Application Characterization using Oxbow Toolkit and PADS Infrastructure

Sarat Sreepathi*, M. L. Grodowitz*, Robert Lim†, Philip Taffet‡
Philip C. Roth*, Jeremy Meredith*, Seyong Lee*, Dong Li*, Jeffrey Vetter*
*Oak Ridge National Laboratory, Oak Ridge, TN, USA
Email: {sarat,grodowitzml,rothpc,jsmeredith,lees2,lid1,vetter}@ornl.gov
†University of Oregon, Eugene, OR, USA
Email: roblim1@cs.uoregon.edu
‡Rice University, Houston, TX, USA
Email: ptaffet@rice.edu

***Abstract*—Characterizing the behavior of a scientific application and its associated proxy application is essential for determining whether the proxy application actually does mimic the full application. To support our ongoing characterization activities, we have developed the Oxbow toolkit and an associated data store infrastructure for collecting, storing, and querying this characterization information. This paper presents recent updates to the Oxbow toolkit and introduces the Oxbow project's Performance Analytics Data Store (PADS). To demonstrate the possible insights when using the toolkit and data store, we compare the characterizations of several full and proxy applications, along with the High Performance Linpack (HPL) and High Performance Conjugate Gradient (HPCG) benchmarks. Using techniques such as cluster visualizations of PADS data across many experiments, we found that the results show unexpected similarities and differences between proxy applications, and a greater similarity of proxy applications to HPCG than to HPL along many dimensions.**

I. Introduction

Roadmaps for extreme scale high performance computing (HPC) systems [1], [2] forecast significant changes to system architectures, forcing dramatic changes to HPC software in order to achieve good performance and power efficiency. To make these changes efficiently and effectively, the behavior of applications that will run on future extreme scale systems must be understood at a deeper level than is possible using only traditional metrics such as execution speed. With better understanding of their applications' behaviors, application designers can provide more precise statements of their applications' computation and communication requirements, and identify new ways to improve execution along dimensions of performance, power, and reliability. With better information about application requirements, extreme scale system designers can tailor their designs to better satisfy the applications' requirements. This co-design of hardware and software has the potential for substantial improvements in scientific computing productivity.

To support the characterization of scientific applications, we have developed the **Oxbow** toolkit and Performance Analytics Data Store (**PADS**). The Oxbow toolkit provides a variety of mechanisms to collect information about applications' static and dynamic characteristics, and about the systems on which they execute. PADS provides an infrastructure for tracking this information across different experiments, over changes in application versions, and between systems. PADS also provides a web-based visualization portal accessible at oxbow.ornl.gov by following the "Portal" link at the top of the page. Many figures presented in this paper were generated using this web interface.

To focus on specific application-system interactions and to control access to applications from sensitive domains, HPC co-design researchers often develop "proxy" applications that mimic real application behavior along one or more dimensions, but use a smaller code base [3]. Since its inception, a primary goal of the Oxbow project has been to study how well proxy applications mimic their respective full applications in practice [4].

In this paper, we provide new insights into current co-design proxy applications produced by the various U.S. Department of Energy (DOE) Co-design centers. We also contextualize these proxy applications by comparing their behavior to other benchmarks, micro-kernels, and full applications of interest to DOE and other extreme scale scientific computing centers. Table I shows the list of full applications and proxy applications which have been profiled and uploaded to our data store.

The rest of the paper is organized as follows. After discussing some existing work in application characterization in Section II, we detail the Oxbow project tools, metrics used for our characterization studies, the design of the PADS infrastructure, and the expected workflow of the toolchain in Section III. Section IV presents our characterizations of

TABLE I: Profiled applications grouped by co-design set

Source	Benchmark
CESAR	XSBench, RSBench, NEKBONE, NEKmk
ExaCT	Exp_CNS_NoSpec, Multigrid_C, vodeDriver
ExMatEx	LULESH
LLNL ASC	AMG2013, AMGmk, MCB, UMT2013, UMTmk
Mantevo	miniAMR, miniFE
Full Application	LAMMPS, eavl, VisIt, Nek5000, QMCPack
Other Benchmark	HPL, HPCG, KMI Hash, GFMCmk, HACCmk

978-1-4799-7564-8/14 $31.00 © 2014 IEEE
DOI 10.1109/Co-HPC.2014.11

computation and communication behavior of the proxy applications, microkernels, and benchmarks listed in Table I. In Section V, we summarize the findings of our application characterization studies and outline some directions for future work with PADS and the Oxbow toolkit.

II. Previous & Related Work

In our earlier study using Oxbow [4], we reported on characterizations of several co-design applications. This work extends our previous work with new characterization results, a description of new analysis techniques such as clustering of application characterization data, and a description of our new data storage infrastructure and its web-based interface.

The MIAMI Toolkit [5] used in previous Oxbow work has been augmented and has become an independent software package. As in our earlier Oxbow work, we use the Oxbow version of MIAMI for studying an application's dynamic instruction mix. Like other Oxbow tools (see Section III), our version of MIAMI is integrated into the Oxbow tools build system. It is integrated into the common Oxbow execution and upload infrastructure.

The Tuning and Analysis Utilities (TAU) [6] has a performance database [7] which is tightly coupled to the TAU ecosystem. TAU provides in-depth application analysis. Our work is complementary to this effort, focusing on extracting high level inter-application comparisons from a combination of experimental metrics.

Similarly, HPCToolkit [8] provides data storage and visualization on a local system for the purpose of in-depth analysis. In contrast, the Oxbow data store is intended to be shared and available over the Internet to multiple users.

Scalasca [9] collects large amounts of communication data and can analyze this data to automatically identify communication-related performance problems. In contrast, the mpiP library used in Oxbow is a lightweight profiling tool used to generate statistics and communication topology data. In Oxbow, we use this communication data to compare communication patterns and data transfer volumes across different applications and their input sets.

Other approaches to understand application characteristics require running applications on various different platforms and simulations to compare performance metrics [10], [11]. Our work complements such efforts by 1) providing architecture independent characterization and 2) maintaining experimental platform attributes in our data store to correlate performance results with specific systems.

Compiler based tools, such as PALM [12] and Byfl [13], can generate application models from source code. These require specific compiler toolchains. Oxbow works with a wide range of program development toolchains and includes support for the profiling of pre-compiled system libraries linked to an application's executable program.

III. Oxbow Project Methodology

Figure 1 shows the workflow used to perform the study presented in this paper. First, the Oxbow toolkit is installed on the target compute platform. Some number of applications

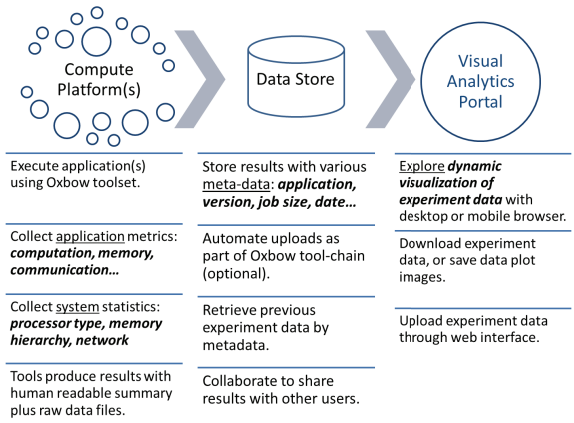


Fig. 1: Overall Oxbow + PADS Workflow

are compiled and executed using a variety of tools to profile computation, communication, and memory behavior. In addition to runtime metrics, we collect source code characteristics that are correlated with any experiments using that source. The system itself is also profiled to determine hardware features.

Once data has been collected on the target platform, it is uploaded to the data store. When data is uploaded, it is tagged with meta-data to allow sorting, analysis, and retrieval of experiments by information such as date, application version, or other user defined naming information.

Any data in the data store will be dynamically visualized through the web portal. Users can explore their data and compare it with data from other applications. The web portal can also be used to upload and download experiment data through a browser.

A. Application Characterization

Oxbow can be used to evaluate application with a variety of different metrics using different tools and methods. This flexibility is useful when studying co-design applications that are meant to model only a specific behavior of a parent application. For example, one would not need to examine communication data for a proxy application that is only modeling memory access patterns.

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

Computation Profiling: The computational profile of an application execution is described by the mix of executed micro-operations. These are not instruction counts, which would be specific to a given ISA, but the breakdown of x86 instructions into their components.

The decoding and breaking down of instructions is done using the MIAMI toolkit [5]. MIAMI is an extensible set of tools built on top of the PIN dynamic binary instrumentation tools from Intel [14]. The details of how instructions are decoded and analyzed is described previously in [4].

Micro-operations are grouped into coarser categories which are listed in table II. At a high level, operations are grouped into memory, control, and arithmetic. The categorization is based on which type of microarchitectural components are needed to support each type. Based on the categorization we are able to say, for example, whether a code is using vector arithmetic units.

Communication Analysis: Since we are targeting current parallel scientific applications, communication analysis is done by analyzing the application use of the MPI library. Oxbow uses version 3.3 of the mpiP lightweight MPI profiling library. The Oxbow version of mpiP includes several not-yet-released updates to this library to collect data about all point-to-point and collective communication between MPI ranks in a computation.

The volume of data transferred between ranks is output in an adjacency matrix. This can be visualized to see the communication topology for an application run. For example, a perfect symmetry across the diagonal represents a zero-noise communication pattern common to many benchmarks, whereas real applications may display more asymmetrical communication between ranks.

Memory Behavior Measurement: Oxbow includes several metrics to measure memory behavior, including bandwidth and a reuse distance metric to estimate data locality. These metrics are not used in this work, but are detailed here [4] and on the Oxbow web site.

Source Code Analysis: A recently added feature allows Oxbow to perform static analysis on application sources. Given a set of source files, the toolkit will determine languages used, lines of code, types of parallelism, number of functions and cyclomatic complexity. This is collected once for a source tree, and will be correlated to any experiments using that source.

Software cyclomatic complexity uses the constructed control flow graph of a software module to derive a measure of the amount of decision logic required by the software. Cyclomatic complexity is defined by McCabe et al. [15] for each module to be $e - n + 2$, where e and n are the number of edges and nodes in the control flow graph, respectively.

B. System Statistics Collection

During install and configuration, Oxbow probes a target compute architecture to detect hardware features. The analysis is done using low-level Linux probes that are available on most HPC clusters. Currently, the analysis uses `/proc/cpu`, `cpuid` and `lshw` calls, which are available on most HPC Linux cluster environments.

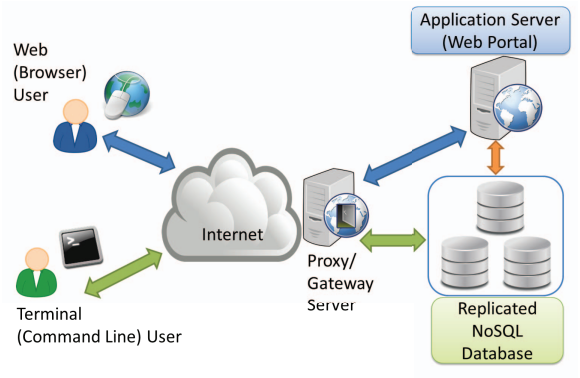


Fig. 2: PADS Architecture

Oxbow also includes a third-party tool, `hwloc` [16], that generates an abstract diagram of architectural topology including cache size and structure. Table III shows the information collected for our experimental platform, which is displayed through our web portal.

In addition to maintaining provenance information for experiments run on each platform, system analysis generates a machine description file for the MIAMI toolkit. Previously, this file had to be hand-generated by consulting a manual for each system to determine hardware features such as cache hierarchy and specific instruction set support.

C. Performance Analytics Data Store (PADS) Design Goals

Between system profiles, code profiles, and experimental results, Oxbow produces a large amount of data to be stored, reviewed, and correlated. To handle this task, we have deployed the PADS (Performance Analytics Data Store) with associated web portal that are tied to the Oxbow toolkit. We have identified several important goals to be addressed by this data store architecture to allow usability and future extensibility.

Enable New Insights: The primary goal of an online data store is to collect together many different experimental results in a common location for the purpose of better understanding applications. An example of this use is the hierarchical/agglomerative clustering analysis described in section III-F.

Flexible Data Model: The data store should accommodate diverse data formats that emanate from performance tools. The data model needs to account for evolution of schema as tool outputs change as well as addition of new tools and metrics.

Transparency: An open access model allows domain scientists, computer scientists and other stakeholders to collaborate effectively. Researchers typically strive to publish relevant experiment data as an addendum to their technical publications. Although such mechanisms satisfy the transparency goal, such data is not easily accessible for programmatic analysis.

Accessibility: The PADS data infrastructure was designed to facilitate easy access by a variety of clients. It supports use cases ranging from summary reports meant for human consumption to programmer friendly APIs for custom analysis of raw data.

D. PADS Architecture

The PADS architecture is shown in Figure 2. PADS is built on MongoDB [17], a well established, open-source, NoSQL solution to provide a flexible data model. MongoDB uses a schema-less database design. Software logic ensures that the state of the data model at any point in time is kept in sync with the application logic that powers the PADS web portal and other database tools. Changes in the data model are expected and handled gracefully, without invalidating older stored data.

Failover, Monitoring & Backup: PADS is deployed as a replicated database configuration to provide failover capability with one primary and two secondary databases. If the primary goes down, one of the secondary databases takes over ensuring continued database function. Continuous monitoring of the database and application server infrastructure is put in place for performing usage analytics, understanding load characteristics as well as immediate notification of any problems. Automated nightly backups of the entire database to on-site attached network storage allows recovery from catastrophic hardware failure.

Web Portal: A visual analytics portal allows users to interactively explore the performance data in PADS. The portal is deployed on an application server and utilizes Python web technologies to power the backend functionality, and database connectivity is provided through a lighweight REST API [18]. Modern client-side web technologies including JavaScript, AJAX, Bootstrap [19], jqPlot [20] and HTML5 allow for a rich web experience that supports desktop and mobile browser access.

The replicated database and application server powering the web portal are both behind a site-wide firewall and shielded from direct Internet access by a gateway server.

E. Potential Use Cases

The separation of concerns between toolkit, database, and web portal allows users varying levels of effort and control. A user can use default tool configurations and browser uploads to collect information from a desktop and explore results online. With the same infrastructure, a user can become very familiar with the raw data output from a given tool, and use command line access to store and retrieve experiment results that they analyze with their own custom toolchain.

Online availability also allows remote collaborations. Consider the scenario where a domain scientist working with a particular application is the best person to choose input parameters for an application. Vendors or systems researchers might want to study that application, but might not choose the most representative configurations for experiments. The ability to easily compare expected results in an open data store would reduce the chance of erroneous results or wasted efforts.

TABLE III: Platform information

Attribute	Value
Model Name	Intel(R) Xeon(R) CPU X5660 @ 2.80GHz
Core Speed	2800 MHZ
Cores	6 cores
L1 data (per core)	32KB
L1 instruction (per core)	32KB
L2 (per core)	256KB
L3 (shared)	12MB

F. Multivariate Application Clustering

With the growing amount of results in the PADS data store, making comparisons with traditional scatter or line plots quickly becomes overwhelming. We use a hierarchical/agglomerative clustering approach to more easily identify similarities across different applications using any number of normalized metrics.

A notable advantage of this approach over similar techniques like k-means clustering is the ability to omit explicit specification of the number of clusters *a priori*. Hence, the hierarchical clustering analysis is more amenable to identification of the natural number of clusters in the given dataset.

The algorithm begins with a forest of clusters that will finally form a hierarchical tree. At the outset, each data point forms its own cluster. When two clusters a, b from this forest are combined into a single cluster c, a and b are removed from the forest, and c is added to the forest. When only one cluster remains in the forest, the algorithm stops, and this cluster becomes the root.

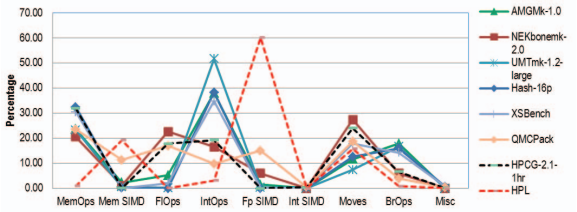
The output of this process is a dendogram [21] tree image that groups applications by similarity and indicates a measure of similarity between application groupings.

IV. Application Insights

This section presents computational and communication patterns of proxy applications in various configurations in comparison to top supercomputing benchmarks – HPCG and HPL.

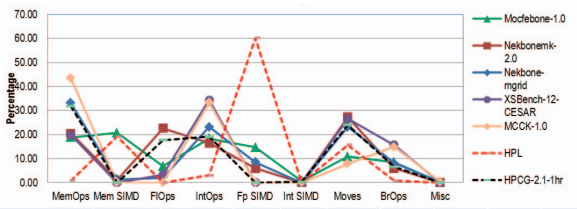
A. Comparison of HPCG, HPL, and Proxy Applications

Figure 3 shows a comparison of micro-operation ratios for High Performance Linpack (HPL), version 2.1 of High



	MemOps	Mem SIMD	FlOps	IntOps	Fp SIMD	Int SIMD	Moves	BrOps	Misc
AMGMk-1.0	23.58	2.25	5.16	38.10	1.45	0.00	11.65	17.81	0.00
NEKbonemk-2.0	20.55	1.09	22.61	16.53	5.94	0.00	27.36	5.93	0.00
UMTmk-1.2-large	23.22	0.50	0.24	51.69	0.37	0.03	7.33	16.62	0.01
Hash-16p	32.38	0.08	0.00	38.31	0.00	0.02	12.63	15.92	0.64
XSBench	30.48	0.00	2.02	34.75	0.00	0.00	17.82	14.27	0.66
QMCPack	23.50	11.34	16.92	9.76	15.01	0.56	18.70	3.94	0.28
HPL	0.9	19.2	0.1	3.1	60.2	0	15.7	0.8	0
HPCG-2.1-1hr	31.85	0.040	17.76	19.23	0.002	0.40	24.034	6.62	0.036

Fig. 3: Mix of Instruction Micro-operations for HPCG, HPL, microkernel and proxy applications



	MemOps	Mem SIMD	FlOps	IntOps	Fp SIMD	Int SIMD	Moves	BrOps	Misc
Mocfebone-1.0	18.90	20.76	6.82	18.39	14.72	0.75	10.86	8.51	0.29
Nekbonemk-2.0	20.55	1.09	22.61	16.53	5.94	0.00	27.36	5.93	0.00
Nekbone-mgrid	33.13	1.44	2.07	23.20	8.59	0.00	23.24	8.20	0.14
XSBench-12-CESAR	19.81	0.00	3.44	34.41	0.00	0.00	26.60	15.70	0.04
MCCK-1.0	43.48	0.00	0.03	33.39	0.01	0.00	7.86	14.89	0.34
HPL	0.9	19.2	0.1	3.1	60.2	0	15.7	0.8	0
HPCG-2.1-1hr	31.86	0.04	17.77	19.24	0.00	0.40	24.03	6.62	0.04

Fig. 4: Mix of Instruction Micro-operations for CESAR proxy applications and HPCG

Performance Conjugate Gradient (HPCG), three microkernel benchmarks (denoted with mk), and three proxy applications from various areas. These selections were made to show a large range of application types for comparison.

Figure 4 compares the proxy applications from the CESAR co-design center with HPCG and HPL. Each application instruction counts are represented as percentages. This normalizes the instruction counts for each application to a set of real numbers that sum to 100.

These comparisons were made to compare both the traditional and new top500 benchmarks with the behavior of proxy applications and kernels. Benchmarks like HPL and HPCG have typically influenced HPC system design decisions. A reason given for launching an alternative HPCG-based top500 list is to generate rankings based on more realistic scientific application behavior.

The results we collected for instruction mix show that the HPL results are very different than all of the other experiments. Even the microkernels, which isolate computation from other phases, show at most 15% floating point SIMD instructions. HPL shows 60% floating point SIMD instructions due to the efficient vectorization the very regular computation particular to this type of dense linear solver.

By comparison, HPCG presents a much more representative mix of computational requirements. In each category, it falls somewhere between the peaks of highs and lows, with average amounts of floating point, integer, memory, and control. It does not currently present a large percentage of SIMD operations, but we expect this to change as it becomes more optimized over time.

Though we made these comparisons to investigate behavior of proxy applications and benchmarks, other observations can be made using the same data set.

A surprising secondary result here is the similarity between the profiles of KMI-Hash and UMTmk. KMI Hash is a data-centric application benchmark that represents genome assembly and mapping analysis. UMTmk is the most CPU intensive kernel from the UMT radiation transport proxy application. Based on this similarity, CPU resource requirements of compute intensive sections of KMI Hash are represented by the UMT microkernel.

B. Application Clustering: Instruction Mix

Line graphs of micro-operation mixes were shown in section IV-A to compare applications. These graphs become hard to read with a relatively small number of results. To generate a more rich diagram of application similarity, we used a hierarchical/agglomerative clustering over all instruction categories.

This type of clustering uses a distance metric to measure the difference between each experiment. A tree is built iteratively. At every iteration, each cluster is joined together with the nearest neighboring clusters, resulting in the final relational hierarchy. In general, a distance metric for this type of clustering can be any function.

In this case, we defined distance as the euclidean distance in 9-dimensional space between points at the coordinates defined by the percentage-wise instruction mix of an experiment.

The following equation defines our distance metric between two experiments, α and β. Each experiment has a set of instruction percentages, I, which are a set of numbers that sum to 100, and represent the categories of micro-operations defined in table II.

$$\sqrt{\sum_{i_\alpha \in I_\alpha, i_\beta \in I_\beta} (i_\alpha - i_\beta)^2}$$

Figure 5 shows the results of listed along the axes of previous figures and in table II. The leaves of the dendogram represent data points, and are labeled with a string concatenation of application name and experiment name (a user defined value).

A user defined cut-off threshold truncates leaves below certain level to aggregate into a finite number of clusters. Figure 5 shows a cutoff threshold of the euclidean distance 62 (vertical line at 62 on the x-axis) to identify nine clusters.

Leaves under a common branch are more similar to each other than leaves under a different branch. The vertical branching points represent a relative degree of difference between branches. For example, clusters 7 and 8 show more similarity to each other than they do to cluster 9. Drout and Smith [21] provide a good introduction to dendrogram interpretation.

In Fig. 5, it is interesting to note that HPCG and VisIt belong to the same cluster and mostly differ in the percentage of Int SIMD and FlOps instructions. Moreover, VisIt is an outlier among the instruction mix profiles wherein it exhibits 10.26% Int SIMD instructions and the next highest is HACCmk with 2.18% Int SIMD instructions. In this specific problem configuration, VisIt performed an isosurface operation.

The data store contains a large number of different experiments with different goals. Some users look for different configurations of thread count, as in the HACCmk:N_threads experiments. Other users look at different problem inputs, as in the miniAMR experiments. Combining this jumble of data together lets someone search freely for correlations.

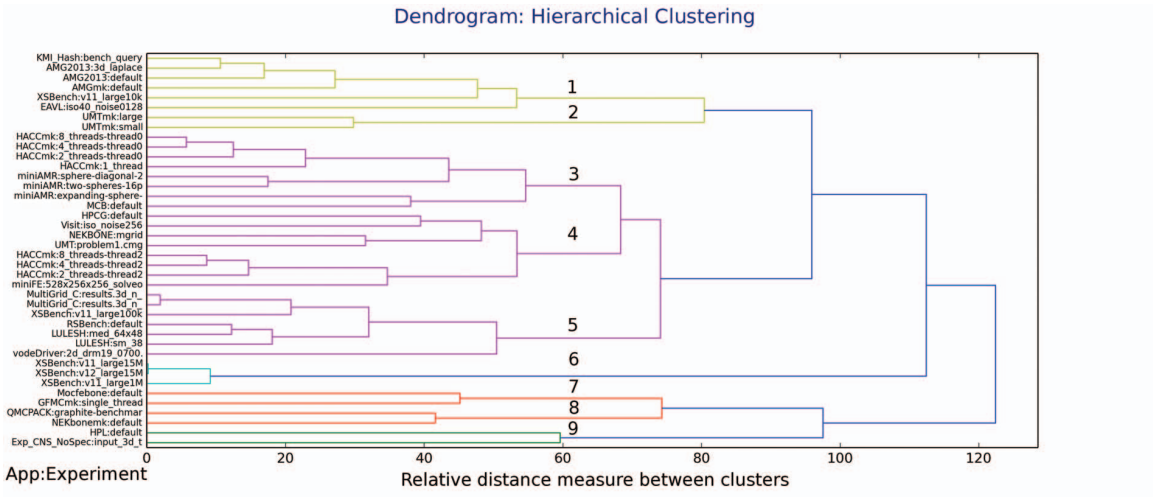


Fig. 5: Hierarchical Clustering Results. The numbers indicates the cluster assignment if we aggregate leaves in the same sub-tree.

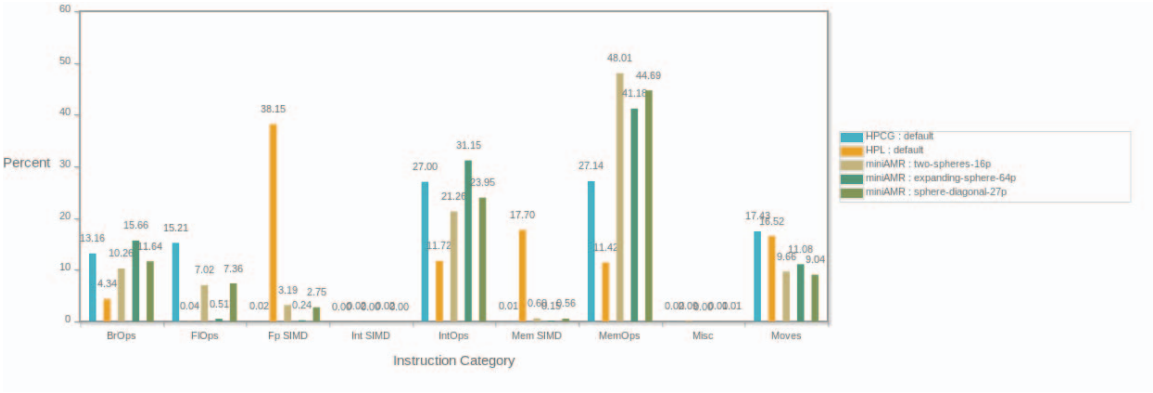


Fig. 6: Instruction Mix of Micro-operations for HPCG, HPL, and three configurations of miniAMR

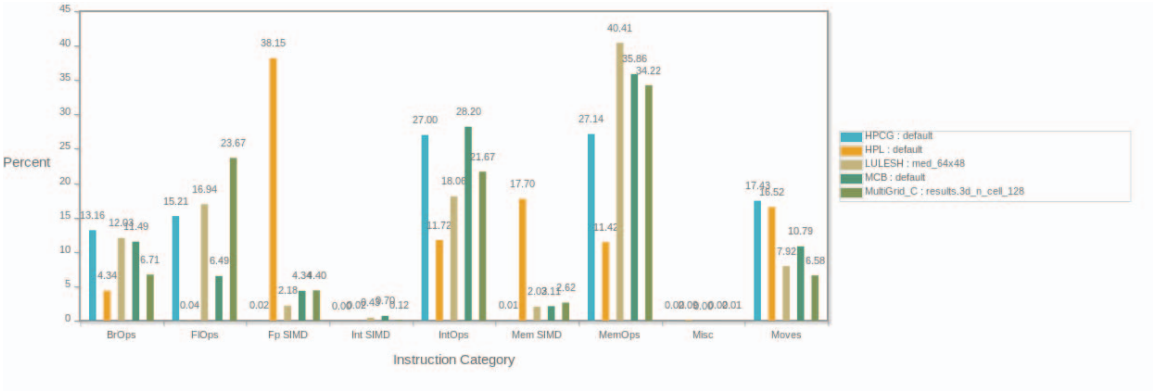


Fig. 7: Instruction Mix of Micro-operations for HPCG, HPL, and three proxy applications

For example, the results of running miniAMR with the three different recommended problem types are close together, as expected, but the difference in length between the subtree for expanding sphere and the other two problem types is rather large.

Using the web portal, those experiments are selected along with HPL and HPCG, and the image in figure 6 is dynamically generated. It is clear that the expanding sphere experiment has very low floating point operations and less memory access than the other two scenarios. We also note that all three scenarios have many more memory operations than either HPCG or HPL. This is all accomplished in less than 10 minutes of using the web portal, rather than generating graphs by hand until something interesting appears.

C. Communication Behavior

When visiting the communication page of the web portal, any uploaded mpiP communication volume data are visualized as heat maps such as those shown in figure 8. The images on the left include mouse-over data and a boxplot of data range as it appears on the web portal.

The communication volume graphs of HPL (Fig. 8c and HPCG (Fig. 8a) show that the communication patterns of these two benchmarks are very different. HPL has communication distributed evenly throughout all ranks. Hovering the mouse over (55,15) shows that large amounts of data are being communicated, whereas no other application here shows any communication between those ranks.

Again, HPL shows heavily optimized, atypical behavior. In this case, it would favor an architecture that provided a large amount of point-to-point bandwidth over the entire machine. By contrast, the rest of the applications here, including HPCG have hierarchical communication that would favor a system with higher bandwidth between nodes hosting neighboring ranks.

LULESH (Fig. 8b) in particular appears nearly identical to HPCG in terms of communication. According to the LULESH documentation, it is not meant to be very interesting in terms of communication, and should scale its stencil computation easily to very large machines. HPCG has a similar goal, but stresses both computation and memory, as bottlenecks, over communication.

A different pattern emerges in miniAMR (Fig. 8e) and MCB (Fig. 8f), which darken toward the origin, showing an all-to-one reduction that funnels data to rank 1. Running the mouse cursor along the bottom of the x-axis of miniAMR shows each rank sending data to rank 1, with a periodic darkening indicating a tree reduction. This type of communication is not represented by HPCG. MCB is meant to test OpenMP+MPI scalability, and tests it in a very different way than HPCG.

Having found these patterns in the communication data, it is easy to generate the micro-operation chart in figure 7 from the data store. While LULESH and HPCG have the same communication structure, they differ in processing requirements. In particular, LULESH is more memory operation intensive than HPCG.

Finally, an application's communication behavior is highly dependent on the problem domain and specific experiment configuration. For instance, there is notable change in communication behavior for miniAMR for different experiments (Fig. 9).

Fig. 9: Communication behavior of miniAMR with different problem configurations (two moving spheres:16 ranks, expanding sphere:64 ranks, sphere moving diagonally:27 ranks)

D. Using Oxbow in Co-design efforts

Recently, this infrastructure resulted in an improvement to a proxy application. In this case, we collected results from an application, which resulted in communication patterns which were very different than expected. The original team was contacted, and they reviewed the results through the web portal at their home location.

As a result, the application was redistributed to us with an improved test input parameter set and build system changes to drive the behavior toward the expected application behavior.

The real contribution of this work is the ability to "close the loop" in a codesign effort, by providing easily accessible, high level views of application behavior.

V. Conclusions & Future Work

The Oxbow toolkit and data store infrastructure is currently undergoing testing by friendly users. An upcoming public release will make the software available to the broader co-design community.

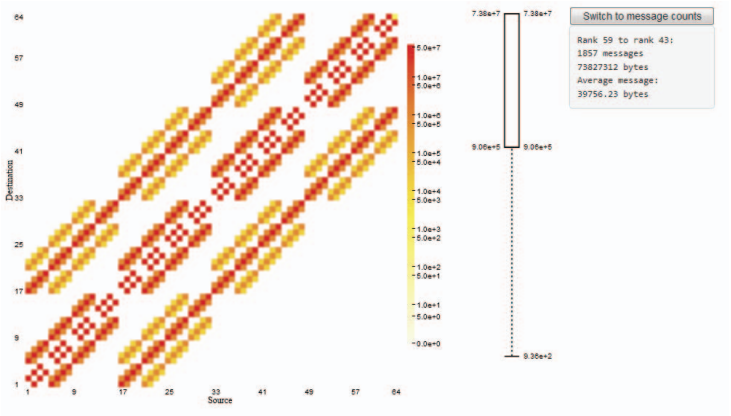
We have shown that even with large number of experiments, the web portal and data store allow a user to easily identify interesting correlations. Having found interesting sets of experiments, the user can drill-down into those details.

We propose this as a platform for information dissemination and a vehicle for collaboration between domain scientists, applied mathematicians, computer scientists, and hardware architects.

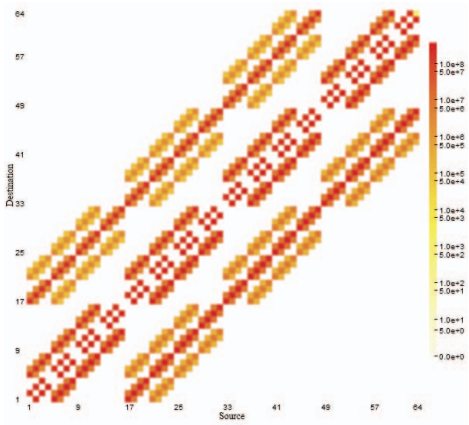
Acknowledgment

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research.

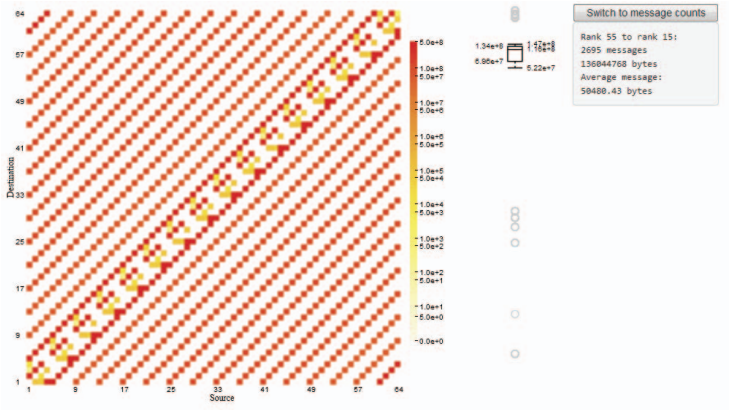
This material presents work performed by Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC under Contract #DE-AC05-00OR22725 to the U.S. Government. Accordingly, the U.S. Government retains a non-exclusive, royalty-free license to publish or reproduce the published form of this contribution, or allow others to do so, for U.S. Government purposes.



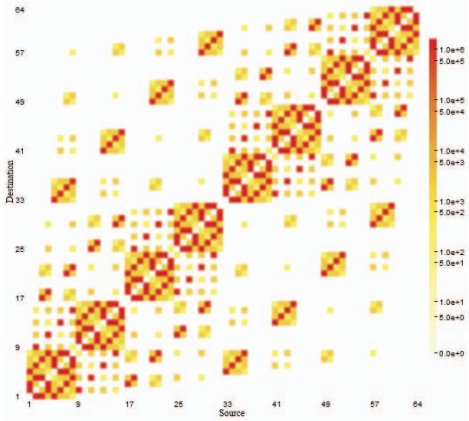
(a) HPCG communication volume with boxplot and mouse-over data



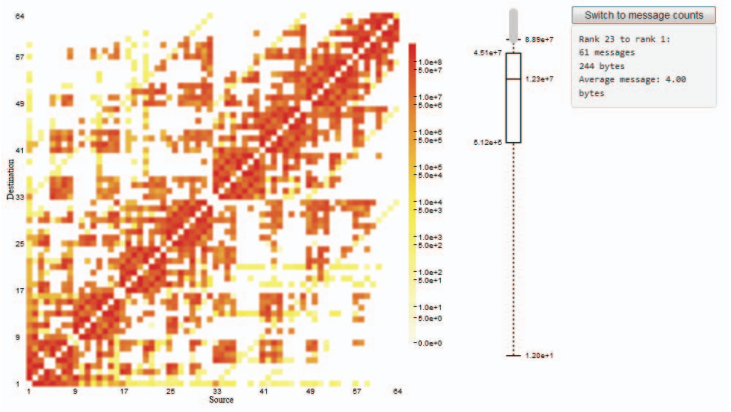
(b) LULESH communication volume



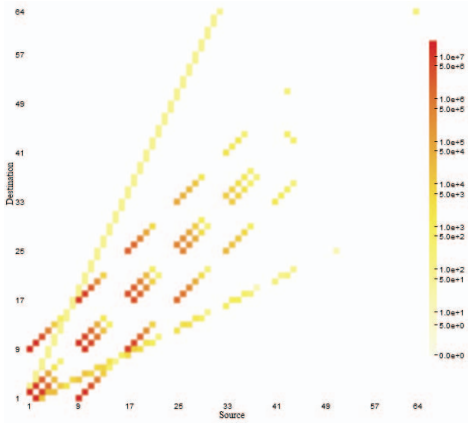
(c) HPL communication volume with boxplot and mouse-over data



(d) Multigrid_C communication volume



(e) miniAMR:expanding-sphere communication volume with boxplot and mouse-over data



(f) MCB communication volume

Fig. 8: Communication Volume for 64 rank executions of two benchmarks and two proxy applications

REFERENCES

[1] J. Dongarra, P. Beckman, T. Moore, P. Aerts, G. Aloisio, J.-C. Andre, D. Barkai, J.-Y. Berthou, T. Boku, B. Braunschweig, F. Cappello, B. Chapman, X. Chi, A. Choudhary, S. Dosanjh, T. Dunning, S. Fiore, A. Geist, B. Gropp, R. Harrison, M. Hereld, M. Heroux, A. Hoisie, K. Hotta, Z. Jin, Y. Ishikawa, F. Johnson, S. Kale, R. Kenway, D. Keyes, B. Kramer, J. Labarta, A. Lichnewsky, T. Lippert, B. Lucas, B. Maccabe, S. Matsuoka, P. Messina, P. Michielse, B. Mohr, M. S. Mueller, W. E. Nagel, H. Nakashima, M. E. Papka, D. Reed, M. Sato, E. Seidel, J. Shalf, D. Skinner, M. Snir, T. Sterling, R. Stevens, F. Streitz, B. Sugar, S. Sumimoto, W. Tang, J. Taylor, R. Thakur, A. Trefethen, M. Valero, A. van der Steen, J. Vetter, P. Williams, R. Wisniewski, and K. Yelick, "The international exascale software project roadmap," *International Journal of High Performance Computing Applications*, vol. 25, no. 1, pp. 3–60, 2011.

[2] P. Kogge, K. Bergman, S. Borkar, D. Campbell, W. Carlson, W. Dally, M. Denneau, P. Franzon, W. Harrod, K. Hill, J. Hiller, S. Karp, S. Keckler, D. Klein, R. Lucas, M. Richards, A. Scarpelli, S. Scott, A. Snavely, T. Sterling, R. S. Williams, and K. Yelick, "Exascale computing study: Technology challenges in achieving exascale systems," DARPA Information Processing Techniques Office, Tech. Rep., 2008.

[3] R. F. Barrett, S. Borkar, S. S. Dosanjh, S. D. Hammond, M. A. Heroux, X. S. Hu, J. Luitjens, S. G. Parker, J. Shalf, and L. Tang, "On the Role of Co-design in High Performance Computing," vol. 24, pp. 141 – 155.

[4] J. S. Vetter, S. Lee, D. Li, G. Marin, C. McCurdy, J. Meredith, P. C. Roth, and K. Spafford, "Quantifying Architectural Requirements of Contemporary Extreme-Scale Scientific Applications," in *International Workshop on Performance Modeling, Benchmarking and Simulation of HPC Systems (PMBS13)*, Denver, CO, 2013.

[5] G. Marin, J. Dongarra, and D. Terpstra, "MIAMI: A framework for application performance diagnosis," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2014.

[6] S. S. Shende and A. D. Malony, "The TAU parallel performance system," *International Journal of High Performance Computing Applications*, vol. 20, no. 2, pp. 287–311, 2006.

[7] K. A. Huck, A. D. Malony, R. Bell, and A. Morris, "Design and implementation of a parallel performance data management framework," in *Parallel Processing, 2005. ICPP 2005. International Conference on*. IEEE, 2005, pp. 473–482.

[8] L. Adhianto, S. Banerjee, M. Fagan, M. Krentel, G. Marin, J. Mellor-Crummey, and N. R. Tallent, "HPCToolkit: Tools for performance analysis of optimized parallel programs," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 685–701, 2010.

[9] M. Geimer, F. Wolf, B. J. Wylie, E. Ábrahám, D. Becker, and B. Mohr, "The Scalasca performance toolset architecture," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 6, pp. 702–719, 2010.

[10] C. Vaughan, M. Rajan, R. Barrett, D. Doerfler, and K. Pedretti, "Investigating the impact of the Cielo Cray XE6 architecture on scientific application codes," in *Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on*. IEEE, 2011, pp. 1831–1837.

[11] S. Dosanjh, R. Barrett, D. Doerfler, S. Hammond, K. Hemmert, M. Heroux, P. Lin, K. Pedretti, A. Rodrigues, T. Trucano *et al.*, "Exascale design space exploration and co-design," *Future Generation Computer Systems*, vol. 30, pp. 46–58, 2014.

[12] N. R. Tallent and A. Hoisie, "Palm: easing the burden of analytical performance modeling," in *Proceedings of the 28th ACM international conference on Supercomputing*. ACM, 2014, pp. 221–230.

[13] S. Pakin, "Byfl: Analysis of low-level application characteristics," Los Alamos National Laboratory (LANL), Tech. Rep., 2011.

[14] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, ser. PLDI '05. New York, NY, USA: ACM, 2005, pp. 190–200.

[15] T. J. McCabe and A. H. Watson, "Software complexity," *Crosstalk, Journal of Defense Software Engineering*, vol. 7, no. 12, pp. 5–9, 1994.

[16] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: a Generic Framework for Managing Hardware Affinities in JHPC Applications," *IEEE 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, 2010.

[17] "MongoDB - open-source NoSQL document database," http://mongodb.org, 2014.

[18] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000.

[19] "Bootstrap - HTML, CSS, and JS framework," http://getbootstrap.com/, 2014.

[20] "jqPlot - JavaScript plotting library," http://jqplot.com/, 2014.

[21] 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.