Evaluating Celerity: A 16-nm 695 Giga-RISC-V Instructions/s Manycore Processor With Synthesizable PLL

Austin Rovinski, Chun Zhao, Khalid Al-Hawaj, Paul Gao, Shaolin Xie, Christopher Torng, Scott Davidson, Aporva Amarnath, Luis Vega, Bandhav Veluri, Anuj Rao, Tutu Ajayi, Julian Puscar, Steve Dai, Ritchie Zhao, Dustin Richmond, Zhiru Zhang, Ian Galton, Christopher Batten, Michael B. Taylor, and Ronald G. Dreslinski

***Abstract*—This letter presents a 16-nm 496-core RISC-V network-on-chip (NoC). The mesh achieves 1.4 GHz at 0.98 V, yielding a peak throughput of 695 Giga RISC-V instructions/s (GRVIS), a peak energy efficiency of 314.89 GRVIS/W, and a record 825 320 CoreMark benchmark score. Unlike previously reported [1], this new score was obtained without modifying the core benchmark code. The main feature is the NoC architecture, which uses only 1881 μm^2 per router node, enables highly scalable and dense compute, and provides up to 361 Tb/s of aggregate bandwidth.**

***Index Terms*—Celerity, manycore, network-on-chip (NoC), RISC-V.**

I. Introduction

Complex, data-parallel workloads continue to push toward edge devices, such as mobile and Internet-of-Things (IoT) platforms. In particular, streaming-based workloads like real-time computer vision and machine learning are steadily increasing in demand. Mobile devices demand high energy efficiency to attempt these computationally intensive workloads. At the same time, the hardware must remain flexible to perform state-of-the-art algorithms as well as workloads that emerge post-fabrication. Prior manycore architectures [2]–[4] that target streaming workloads have yielded high area and energy efficiencies (Table I). However, much of the die area for these architectures were dedicated toward the network-on-chip (NoC), including cache-coherence protocol controllers, which restricts potential compute density and efficiency. We demonstrate a novel NoC architecture that enables fast internode communication with significantly reduced die area (2.5×–44×) compared to prior work. The processor is composed of a 496-core array of 5-stage, in-order RISC-V RV32IM cores in a mesh configuration (Fig. 1). It achieves a peak of 695 Giga RISC-V instructions/s (GRVIS), and a record 825 320 CoreMark benchmark score.

II. Manycore Architecture

In order to achieve a high compute density, the network architecture (Fig. 1) differs significantly from a traditional coherent shared-memory model as described in this section.

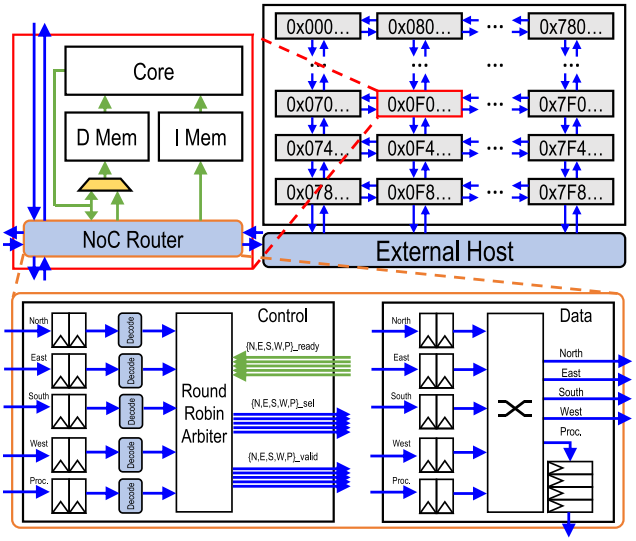

Fig. 1. Manycore mesh architecture with callouts to an individual tile and a tile's router architecture.

A. Partitioned Global Address Space

Instead of caches, the manycore processor has a partitioned global physical address space across all network nodes. Fig. 1 illustrates the manycore mesh, and the memory address mapping across nodes in the network using a 32-bit addressing scheme. This mapping extends to nodes below the bottom edge of the network to allow messages to be sent in and out of the network. For Celerity, we use asynchronous buffers to communicate with four 64-bit general-purpose RISC-V (RV64G) cores as hosts. These host cores are capable of running a full operating systems such as Linux.

The use of partitioned global address space (PGAS) allows significant area improvement over a traditional shared memory system. Fig. 2 shows the area overhead of a directory-based coherent cache versus Celerity's PGAS system, demonstrating that PGAS offers over a 20× reduction in area overhead. The comparison system breakdown was extracted from Celerity's RV64G control cores, with directory area conservatively estimated from Sanchez and Kozyrakis [5]. The cost of removing these structures is mainly the ease of programming that comes from shared memory. However, streaming and highly parallel workloads often have well-defined dataflow patterns, which can enable compilers to manage data movement and mitigate this cost. Such strategies are discussed further in Section IV.

B. Remote Store Programming

The mesh uses the remote store programming (RSP) model [6] to send messages over the network. As opposed to a shared memory

Manuscript received August 17, 2019; revised October 9, 2019 and October 30, 2019; accepted November 4, 2019. Date of publication November 18, 2019; date of current version December 20, 2019. This article was approved by Associate Editor Xin Zhang. This work was supported in part by DARPA under Award HR0011-16-C-0037, and in part by NSF under Award 1059333, Award 1512937, Award 1563767, Award 1565446, and Award 1337240. *(Corresponding author: Austin Rovinski.)*

A. Rovinski, A. Amarnath, T. Ajayi, and R. G. Dreslinski are with the Electrical Engineering and Computer Science Department, University of Michigan, Ann Arbor, MI 48109 USA (e-mail: rovinski@umich.edu).

C. Zhao, P. Gao, S. Xie, S. Davidson, L. Vega, B. Veluri, D. Richmond, and M. B. Taylor are with the Paul G. Allen School of Computer Science & Engineering, University of Washington, Seattle, WA 98195 USA.

K. Al-Hawaj, C. Torng, S. Dai, R. Zhao, Z. Zhang, and C. Batten are with the School of Electrical and Computer Engineering, Cornell University, Ithaca, NY 14853 USA.

A. Rao, J. Puscar, and I. Galton are with the Electrical and Computer Engineering Department, University of California at San Diego, La Jolla, CA 92093 USA.

Digital Object Identifier 10.1109/LSSC.2019.2953847

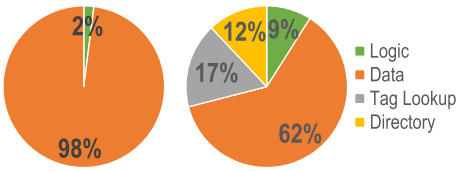

Fig. 2. Area breakdown for Celerity's PGAS memory system (left) versus a comparable directory-based coherent cache (right).

model where a node can load or store to any address, RSP disallows loads from remote memory. A node can freely load or store to its local memory, but can only perform stores to remote memory. Using RSP both reduces router area (10% less than a router with remote loads) and prevents pipeline stalls associated with long-latency remote loads. Along with using two physical networks and dimension-ordered routing, RSP guarantees deadlock-free, in-order delivery.

C. Single-Flit Packets

Celerity implements a different flow control scheme compared to prior work. While wormhole routing is common due to its relative efficiency, it still has inefficiencies related to packet ingestion in the network. Most wormhole schemes require head and/or tail flits to reserve routes and communicate metadata. This results in network overhead, as sending a single data flit results in 2 to 3 flits being injected into the network. In addition, wormhole routing can cause head-of-line blocking when one packet's route reservation conflicts with another.

Celerity instead implements a single-flit packet protocol, where the command, address, and data of a packet is contained in a single flit. This flow control scheme offers several benefits over wormhole routing.

1) No head or tail flits—no overhead flits in a packet.
2) Head-of-line blocking is not possible as routes are not reserved (congestion can still occur).
3) Small core-to-core latency, especially for adjacent cores.
4) An in-order pipeline can execute one store per cycle, because a store injects only one flit into the network.

The single-flit flow control scheme is further discussed in Section VI with comparisons to prior work.

III. Manycore Implementation

Fig. 3 shows the layout of a single tile, which contains a "Vanilla-5" core and the routing logic for that node. A core implements the 32-bit RISC-V base instruction set and the multiply/divide extension (RV32IM) in a 5-stage pipeline. Each tile contains 2×4 KB SRAMs for instruction/data memories (IMEM/DMEM), and a 32-entry, 32b register file implemented using two 1r1w latch-based memories. The router is a single-stage design, allowing it to arbitrate, route, and send flits in a single cycle. In addition to providing low latency, the area of the router is reduced over a multistage design. Because there are no pipeline registers between nodes, flits take only 1 cycle per hop. Two-element FIFOs are used at the input for each direction to hold packets in case of congestion. To implement both rate limiting and memory fences, we use a source-controlled credit counter. The credit counter is decremented on each packet injected into the network from a remote store, and incremented when a remote store completes. Credits are returned over a separate 9-bit NoC with the same architecture as in Fig. 1. The per-module physical area breakdown is listed in Fig. 3, with the NoC occupying only 1881 μm^2 (7.8%) of the tile. The router supports 80b transfers per cycle, which packages data, address, and commands into a single flit. The router

Cell Type	Area (μm^2)	%
IMEM	6691	27.59
DMEM	6691	27.59
RF	2008	8.28
Core logic	2473	10.20
ALU	485	2.00
Div	412	1.70
Mult	301	1.24
Pipeline/other	1275	5.26
NoC	1881	7.76
Endpoint FIFO	303	1.25
Credit counter	23	0.09
Router	1555	6.41
Endcap/welltap	281	1.16
Filler	1635	6.74
Unutilized	2591	10.68
Total	24251	100.00

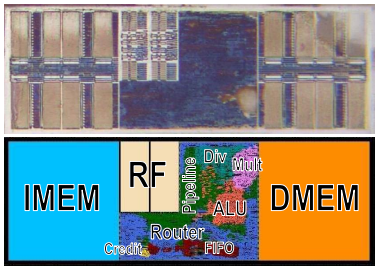

Fig. 3. (Left) Physical area breakdown of each manycore tile. (Top right) Tile die photograph. (Bottom right) Tile floorplan.

and core run on the same clock domain up to 1.4 GHz, allowing each tile to both transfer 750 Gb/s and process 1.4 GRVIS. Several gaps were created between rows of tiles to allow for ESD cells and in-cell overlays (ICOVL) as required for fabrication. The total die area of the manycore is 15.25 mm^2 as fabricated with ESD and ICOVL (or 12.03 mm^2 without). This yields an area efficiency of 45.57 GRVIS/mm^2 (57.77 GRVIS/mm^2).

IV. Porting Workloads

Software programs are compiled using a different workflow from shared memory systems. Because each tile is a RISC-V core, C/C++ programs can be compiled using the standard RISC-V toolchains. We use a custom GCC linker script which maps data and instructions to separate 4-KB segments such that instructions and data may be placed into the respective IMEM and DMEM. When compiling, the program must target a single tile and fit within a tiles IMEM/DMEM. For single-program, multiple-data (SPMD) class programs, the same program can simply be replicated across each tile in the mesh. Larger programs can be constructed by partitioning instructions across tiles and explicitly passing data between them. For example, a large program can be split into multiple program segments. Each segment is stored in a different tile's IMEM, and data is passed between tiles. In the case of streaming applications, this works particularly well for separating consumer and producer functions across tiles and streaming data between them. Infrastructures have been developed to simplify compiling such workloads, such as StreamIt [7], an infrastructure to automatically partition programs using annotations, and `bsg_manycore_lib`, our library for sending, receiving, and synchronizing data across tiles. These infrastructures can allow programmers to write code segments, annotate dependencies, and allow compilers/libraries to orchestrate the data transfer.

A. CoreMark

The primary workload we use to benchmark our processor is CoreMark, a computationally intensive benchmark that stresses pipeline performance. We port CoreMark to the manycore platform by starting with the "barebones" implementation provided by EEMBC. With this implementation, we create a simple linker script to identify which functions to distribute to the manycore tiles versus the functions to run on the host control cores. We then use CoreMark's parallelization interface to load the program binaries to all manycore tiles and run the program. The CoreMark benchmark enumerates the criteria to submit a valid CoreMark score, which we adhere to. Unlike previously reported [1], the scores we report in Section VI do not use modified core benchmark code. A change in compiler version

TABLE I
COMPARISON TO RELATED WORKS

	ISSCC '08 [2]	HPCA '18 [3]	JSSC '17 [4]	ESSCIRC '14 [11]	This work
ISA	VLIW	SPARC V9	RISC	RISC-V	RISC-V
Datapath Width	32-bit	64-bit	16-bit	64-bit	32-bit
Technology	90nm Planar	32nm SOI	32nm SOI	45nm SOI	16nm FinFET
Voltage	0.90 - 1.30 V	0.80 - 1.20 V	0.67 - 1.10 V	0.65 - 1.20 V	0.60 – 0.98 V
Area[a]	232.16 mm^2	29.37 mm^2	57.41 mm^2	3.08 mm^2	15.25 (12.03[c]) mm^2
Normalized Area[ab]	32.65 mm^2	14.08 mm^2	27.52 mm^2	0.69 mm^2	15.25 (12.03[c]) mm^2
Normalized NoC Router Area[ab]	~82894 μm^2 (5x32 bit)	16214 μm^2 (3x64 bit)	4784 μm^2 (16 + 2x16 bit)	–	**1881 μm^2** (80 + 9 bit)
Cores (Threads)	64 (64)	25 (50)	1000 (1000)[d]	2 (2)	496 (496)
Frequency	750 MHz	500 MHz	1770 MHz	200 - 1300 MHz	10 - 1400 MHz
Power	10.8 W	2 W	39.6 W[d]	0.96 W	7.47 W
Norm. Throughput[e]	144 GOPS	5 GOPS	885 GOPS	5.2 GRVIS	695 GRVIS
Network Aggregate Bandwidth[f]	33.79 Tb/s	11.33 Tb/s	53.4 Tb/s (wormhole) 335 Tb/s (circuit)	–	361 Tb/s
Network Bisection Bandwidth[g]	1.92 Tb/s	0.96 Tb/s	0.58 Tb/s (wormhole) 3.65 Tb/s (circuit)	–	4.00 Tb/s
Routing Model	Wormhole	Wormhole	Wormhole+circuit	–	Single-flit packet
Energy Efficiency[e]	13.33 GOPS/W	2.50 GOPS/W	22.35 GOPS/W[d]	5.42 GRVIS/W	**93.04 GRVIS/W**
Normalized Area Efficiency[abe]	4.41 GOPS/mm^2	0.36 GOPS/mm^2	32.16 GOPS/mm^2	7.54 GRVIS/mm^2	**45.57 (57.77[c])** GRVIS/mm^2

[a] Area only includes die area allocated to tiles
[b] Area normalized to 16nm based on Contacted Poly Pitch (CPP) scaling
[c] Excluding ESD and ICOVL area
[d] KiloCore can only power 160 cores from its package. Power extrapolated to 1000 cores
[e] Throughput normalized to 32-bit GOPS/GRVIS
[f] Network Aggregate Bandwidth = (# usable links) * (link bandwidth)
[g] Network Bisection Bandwidth = (min. # links cut to bisect network) * (link bandwidth)

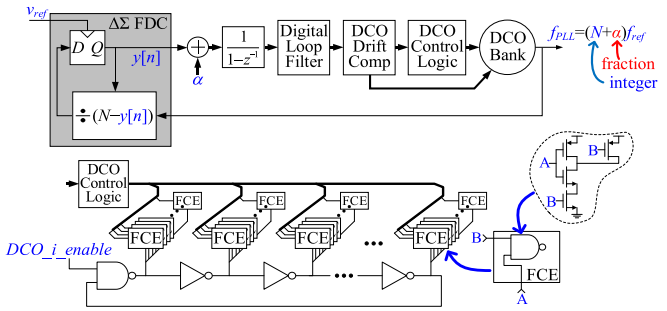

Fig. 4. Synthesizable PLL architecture.

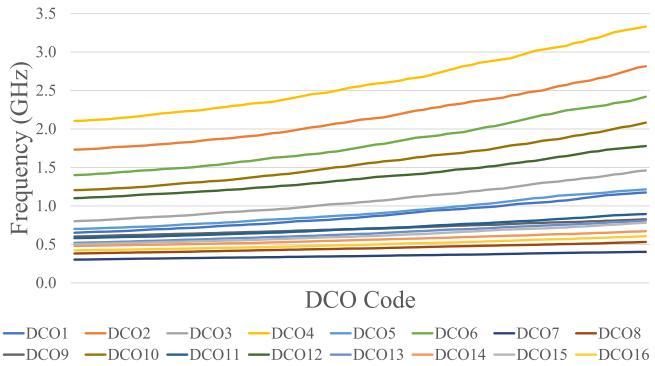

Fig. 5. PLL DCO code versus simulated output frequency.

and compiler flags allowed us to fit the benchmark within a single tile's IMEM, as well as modestly improve the score.

V. DIGITAL PLL

The manycore clock is supplied by a custom, fully synthesized, and automatically placed and routed clock generator. It operates from an isolated 0.8 V supply and occupies 5898 μm^2. With a reference frequency of $f_{\text{ref}} = 26$ MHz, its output frequency is tunable from 10 MHz to 3.3 GHz with minimum increments of no more than 2%, and consumes 1.5–3.5 mW at the min and max frequencies, respectively. The PLL achieves a (simulated worst-case) period jitter of 2.5 ps. Jitter was obtained using a bit-exact, event-driven simulation which accounts for phase noise. The simulation forgoes supply noise, as the design was done in parallel to the SoC before supply characteristics were known. However, the synthesizable architecture was created to be tolerant of supply noise. The PLL locks both frequency and phase with a simulated worst-case lock time of 230 µs.

The clock generators PLL core (Fig. 4) consists of a first-order $\Delta\Sigma$ frequency-to-digital converter [8], an α adder, a frequency-to-phase accumulator, a digital low pass loop filter, a DCO drift compensation logic, a DCO control logic, and a bank of 16 DCOs. The 16 DCOs together cover a frequency range of 1.3–3.3 GHz, and only one DCO is enabled for each output frequency setting. Each DCO (Fig. 4) is a ring oscillator wherein each inverting delay element is loaded with a bank of NAND gate frequency control elements (FCEs) [9]. We target a 50% frequency range overlap above and below for each DCO in order to margin against process, voltage, and temperature variation (Fig. 5). The DCO drift compensator dynamically controls 37 of the FCEs to compensate for drift of the DCOs center frequency over temperature and supply. The DCO control logic partitions its input into integer and fractional parts. The integer part drives all but 8 of the remaining FCEs with an update rate of f_{ref}. The fractional part is oversampled by a second-order $\Delta\Sigma$ modulator followed by a dynamic element matching encoder, the output of which drives the final 8 FCEs.

Ur Rahman *et al.* [10] proposed a similar architecture to this letter, however a key distinction is that this letter uses NAND gates as loading elements to vary node capacitance, whereas ur Rahman *et al.* use inverters in parallel to vary drive current. NAND gate loading is compatible with synthesis tools, whereas parallel driving cells are usually not, due to a lack of tristate devices in most digital cell libraries.

VI. EXPERIMENTAL RESULTS

To validate the manycore processor, we run CoreMark distributed across all cores simultaneously. CoreMark is structured as self-validating benchmark: each iteration depends on the previous iteration and a hash of the final state is used to check correctness. Fig. 6 identifies the operating configurations where CoreMark reports a correct result for all tiles. The processor achieves a max throughput of 695 GRVIS at 1.4 GHz and 0.98 V—the highest single-chip RISC-V throughput to date—and a max energy efficiency of 314.89 GRVIS/W at 500 MHz and 0.60 V. It achieves a record CoreMark score of 825 320, outperforming the next best score by more than 2×, as well as our previously reported score [1] by a small margin. Our evaluation uses GRVIS as a measure of performance because it signifies compliance with the RISC-V ISA. A custom ISA can increase efficiency by tailoring instructions, but this extricates the architecture

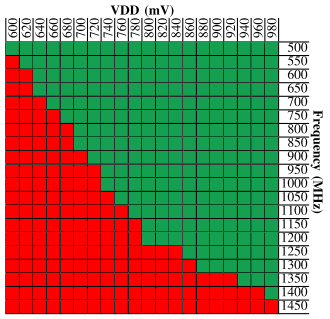

Fig. 6. Shmoo plot of operation points.

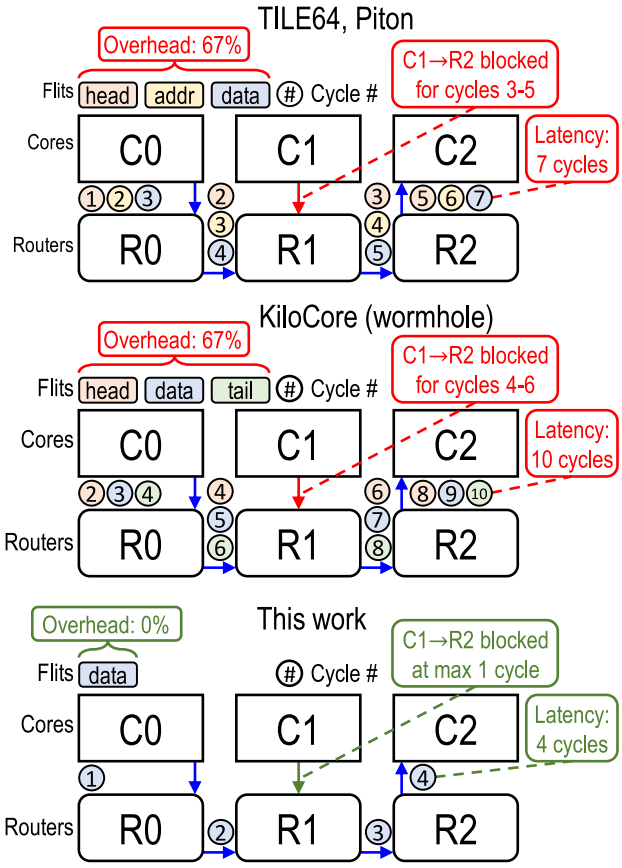

Fig. 7. Example of sending a packet with one data flit from core 0 to core 2 for each flow control model.

from the benefits of open-source software and toolchains. In our comparison, we quantify non-RISC-V performance with giga-operations per second (GOPS). For direct comparisons, GRVIS can be treated as GOPS. Table I compares our work against prior manycore works. In most metrics, this letter compares very favorably against related works. Celerity exceeds all compared works for normalized NoC area (2.5×–44×), area efficiency (1.8×–160×), and energy efficiency (4.2×–37×). Throughput measurements are normalized to 32-bit operations, under the optimistic assumption that two 16-bit operations are equivalent to one 32-bit operation.

Table II provides a comparison of our single-flit flow control model versus the related work, and Fig. 7 provides an example of each flow control model sending one flit of data to another node. Our model allows our network to outperform prior works in both packet throughput and latency for small data transfers. The difference in latency between the models diminishes toward larger data transfers, however

TABLE II
COMPARISON OF FLOW CONTROL MODELS

Work	Routing model	Arbitrary destination	Head-of-line blocking	Packet throughput	Min. latency (cycles)	Overhead flit fraction
TILE64[1], Piton[2]	Wormhole	Yes	Yes	0.33 / cycle	h + n + t + 1	2 / (n + 1)
KiloCore[3]	Wormhole	Yes	Yes	0.33 / cycle	2h + n + 1*	2 / (n + 2)
	Circuit switched	No	N/A+	Not Reported	Not Reported	Not Reported
This work	Single-flit packet	Yes	No	1 / cycle	h + n - 1	0

h hops, n data flits, t turns in the network path. Core↔router is 1 hop.
* KiloCore's network is GALS and requires synchronization for each hop.
+ KiloCore's circuit switch NoC can only be reprogrammed during the processor configuration phase

data streaming workloads favor smaller transfer sizes with smaller latency in order to allow processing at the next node sooner. Critical paths in the design lie in both the core and NoC, although experiments show that the NoC tends to be the limitation on frequency. In terms of impact on improvement over related work, the NoC and core architecture both contribute significantly.

KiloCore [4] modestly exceeds this letter in network aggregate and bisection bandwidth, although a majority of its bandwidth comes from the statically routed circuit-switched network. KiloCore also uses only 1.1 KB memory per tile, whereas Celerity uses 8 KB per tile. In terms of RISC-V performance, Lee *et al.* [11] reported state-of-the-art in single-chip GRVIS throughput, which we outperform by 267×.

ACKNOWLEDGMENT

This letter employed a BaseJump ASIC Motherboard; the bringup effort was partly funded by the DARPA/SRC JUMP ADA center.

REFERENCES

[1] A. Rovinski *et al.*, "A 1.4 GHz 695 Giga Risc-V inst/s 496-core manycore processor with mesh on-chip network and an all-digital synthesized PLL in 16nm CMOS," in *Proc. Symp. VLSI Circuits*, Jun. 2019, pp. C30–C31.
[2] S. Bell *et al.*, "TILE64-processor: A 64-core SoC with mesh interconnect," in *IEEE Int. Solid-State Circuits Conf. Dig. Tech. Papers*, Feb. 2008, pp. 88–89.
[3] M. McKeown *et al.*, "Power and energy characterization of an open source 25-core manycore processor," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit. (HPCA)*, Feb. 2018, pp. 762–775.
[4] B. Bohnenstiehl *et al.*, "KiloCore: A 32-nm 1000-processor computational array," *IEEE J. Solid-State Circuits*, vol. 52, no. 5, pp. 891–902, Apr. 2017.
[5] D. Sanchez and C. Kozyrakis, "SCD: A scalable coherence directory with flexible sharer set encoding," in *Proc. IEEE Int. Symp. High Perform. Comput. Archit.*, Feb. 2012, pp. 1–12.
[6] H. Hoffmann, D. Wentzlaff, and A. Agarwal "Remote store programming," in *Proc. High Perform. Embedded Archit. Compilers*, Jan. 2010, pp. 3–17.
[7] W. Thies, M. Karczmarek, and S. Amarasinghe, "StreamIt: A language for streaming applications," in *Proc. Int. Conf. Compiler Constr.*, Apr. 2002, pp. 179–196.
[8] R. Beards and M. Copeland, "An oversampling delta-sigma frequency discriminator," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process.*, vol. 41, no. 1, pp. 26–32, Jan. 1994.
[9] P.-L. Chen, C.-C. Chung, and C.-Y. Lee, "A portable digitally controlled oscillator using novel varactors," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 52, no. 5, pp. 233–237, May 2005.
[10] F. ur Rahman, G. Taylor, and V. Sathe, "A 1–2 GHz computational-locking ADPLL with sub-20-cycle locktime across PVT variation," *IEEE J. Solid-State Circuits*, vol. 54, no. 9, pp. 2487–2500, Sep. 2019.
[11] Y. Lee *et al.*, "A 45nm 1.3GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators," in *Proc. 40th Eur. Solid-State Circuits Conf. (ESSCIRC)*, Sep. 2014, pp. 199–202.