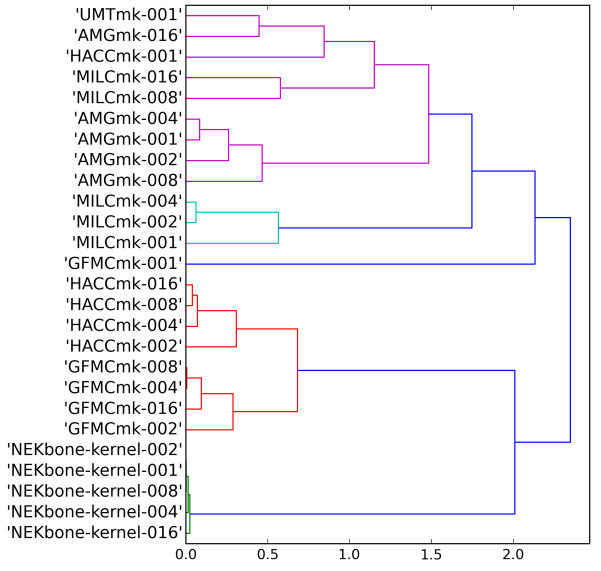

Fig. 3. Hierarchical Clustering: Type M-I Combined Tool Analysis

see a large change in memory access behavior at 16 threads in MILCmk, meaning that the previous observation of instruction mix changes at 16 threads do not carry over into memory access behavior.

Upon closer examination, these effects result from the size of the working set for MILCmk calculations increasing much faster than other applications in order to provide enough work for increased thread counts.

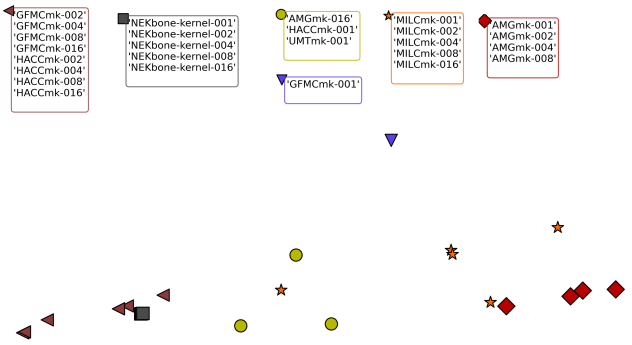

Fig. 4. Kmeans Clustering: Type M-I Combined Tool Analysis

Other notable experiments are HACCmk at 1 and 16 threads, which are dissimilar from other thread counts for that kernel. This results from shared write activity along the boundaries of thread data sets. In the case of 1 thread, there is no sharing, so data does not get pushed to DRAM as often as in the case of 2, 4, and 8 threads, which cause the shared L3 cache to eject data more often. In the case of 16 threads, the data sharing required pushing even more data into DRAM because there are two physical processors.

The final observation of Figure 2 is that the vertical ordering of applications has shifted overall. So, while Nekbone-kernel and AMGmk have very similar computational requirements, they have rather different memory requirements. The next figure will combine results to show which of these factors has a higher level of difference, i.e. does the memory or the computation similarity matter more?

Figure 3 combines the results of micro-operation mix and memory bandwidth. To answer the previous question, given our weighting algorithm, the differences in memory between AMGmk and Nekbone-kernel outweigh the similarities in computational requirements.

Overall, Figure 3 shows a good melding of the results between Figures 1 and 2, as would be expected. The surprise here is the change in relative position of UMTmk. Looking back at the previous figures, UMTmk showed similarities to MILCmk, GFMCmk, and Nekbone-kernel. However, Figure 3 shows that its computational similarities to AMGmk and MILCmk carry much more weight than its memory similarities to GFMCmk and Nekbone-kernel.

Figure 4 uses the same coordinates for experiments as Figure 3. In this case, the goal is to make 6 clusters using the k-means procedure described in section II. Under this analysis, we see that applications generally cluster together, independent of thread count. However, we see that several experiments fall into odd groupings. As in previous observations, there is a pattern of kernels tending to behave differently at single core counts and at 16 cores (two physical processors).

B. Analysis of kernel subphases

AMGmk and MILCmk kernels both contain subphases, as indicated by the software developers. All subphases were individually measured in all experiments. The previous figures showed results of the sum of all subphases.

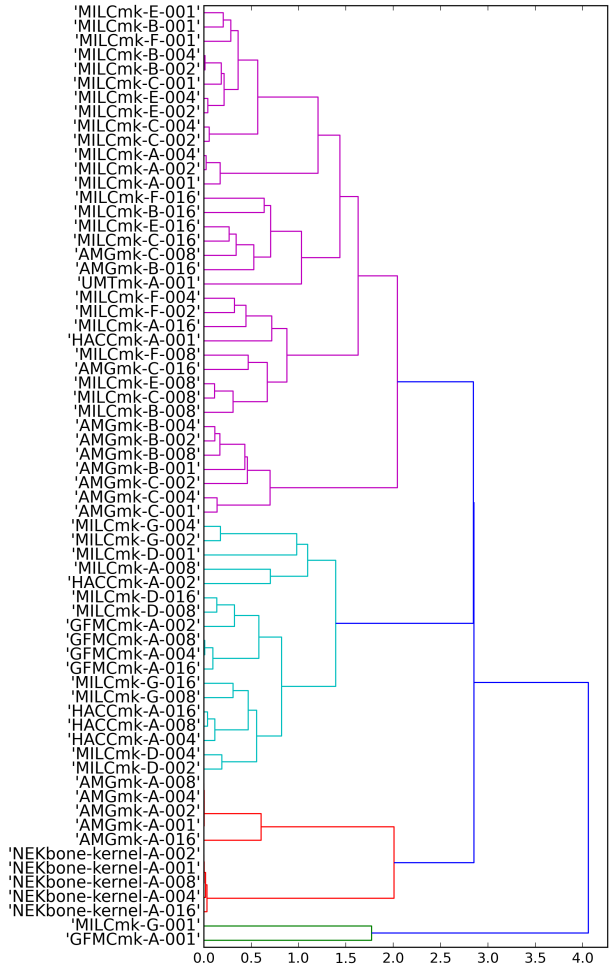

Fig. 5. Hierarchical Clustering of Subphases: Type M-I

Figure 5 shows the breakdown of all kernels into subphases. Rather than try to put section names, which made the figure unreadable, sections are numbered A, B, C... So each label is comprised of kernel name, section letter, and thread count. In all kernels other than AMGmk and MILCmk, there is only a section A.

In the section breakdown, we see that Nekbone-kernel is most similar to AMGmk section A. This section dominates the micro-operation mix metrics, which is why Nekbone-kernel appears most similar to AMGmk in Type I clustering. However, AMGmk sections B and C dominate the memory bandwidth metrics, which is why Nekbone-mk is dissimilar from AMGmk in Type M clustering. The section breakdown lets us know where to look to see why these applications appear similar in one clustering, and dissimilar in another.

The large number of sections of MILCmk makes it hard to process exactly the results from Figure 5. However, it appears

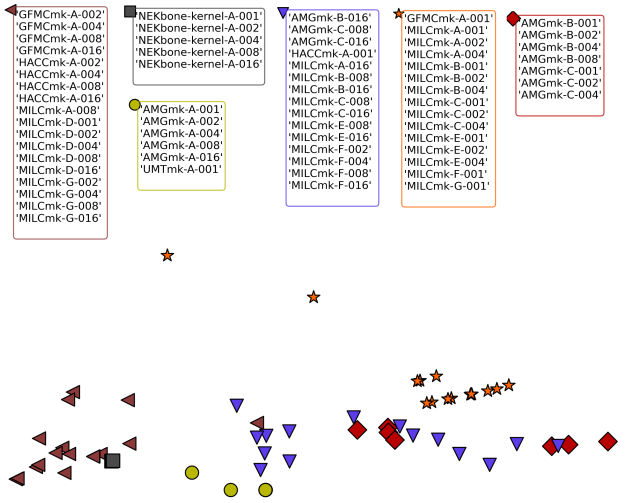

Fig. 6. Kmeans Clustering of Subphases: Type M-I

that for several sections, the thread count is a much better indicator of similarity than the subsection itself.

Figure 6 provides a more simple visualization in the form of 6 clusters. Here, it becomes more obvious that MILCmk sections A, B, C, E, and F have behavioral similarity determined by thread count, whereas section D and G are similar based on the phase. This figure also shows that AMGmk sections B and C are those which have the most variation due to thread count.

This kind of result would be relevant to a software designer attempting to understand how to write these algorithms such that they would perform well on many different system types. Or, on the other hand, this would be relevant to a system designer trying to understand why a certain application behaves unexpectedly at higher levels of parallelism. By being able to quickly identify which phases of an application are showing more change, the problem of studying application behavior can be simplified to studying the behavior of certain sections under certain conditions.

V. Conclusion

In this work, we have shown how the big-data analytical methods of hierarchical clustering and k-means analysis can be applied to the study of performance characterization of HPC computations.

By using the collected performance metrics as points in a multi-dimensional space, and experiments run under varying conditions as the individual observations, we were able to identify patterns of behavior across applications. In particular, these methods were fast way to identify the pattern of behavioral change when moving from 1 to 2 physical processors in a computational node. This type of pattern will be important to identify as the node architecture of extreme scale systems will become more complex, with multiple processors, accelerators, and deeper memory hierarchies.

We were also able to show that analysis of application sub-phases using clustering methods can show which phases dominate changes in behavior. This kind of analysis can provide software developers a faster path to identifying which parts of a code are the cause of some change in behavior. This will be an important corollary to the problem of identifying behaviors on more complex nodes in future extreme scale systems.

These methods are being integrated into our ongoing work building a results database of many scientific computations being run under varying conditions. In future, we will provide more sophisticated analytic methods and better visualization tools. Also, as this work has integrated results from multiple tools to give a unified picture of computational and memory behavior. Future work will integrate communication behavior and extend these methods into MPI parallelized codes on multiple computational nodes.

Acknowledgment

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan http://energy.gov/downloads/doe-public-access-plan. This research is sponsored by the Office of Advanced Scientific Computing Research in the U.S. Department of Energy.

References

[1] 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 *et al.*, "The landscape of parallel computing research: A view from berkeley," Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Tech. Rep., 2006.

[2] W. Joubert and S.-Q. Su, "An analysis of computational workloads for the ornl jaguar system," in *Proceedings of the 26th ACM international conference on Supercomputing*. ACM, 2012, pp. 247–256.

[3] S. Sreepathi, M. L. Grodowitz, R. Lim, P. Taffet, P. C. Roth, J. Meredith, S. Lee, D. Li, and J. Vetter, "Application Characterization Using Oxbow Toolkit and PADS Infrastructure," in *Proceedings of the 1st International Workshop on Hardware-Software Co-Design for High Performance Computing*, ser. Co-HPC '14. IEEE Press, 2014, pp. 55–63. [Online]. Available: http://dx.doi.org/10.1109/Co-HPC.2014.11

[4] P. Balaprakash, D. Buntinas, A. Chan, A. Guha, R. Gupta, S. H. K. Narayanan, A. A. Chien, P. Hovland, and B. Norris, "Exascale workload characterization and architecture implications," in *Proceedings of the High Performance Computing Symposium*, ser. HPC '13. San Diego, CA, USA: Society for Computer Simulation International, 2013, pp. 5:1–5:8. [Online]. Available: http://dl.acm.org/citation.cfm?id=2499968.2499973

[5] CORAL Collaboration, "CORAL Request for Proposal B604142," February 2014. [Online]. Available: https://asc.llnl.gov/CORAL/

[6] M. Bach, M. Charney, R. Cohn, E. Demikhovsky, T. Devor, K. Hazelwood, A. Jaleel, C.-K. Luk, G. Lyons, H. Patil *et al.*, "Analyzing parallel programs with pin," *Computer*, vol. 43, no. 3, pp. 34–41, 2010.

[7] M. Drout and L. Smith, "How to read a dendrogram?" http://wheatoncollege.edu/lexomics/files/2012/08/How-to-Read-a-Dendrogram-Web-Ready.pdf, 2014.