KiloCore: A Fine-Grained 1,000-Processor Array for Task-Parallel Applications

KiloCore is an array of 1,000 independent processors and 12 memory modules designed to support applications that exhibit fine-grained task-level parallelism. Each programmable processor occupies 0.055 mm^2 and supports energy-efficient computation of small tasks. Processors are connected using circuit and packet-based networks. Fine-grained tasks have low communication link densities, allowing most links to be assigned to the energy-efficient, high-performance circuit network.

Brent Bohnenstiehl
Aaron Stillmaker
Jon Pimentel
Timothy Andreas
Bin Liu
Anh Tran
Emmanuel Adeagbo
Bevan Baas
University of California, Davis

Parallel processing offers well-known benefits in performance and efficiency, with many modern chip designs focusing on integrating increasing numbers of processors on a single die instead of increasing the complexity of a smaller number of processors.[1–5] Many current and future computing applications, ranging from embedded Internet-of-Things devices to cloud datacenters, are placing increased emphasis on hardware solutions that provide high energy efficiency alongside high performance.[6]

Semiconductor fabrication technologies continue to provide increasing levels of integration,[7] offering opportunities for new architecture designs. However, increasing fabrication costs continue to motivate the development of programmable and/or reconfigurable architectures, which can address the needs of a range of applications in varying computing domains.

In this article, we discuss KiloCore, a chip containing a many-core programmable processor array for applications that exhibit fine-grained task-level parallelism. KiloCore addresses the aforementioned factors with a massively parallel computing platform that is energy efficient for a wide variety of workloads, capable of high performance, easily scalable to higher processor counts, and suitable for a range of applications and critical kernels, either acting alone or as a coprocessor in a heterogeneous system.

KiloCore Architecture

KiloCore consists of an array of 1,000 independently programmable processors along with 12 memory modules each containing 64 Kbytes (768 Kbytes total), connected in a mesh fabric.[8] Figure 1 displays a highlighted die photo, along with approximate

0272-1732/17/$33.00 © 2017 IEEE Published by the IEEE Computer Society

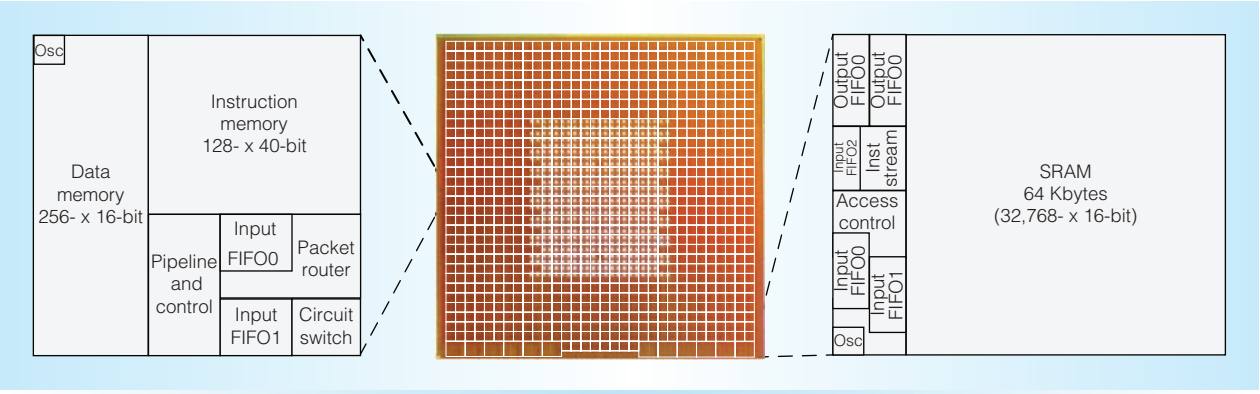

Figure 1. Die photo of the KiloCore chip, with borders between individual processors and memories highlighted. Approximated block layouts for a single processor (left) and a single independent memory (right) are shown.

block boundaries within the processor and memory tiles.

Processors

Each processor contains a 256- × 16-bit data memory, a 128- × 40-bit instruction memory, and a 16-bit datapath, and uses single-issue, in-order execution of memory-to-memory instructions.[9] Processors support 72 instruction types, branch prediction, predication, and loop acceleration, and include a multiply-accumulate unit with a 40-bit accumulator. Data types larger than 16 bits are supported through carry operations for add and subtract, or partial product accumulation for multiplies.

Independent Memories

Independent memory modules are located along the bottom of the array, with each module connecting to two neighboring processors and providing 64 Kbytes of storage. The memory can be used to source data or instructions. When sourcing instructions, the memory module takes over program control from a neighboring processor, replacing the standard 7-bit program counter with a 16-bit counter and extending the maximum size of a single program from 128 to 10,922 instructions.

Network

Communication between processors is handled by complementary circuit and packet networks. The circuit network is statically configured during programming to implement the most-trafficked communication paths, with any remaining traffic being transferred using the packet network. Each processor tile supports two circuit links and one packet link per side and per direction, with a circuit switch and a packet router located in each of the 1,000 tiles. Packet routers use wormhole routing, and both networks use source-synchronous communication.

Clocking

Globally asynchronous, locally synchronous clocking[10] is implemented in KiloCore, with each processor, packet router, and independent memory having its own local oscillator, for a total of 2,012 oscillators. These are self-timed ring oscillators that do not use phase-locked loops, contain configurable delay elements, are configured according to their local core's maximum operating frequency, and may independently halt or restart as they wish without requiring an external reference clock.

When cores are idle and waiting for work, they halt their local oscillator after a short delay, and restart it when work is available. An idle processor consumes zero active power, with leakage amounting to 1.1 percent of its typical active power. This low leakage is achieved through heavy use of high-threshold transistors in the design. This feature allows KiloCore to maintain energy-efficient operation when applications are not able to keep all processors supplied with work.

Chip

KiloCore was fabricated in a 32-nm partially depleted silicon on insulator technology.

The die occupies 64 mm^2, with the processor and memory array occupying 60 mm^2. KiloCore contains 621 million transistors. Some of the key measurements include the following:

- Processors, packet routers, and independent memories operate from a maximum voltage of 1.1 V down to minimum voltages of 560, 670, and 760 mV, respectively.
- Processors support an average clock rate of 1.24 GHz when operating at 0.9 V, increasing up to 1.78 GHz at 1.1 V, with similar clock rates for the independent memory modules.
- Circuit network links transfer up to 28.5 Gbits per second (Gbps) each, packet network links transfer up to 9.1 Gbps each, and the combined networks support a bisection bandwidth of 4.2 Tbits per second (Tbps) at 1.1 V.
- An individual processor consumes 17 mW when 100 percent active with a typical workload and operating at 0.9 V.

KiloCore's physical design was implemented in 34 days from access to the full design libraries to tape out. The prototype chip's processors, memories, and network are fully functional, except for hold-time violations on some network paths. The prototype chip uses stock packaging designed for a smaller die, which unfortunately delivers direct power to only the central portion of the array. At higher voltages and activities, processors on the outside of the array operate at reduced frequencies. Full array performance estimates are given assuming a custom package design that would not have this limitation.

Example Applications

Several applications have been implemented for KiloCore and expanded to use most of the array. Application performance is estimated by simulations that are cycle accurate within a core, use subcycle precision for core interactions, fully model varied per-core frequencies, and utilize subinstruction energy measurements. Application code has been lightly to moderately optimized, and additional effort would yield significant improvements. Performance is given for operation at 0.9 V.

An Advanced Encryption Standard (AES) engine is implemented with 974 processors for 128-bit keys. It supports a throughput of 14.5 Gbps while using 6.7 W.

A low-density parity-check (LDPC) decoder is implemented with 968 processors and 12 independent memories for a 4,095-bit code length. With four decoding iterations and a partial fifth for valid code-word detection, it has a throughput of 138 Mbits per second (Mbps) while using 4.1 W.

A 4,096-point complex fast Fourier transform (FFT) application is implemented with 980 processors and 12 independent memories, operating on 16-bit complex data. It transforms 565 MSamples per second using 4.1 W.

The first phase of an "external" record sort is implemented with 1,000 processors. Here, 100-byte records contain a 10-byte sorting key and are processed into sorted blocks of 185 Kbytes, in support of the second merging phase of the external sort. It sorts 1.47 Gbytes per second using 1.2 W.

Programming the Array

KiloCore is designed for high cooperation between processors, in which each processor executes a task of up to 128 instructions. Mapping an application to this architecture involves applying a series of task-partitioning transformations, wherein the final tasks are mappable to the processors. These transforms are loosely categorized as serial and parallel partitioning.

Application Task Partitioning

Serial partitioning transforms sections of code into a sequence of tasks that form a computation pipeline. Live variables at the code separation points are transferred between tasks using message passing. Variables can be transferred from producers to consumers directly, through intermediate tasks in the chain, or using a mixture of these methods. Partitioning can produce tasks with as little as one instruction that directly reads data from the network, performs an operation, and writes the result back to the network.

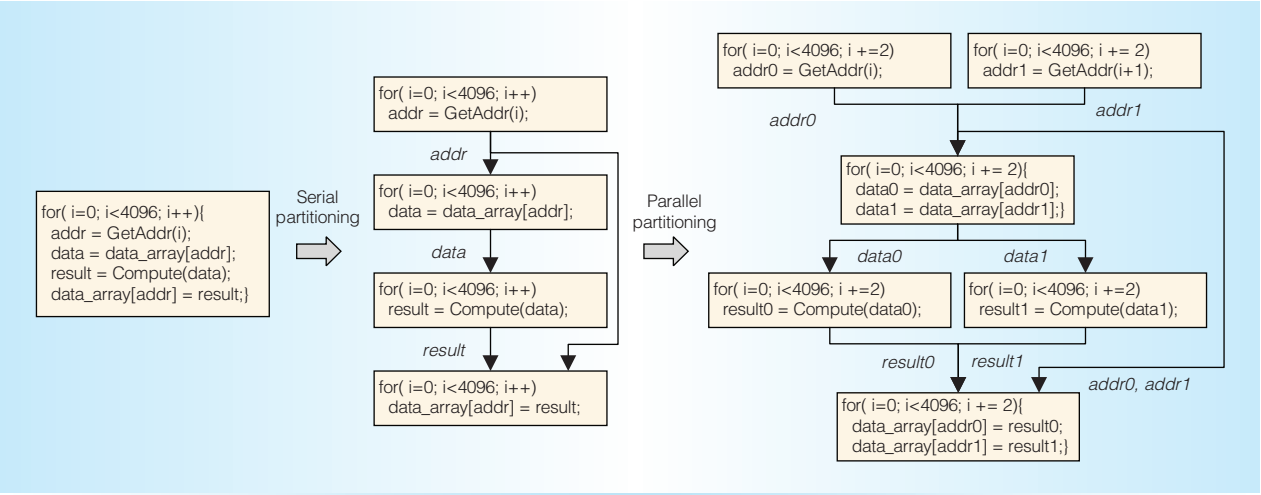

Figure 2. Example of serial and parallel task partitioning. Serial partitioning reduces instruction counts per task and isolates large data structures, whereas parallel partitioning improves the throughput of critical paths.

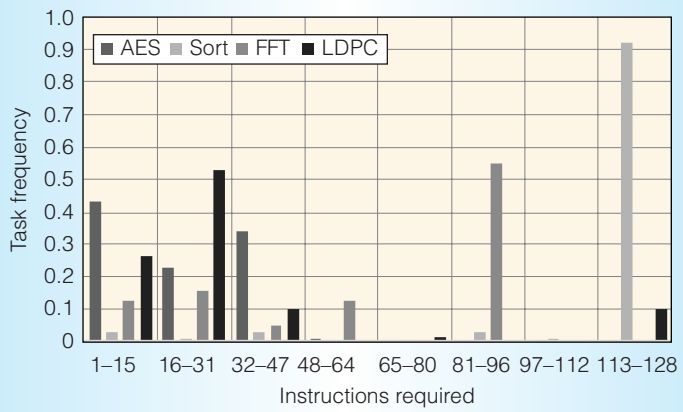

Figure 3. Number of instructions required by tasks in the example applications after task partitioning. All tasks fit within the 128-word instruction memory of a single processor. (AES: Advanced Encryption Standard; FFT: fast Fourier transform; LDPC: low-density parity check.)

Parallel partitioning performs task replication to increase the throughput of critical paths in the application that exhibit data parallelism. This transform is typically applied to loop bodies or is used to implement vector operations. This partitioning introduces overhead for splitting and joining the data being processed by the replicas, which can involve inserting additional data routing tasks if many replicas are formed. When task execution time varies significantly with the data, intelligent distribution can be used to supply data to tasks as they finish their computations.

Serial and parallel partitioning transformations are applied to the application multiple times, progressing from the original code to that which will be mapped to KiloCore. Figure 2 shows an example of partitioning a task that processes elements in a 4,096-element data array. This task is partitioned serially to isolate the data access tasks from the main workload, replicating the loop iterator to maintain correct task execution counts. Parallel partitioning is then applied to accelerate the address generation and data computation tasks, with appropriate loop count modifications.

Figure 3 shows the number of instructions required for tasks after partitioning was performed for the sampled applications. All tasks fit within the 128-word instruction memory of a KiloCore processor.

Figure 4 shows the amount of data memory required by these same tasks. 98.7 percent of tasks fit within the 512-byte data memory of a KiloCore processor. The remaining tasks include those that access larger data structures in the FFT and LDPC applications. These tasks are mapped to the 24 processors neighboring the 12 independent memories in KiloCore.

Task partitioning introduces overhead for intertask data transfers. This overhead is partially hidden in KiloCore by allowing instructions to directly access the network

links as part of their source and destination fields. In the sampled applications, after partitioning, communication overhead accounts for 30 percent of overall energy usage, including network energy, along with instructions for reading or writing the networks. This energy partially replaces that which would be spent on writing and subsequently reading variables from local data memories. In some situations, partitioning will directly lower application energy, as a variable transferred using the circuit network over a short distance requires as little as 25 percent of the energy used when storing and reading the same variable from a local data memory.

Figure 5a shows throughput scaling for the sampled applications as the core count increases. Figure 5b shows the corresponding energy efficiency of the applications. AES, FFT, and LDPC show approximately linear growth in throughput with core count, with energy efficiency remaining steady when going to large numbers of cores. We omit the Sort algorithm here because it uses additional cores to increase the size of sorted blocks, and the amount of work being done at different core counts is not directly comparable.

Programming

Program code is written in C++ or assembly language. We use a many-core simulator, written in C++ and customized for KiloCore, to verify program correctness during development, estimate performance, and generate code profiles to be used for optimization.

Early profiling, using unpartitioned or partially partitioned task code, is used to gather code execution statistics that identify hot spots and help guide the partitioning effort. The profile includes network traffic measurements to aid in assigning core links to the circuit or packet networks.

Once the final partitioned task code has been generated, the tasks are mapped to processors using an automated mapping tool. This tool includes considerations such as avoidance of faulty or partially functional processors; optimizations to take advantage of process, voltage, and temperature variations; self-healing for failures due to wear-out effects; and organization of packet traffic to reduce congestion and energy usage.

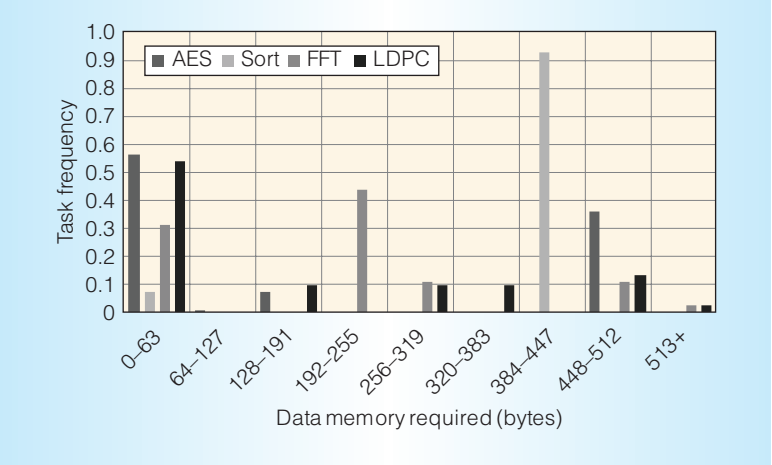

Figure 4. Amount of data memory required by tasks in the example applications after task partitioning. Most tasks fit within the 512-byte data memory of a single processor, with a small number of tasks requiring the assistance of the independent memory modules.

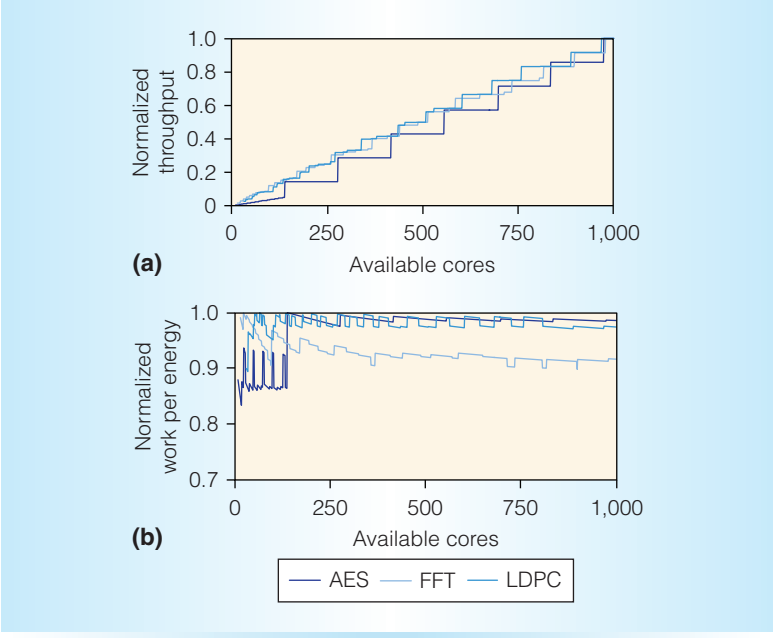

Figure 5. Normalized application (a) throughput and (b) energy efficiency as the number of cores available to the application is increased to 1,000.

Network Design

An important consideration for many-core architectures is how to transfer data between cores in a manner that is energy efficient, avoids network congestion, and supports intertask synchronization.

An application's communication requirements depend heavily on the implementation method. In our sampled applications,

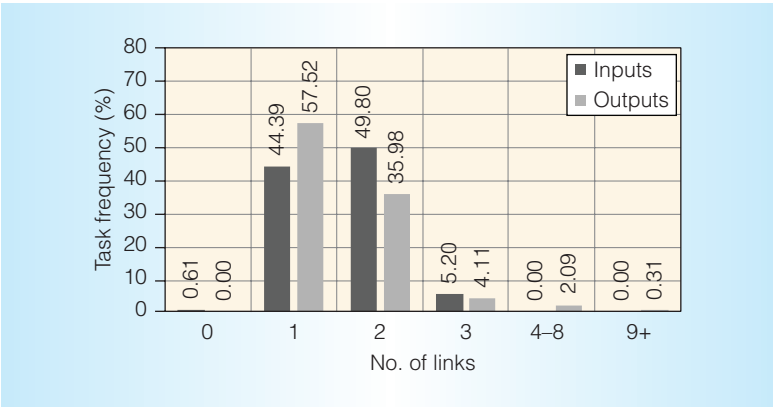

Figure 6. Across sampled applications, partitioned tasks exhibit a low density of intertask communication, with a large majority of tasks having fewer than three inputs and fewer than three outputs.

algorithms were chosen that, once partitioned into fine-grained tasks, exhibited low densities of intertask communication links. When mapping one task to each processor, on average only 0.15 percent of possible links are used, as compared to maximally dense all-to-all communication.

In Figure 6, tasks are categorized by their number of required input or output connections. The figure shows that 95 percent of the tasks have a fan-in of two or fewer, and 93 percent have a fan-out of one or two. This result is partially influenced by the partitioning algorithms used, which favor reducing the number of links needed.

KiloCore uses complementary circuit and packet networks to efficiently support these links. The low-area, low-energy, high-throughput circuit network supports two inputs and up to eight outputs per processor. This network supports 95 percent of links in the sampled applications. The remaining links are assigned to the packet network, which is designed for medium throughput to reduce packet router area overhead. The packet network is also used to support general administrative signaling.

Task synchronization occurs during communication, with any network write instruction being matched to a read instruction at the destination. Processors attempting to read an unavailable input will pause and wait for the data to arrive. Interprocessor first-in, first-out data buffers[11] hold 32 words each and allow a transmitting processor to continue its program after a write, forcing a pause only when a buffer is full.

It is near certain that more chips with 1,000 or more independent processors will be designed and built in the future.[6,12] These chips will need to consider challenging questions regarding how these processors will communicate and synchronize with each other and with external system components, as well as how software will be developed for them, in light of their targeted applications. Our KiloCore chip explores some answers to these questions, demonstrating the feasibility and potential advantages of these many-core architectures. MICRO

Acknowledgments

We gratefully acknowledge support from DoD and ARL/ARO grant W911NF-13-1-0090; NSF grants 0903549, 1018972, 1321163, and CAREER Award 0546907; SRC GRC grants 1971 and 2321, and CSR grant 1659; and C2S2 grant 2047.

References

1. Z. Yu et al., "AsAP: An Asynchronous Array of Simple Processors," *IEEE J. Solid-State Circuits*, vol. 43, no. 3, 2008, pp. 695–705.
2. S. Vangal et al., "An 80-Tile Sub-100-W TeraFLOPS Processor in 65-nm CMOS," *IEEE J. Solid State Circuits*, vol. 43, no. 1, 2008, pp. 29–41.
3. D.N. Truong et al., "A 167-Processor Computational Platform in 65 nm CMOS," *IEEE J. Solid-State Circuits*, vol. 44, no. 4, 2009, pp. 1130–1144.
4. S. Bell et al., "TILE64 Processor: A 64-Core SoC with Mesh Interconnect," *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2008, pp. 88–89.
5. M. Butts and A.M. Jones, "TeraOPS Hardware and Software: A New Massively-Parallel, MIMD Computing Fabric IC," *Proc. IEEE Hot Chips Symp.*, session 5, 2006.
6. K. Kim, "Silicon Technologies and Solutions for the Data-Driven World," *IEEE Int'l Solid-State Circuits Conf.*, 2015, pp. 1–7.
7. W.M. Holt, "Moore's Law: A Path Going Forward," *Proc. IEEE Int'l Solid-State Circuits Conf.*, 2016, pp. 8–13.

8. B. Bohnenstiehl et al., "KiloCore: A 32 nm 1000-Processor Array," *Proc. IEEE Hot Chips Symp. High-Performance Chips*, session 8, 2016.
9. B. Bohnenstiehl et al., "A 5.8 pJ/Op 115 Billion Ops/Sec, to 1.78 Trillion Ops/Sec 32 nm 1000-Processor Array," *Proc. Symp. VLSI Circuits*, 2016, pp. 1–2.
10. D.M. Chapiro, "Globally-Asynchronous Locally-Synchronous Systems," PhD dissertation, Computer Science Dept., Stanford Univ., 1984.
11. R.W. Apperson et al., "A Scalable Dual-Clock FIFO for Data Transfers between Arbitrary and Haltable Clock Domains," *IEEE Trans. VLSI Systems*, vol. 15, no. 10, 2007, pp. 1125–1134.
12. S. Borkar, "Thousand Core Chips: A Technology Perspective," *Proc. 44th Ann. Design Automation Conf.*, 2007, pp. 746–749.

Brent Bohnenstiehl is a PhD student in electrical and computer engineering at the University of California, Davis. His research interests include processor architecture, VLSI design, hardware–software codesign, dynamic voltage and frequency scaling algorithms and circuits, and many-core compilers and other programming and simulation tools. Bohnenstiehl received a BS in electrical and computer engineering from the University of California, Davis. Contact him at bvbohnen@ucdavis.edu.

Aaron Stillmaker is an assistant professor in the Electrical and Computer Engineering Department at California State University, Fresno. His research interests include many-core processor architecture, many-core applications, and VLSI design. Stillmaker received a PhD in electrical and computer engineering from UC Davis, where he completed the work for this article. Contact him at astillmaker@ucdavis.edu.

Jon Pimentel is a PhD candidate in electrical and computer engineering at the University of California, Davis. His research interests include floating-point architectures, VLSI design, synthetic aperture radar imaging, scientific applications, and many-core processor architecture. Pimentel received an MS in electrical and computer engineering from the University of California, Davis. Contact him at jjpimentel@ucdavis.edu.

Timothy Andreas is a PhD student in electrical and computer engineering at the University of California, Davis. His research interests include energy-efficient and high-performance machine learning algorithms and processor architectures. Andreas received a BS in electrical and computer engineering from the University of California, Davis. Contact him at tjandreas@ucdavis.edu.

Bin Liu is a software engineer on the machine learning and risk team at Uber. His research interests include high-performance many-core processor architecture, dynamic supply voltage and frequency scaling algorithms and circuits, and parallel encryption engine implementations. Liu received a PhD in electrical and computer engineering from UC Davis, where he completed the work for this article. Contact him at binliu@ucdavis.edu.

Anh Tran is a lead hardware research engineer at Cavium. His research interests include VLSI designs, multicore architectures, on-chip interconnects, and reconfigurable systems on chip. Tran received a PhD in electrical and computer engineering from UC Davis, where he completed the work for this article. Contact him at anhtr@ucdavis.edu.

Emmanuel Adeagbo is a physical design engineer with Intel Platform Engineering Group's Big Core division. His research interests include energy-efficient regular expression applications, VLSI, and digital architecture design. Adeagbo received an MS in electrical and computer engineering from UC Davis, where he completed the work for this article. Contact him at eoadeagbo@ucdavis.edu.

Bevan Baas is a professor in the Electrical and Computer Engineering Department at the University of California, Davis. He leads projects in architectures, hardware, applications, and software tools for VLSI computation. Baas received a PhD in electrical engineering from Stanford University. Contact him at bbaas@ucdavis.edu.