A Principled Kernel Testbed for Hardware/Software Co-Design Research

Alex Kaiser, Samuel Williams, Kamesh Madduri, Khaled Ibrahim, David Bailey, James Demmel, Erich Strohmaier
{ADKaiser, SWWilliams, KMadduri, KZIbrahim, DHBailey, EStrohmaier}@lbl.gov, demmel@eecs.berkeley.edu

FUTURE TECHNOLOGIES GROUP

Motivation

Current research is focused on how to effectively use an ever-dwindling array of parallel processors. As such, the community is being driven into an evolutionary and architecturally-driven mindset. We believe this will yield substantial results. For hardware/software co-design to truly be effective, we must start from the core computational methods we wish to accelerate, not code extracted from existing applications.

This project is focused on creating a kernel testbed based on the core computational methods found in high-performance computing. We believe the core methodology (if not some of the kernels) are applicable in performance computing. We believe the core methodology mandates creation of:

- Benchmarks that may not fully enable interdisciplinary research
- Methodology (if not some of the kernels) are applicable in performance computing
- Performance computing

We believe the core methodology mandates creation of:

- Benchmarks that may not fully enable interdisciplinary research
- Methodology (if not some of the kernels) are applicable in performance computing
- Performance computing

(1) Problem Specification

- The problem specification for a kernel mathematically or quantitatively defines the functional relationship between input and output.
- We strive not to use array notation or other programming language-based constructs (e.g., loops for parallel constructs) in our definitions.
- For example, in numerical linear algebra, we define problems using the well-developed lexicon of linear algebra (scalars, vectors, matrices) and operators (addition, multiplication, transpose, inverse, sum, etc.)

(2) Scalable Input Dataset

- Wherever possible, each kernel problem definition is accompanied by a scalable input generation scheme.
- They should be amenable to straightforward and independent verification while guaranteeing the existence of a solution (random inputs may not suffice).
- When performing distributed or novel HW/SW design, researchers should re-implement the input generators.

(3) Verification Scheme

- We wish to verify problems independently from their definitions. One shouldn’t use reference codes/HW to verify novel hardware/software designs.
- In many domains, for carefully constructed inputs, we may provide an analytic solution based on the calculus of the underlying mathematics. (see example in next column)
- Some kernels are simple functions (they’re not solvers). For them, complex verification schemes are usually not needed.

Reference Implementation

- To provide some guidance as how one might implement such a kernel using existing languages, programming models, and hardware, we provide a reference implementation for each kernel.
- The reference implementation is either a sequential C or MATLAB program including the input generation and verification components (where applicable). The reference implementations should never be used as the basis for benchmarking. It is incumbent upon researchers to produce appropriate implementations for their field of research.

Quality of HW/SW Solution

- If this testbed were used only for SW optimization, then the quality of the optimized implementations is primarily time or energy.
- If used for HW/SW co-design, hardware design cost and portability should be considered.
- If used for programming model or language research, productivity might be of interest.

Input / Verification Example

- Consider solving the heat equation PDE on a rectangular N-dimensional domain. Virtually every non-trivial kernel has an associated scalable verification scheme.
- Additionally, we have created sequential C or MATLAB reference implementations for most of them.
- We list their status below and categorize them based on the original seven dwarfs or Berkeley’s subsequent 13 Motifs.

Our Kernel Testbed Today

- To date, we have created a testbed of over 40 kernels.
- Virtually every non-trivial kernel has an associated scalable verification scheme.
- Additionally, we have created sequential C or MATLAB reference implementations for most of them.
- We list their status below and categorize them based on the original seven dwarfs or Berkeley’s subsequent 13 Motifs.