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, SWilliams, 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 diversifying array of parallel processors. As such, the scalable kernel space is being driven into an evolutionary and architecturally-driven mindset. We believe this will yield suboptimal 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.

Thus, 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 other domains. Previous attempts have created benchmarks that may not fully enable inter-disciplinary research.

We believe this will yield a scalable mindset. We wish to verify problems independently from their existence of a solution (random inputs may not suffice).

Additionally, we may provide an analytic solution based on the calculus of the underlying mathematics.

3. Verification Scheme

We wish to verify problems independently from their definitions. (one shouldn’t use reference codes 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.

Optional 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 implementation 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 was 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.

By carefully selecting the initial and boundary conditions, we may analytically solve the problem.

Conversely, we may solve the problem numerically using one of 6 different methods (spanning three dwarfs)

All methods should produce the same answer as a sampling of the analytic solution.

We may aggressively push the complexity in the sparse arena by permitting the grid enumeration (rows/columns) or randomly adding explicit zeros.

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 seven dwarfs.

Testbed Components

Our testbed is composed of a series of kernels.

For each kernel, the testbed mandates creation of:

- a formal problem specification in a mathematical, or domain-appropriate language
- 2. a scalable input generator
- 3. a scalable verification scheme

Optionally, we provide a reference implementation in commonly used programming languages.

Additionally, we may provide an optimized reference implementation that provides insights into the bottlenecks on existing hardware and researcher’s optimizations to eliminate, hide, or mitigate them.