The MachSuite Benchmark
Brandon Reagen, Robert Adolf, Gu-Yeon Wei, David Brooks
Harvard University, Cambridge, MA, USA
{reagen, rpadolf, guyeon, dbrooks}@eecs.harvard.edu

I. INTRODUCTION
With the demise of Dennard scaling, today’s architects are confronting chips filled with more transistors than can be fully powered. One possible mechanism to continue improving performance on a power budget is the use of hardware accelerators, fixed-function hardware blocks which compute a specific task at a fraction of the cost of a general-purpose processor.

Unfortunately, designing hardware accelerators in RTL languages like Verilog or VHDL is time-consuming and complicated. To improve productivity, the CAD community introduced High-Level Synthesis (HLS) tools, which automatically synthesize RTL code from a high-level language like C or C++. Modern HLS frameworks can produce code with performance on par with hand-written RTL for some workloads. However, as designers tackle more complicated, irregular applications, these tools are struggling.

The interest in hardware specialization has brought about challenges at the architecture level as well. Fine-grained heterogeneity has appeared in processors, and the rising popularity of systems-on-chip (SoCs) in the mobile domain is making hardware accelerators commonplace. Architects now have to contend with a much larger range of on-chip interactions, and with so many different possible tactics and mechanisms, it is a daunting task to choose the right design direction.

These trends are two sides of the same coin: advances in accelerator-centric research have caught us unprepared to quantitatively and objectively evaluate the relative strengths and weaknesses of such a diverse collection of techniques. Much of this can be ascribed to a simple lack of standardization. A survey of recent publications involving hardware accelerators revealed that of the 88 distinct benchmarks used across 25 papers, 64 of them were only ever used once, as shown in Figure 1. The most popular benchmarks appeared in less than half of those. The community needs standardization and commensurability.

Consequently, we present MachSuite: a new accelerator-centric benchmark suite tailored to the needs of both the HLS and architecture communities. MachSuite is a set of 19 benchmarks spanning 12 different kernels, written to cover a diverse set of application domains and to incorporate distinct algorithmic choices. All the benchmarks in MachSuite are HLS synthesizable, providing architecture researchers an easy way to quickly generate a diverse set of hardware accelerators.

II. THE NEED FOR A NEW BENCHMARK SUITE
In this section, we draw a distinction between benchmark kernels, algorithms, and implementations and discuss the importance of standardization across each of them. We then examine existing, related benchmarks and show why they are unsuitable for accelerator-centric research.

A. The Need for Standard Kernels
Absent any established standard, researchers will choose the most appropriate workloads for their particular situation. Unfortunately, the wide variety of different projects leads to divergence in the choice of benchmarks used. This makes direct, objective comparisons between novel solutions difficult. Standardized benchmark kernels allows the community to understand and quantify where fundamental advances are being made.

B. The Need For Standard Algorithms
A more subtle issue is differences between which algorithms are used to solve a particular kernel. We do not want to discourage different approaches, and it is important for the community to be able to study and compare different algorithms, but we need transparency. For example, a case study evaluating \( k \)-means clustering accelerators demonstrated that even solving the same problem with the same inputs produced a performance gap of \( 1.6 \times 12.8 \times \) between different algorithms [4]. Without agreement on a common algorithm, these discrepancies can be hidden in published research results.

C. The Need For Standard Implementations
In a similar vein, differences in coding style and implementation can cause significantly different results. While this effect shows up in benchmarks for conventional platforms, accelerators are more sensitive to these changes, especially designs produced using HLS tools. One recent example involved a

---

1MachSuite was originally published by B. Reagen et. al in IISWC 2014 [2]
parametric sweep over a space of implementations of the same kernel and algorithm [3]. The authors showed that by changing only the organization of the inner loop of a parallel scan code, the Pareto-optimal points for power and performance can vary by an order of magnitude, even when given similar hardware resources, memory bandwidth, and parallelism.

D. Existing Benchmark Suites are Not Applicable

To the best of our knowledge, CHStone [1] is the only existing HLS benchmark suite. Designed to be an evaluation mechanism for HLS tools, CHStone focuses on a small number of low-level computations. While useful for evaluating the capabilities of older HLS frameworks, it falls short when put in the context of the complex designs handled by modern CAD toolchains and the expansive system designs that many architects are contemplating and constructing.

Many high-quality benchmarks have emerged from the GPGPU community in the last several years. Large numbers of vector processing units argue for a data parallel programming paradigm; a deep, complicated memory hierarchy demands attention at the algorithm level; the split main memory system strains application writers. Most GPU benchmarks do an admirable job of contending with these constraints, but accelerator-centric architectures have a completely different design space, one characterized by many more degrees of freedom. GPU benchmarks are unsuitable for this space.

III. THE DESIGN OF THE MACHSUITE BENCHMARKS

To address these needs, we created MachSuite, a set of 19 benchmarks covering 12 different application kernels. In this section we present an overview of MachSuite’s features and how we arrived at the design decisions we made.

A. Kernel Selection

The MachSuite workloads were chosen to satisfy two basic criteria: diversity and coverage. To achieve diversity, we used a battery of workload characterization metrics to judge whether any two programs had overly similar execution behavior. To achieve coverage, we first looked at our literature survey. Directly including every code in that list would be prohibitive, but we can match each benchmark published more than once with a similar one of our benchmarks, ensuring that the behavior of MachSuite encompasses workloads the community cares most about. We augmented our set of kernels to include additional programming patterns that provide new targets for accelerator designers and system architects to evaluate against. In Table I, each benchmark is described and assigned to an application pattern to show, at a high level, the application space covered by MachSuite.

B. Algorithm Selection

For each kernel, we select an algorithm representative of the tactics commonly used to solve it. While we could have chosen cutting-edge methods and invested heavily in heuristics for our benchmarks, we elected to favor simplicity over optimality for two reasons. First, optimality is not portable, and an algorithm which performs well given one set of assumptions may flop when run in another context. Codes with this “sometimes-optimal” flavor are rarely useful as benchmarks. Secondly, simplicity is a virtue for experimentation. Benchmark suites are continually used and abused in ways their designers never predicted.

<table>
<thead>
<tr>
<th>Kernel/Algorithm</th>
<th>Description</th>
<th>Berkeley Dwarf</th>
</tr>
</thead>
<tbody>
<tr>
<td>AES/AES</td>
<td>AES encryption</td>
<td>Combinational logic</td>
</tr>
<tr>
<td>BKF/BULK</td>
<td>Breadth-first search</td>
<td>Unstructured grids</td>
</tr>
<tr>
<td>BFS/QUEUE</td>
<td>Breadth-first search</td>
<td>Graph traversal</td>
</tr>
<tr>
<td>F/F/TRANSPOSE</td>
<td>Fast Fourier transform</td>
<td>Spectral methods</td>
</tr>
<tr>
<td>GEMM/NCUBED</td>
<td>Matrix multiplication</td>
<td>Dense linear algebra</td>
</tr>
<tr>
<td>GEMM/BLOCKED</td>
<td>Matrix multiplication</td>
<td>Dense linear algebra</td>
</tr>
<tr>
<td>KMP/KMP</td>
<td>String matching</td>
<td>Finite state machines</td>
</tr>
<tr>
<td>M/L/N</td>
<td>Molecular dynamics</td>
<td>N-body methods</td>
</tr>
<tr>
<td>MD/GRID</td>
<td>Molecular dynamics</td>
<td>N-body methods</td>
</tr>
<tr>
<td>NW/NW</td>
<td>DNA alignment</td>
<td>Dynamic programming</td>
</tr>
<tr>
<td>SORT/ERANGE</td>
<td>Sorting</td>
<td>Map reduce</td>
</tr>
<tr>
<td>SPMV/CRS</td>
<td>Sparse matrix/vector multiplication</td>
<td>Sparse linear algebra</td>
</tr>
<tr>
<td>SPMV/ELLPACK</td>
<td>Sparse matrix/vector multiplication</td>
<td>Sparse linear algebra</td>
</tr>
<tr>
<td>STENCIL/STENCIL2D</td>
<td>Stencil computation</td>
<td>Structured grids</td>
</tr>
<tr>
<td>VITERBI/VITERBI</td>
<td>Hidden Markov model estimation</td>
<td>Graphical models</td>
</tr>
</tbody>
</table>

TABLE I: The MachSuite benchmarks.

C. Future Directions

MachSuite is not without limitations. Each code we provide was written to have a memory footprint of 32 kB, around the size of an L1 data cache. This is useful when examining research strengths and weaknesses between kernels, and it is representative of much prior work in accelerator design. However, many researchers are now looking at larger projects and integrated designs, and these efforts require substantially larger input and working set sizes to exercise memory hierarchies and provide realistic data movement behavior.

Our current research is on designing a scalable version of MachSuite. This requires not only larger input sizes, but also implementations which automatically adjust to data and platform characteristics. For example, a larger input for a matrix multiplication code means the blocking factor and loop bounds would also need to change in order to maintain realistic program characteristics like computational density and memory reuse. For more irregular workloads, realistic scaling is even more important.

IV. CONCLUSION

MachSuite is a new, accelerator-centric benchmark suite designed to provide standard kernels and implementations for accelerator-related research. We provide a clean, HLS-synthesizable codebase to allow tool developers and computer architects to explore new optimizations, handle more complex program behaviors, and allow their results to be analyzed on common ground.

REFERENCES


