EDITORS: Kathryn Mohror, mohror1@llnl.gov
John M. Shalf, jshalf@lbl.gov

DEPARTMENT: LEADERSHIP COMPUTING

Preparing for the Future—Rethinking Proxy Applications

Satoshi Matsuoka, Jens Domke, Mohamed Wahib, and Aleksandr Drozd, *RIKEN Center for Computational Science, Kobe, 650-0047, Japan*

Andrew A. Chien and Raymond Bair, *Argonne National Laboratory, Lemont, IL, 60439, USA*

Jeffrey S. Vetter, *Oak Ridge National Laboratory, Oak Ridge, TN, 37831, USA*

John Shalf, *Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA*

A considerable amount of research and engineering went into designing proxy applications, which represent common high-performance computing (HPC) workloads, to co-design and evaluate the current generation of supercomputers, e.g., RIKEN's supercomputer Fugaku, ANL's Aurora, or ORNL's Frontier. This process was necessary to standardize the procurement while avoiding duplicated effort at each HPC center to develop their own benchmarks. Unfortunately, proxy applications force HPC centers and providers (vendors) into an undesirable state of rigidity, in contrast to the fast-moving trends of current technology and future heterogeneity. To accommodate an extremely heterogeneous future, we have to reconsider how to co-design supercomputers during the next decade, and avoid repeating past mistakes.

Supercomputing is the art of mapping a scientific question onto hundreds of trillions or quadrillions of transistors, as in the case of the currently fastest supercomputers in the world, by exploiting the problem's underlying concurrency. Unfortunately, this requires numerous transformations: *question→algorithm→parallelization→language→compilation→execution*, and intermediate bottlenecks, such as Amdahl's law, are complicating an efficient utilization of the available transistors. While society's problems are somewhat immutable, until solved, we see an increase in available choices in the remainder of this chain of transformations.

Historically, the sciences' demand for more compute capabilities was met by increasing the transistor count and density, and hence, the users had autonomy all the way from problem to compilation, while the high-performance computing (HPC) community focused primarily on developing parallelization techniques and on perfecting component integration to assemble the supercomputers. But the projected end of Moore's law and Dennard's scaling in the early 2000s required a rethinking, culminating in an intensified co-design effort at supercomputing centers. We had to take a closer look at our workloads, resulting in scaled-down versions of important scientific applications, so-called *mini* or *proxy applications*,[1] which represent the workload from problem to language, and which redefined a new overlapping between HPC users, centers, and vendors. Consequently, HPC centers and vendors tailored the hardware architectures, i.e., many-core CPUs and/or GPUs paired with high-bandwidth memory, and improved the languages, such as CUDA, to meet the proxy's requirements, while the users were required to modify their parallelization approaches and data layout, and potentially had to rewrite kernels.

We believe that this approach, while being somewhat successful for current (pre)exascale supercomputers, is not sustainable in the future. The primary reasons being an explosion in workload complexity; number of proxy applications; hardware choices; programing languages; and parallelization strategies. The availability of noisy intermediate-scale quantum

1521-9615 © 2022 IEEE
Digital Object Identifier 10.1109/MCSE.2022.3153105
Date of current version 27 May 2022.

computers, FPGAs and ASICs, in-memory processing, SmartNICs, and the commercial success of deep learning and cryptocurrencies, increase the choice for processors beyond just traditional CPUs and GPUs. Furthermore, domain experts crave for higher level languages, such as Julia or Python, or domain-specific languages (e.g., Tensorflow) to increase productivity, while HPC experts seek performance portability and exploit concurrency with OneAPI, Kokkos, RAJA, Chapel, etc.

Hence, the interaction between HPC users and providers has to change, yet again, to improve future supercomputing. The questions we want to address in the upcoming sections are as follows. What are better alternatives to proxy applications to tackle this increasing complexity? Can we maintain an efficient mapping from problem to execution within the given limits? Can we improve, automate, and streamline the co-design and procurement process for the next generation of supercomputers?

STATE OF THE ART FOR CO-DESIGN TOOL CHAINS AND PROCUREMENT BENCHMARKS

In the build-up to the petascale era in the early 2000s, the community recognized the necessity for moving away from relying on workload model benchmarks given how working with such models is complex for small companies and also due to the lack of their utility for assessing heterogeneity. The HPC challenge benchmark suite in 2005[2] and later the HPCS program by DARPA paved the way for the first transition to small and simple benchmarks.[3] However, HPCS was considered to not have enough fidelity for procurement activities. That is mainly due to 1) the inability to relate the numbers reported by the benchmarks to real workloads, and 2) the rigidity of the benchmarks that hindered efforts for emphasizing memory hierarchy and locality or exploring heterogeneity.

Evolution of Proxy Applications

After recognizing the limitations of the benchmarking approach of HPCS, the community moved to an alternative method to characterize a broad spectrum of applications. To aid co-design, governments commissioned, and HPC centers developed, numerous sets of benchmarks that reflected the priority applications of each individual nation. The community did not waste effort in trying different approaches for designing proxy applications and benchmarks to be used in procurement. The resulting proxy apps and benchmarks were used to drive the development process of this generation of (pre)exascale systems. Examples of this effort are the Exascale Computing Project (ECP) Proxy Applications, Center for Efficient Exascale Discretizations (CEED) Miniapps, and CORAL-2 Benchmarks, all developed by the DoE national labs of the USA. The European Union's PRACE consortium hosts the Unified European Application Benchmark Suite (UEABS). Meanwhile, RIKEN assembled multiple Fiber/proxy apps, spanning the social and scientific priority areas of Japan, to assist the hardware and software R&D for the supercomputer Fugaku.[4]

Overall, many of the aforementioned proxy applications have been widely used for procurement decisions, vendor interactions, evaluation of programing models and testbeds, and supercomputing research all the way from transistor to full system scales (see various related scientific papers and presentations[5–8]). Other commonly used benchmarking sets have been curated by organizations, such as the Standard Performance Evaluation Corporation (SPEC), the International Open Benchmark Council (BenchCouncil), or MLCommons—with proxys for CPUs, distributed systems, accelerators, artificial intelligence—or have been developed by companies, such as Intel (HiBench), Baidu (DeepBench), and Google (PerfKit). Finally, traditional peak performance benchmarks, such as High Performance Linpack (HPL) and Conjugate Gradient, stream, or Intel's MPI Benchmarks, continue to be relevant during procurement and for system evaluation.

LIMITATIONS OF EXISTING CO-DESIGN TOOL CHAINS AND PROCUREMENT BENCHMARKS

An in-depth, successful co-design of modern systems without the assistance of proxy apps and workload analysis would have been unlikely, but the current state of our co-design "toolbox" contains some remediable and some fundamental flaws that we (HPC centers) and vendors uncovered while applying it, as listed hereafter.

Implementation Biases

Any implementation of a benchmark, no matter the size, entails multiple implicit biases not only toward programing language, data layout, and parallelization approach, but also toward the underlying algorithm, which is used to solve a problem. Since many of the ECP, Fiber, and SPEC proxy application are scaled-down versions of existing HPC workloads, they are still predominantly implemented in Fortran and C, and have been tuned over the years, and sometimes decades, for cache-based CPU architectures. This over-commitment of co-design in one area can lead to lack of attention in other areas, causing, for example, underperforming C++ applications with certain compilers. Similarly, the extensive focus on GPU-based

supercomputing resulted in highly tuned CUDA (and "legacy" CPU) proxy applications from the ECP side with memory/data layouts optimized for those architectures. These codes and layouts are not easily transferable to OneAPI/SYCL, which will be used in ANL's Aurora exascale supercomputer, and hence, more engineering and time was required to port the codes to another programing paradigm.

Complexity Trap and Performance Portability Myth

Unfortunately, and despite best efforts by the HPC community, achieving performance portability across slightly or widely varying architectures is still an unobtainable goal. Modern features in OpenMP or alternative programing paradigms, such as OpenCL, demonstrate that applications can be written (often without separate code paths for different architectures) in a way to easily migrate between different CPUs, GPUs, and vendors while preserving correctness; however, usually such migration results in severe performance drops.[9–11] Hence, manual code refactoring is still required, to change algorithms, data layouts, parallelization strategies, etc., to fully utilize any given architecture. However, domain scientists estimate the cost of such refactoring efforts (for example, to target FPGAs) of upward of $11 million per 100,000 lines of code.[12] Such cost for a single proxy is obviously unjustifiable for vendors and HPC centers[8,13] during the early phases of the co-design, i.e., while exploring diverging architecture options, especially when considering the growing number of proxies and code complexity thereof. Therefore, the focus on proxies of large/legacy scientific codes, which had been tuned for previous architectures, traps the co-design participants into considering only similar architectures.

Insufficient Input Configurations, Testing, and Documentation

We noticed additional shortcomings while working with many of these proxy applications in multiple research and co-design projects. For example, some applications rely on third-party, sometimes closed-source, libraries or inputs, which were inaccessible because the documented hyperlinks were outdated and content had been moved. Working with different compilers or even compiler versions proved challenging, since many of the scaled-down workloads have not been tested across a range of compilers, and changes in the environment exposed implementation errors or numerical instabilities. Furthermore, the documentation of the proxy applications often ranges in quality, even within certain sets of benchmarks from the same institution, with lack of information such as 1) which parts of the code are performance critical, 2) is bit-reproducibility required, 3) what is/are the implemented algorithms and do alternatives exist, and 4) which system characteristics does this input evaluate? (to name just a few). Last but not least, many proxy applications lack variability in input selection, i.e., they lack strong- or weak-scaling tests, have fixed requirements for memory, OpenMP threads, MPI ranks, etc., or lack meaningful inputs that are usable with slow, high-fidelity simulators. These shortcomings can severely impede the co-design approach, yet most of them are preventable with adequate funding, community effort, and predetermined guidelines for code/input/documentation quality.

Lack of Efficiency Reporting

It is important for performance engineering practitioners and end users to understand the efficiency a given benchmark could achieve on a given hardware target. Efficiency in this context can be viewed as *how much performance was achieved, in comparison to the peak theoretical performance*. All the aforementioned benchmark suites lack efficiency reports. Explicitly reporting efficiency becomes increasingly important, as we head into an era of more complexity in systems. Without keeping all stakeholders aware of the efficiency considerations, we run the risk of designing complex systems that are poorly utilized and/or hard to program.

Current Efforts to Overcome the Shortcomings

With the focus of centers shifting to prepare their priority workloads for delivered systems, the work on proxy apps has slowed. Few efforts, e.g., filtering duplicated proxies via $\cos\theta$-metric,[8] are still ongoing, but no coordinated and noticeable activity is dedicated toward resolving the proxy-application limitations we mentioned. In the rest of this article, we lay out our vision for the future: evolving beyond proxy applications.

CO-DESIGN TOOLBOX OF THE FUTURE

Since neither microbenchmarks nor application proxies seem to be the right fit for the required co-design in the next decade, one being too detached from reality and the other too static and backward looking, we require something in-between. We envision a form of highly parameterizable, easily amendable, motif-like representations of algorithms or kernels. Such complex "operations" could be anything from fast Fourier transform or sparse matrix-multiplication over data-structure padding operations for

stencils to remeshing operations in load-balanced solvers. For their shape-shifting nature, we call them *Octopodes* instead of scientific motifs[14] (e.g., NAS Parallel Benchmarks), since they are more abstract and meant to be closer to algorithms (supported by one/many reference implementations) than fixed proxy-implementation. The utility of *Octopodes* and how they fit into the remaining co-design toolchain is the subject of the following sections.

Representative *Octopodes* for Co-Design

The goal is to capture the algorithms or complex operations instead of the implementation, which make up modern workloads, and which can be optimized, transformed, or replaced. Assuming we have such kernel-/operator-generation capabilities for a wide range of typical scientific problems, then we could use machine learning for the following:

- automatically identifying compute phases or regions-of-interest across all scientific codes and projects running on a given supercomputer;
- determining the right parameterization to correlate an *Octopode* to a real application region;
- finding similarities between applications (for example, by using the $\cos\theta$-metric[8]).

In this way, hardware and software tuning can be performed more efficiently for a given problem class instead of multiple different (proxy-) applications, which all exhibit similar behavior.

Such high-level specifications and parameterizations of interest can be used to convey the mathematical problem and underlying compute pattern to a co-design team or hardware vendor in a more descriptive manner than a million lines of source code and scientific papers.

What we mean by "high-level specifications and parameterizations" can be more easily understood when looking at a potential *Octopode*, namely matrix–matrix-multiplications or in short *matmul*, which everyone in the HPC community is familiar with. Instead of just benchmarking double-precision `dgemm` with HPL, a single matmul-*Octopode* would support various input shapes (such as squared, rectangular, and tall/skinny) and numerical precisions (i.e., from quadruple precision `fp128`, as used in some quantum simulations, all the way down to low precision, such as `bfloat16` and the like), and batched and non-batched execution. Furthermore, the matmul-*Octopode* shall not only cover dense matrix–matrix operations, but also matrix–vector and sparse matrices. Many of the sparse matrix formats exploit local memory to some degree to achieve computational performance, but such blocking needs to be supported by the input matrix, and hence, a simple randomly generated sparse matrix is unsuitable as general input representation, and should only be used as one of many. Other realistic sparse inputs for the *Octopode* can be sampled from public repositories or various existing HPC workloads (e.g., resulting from structured meshes) and DL problems, which often result in highly regular and blocked sparsity.

Future Co-Design Cycle With *Octopodes*

Our vision for a modernized co-design requires even tighter collaboration between the HPC users and the co-design teams, and allows more flexibility for the vendors. The first step requires the users and co-design teams to analyze the dominant HPC workloads (w.r.t. the consumed node-hours). This process consists of:

1) profiling the execution;
2) identifying regions-of-interest;
3) collecting performance-relevant data, such as execution time and hardware counters; and
4) categorizing the regions into algorithms and complex operations.

In the past, 1)–3) were commonly done by users, but 4) is necessary as well for a holistic view—for machine learning, and for an improved co-design. These regions can then be mapped to *Octopodes*, which correlate to true HPC and data center workloads, if parameterized appropriately.

To demonstrate the hardware capabilities, the vendors shall be allowed more tuning freedom for the *Octopodes*, i.e., changes of algorithm, implementation, integer/floating-point precision, data layout, etc., as long as the intended result remains the same, the changes are properly communicated, and not only the algorithm is benchmarked (but also the necessary time for the pre/postexecution transformation of the data). The knowledge for mapping an *Octopode* to a given hardware can then be used on a limited set of proxy applications to serve as benchmark suite for acceptance testing and to serve as demonstrator for users on how to change/tune their codes.

The crucial aspects for this design cycle to succeed are better tools, extensive automation in workload analysis and architecture modeling and evaluation, and increased bidirectional knowledge transfer between users, system operators, co-designers, and hardware/software vendors. Furthermore, *Octopodes*, as well as microbenchmarks and proxies, need to be appropriately documented. Finally, all (reference-) implementations of *Octopodes* should, in addition to reporting architecture-

independent performance metrics (e.g., work/time), report the efficiency for the implementation when run on a target hardware.

OUTLOOK AND SUMMARY

The use of proxy applications expanded and improved the co-design capability of modern supercomputers, but we believe that current hardware trends and software complexities require a new set of tools for the co-design of postexascale supercomputers and federated HPC/data centers to better capture, analyze, and model existing and future workload demands. To open the floor for future, community-wide discussions, we have outlined the state of the art and its shortcomings, and propose an alternative, hopefully better suited set of highly parameterizable, easily amendable, motif-like problem representations that we call *Octopodes*. These algorithms or complex operations shall not replace proxy applications entirely but supersede them as the primary target in the co-design process. *Octopodes* will be the common language between HPC users, system operators, co-designers, and vendors to describe the to-be-solved scientific challenges—what needs to be computed, and how it can be computed—in an abstract way. This approach allows for more flexibility in the hardware/software design and selection to match the users' needs with the best architecture, instead of fine-tuning legacy architectures to legacy implementations. We expect that our conception of a community-driven and well-curated set of *Octopodes* is able to improve the overall co-design cycle, while also being able to alleviate the increased complexity and labor cost associated with modern proxies.

REFERENCES

1. R. Barrett *et al.*, "On the role of co-design in high performance computing," *Adv. Parallel Comput.*, vol. 24, pp. 141–155, 2013, doi: 10.3233/978-1-61499-324-7-141.
2. P. Luszczek *et al.*, "Introduction to the HPC challenge benchmark suite," Innovative Computing Laboratory, Knoxville, TN, USA, Tech. Rep. ICL-UT-05-01, 2005.
3. J. Dongarra *et al.*, "DARPA's HPCS program: History, models, tools, languages," in *Proc. Adv. Comput.*, vol. 72, pp. 1–100, 2008, doi: 10.1016/s0065-2458(08)00001-6.
4. M. Sato *et al.*, "Co-design for A64FX manycore processor and 'FUGAKU'," in *Proc. IEEE SC20: Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2020, pp. 651–665, doi: 10.1109/SC41405.2020.00051.
5. J. Domke *et al.*, "HyperX Topology: First at-scale implementation and comparison to the fat-tree" in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2019, pp. 1–23, doi: 10.1145/3295500.3356140.
6. O. Aaziz *et al.*, "A methodology for characterizing the correspondence between real and proxy applications," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2018, pp. 190–200, doi: 10.1109/CLUSTER.2018.00037.
7. J. Ang *et al.*, "ECP report: Update on proxy applications and vendor interactions" *Sandia Nat. Lab.*, Albuquerque, NM, USA, Tech. Rep. SAND-2020-3852R, 2020.
8. D. Richards *et al.*, "Best practices for using proxy applications as benchmarks," *Presentation IDEAS Productiv. Project Webinar*, 2020. [Online]. Available: https://proxyapps.exascaleproject.org/wp-content/uploads/2020/03/ProxiesAndBenchmarks_small.pdf
9. J. C. Saez *et al.*, "Enabling performance portability of data-parallel OpenMP applications on asymmetric multicore processors," in *Proc. 49th Int. Conf. Parallel Process.*, 2020, pp. 1–11, doi: 10.1145/3404397.3404441.
10. R. Gayatri *et al.*, "A case study for performance portability using OpenMP 4.5" in *Proc. Int. Workshop Accelerator Program. Using Directives*, 2019, pp. 75–95, doi: 10.1007/978-3-030-12274-4_4.
11. J. Pennycook *et al.*, "An investigation of the performance portability of OpenCL," *J. Parallel Distrib. Comput.*, vol. 73, no. 11, pp. 1439–1450, 2013, doi: 10.1016/j.jpdc.2012.07.005.
12. B. Sorensen *et al.*, "Special report for NASA: Exploring options for a bespoke supercomputer targeted for weather and climate workloads," Hyperion Research, Saint Paul, MN, USA, Tech. Rep. HR4.0046.10.01.2019, 2019.
13. B. Begole *et al.*, "Position paper: Codesign beyond exascale," Advance Micro Devices, Santa Clara, CA, USA, Tech. Rep. 18435741, 2021.
14. K. Asanović *et al.*, "The landscape of parallel computing research: A view from Berkeley," Dept. Elect. Eng. Comput. Sci., UC Berkeley, Berkeley, CA, USA, Tech. Rep. UCB/EECS-2006-183, 2006.

SATOSHI MATSUOKA is the director at the RIKEN Center for Computational Science, Kobe, 650-0047, Japan. Contact him at matsu@acm.org.

JENS DOMKE is a research scientist at the RIKEN Center for Computational Science, Kobe, 650-0047, Japan. Contact him at jens.domke@riken.jp.

MOHAMED WAHIB is the team leader at RIKEN Center for Computational Science (RIKEN-CCS), Kobe, 650-0047, Japan. Contact him at mohamed.attia@riken.jp.

ALEKSANDR DROZD is a research scientist at the RIKEN Center for Computational Science, Kobe, 650-0047, Japan. Contact him at alex@blackbird.pw.

ANDREW A. CHIEN is a William Eckhardt professor in computer science and the director at the CERES Center for Unstoppable Computing, The University of Chicago, Chicago, IL, 60637, USA, and a Senior Computer Scientist at Argonne National Laboratory, Lemont, IL, 60439. Contact him at achien@mcs.anl.gov.

RAYMOND BAIR is a chief computational scientist for applications at Computational Science Division, Argonne National Laboratory, Lemont, IL, 60439, USA. Contact him at raybair@gmail.com.

JEFFREY S. VETTER is a corporate fellow with Oak Ridge National Laboratory (ORNL), Oak Ridge, TN, 37830, USA, where he is also the section head for advanced computer systems research and the founding director at the Experimental Computing Laboratory. For more, visit https://ft.ornl.gov/~vetter/. Contact him at vetter@computer.org.

JOHN SHALF is currently the department head for computer science at Lawrence Berkeley National Laboratory, Berkeley, CA, 94720, USA, and was formerly the deputy director of hardware technology for the DOE Exascale Computing Project and the leader of the Green Flash Project. Contact him at jshalf@lbl.gov.