A Testbed of Parallel Kernels for Computer Science Research

David Bailey, James Demmel, Khaled Ibrahim, Alex Kaiser, Alice Koniges,
Kamesh Madduri, John Shalf, Erich Strohmaier, Samuel Williams
estrohmaier@lbl.gov
Computational Research Division
Lawrence Berkeley National Laboratory, Berkeley, CA 94720

For several decades, computer scientists have sought guidance on how to evolve architectures, languages, and programming models for optimal performance, efficiency, and productivity. Unfortunately, this guidance is most often taken from the existing software/hardware ecosystem. Architects attempt to provide micro-architectural solutions to improve performance on fixed binaries. Researchers tweak compilers to improve code generation for existing architectures and implementations, and they may invent new programming models for fixed processor and memory architectures and computational algorithms. In today's rapidly evolving world of on-chip parallelism, these isolated and iterative improvements to performance may miss superior solutions in the same way gradient descent optimization techniques may get stuck in local minima.

In an initial study, we have developed an alternate approach that, rather than starting with an existing hardware/software solution laced with hidden assumptions, defines the computational *problems* of interest and invites architects, researchers and programmers to implement novel hardware/software co-designed *solutions*. Our work builds on the previous ideas of computational dwarfs, motifs, and parallel patterns by selecting a representative set of essential problems for which we provide: An algorithmic description; scalable problem definition; illustrative reference implementations; verification schemes.

For simplicity, we focus initially on the computational problems of interest to the scientific computing community but proclaim the methodology (and perhaps a subset of the problems) as applicable to other communities. We intend to broaden the coverage of this problem space through stronger community involvement.

Previous work has established a broad categorization of numerical methods of interest to the scientific computing, in the spirit of the NAS Benchmarks [3], which pioneered the basic idea of a "pencil and paper benchmark" in the 1990s. The initial result of the more modern study was the seven *dwarfs*, which was subsequently extended to 13 *motifs* [1, 2, 4]. These motifs have already been useful in defining classes of applications for architecture-software studies. However, these broad-brush problem statements often miss the nuance seen in individual kernels. For example, the computational requirements of particle methods vary greatly between the naive (but more accurate) direct calculations and the particle-mesh and particle-tree codes.

Thus we commenced our study with an enumeration of problems, but then proceeded by providing not only reference implementations for each problem, but more importantly a mathematical definition that allows one to escape iterative approaches to software/hardware optimization. To ensure long term value, we have augmented each of our reference implementations with both a scalable problem generator and a verification scheme.

In a paper we have prepared that documents our efforts [5], we describe in detail this process of problem definition, scalable input creation, verification, and implementation of reference codes

for the scientific computing domain. Table 1 enumerates and describes the level of support we've developed for each kernel. We group these important kernels using the Berkeley dwarfs/motifs taxonomy using a red box in the appropriate column. As kernels become progressively complex, they build upon other, simpler computational methods. We note this dependency via orange boxes.

After enumeration of the important numerical problems, we created a domain-appropriate high-level definition of each problem. To ensure future endeavors are not tainted by existing implementations, we specified the problem definition to be independent of both computer architecture and existing programming languages, models, and data types.

Then, to provide context as to how such kernels productively map to existing architectures, languages and programming models, we produced reference implementations for most of the kernel (see table). These sample codes should be viewed as "hints," designed to show how other designers have mapped a problem's operands and operators to existing hardware and software. Since we wanted such implementations to be illustrative, we tried to ensure they were the most straightforward implementation in the easiest to understand languages using familiar architectures. To that end, most of the linear algebra-oriented computations are written in MATLAB using array indexing to process matrices, rather than one-line library calls to compute the same kernel. This ensures that the kernel's computation is explicit and readable in the implementation and not hidden behind a library. For other problems, such as the Barnes-Hut n-Body solver, the implementations were written in pure C, without any supporting library computations.

Along this line, we have also created a scalable problem generator to accompany each computation. In some cases this generator may be nothing more than a means of specifying problems using the underlying method's high-level description language. In other cases, code is written to create input datasets. In either case, the problem size is independent of implementation or mapping to architecture.

Finally, we have specified a means to verify the validity of a solution. In many cases, we construct problems whose solutions are known a priori or can be calculated with minimal cost. For example, we verify the symmetric eigensolver by constructing randomized matrices with known eigenvalues. To obtain such a matrix, one forms a diagonal matrix D composed of the desired eigenvalues and a randomized orthogonal matrix Q. The test matrix is the product $Q^T DQ$. This "reverse diagonalization" produces a randomized matrix with pre-determined eigenvalues, the eigenvalues of which can be selected to be as numerically challenging or clustered as the user desires.

While the work done so far has satisfactorily developed this concept from its initial conception, we feel that much remains to be done, particularly as the community in general (and DOE-OS in particular) moves to exascale computer systems. First of all, optimized parallel reference implementations must be prepared for all of the kernels implemented so far (plus any additional ones that may be added to the set). Secondly, the implications of these kernels for computer science research in the field needs to be explored. In particular, we need to explore the issue programming models in the context of these motifs. In addition, we need to examine how these kernels (parallel and serial) behave in a cache memory hierarchy, and how this hierarchy needs to change as to move to highly multicore and exascale systems.

Acknowledgments

This work was supported by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Kernel	Dense Linear Alg.	Sparse Linear Alg.	Structured Grids	Unstructured Grids	Spectral	Particles	Monte Carlo	Graphs & Trees	Sort	Definition	Reference	Optimized	Scalable Inputs	Verification
Scalar-Vector Mult.	■									✓	✓		✓	
Elementwise-Vector Mult.	■									✓	✓		✓	
Matrix-Vector Mult.	■									✓	✓		✓	
Matrix-Matrix Mult.	■									✓	✓		✓	
LU Factorization	■									✓	✓	✓	✓	✓
Symmetric Eigensolver (QR)	■									✓	✓		✓	✓
Cholesky Factorization	■									✓			✓	
SpMV (y=Ax)		■								✓	✓	✓	✓	✓
SpTS (Lx=b)		■								✓	✓		✓	✓
Matrix Powers (y_k=A^kx)		■								✓	✓		✓	✓
CG	▪	■								✓	✓		✓	✓
KSM/GMRES	▪	■								✓	✓		✓	✓
SpLU	▪	■												
Finite Difference Derivatives			■							✓	✓		✓	✓
FD/Laplacian			■							✓	✓	✓	✓	✓
FD/Gradient			■							✓	✓	✓	✓	✓
FD/Divergence			■							✓	✓	✓	✓	✓
FD/Curl			■							✓	✓		✓	✓
FD/Solve PDE, Explicit			■							✓	✓		✓	✓
FD/Solve PDE, Implicit Iter.	▪		■							✓	✓		✓	✓
FD/Solve PDE, Multigrid	▪		■							✓	✓		✓	✓
1D FFT (complex→complex)					■					✓	✓		✓	✓
3D FFT (complex→complex)					■					✓	✓		✓	✓
Convolution	▪				■					✓	✓		✓	✓
Solve PDE via FFT	▪		■		■					✓	✓		✓	✓
2D N^2 Direct						■				✓	✓		✓	
3D N^2 Direct						■				✓	✓		✓	
2D N^2 Direct (with cut-off)						■				✓	✓		✓	✓
3D N^2 Direct (with cut-off)						■				✓	✓		✓	✓
2D Particle-in-Cell (PIC)			▪			■								
3D Particle-in-Cell (PIC)			▪			■								
2D Barnes Hut						■		▪		✓	✓		✓	✓
3D Barnes Hut						■		▪		✓	✓		✓	✓
2D Fast Multipole Method					▪	■		▪						
3D Fast Multipole Method					▪	■		▪				✓	✓	
Quasi-Monte Carlo Integration							■			✓	✓		✓	✓
EP Summation							■			✓	✓			✓
Graph traversal								■		✓	✓		✓	✓
Betweenness centrality								■		✓	✓		✓	✓
Integer Sort									■	✓	✓		✓	✓
100 Byte Sort									■	✓	✓		✓	✓
Spatial Sort									■	✓			✓	

Table 1: Brief overview of enumerated kernels with their mapping to dwarfs. Check marks denote progress we've made towards a practical testbed for scientific computing. Note, orange boxes denote the mapping of supporting kernels to dwarfs.

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, and K.A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report UCB/EECS-2006-183, University of California, Berkeley, December 2006.

[2] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J.D. Kubiatowicz, E.A. Lee, N. Morgan, G. Necula, D.A. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K.A. Yelick. The parallel computing laboratory at U.C. Berkeley: A research agenda based on the Berkeley view. Technical Report UCB/EECS-2008-23, University of California, Berkeley, March 2008.

[3] D.H. Bailey, E. Barszcz, J.T. Barton, D.S. Browning, R.L. Carter, L. Dagum, R.A. Fatoohi, P.O. Frederickson, T.A. Lasinski, R.S. Schreiber, H.D. Simon, V. Venkatakrishnan, and S.K. Weeratunga. The NAS parallel benchmarks. *Int'l. Journal of High Performance Computing Applications*, 5(3):63–73, 1991.

[4] P. Colella. Defining software requirements for scientific computing, 2004. DARPA HPCS presentation.

[5] Alex Kaiser, Samuel Williams, Kamesh Madduri, Khaled Ibrahim, David Bailey, James Demmel, and Erich Strohmaier. Principled kernel testbed for hardware/software co-design research. *manuscript to appear at HotPar'10*, 2010.