REDUCT: Keep it Close, Keep it Cool!

Efficient Scaling of DNN Inference on Multi-core CPUs with Near-Cache Compute

Anant V. Nori[†], Rahul Bera[*], Shankar Balachandran[†], Joydeep Rakshit[†], Om J. Omer[†],
Avishaii Abuhatzera[‡], Belliappa Kuttanna[‡] and Sreenivas Subramoney[†]

[†]Processor Architecture Research Lab, Intel Labs, Bangalore (Corresponding Author: anant.v.nori@intel.com)
[*]ETH Zürich (work was done while at Intel)
[‡]Intel Corporation

***Abstract*—Deep Neural Networks (DNN) are used in a variety of applications and services. With the evolving nature of DNNs, the race to build optimal hardware (both in datacenter and edge) continues. General purpose multi-core CPUs offer unique attractive advantages for DNN *inference* at both datacenter [60] and edge [71]. Most of the CPU pipeline design complexity is targeted towards optimizing general-purpose single thread performance, and is overkill for relatively simpler, but still hugely important, data parallel DNN inference workloads. Addressing this disparity efficiently can enable both raw performance scaling and overall performance/Watt improvements for multi-core CPU DNN inference.**

We present REDUCT, where we build innovative solutions that bypass traditional CPU resources which impact DNN inference power and limit its performance. Fundamentally, REDUCT's *"Keep it close"* policy enables consecutive pieces of work to be executed close to each other. REDUCT enables *instruction delivery/decode close to execution* and *instruction execution close to data*. Simple ISA extensions encode the fixed-iteration count loop-y workload behavior enabling an effective bypass of many power-hungry front-end stages of the wide Out-of-Order (OoO) CPU pipeline. Per core performance scales efficiently by distributing light-weight tensor compute near all caches in a multi-level cache hierarchy. This maximizes the cumulative utilization of the existing architectural bandwidth resources in the system and minimizes movement of data.

Across a number of DNN models, REDUCT achieves a 2.3× increase in `convolution` performance/Watt with a 2× to 3.94× scaling in raw performance. Similarly, REDUCT achieves a 1.8× increase in `inner-product` performance/Watt with 2.8× scaling in performance. REDUCT performance/power scaling is achieved with no increase to cache capacity or bandwidth and a mere 2.63% increase in area. Crucially, REDUCT operates entirely within the CPU programming and memory model, simplifying software development, while achieving performance similar to or better than state-of-the-art Domain Specific Accelerators (DSA) for DNN *inference*, providing fresh design choices in the AI era.

I. Introduction

New data-centric paradigms of compute have made machine learning (ML) and Deep Neural Networks (DNN) pervasive in all fields of human endeavor. The race to build hardware for DNN execution continues with custom Domain Specific Accelerators (DSA), programmable FPGAs, and general-purpose GPUs and CPUs all throwing their hats in the ring [63] [39]. The goodness of these hardware platforms is judged on multiple parameters including performance (throughput/latency), power, area, cost, deployment form-factor, software ecosystem and programmability. While each hardware platform has innate goodness on a subset of these parameters, none are currently optimal on all of them. Industry and academic research and development continues for each of these hardware platforms to improve their offerings on all fronts.

CPU vendors continue to invest on both the hardware and software fronts towards improving DNN inference. Both Intel and Apple have announced new compute directly targeting matrix operations through Advanced Matrix Extension (AMX) [12], [27]. In fact, despite having a dedicated Neural Accelerator, Apple A13 Bionic added AMX units in the CPU Lightning cores [27] by extending the ARMv8.4 ISA. Furthermore, additional support for multiple data formats like int8 [2] and BF16 [11] have been added to the CPU to stay abreast of DNN algorithmic research. High performance libraries from CPU vendors are continuously being developed and tuned to extract maximum performance from new CPU compute. For example, Intel's MKL-DNN library achieved up to 198X performance improvement [3] on state-of-the-art CPU configurations. Smart algorithms [35] and co-processor solutions like Centaur [47] push the CPU frontier further.

CPUs currently constitute a significant share of the compute resources employed in industry for DNN inference. The wide prevalence of CPUs already in datacenters (for example, at Facebook) provisioned for peak load levels in conjunction with diurnal load cycles, leads to the abundant availability of *free* CPU compute for DNN inference [60], [71]. The programmable general-purpose nature of CPUs with their rich and mature ecosystem of tools and programming models enable quick development and deployment [73]. CPUs also offer latency benefits (see [44], [74] and MLPerf Inference Results [17]), since they don't rely on a driver-offload for DNN tasks, enabling a close interaction between DNN and non-DNN tasks in real-time inference. DNN tasks with limited parallelism, like Recurrent NNs, fit more naturally to CPUs with higher clock frequencies [76].

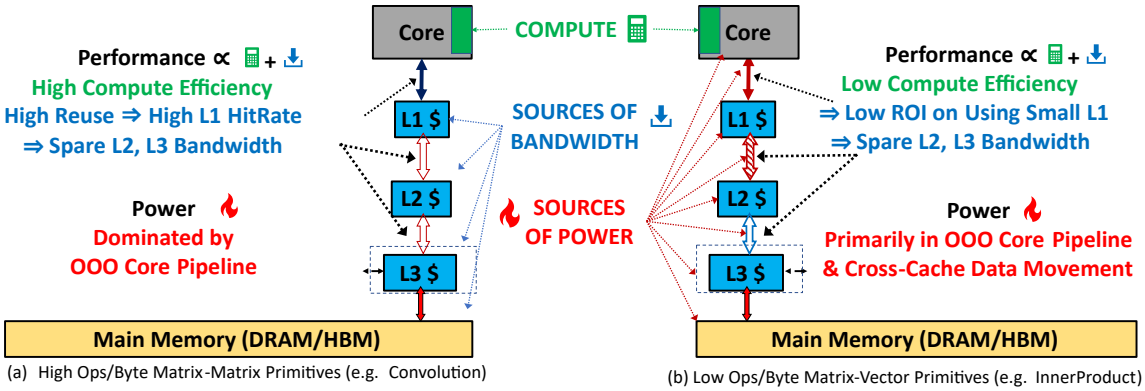

Fig. 1. Modern CPUs with multi-level memory hierarchies and how inference primitives use them

Wide and deep OoO CPU pipelines, designed to expose Instruction Level Parallelism (ILP) for single thread performance draw significant power. All instructions are unrolled in hardware and flow through everyone of these power hungry pipeline stages. All compute is placed "monolithically" atop a serially accessed multi-level cache hierarchy designed to minimize average load latency. All loads and stores must go through the L1 cache, thus restricting it to be the primary source of bandwidth.

There is a stark disparity in CPU requirements for single-thread performance with limited ILP and DNN-inference primitives like `convolution` or `inner-product` that are heavily data-parallel. DNN performance is governed by raw compute throughput and a required data throughput (bandwidth) to feed it. Generational compute scaling in CPUs is achieved via both intra-core scaling [2], [29] and multi-core scaling. The required bandwidth to feed the compute however, depends on primitives themselves; the higher the compute intensity (Ops/Byte) the lower the bandwidth required, and vice-versa. As DNN usages and topologies evolve, there is an increasing heterogeneity in their Ops/Byte (Compute/Bandwidth) requirements (see Section II).

The monolithic core-centric CPU organization results in sub-optimal performance, resource utilization and power when executing DNN primitives with diverse Ops/Byte requirements. Matrix-Matrix primitives (like convolution) illustrated in Figure 1(a) typically have high Ops/Byte. High reuse and hit-rate in the L1 cache delivers sufficient bandwidth for current compute resources. However, due to serialized accesses through the hierarchy, further compute scaling will hit a L1 cache bandwidth limit, despite L2 and L3 cache bandwidths being heavily underutilized. Matrix-Vector primitives (like inner-product), shown in Figure 1(b) have much lower Ops/Byte. Low hit-rates and bandwidth from small L1 caches delivers insufficient bandwidth for current compute resources. Serializing through the hierarchy unnecessarily introduces a L1 bottleneck despite larger L2 and L3 caches (with better hit-rates) being heavily underutilized in bandwidth. CPU execution unrolls every instruction instance through all pipeline stages. In particular wide fetch, decode, rename (RAT) and dispatch (RS) stages designed to expose ILP are extremely power hungry, raking up as much as 59% of total power consumed (see Section II-B4). Much of the hardware complexity in these stages isn't useful for highly repetitive data-parallel, fixed iteration count DNN kernels.

We present REDUCT, where we employ a *"Keep it close"* policy that enables consecutive pieces of work to be executed close to each other. REDUCT enables *instruction delivery/decode close to execution* and *instruction execution close to data*. REDUCT proposes simple "REDUCT Support Extensions" (rSX) ISA extensions to encode the structured loop-y workload behavior. This allows an effective bypass of many power-hungry stages of the wide OoO CPU pipeline into light-weight Tensor Functional Units (TFU). The core does fewer fetches and decodes, with a bulk offload of decoded tensor work (load/store/compute) to the TFUs, keeping instruction delivery/decode close to execution. The TFUs are distributed near all caches in a multi-level cache hierarchy, unshackling the binds of serialization through the hierarchy. This allows bypassing inner cache-levels as required, keeping instruction execution close to data. REDUCT's near-cache approach maximizes the cumulative utilization of existing architectural bandwidth resources in the system demanding no increase in cache capacity or bandwidth. Crucially, REDUCT operates entirely within the CPU programming and memory model, supporting existing virtual memory and hardware coherence along with hardware memory consistency and distribution of work to all compute by leveraging Simultaneous Multi Thread (SMT) CPU capabilities.

We make the following key contributions in this paper that address the disparity in requirements for single thread performance vs DNN workloads.

- We present **REDUCT**, where we distribute light-weight Tensor Functional Units, near each level of cache. REDUCT maximizes efficient utilization of the existing cumulative bandwidth in the system and minimizes data movement. REDUCT adds minimal additional hardware with no increase in cache capacity or bandwidth to the CPU, while scaling performance to levels matching state-of-the-art DNN DSAs.

- We develop simple "REDUCT Support Extensions" to the ISA that condense and encode multiple loops of fixed iteration count information. This allows many unnecessary power-hungry stages of the OOO CPU pipeline to be bypassed with all work (unrolling and execution) performed in close proximity to the data, drastically improving achieved performance/Watt.
- To our knowledge, REDUCT is the first practical implementation of near-memory processing that uses only the existing hardware and software architectural interfaces (like existing on-die cache ports and SMT hardware contexts) and thus requires no change to the CPU programming or memory model.

Evaluated across multiple DNN models, REDUCT achieves a 2.3× improvement in convolution performance/Watt with a 2× to 3.94× scaling in raw performance. Similarly, REDUCT achieves a 1.8× increase in inner-product performance/Watt with 2.8× performance. With full support for the existing CPU programming model, no increase in cache capacity or bandwidth and a mere 2.63% additional area, REDUCT enables unprecedented CPU efficiency gains and TCO savings for datacenters while matching performance levels of state-of-the-art DNN DSAs.

II. Characterization and Opportunity

We first perform an in-depth power and performance characterization of multiple primitives common to DNN inference on state-of-the-art CPU configurations. The goal is to derive insights that can lead to efficient performance and performance/Watt scaling.

A. Programming and Execution Model

DNN models are specified in frameworks like TensorFlow [30], Caffe [51], PyTorch [22], MXNet [4], OpenVino [21] etc. These frameworks provide developers with easy-to-use APIs to describe topologies and model parameters while abstracting away the underlying hardware. The frameworks in-turn leverage highly-optimized, platform-specific libraries like Intel oneDNN [19] and MKL-DNN [14] [26] [13], AMD's BLIS or libFLAME [9], ARM's compute libraries [5] and Nvidia's cuDNN [18] to extract maximum performance from the underlying hardware. The inner-most loops of DNN primitives are typically implemented in a vectorized, highly optimized (often JITed [42]) manner for optimal performance and maximum data reuse (see Figure 2). The outer-most loops in the primitives (computing different sets of outputs like in the example in Figure 2) are parallelized using well established threading run-times (OpenMP [20], TBB [16]) to distribute work across compute cores in the target hardware (see Figure 3).

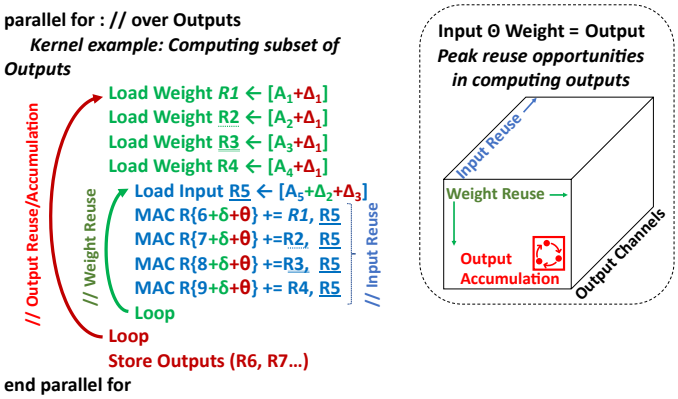

Fig. 2. 1x1 Convolution kernel example

B. Performance and Power Analysis

We evaluate state-of-the-art oneDNN [19] primitives [1]. We focus on int8 data types since several seminal studies [43] have shown that 8 bit (or lower) precision is sufficient for inference accuracy. The use of lower precisions reduces dependence on expensive off-chip DRAM bandwidth enabling us to focus on cache bandwidths that are on-die.

The compute intensity or Ops/Byte property of any DNN primitive plays a fundamental role in determining its bandwidth requirements (and hence compute efficiency) from the system. This metric needs to be evaluated at multiple levels of abstraction:

- ***Algorithm:*** The theoretical peak Ops/Byte is based on the work being performed if we had an "infinite" register file (RF) holding data. For example, in 1x1 `convolution` (Figure 2), every weight element is reused across all input plane elements in the same input channel, for different output plane elements. Similarly, input and output elements also have peak possible reuse opportunities.
- ***Kernel:*** The software implementation extracts reuse out of the finite RF of the compute core depending on both the RF size and the data-flow implemented. RF usage determines the minimum number of loads and stores to be executed. For the kernel in Figure 2, the

[1]Our evaluation methodology, system configuration and workloads are described in detail in Section IV.

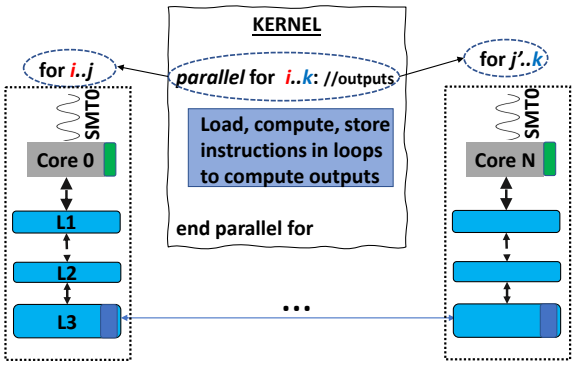

Fig. 3. Program flow in the baseline multi-core processor

number of weight loads (reused across inputs in the inner-most loop) and the number of iterations of the innermost loop (reusing weights to compute different outputs) are governed by the RF size.

- ***Hardware:*** On-die cache hit-rates determine bandwidth delivery to the core and cross-cache data movement when kernels are executed on the underlying hardware. We define `Data Movement Overheads` introduced by the hardware as the ratio of cumulative cross-cache data movement (fills and evictions) to the stores and loads from/to the compute core's RF (determined by the kernel).

We analyze and summarize these metrics in Table I for the convolution and FC primitives with key highlights in red.

1) `Convolution` Characterization: We evaluate the optimized oneDNN implementation that fuses convolution and ReLU (non-linear function on output) primitives. Table I details our characterization of the high Ops/Byte matrix-matrix convolution primitive across all convolutional layers of ResNet-50 [46] and shows several interesting insights (highlighted in red). First, the kernels employ output-stationary data-flows (maximizing output reuse in RF), with very low store bandwidth (Stores/MAC-Instr). Input and weight reuse variability across layers is subsumed within the RF resulting in a fairly steady 0.5 Loads/MAC-Instr requirement across all layers. Second, high average L1 hit-rates (86%) result in high compute efficiency (120 MACs/cycle out of a peak 128). Fills and evictions at the L1 cache still add an average 20% overhead in data movement. `conv1` layer of ResNet50 has the lowest L1 hit-rate, adding 69% data movement overhead at L1, with performance as low as 100 MACs/cycle.

Performance Opportunity: High L1 hit-rates mean that we use only about 60% of available L1 bandwidth (0.5 loads/MAC-Inst, two MAC-Inst/core/cycle vs two loads/-cycle/core at L1). Furthermore, the L2 and L3 bandwidths are still hugely under-utilized due to cache-hierarchy serialization. Placing tensor compute near each of these caches, would enable further scaling of performance without any increase in overall on-die capacity or bandwidth. More re-use directly from these caches also reduces data movement to the L1 caches. The 0.5 loads/MAC-instr requirement and peak bandwidths of each cache determines the peak compute required near each cache.

TABLE I
RESNET-50 CONVOLUTION AND TRANSFORMER INNER PRODUCT CHARACTERIZATION

	ResNet50 Convolution			Transformer InnerProduct		
Metric	**Avg.**	**Min**	**Max**	**Avg.**	**Min**	**Max**
Ops/Byte: Based on Algorithm						
Input	1021	32	4608	1727	1024	33708
Weight	2245	100	25600	1	1	1
Output	998	64	4608	2057	1024	33708
Memory Transactions/Op-Instr: Based on Kernel						
Loads	0.49	0.39	0.59	1.35	1.19	1.41
Stores	0.058	0.003	0.25	7E-04	3E-05	9E-04
Hardware: On-Die Cache Hit-Rate						
L1 $	86%	57%	98%	23%	15%	26%
L2 $	88%	51%	99%	72%	0%	100%
L3 $	99.4%	97.6%	99.9%	99%	64%	100%
Hardware: Data Movement Overhead						
L1-L2	20%	1%	69%	109%	81%	121%
L2-L3	2%	0.3%	5.8%	47%	27%	99%
Total	22%	2%	71%	156%	147%	181%
Hardware: Performance (Peak=128 MACs/Cycle/Core)						
Ops/Cyc	120.4	100.0	127	12.99	8.81	14.02

2) `Inner-Product` Characterization: Inner-Product primitives involve matrix-vector operations and have lower peak Ops/Byte compared to convolutions. These primitives are prominent in Recurrent DNN models and Sequence-to-Sequence models (eg. Transformer [70]) which are heavily used in applications like natural language processing. Table I also details our inner-product characterization for all layers in Transformer. This primitive has a poor 23% L1 hit-rate, which coupled with a high 1.35 Load/MAC-instr bandwidth requirement results in low compute efficiency achieving only 13 MACs/cycle. Furthermore, we see up to a whopping 156% overhead in cross-cache data movement.

Performance Opportunity: While L1 hit-rates are low, hit-rates in the larger L2 (1MB) and L3 (1.375MB per core) are significantly better. Tensor compute placed directly near these caches, bypassing the small L1 entirely, would leverage the higher hit-rates for higher bandwidth to feed the compute. Along with eliminating all data movement to an under-sized L1, we would see better performance, with no increase in cache capacity or bandwidth.

3) Pooling/Concat: We also evaluate Pooling (dimensionality reduction) and Concat primitives and they mainly involve data movement with low data reuse. Models like DenseNet-169 [50] pass lower level features (outputs) directly to later layers as inputs, using the Concat primitive to prepare data. Execution near L2 and/or L3 caches would reduce this data movement cost.

4) Power Analysis: Figure 4 shows the contribution to total power from various clusters in the CPU. CPUs unroll every instance of every instruction of all loops into every stage of the CPU pipeline. All instructions go through fetch and decode, allocation and dispatch. Register allocation and renaming (RAT) and OOO dispatch (from RS) are extremely expensive in power for wide and deep OOO cores. For compute bound ResNet-50, these stages contribute to 59.7% of total power! For bandwidth bound

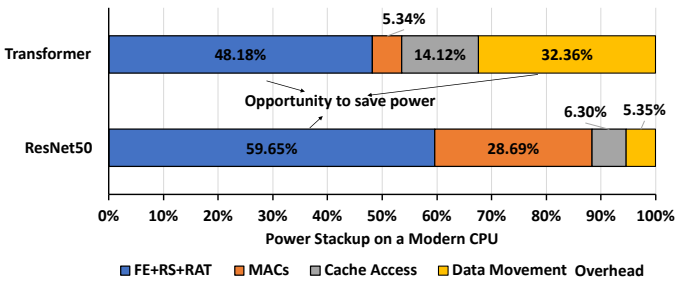

Fig. 4. Stackup of power consumption in convolution dominated ResNet50 and inner product dominated Transformer

TABLE II
CHARACTERIZATION SUMMARY OF DNN PRIMITIVES

Primitive	Observations	Data Movement Overhead	Opportunity
Convolution	Bandwidth over-provisioned w.r.t. compute, Under-utilization of L2/L3 Bandwidth	Mostly at L1-L2	Perform tensor compute near **all** caches (L1,L2,L3)
Inner-product	Compute over-provisioned w.r.t. bandwidth, Poor hitrate at 32KB L1	High at L1-L2 and L2-L3	Place tensor compute near **large** caches (L2 and L3)
Pooling/Concat	Low data reuse Mostly data movement	High at L1-L2 and L2-L3	Execute near **outer** cache levels (L3/L2)
Power dominated by unrolled wide OOO Fetch, Alloc and Dispatch		**Exploit structured/loopy kernel to encode multiple loops**	

inner-product primitives in Transformer, they contribute to 48.2% of total power with cache access and data movement adding another 46.5%.

Power Opportunity: With structured loop-y fixed iteration count DNN kernels, loop unrolling should happen in a "lean" scheduler, close to the tensor compute. Using a "macro"-ISA that encodes this loop information, the CPU can offload multiple loops of decoded (but not unrolled) work to the "lean", low-cost near-cache compute effectively bypassing power-hungry stages of the legacy CPU pipeline for most of the execution.

5) Summary: Table II concludes this section by summarizing our main observations for performance and power. Optimally executing primitives near caches best suited to their requirements can provide performance, power and resource utilization benefits. Furthermore, we should leverage the structured nature of the workloads to bypass power-hungry front-end stages of the legacy CPU pipeline - preferably unrolling and scheduling pre-decoded instructions near the execution units.

III. REDUCT

We present REDUCT[2], which places light-weight "Tensor Functional Units" (TFU) near **all** on-die caches in the system as depicted in Figure 5. In combination with "REDUCT Support Extensions" (rSX) to the ISA, the goal is to efficiently leverage existing system resources (maximizing or minimizing use as required) for performance and power benefits. Crucially, REDUCT also retains the existing CPU programming and memory models which can significantly speed up development and deployment efforts. We now detail the architectural, micro-architectural and programming model aspects of REDUCT.

A. REDUCT: Architectural Support

1) REDUCT Support Extensions (rSX): A close examination of state-of-the-art oneDNN kernels implementing DNN primitives shines light on opportunities to encode multiple loops of information succinctly. Such optimizations would enable minimizing and bypassing the power-hungry front-end stages of the CPU pipeline with unrolling and dispatch of work proximal to the execution units.

[2]*Reduct* has multiple meanings: 1) to reduce; 2) to bring back to memory; 3) to simplify - all relevant to this work

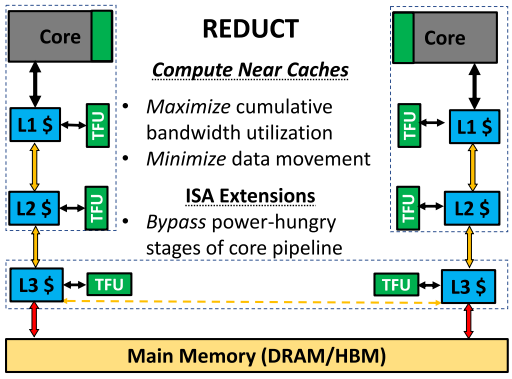

Fig. 5. An overview of REDUCT design

Depicted in Figure 6, is the meta-data information each instruction requires to encode its loop behavior in the kernel. First, we need to know the number of loops and their iteration counts. The ISA can set a limit on the maximum number of loops encoded, and we find that supporting four loops is sufficient for all these kernels (capturing load, compute and store operations). Each instruction needs to know the set of loops it resides within. In the example, weight loads are only executed in the outer loop. Second, loads and stores need a base address as well as an address stride for each loop it resides in. These can be computed since DNN primitive implementation employs structured data layouts to maximize cache port width and capacity. Finally, data dependence is still through registers. Hence, we can require stride values per loop for destination register ids as well. In the example, iterations of the innermost loop reuses weights (being loaded in the

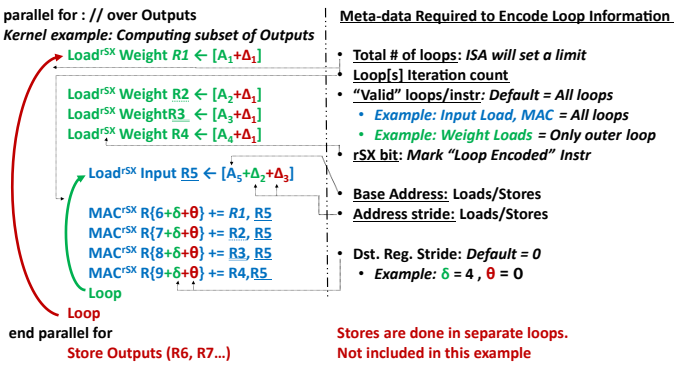

Fig. 6. Requirements for encoding all loop information

Instructions (rSX enabled Kernel)	Comments
❖ TFULoopStart	*Flushes TFU Code Register in the core*
❖ TFULoopCount 2	*Sets total expected loops to 2*
(Loop Iteration Calculation into R0)	*Baseline ISA. Executed in the core.*
❖ TFULoopIteration 1, R0 *//eg. = 64*	*Sets iteration count of outer loop (loop1) to calc. value*
(Loop Iteration Calculation into R0)	*Baseline ISA. Executed in the core.*
❖ TFULoopIteration 2, R0 *//eg. = 4*	*Sets iteration count of inner loop (loop2) to calculated value*
Load^{rSX} Weight *R1* ← [A_1+Δ_1]	***rSX instruction to be executed in TFU near a cache***
❖ TFULoopDisable 2	*Disables outer loop for previously allocated rSX Instr.*
(BaseAddress Calculation into R0)	*Baseline ISA. Executed in the core.*
❖ TFUBaseAddress R0	*Sets BaseAddress for previously allocated rSX Instr.*
(Stride Calculation into R0)	*Baseline ISA. Executed in the core.*
❖ TFUStride 1, R0	*Sets stride for loop1 of previously allocated rSX Instr to calc. value*
…	
MAC^{rSX} R{6+δ+θ} += *R1*, R5	***rSX instruction to be executed in TFU near a cache***

Fig. 7. REDUCT : New rSX instructions and execution semantics

outer loop) to compute different output elements that need to be stored in different registers. Here, the destination register id stride is determined by the number of outputs computed in the innermost loop (4).

Figure 7 illustrates the new rSX instructions and their semantics. Kernel instructions are tagged with a `rSX-bit` (denoting near-cache TFU execution) and are decoded and allocated into new `TFU Code Registers` in the core. Our examination of primitives across multiple DNN models shows 32 registers to be sufficient. However, if a kernel has more than 32 instructions (and/or 4 loops) it would need to be split into smaller kernels that fit within these constraints. New rSX instructions (`TFULoopCount`, `TFULoopIteration`, `TFULoopDisable`, `TFUBaseAddres`, `TFUStride`, and `TFURegStride`) populate their respective meta-data loop information for the instructions tagged with rSX-bit. The meta-data information can be calculated using regular ISA (similar to the way the baseline kernels currently do it). The new `TFULoopStart` instruction flushes the `TFU Code Registers` for new rSX-tagged instructions and the `TFULoopEnd` instruction dispatches the `TFU Code Registers` to the near-cache TFU for unrolled execution. We conservatively estimate each `TFU Code Register` holds 8B of information (opcode, up to 3 registers (or a register and base address),4 loop iteration counts (with a valid bit) and 4 address and register strides). The entire offload takes 32 cycles (using an 8B offload bus width) and this time is amortized by the hundreds of cycles of unrolled execution in the TFU.

2) Tensor Functional Units (TFU): Figure 8 depicts the Tensor Functional Units (TFU), with 32 `TFU Code Registers`. A lean "Unrolling Scheduler" populates two 8-entry in-order `Issue Queues` - one for all compute opcodes and another for loads and stores. The design simplifies scheduling (lower power!) and allows hoisting of loads over compute (to hide load latency) while maintaining strict load/store ordering within the TFU. Loads and stores directly access the cache each TFU is placed proximal to - bypassing any inner levels. Snoops into inner cache levels, as required, are handled through added coherency support to the cache (Section III-B2). A small Translation Cache assists in memory management (Section III-B1).

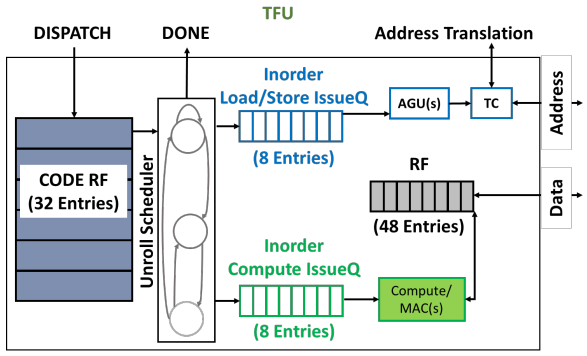

Fig. 8. Schematic of the Tensor Functional Unit (TFU)

Both kernel characterization and performance analysis show that a 48-entry "deep" `TFU Data Register File` per TFU is sufficient - with no register renaming (lower power again!) required. The loads/MAC-instr requirement of the workload and the near cache bandwidth bounds the peak compute "width" (the number of 64B MAC execution units) of the TFU. We evaluate the performance, power and energy implications of different compute widths in Section V. TFU area analysis is included in Section IV.

3) Leveraging SMT: Modern server-class CPUs support 2-way (Intel, AMD) and 4-way SMT (IBM, Sun SPARC). However, since all compute is shared across SMT threads, DNN frameworks disable SMT or use only one thread per core for the primitives, relying on multi-core compute scaling instead. With REDUCT, each TFU is essentially a lean compute engine directly accessing one cache level in the hierarchy. As shown in Figure 9, we leverage SMT to bind each TFU exclusively to one of the logical SMT threads in the physical core. Therefore, each TFU is part of a fully capable, OS-visible hardware context. DNN frameworks can then distribute work across TFUs using existing threading runtimes. This also enables fine grained control over which caches and TFUs to use for a primitive(Section III-C3). rSX ISA, TFU design and SMT usage allow REDUCT to maintain the CPU memory model (Section III-B4).

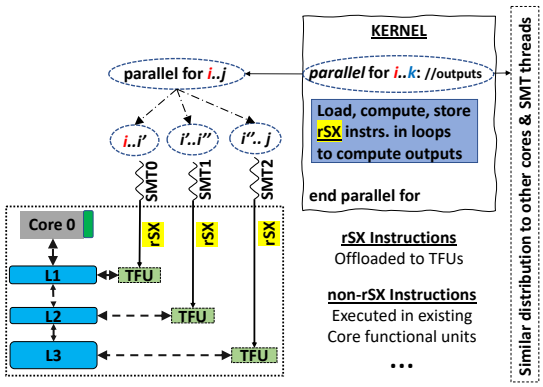

Fig. 9. REDUCT program flow leveraging SMT to bind each TFU to an OS-visible thread and dispatch of rSX to TFUs

B. REDUCT: Micro-Architectural Support

1) **Virtual Memory:** The TFUs need virtual memory to physical translation since CPU caches are physically tagged. oneDNN optimizations use special layouts, customized to feed compute, with a high spatial locality of tensor accesses [28] (core TLB hit-rates > 98%). We find that a small 8-entry local Translation Cache (TC) per TFU, holding recently observed virtual to physical mappings, can achieve a 95% hit-rate. Each TFU needs to snoop the local core TLB (which is close by since the TFUs are near caches physically co-located with the core) to populate the TC. With the high hit-rates, we observe no performance impact of TC misses or TLB snoops.

2) **Coherence Support:** Since REDUCT operates in the cache coherent address space, hardware coherence uses much of the existing framework and is supported through minor additions to the L2/L3 caches and controllers. The near-L2 TFU adds the requirement of a core-valid bit at the L2 (denoting ownership by the local near-L2 TFU) and the ability to generate a Request for Ownership (RFO) for its stores. Similarly the near-L3 TFU requires the directory entries at L3 to distinguish owners between the existing cores and the local near-L3 TFUs through additional core-valid bits per near-L3 TFU. The L3 controller must also generate RFOs for stores generated by its local near-L3 TFU. Snoops that may be required due to these RFOs use the existing coherence framework. Since oneDNN uses output stationary kernels, each TFU operates on its own set of output elements. Therefore, we see a negligible increase in snoop traffic, limited to cases when different output elements share a 64B cacheline.

3) **Distributed L3 Caches:** L3 caches in modern CPUs are distributed and shared across multiple-cores. Each near-L3 TFU per core is placed near the shared L3 slice that is co-located with the core. While L1 and L2 caches are private per core, and can have duplication of data across them, the shared L3 doesn't support data duplication across slices. However, REDUCT requires such duplication for its near-L3 TFUs. For example, they may all need the same weight element to compute their respective outputs. Traversing the L3 interconnect for every address not available locally would cripple near-L3 TFU performance and add significant extra data movement overhead. We enhance existing class-of-service technologies like Intel's CAT [10] to partition a portion of the set-associative cache (a small subset of its total ways) in each L3 bank as a local cache for the attached TFU. Section V includes performance sensitivity to the reserved local cache capacity for each near-L3 TFU.

4) **Memory Ordering:** Within a TFU, strict loads/-store ordering is maintained (TSO). Within any thread, non-TFU operations past any TFU bulk-offload are not allocated till the TFU bulk-offload operations are over. A bulk-offload of hundreds of cycles of work to the TFU (through rSX-ISA) amortizes any performance cost of this serialization. Non-TFU load/store ordering continues to be maintained by the core. Note that since TFUs are on different SMT threads, we do not need to guarantee any ordering in execution of loads and stores across TFUs. To our best knowledge, REDUCT is the only near-memory proposal to fully support the hardware memory consistency model of the CPU.

5) **Context switching and Exceptions:** Under current DNN usage models (example - micro-services in datacenters) context switching or exceptions are extremely unlikely to occur. However, TFUs signal exceptions like existing functional/compute units in the physical cores. TFU context (the `TFU Code Registers` and `TFU Data Registers`) must be included in any context save/restore.

C. REDUCT: Programming Model Support

1) Generating rSX Code: High performance libraries already implement primitives using optimized code that is JITed [42]. The kernel instructions must now be tagged with the `rSX-bit`. Note that this is no different than incorporating any other ISA enhancement (example AVX256 to AVX512). The JITer is already used to compute various loop variables. Note that the programmer need not be aware of the exact TFU and cache level where these rSX instructions will eventually be executed. Optimizing compilers to generate rSX code from native languages without JITing is possible but not explored in our work.

2) Exposing REDUCT Capability at Each Cache Level: Presence of TFUs in each cache level and their corresponding compute width is exposed through the `cpuid` interface that is supported by all modern processors.

3) Optimal TFU selection for primitives: As we characterize and summarize in Table II, and with further analysis in Section V, each primitive has an optimal set of TFUs for power-performance efficiency. We leverage the use of SMT (binding a TFU to a logical thread) and use existing OpenMP APIs to set the affinity of a primitive to a subset of cores. Specifically, we use `KMP_SET_AFFINITY` in the LLVM OpenMP Runtime [24] to achieve this. DNN frameworks (TensorFlow, Caffe etc) would do the same before invoking oneDNN primitives.

4) Distribution of work across TFUs: Since caches across the multi-level hierarchy have different bandwidths, their corresponding TFU will have different compute "widths" as well. For compute bound primitives like convolution, we need to divide work across TFUs proportionate to their compute strength for optimal performance. With high cache hit-rates and predictable performance, such a "static" division of work is sufficient for these workloads. Towards this, we introduce a simple new schedule *kind* called `static_asymmetric` in the LLVM OpenMP runtime. For example, for three TFUs with compute strengths in a 2:2:1 ratio, a static equal division of work would result in unequal thread completion times with the third (weakest) TFU determining final runtime. With a static asymmetric distribution, in the same 2:2:1 ratio, all threads optimally complete at the same time.

TABLE III
MINIMAL SOFTWARE STACK TO SUPPORT REDUCT

Level in Software Stack	REDUCT Support
DNN Frameworks (TensorFlow, Caffe, PyTorch)	Set `KMP_THREAD_AFFINITY` appropriately for each primitive
High Performance Library (oneDNN)	Use rSX extensions for tensor load, store, compute instructions
Threading Runtime (OpenMP)	Asymmetric static scheduling when appropriate (e.g. Convolution)

Table III summarizes the intercepts in the overall software stack by REDUCT. REDUCT operates entirely within the CPU programming and memory model.

IV. Evaluation Methodology

Simulation Framework: We use a modified version of Sniper [34] for our cycle-accurate multi-core simulations. We model a state-of-the-art Intel 28 core datacenter processor [2] but with 4-way SMT whose parameters are listed in Table IV. This baseline supports a peak 128 (2*64) MACs/cycle/core of compute (similar to Intel DL-Boost [64]) with per-core on-die cache bandwidth including two 64B read ports at L1, two 64B read/write ports at L2 and one 64B read/write port at L3.

The simulation framework has been thoroughly validated against silicon by executing all the evaluated oneDNN [14] primitives and verifying the performance trends using a Cascade Lake system running Ubuntu 16.04 with oneDNN v1.0 library. CACTI [57], [67] and McPAT [54] are used to quantify cache and overall energy impact.

TABLE IV
SIMULATOR PARAMETERS

28 Cores	2.6GHz, 4-way SMT, 320-entry ROB, 128 (2*64) MACs/cyc/core
L1 cache	Private, 32kB, 8-way set associative, LRU, 8-enrty MSHR, 2x 64B read ports, 1x 64B write ports, Data access=4 cycles, Tag lookup=1 cycle
L2 cache	Private, 1MB, 16-way set associative, LRU, 48-enrty MSHR, 2x 64B read/write ports , Data access=8 cycles, Tag lookup=2 cycle
L3 cache	Distributed, Non-inclusive, 1.375 MB/slice, 11-way set associative, RRIP, 48-enrty MSHR, 1x 64B read-/write port per slice, Data access latency=10 cycles
Tag directory	MESIF, 10 cycle latency
4x7 Mesh NoC	XY Routing, 22 cycle latency max.

REDUCT: We model all of REDUCT in Sniper allowing sweeps of peak compute at each TFU at different cache levels. The last SMT thread is tied to L1 TFU but not used. We use prefix "**R**" (for REDUCT) or "**M**" (for traditional monolithic core) followed by a number which indicates the peak number of MACs/cycle/core that the configuration has. Further, the notation also attaches an explicit distribution of compute resources across the cache levels as indicated in Table V. **Mxxx** configurations have MAC units that support **xxx** MACs/cycle/core. Table V specifies the hardware resources and the distribution for the REDUCT configurations.

TABLE V
NOTATION FOR REDUCT CONFIGURATIONS

Name	MACs/Cycle/Core	Distribution
R128	128	same as M128
R256	256	128@L1, 64@L2, 64@L3
R320	320	128@L1, 128@L2, 64@L3
R512	512	256@L1, 128@L2, 128@L3
R640	640	256@L1, 256@L2, 128@L3

TABLE VI
AREA BREAKDOWN FOR TFU (IN mm^2)

Registers	MACs	TC,Queues,Control
0.15	0.17	0.06
Total Bytes: 3184		
Total Area: 0.38mm^2		

Area Requirements of REDUCT: We synthesize a Verilog implementation of a TFU instance that is capable of 256 (4*64) MACs/cycle/TFU to support the R640 configuration. Synthesis is done using TSMC 28nm library, setting a target clock of 1 GHz, using Synopsys Design Compiler [25]. The total area per TFU is 0.38mm^2 and the detailed breakup area is given in Table VI. An additional 2KB/core of storage is required for new core-valid bits in L2 and L3 for full coherence. Total area overhead is the sum of this coherence storage and the area required for three TFUs/core. Projecting the total overhead to the 14nm technology [68], [1] in which the Intel Xeon server chips are built, we conservatively estimate the overall overhead due to REDUCT to be a mere 2.63% of a single Xeon core. From observing die-plots([69], [23]), REDUCT takes 35% less than the AVX-512 unit already present in the cores, while delivering a higher peak MACs/cycle/core cumulative compute.

Workloads and Software Stack: We evaluate six DNN topologies end-to-end including ResNet-50 [46], DenseNet-169 [50], MobileNet [48], ResNext-101 [72], Transformer [70] and TwoStream [40]. Of these, Transformer comprises solely of inner-product layers while the others have mostly convolution layers. We use the latest open source oneDNN v1.0 and the Intel C++ compiler 19.0 to ensure we are evaluating state-of-the-art software implementations. Note that these are full system (including DRAM), multi-core evaluations. Since we study int8 inference, most model weights fit in caches. Coupled with high reuse, overall impact on performance and power from DRAM is very low. These workloads are parallelized using the OpenMP multi-threading framework using our new static asymmetric scheduling.

V. Results

We will first present performance and data movement impact of REDUCT near-cache compute for DNN primitives in the ResNet-50 and Transformer models. We then do a detailed power, performance and energy study for REDUCT on ResNet-50 and Transformer. This is followed by results for all six DNN topologies evaluated. We then summarize the performance and performance/Watt goodness of REDUCT and conclude by comparing it with available MLPerf data for GPUs and DSAs.

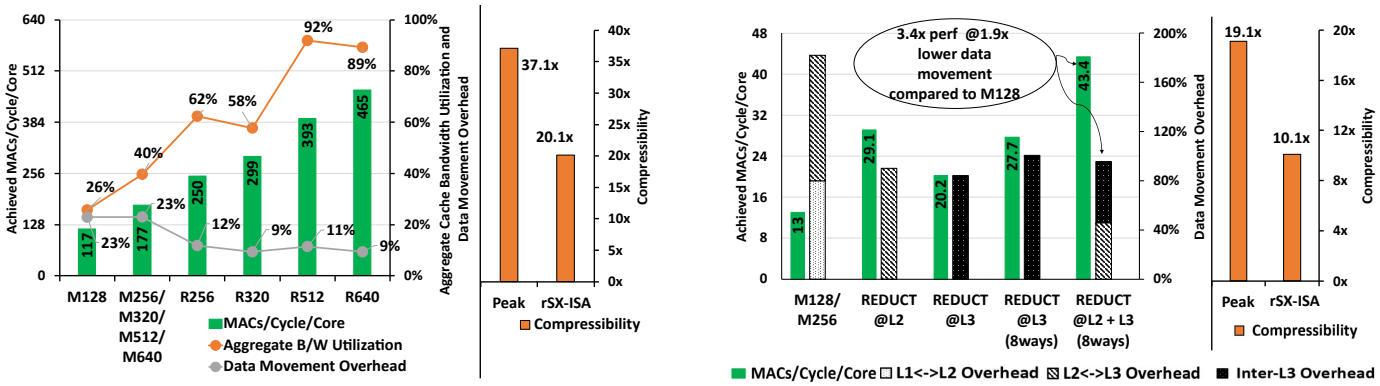

Fig. 10. Impact of REDUCT on ResNet50 convolutional layers (left) and Transformer inner-product layers (right)

A. Convolution

Figure 10 (left) shows the impact of REDUCT on ResNet50 convolutional layers and reveals multiple interesting observations. Traditional scaling of compute plateaus at an average ∼185 MACs/cycle/core from M256 onwards, since L1 bandwidth saturates, despite using only 40% of total available on-die bandwidth. However, REDUCT scales well in performance at all points, achieving between 2x to 3.94x performance over baseline, with up to 90% cumulative bandwidth utilization in the higher peak compute points. REDUCT R256 also has 41% higher performance than M256, achieving near peak performance. M256 needs peak bandwidth (100% hit-rate) from L1 to achieve performance. In contrast, the R256 configuration doesn't require peak bandwidth from any of its caches and is hence able to get higher performance.

We define *compressibility* as the reduction in the number of dynamic instructions that go through the core's FE+RAT with the usage of rSX-ISA, essentially a measure of bulk-offload opportunity. rSX-ISA achieves an average 20× compressibility, which will translate to power savings. Peak compressibility of the data parallel loops is actually 37×, but the new rSX instructions used to populate loop meta-data reduces the compressibility. Finally, REDUCT reduces data movement overheads from 26% down to 9%, mainly going down at the L1-L2 interface.

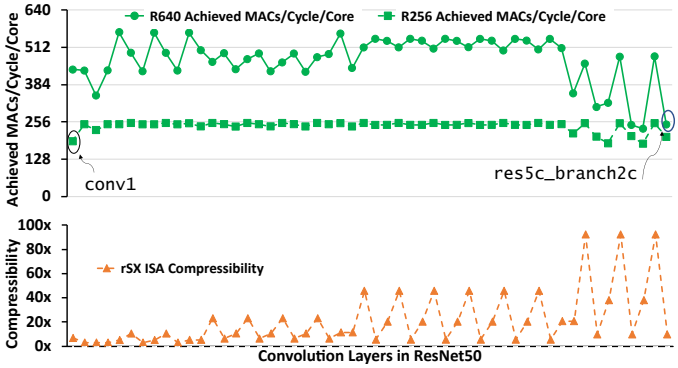

Fig. 11. Per ResNet-50 convolution layer REDUCT performance for R256 and R640 configurations. Compressibility using rSX ISA is at the bottom

Figure 11 shows per-layer performance and compressibility for the 53 convolution layers in ResNet-50 for the R256 and R640 configurations. Sparing a few layers at the start like `conv1` and last few layers like `res5c_branch2c`, R256 configuration achieves peak performance across layers. The initial layers suffer from low cache hit-rates due to model loading. The last few layers have lower total Ops/Byte and the near-L3 TFUs see low hit-rates in their 256KB local partitioned cache (2 out of 11 ways) with high cross-L3 traffic. Increasing the reserved near-L3 ways from 2 to 8 ways improves performance for these layers by 40%-60%. Compressibility increases with increase in the input channel dimension (due to higher accumulation required per output) and is generally lower for 1x1 kernels compared to 3x3 kernels (due to lower input reuse). R640 scales performance over R256 by a factor of 1.86x for a resource scaling of 2.5x.

B. Inner Product

Figure 10 (right) plots the average impact of REDUCT configurations on performance, data-movement for the 106 Transformer inner-product layers on the right. Since these are low Ops/Byte bandwidth bound primitives, we only use the R256 REDUCT configuration but schedule threads selectively at different cache levels. By merely executing the inner-product directly near the large 1MB L2, with better hit-rate and therefore bandwidth delivery, REDUCT achieves 2.2× more performance and a 2.1× reduction in data movement overheads. All L1 traffic has been eliminated. Execution only near-L3, with a 2-way 256KB local cache, also reduces data movement. However, as we can now expect, provisioning more capacity to the near-L3 compute increases performance to match REDUCT near-L2. Executing the primitive at both L2 and L3 improves performance by a huge 3.3× over what can be achieved by the baseline. This is achieved with no increase in on-die cache bandwidth while simultaneously reducing data movement overhead by 2×. Finally, the new rSX-ISA achieves 10× compression, which will directly translate to power and energy savings.

C. Pooling and Concat Layers

For brevity, we summarize the main observations from evaluations of the pooling and concat primitives. Executing the res5c ResNet-50 pooling layer solely near L3 reduces data movement overhead by 95% (103% down to 8%). Similarly, Concat layers in DenseNet-169 see an average 150% data movement overhead, which is brought down by 70%-95% (down to 25%-5% overhead) via near L2 or near L3 execution.

D. Detailed Energy Analysis

Figure 13 represents a detailed energy analysis of the ResNet-50 convolutional layers and Transformer inner-product layers on the baseline M128 and R256 configurations. All the energy components are shown relative to the total energy cost of running on a M128 configuration. We deconstruct the impact of the near-cache and rSX ISA components of the REDUCT proposal separately and when put together.

In ResNet50, the FE and OOO stages of the CPU pipeline dominate overall energy (60%). M256, even while having twice the resources as M128, is iso-energy with M128 as long as near-cache capabilities are not included to them. Adding near-cache compute reduces inter cache data movement costs between L1 and L2 but also increase direct accesses to the larger (costlier in power) L2/L3 caches. Therefore, near-cache compute ends up iso-energy with baseline. The rSX ISA proposal achieves an average 20.1× compression, which translates to a 17× reduction in energy from the FE and OOO stages of the pipeline. REDUCT R256 configuration therefore operates at 42% of baseline energy (translating to a 13% decrease in power, along with a 2x increase in performance).

The bandwidth bound, low Ops/Byte inner-product layers in Transformer present a different story: data movement reduction (primarily by eliminating at L1) brings a 18.6% reduction in energy. Despite lower data reuse compared to convolution, the rSX ISA proposal achieves a 10× compression, bypassing the FE and OOO pipeline stages and providing a 42.8% energy reduction. Put together, we see a 61.5% reduction in overall energy consumption (at 1.5% lower in power and 2.77× better performance).

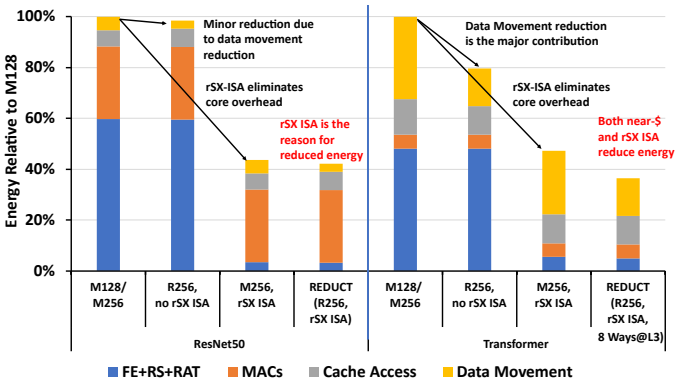

Fig. 13. Stackup of energy consumption

E. End-to-End Performance, Power and Energy

Figure 12 shows performance, energy and power impact of two REDUCT configs (R256 and R640 respectively) over six DNN-inference topologies. Inner-product heavy Transformer is bandwidth bound. It achieves 2.78× better performance at iso-power (65% lower energy) for both REDUCT configs. The rest of the DNN models have mostly convolution layers. With the exception of DenseNet-169, these topologies see around 2× performance with 2× compute (R256 vs M128), at 12%-14% lower power (60% lower energy). This increases to around 3.95× higher performance with 5× more compute (R640 vs M128) at 65% higher power (and 60% lower energy). DenseNet-169 has a number of Concat layers which take nearly 20% of total runtime. REDUCT reduces power for this data shuffling primitive but doesn't impact performance. This results in slightly lower performance improvement for DenseNet-169 with REDUCT, but at slightly better power compared to the other convolution heavy DNN topologies.

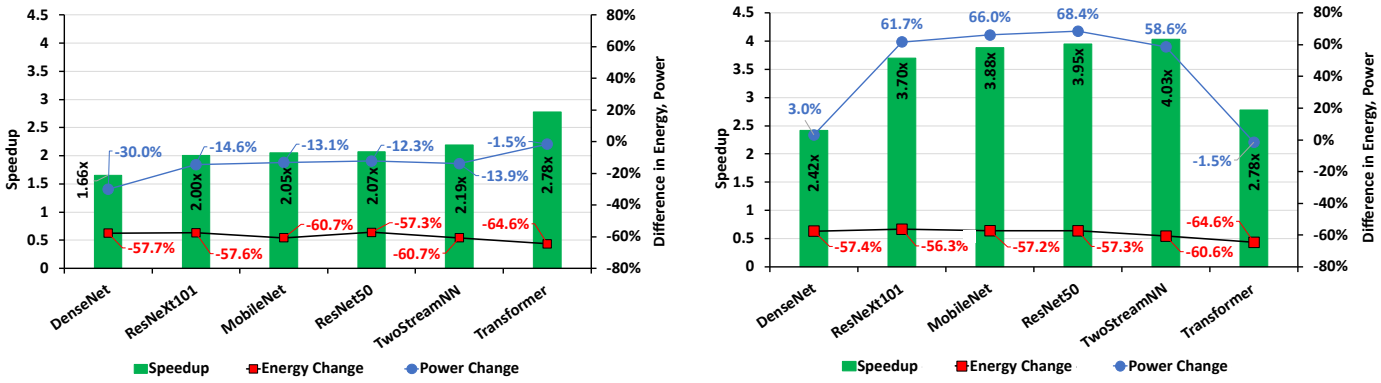

Fig. 12. Overall perf. improvement, energy and power difference for six DNN topologies using REDUCT. R256 (left) and R640 (right) configurations are shown relative to M128. Primary Y-Axis plots performance and secondary Y-Axis plots power and energy change.

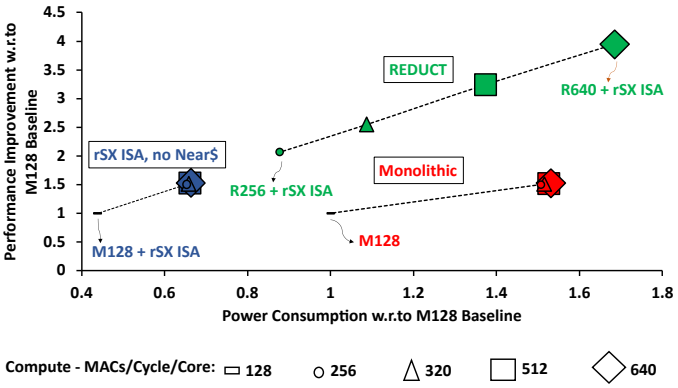

Fig. 14. ResNet-50: Overall performance and power summary

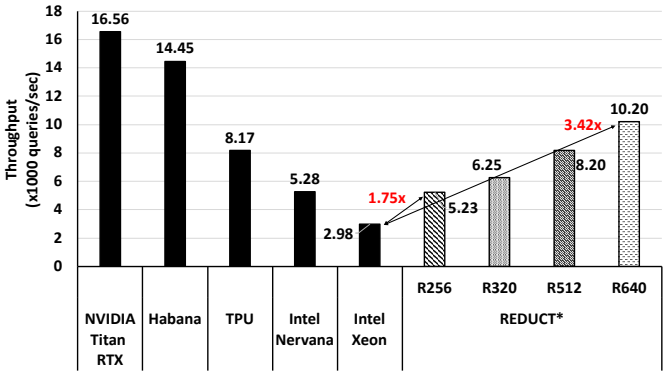

Fig. 15. MLPerf inference throughput of GPUs, accelerators, and CPUs compared with throughput of REDUCT (projected*)

F. Performance Per Watt Benefits

Figure 14 succinctly summarizes the goodness of REDUCT on ResNet-50. On the X and Y-axes, the figure shows performance and power of different configurations normalized to the M128 baseline. Performance/Watt (PPW) improvements across configurations are captured as the ratio of respective Y and X-coordinates. Monolithic performance scaling (shown in red) saturates at M256 with no PPW improvements over M128. Moving to rSX-ISA (blue) improves PPW by 2.3× over any equivalent monolithic configuration. REDUCT (green) leverages near-cache compute on top of the rSX ISA, enabling performance to scale by 2× to 3.94× depending on the available TDP (13% lower power to 68% higher power), while maintaining the PPW gains that the rSX ISA brings. We observed that for inner-product heavy topologies like Transformer, REDUCT achieves a 1.8× increase in `inner-product` performance/Watt with 2.8× improvement performance.

G. Comparison of REDUCT to DSAs and GPUs

We compare REDUCT with other platforms that are used for DNN inference by using state-of-the-art, vendor-provided, publicly available throughput results on ResNet-50 v1.5 from MLPerf Inference results [17], [62]. Figure 15 shows the number of queries processed by a variety of accelerators (Habana Labs [7] , Intel Nervana [15], TPU v3 [52]), GPUs(using a Titan RTX T496X2 card), and Cascade Lake based Intel Xeon Platinum 9200 processor (similar to our CPU baseline). We normalize the data to per node (for accelerators and GPUs) or per socket (for CPUs) and use the best results for each platform.

Incredibly, REDUCT achieves comparable (or even somewhat better) performance to state-of-the-art DSAs (Intel Nervana, Google TPU v3). Note, this performance is achieved with a mere 2.63% additional area, maximizing usage of existing cache capacity/bandwidth, while supporting the CPU programming/memory model and single-thread goodness. In contrast, discrete DSAs add much larger area (separate compute/caches) and power budgets to the system. We don't make any performance/Watt comparisons due to the absence of these metrics in MLPerf. We have shown significant (1.8×-2.3×) improvement in CPU performance/Watt with REDUCT.

VI. REDUCT on low-power, edge CPUs

We have verified the performance/power benefit of REDUCT across a range of compute widths, caches sizes and bandwidths, including lower compute (16/32 MAC/-cycle/core) and cache bandwidths typical to lower power edge CPUs. Edge CPUs can pose two additional challenges. First, they typically have shallower cache hierarchies, often with multiple cores sharing an L2 cache. With REDUCT, we would still have a TFU per core at the L2 cache, but with lower compute strength. The total compute strength across all TFUs near shared-L2 should be proportional to the L2 bandwidth. Second, these cores typically have no SMT capabilities, hence a core would simultaneously schedule to all its TFUs, breaking TSO memory ordering. However, relaxed consistency is fine for ML workloads [6]. For low power and low cost edge SOCs, where budget, latency and battery life requirements reduce the RoI of adding specialized DSA hardware with driver-based offload, we believe REDUCT represents an excellent CPU solution on all cost, performance and energy metrics.

VII. Related Work

Near data processing is an active research topic spanning DRAM to on-die caches, with recent emphasis on the site of processing[59] as well as practical approaches[58] [47] to enable it. Moving processing to in/near memories like DRAM([65],[66],[37]), 3D-stacked HBM[8] or HMC[61]([32], [75],[36],[53],[49],[33]) and even SSDs[56] is one direction. Compute Cache[31], Neural Cache[38], and Duality Cache [41] are all recent works that propose converting the CPU on-die caches into compute units capable of bit-serial /bit-parallel operations **in** the SRAM sub-arrays. While these works show huge gains, they involve changes to highly optimized SRAM sub-arrays, and can degrade or complicate signal integrity, density and floorplan. Livia [55] proposed a data-centric architecture that moves tasks to

cache levels "closest" to the data source but the architecture has several limitations. Livia is meant for irregular workloads and can track only one data element's presence across hierarchy while moving the task. Inference workloads need multiple tensors and accesses to them are very regular. Livia also proposes a new programming model which requires application rewrite. REDUCT takes a light-weight, compute-near-cache approach, reusing existing micro-architectural interfaces and extracting large improvements in performance and power, while supporting the existing programming model.

From the power perspective, REDUCT is different from solutions like Revolver [45] in using explicit instructions to encode repetitive loops, requiring no learning, and by bypassing all of front-end, RS and RAT pipeline stages.

VIII. Summary

As the amount of data created and analyzed grows exponentially, OEMs are willing to harness all forms of available compute to tackle their computational needs. While CPUs are the dominant platform-of-choice for DNN inference in datacenters, our in-depth analysis reveals significant opportunities for more efficient CPU resource utilization leading to drastic improvements in performance/Watt and the ability to scale performance without increasing cache capacity or bandwidth. Using a combination of simple ISA enhancements and light-weight tensor compute *near all caches* in the CPU, REDUCT raises the bar for CPU-based DNN-inference performance and performance/Watt while maintaining the same programming and memory models. REDUCT represents a fundamental re-imagination of the general-purpose CPU, better suited for the AI era.

Acknowledgments

We sincerely thank the anonymous reviewers of ISCA 2021 for their feedback. We also thank Vedvyas Shanbhogue at Intel for discussions on the programming model, coherency and translation issues. We thank Vadim Pirogov, Tatyana Primak and Nikita Shustrov at Intel for discussions on oneDNN kernels and optimizations therein.

References

[1] "14 *nm* Process Technology: Opening New Horizons." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/bohr-14nm-idf-2014-brief.pdf

[2] "2nd Generation Intel Xeon Scalable Processors with Intel C620 Series Chipsets." [Online]. Available: https://www.intel.com/content/www/us/en/design/products-and-solutions/processors-and-chipsets/cascade-lake/2nd-gen-intel-xeon-scalable-processors.html

[3] "Amazing Inference Performance with Intel Xeon Scalable Processors." [Online]. Available: https://www.intel.ai/amazing-inference-performance-with-intel-xeon-scalable-processors/#gs.wb7n2a

[4] "Apache MXNet." [Online]. Available: https://mxnet.apache.org/

[5] "arm Computing Library 19.05." [Online]. Available: https://arm-software.github.io/ComputeLibrary/latest/index.xhtml

[6] "Designing Computer Systems for Software 2.0." [Online]. Available: https://iscaconf.org/isca2018/docs/Kunle-ISCA-Keynote-2018.pdf

[7] "Habana Reports MLPerf Inference Results for the Goya Processor in Available Category." [Online]. Available: https://habana.ai/habana-reports-mlperf-inference-results-for-the-goya-processor-in-available-category/

[8] "High Bandwidth Memory." [Online]. Available: https://www.amd.com/en/technologies/hbm

[9] "HPC Tuning Guide for AMD EPYC Processors." [Online]. Available: http://developer.amd.com/wp-content/resources/56420.pdf

[10] "Improving Real-Time Performance by Utilizing Cache Allocation Technology." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/cache-allocation-technology-white-paper.pdf

[11] "Intel and Facebook Accelerate PyTorch Performance with 3rd Gen Intel Xeon Processors and Intel Deep Learning Boost's new BFloat16 capability." [Online]. Available: https://www.intel.com/content/www/us/en/artificial-intelligence/posts/intel-facebook-boost-bfloat16.html

[12] "Intel Architecture Instruction Set Extensions and Future Features Programming Reference." [Online]. Available: https://software.intel.com/content/dam/develop/public/us/en/documents/architecture-instruction-set-extensions-programming-reference.pdf

[13] "Intel Distribution of Caffe." [Online]. Available: https://github.com/intel/caffe/

[14] "Intel Math Kernel Library for Deep Neural Networks (Intel MKL-DNN)." [Online]. Available: https://github.com/intel/mkl-dnn

[15] "Intel Nervana NNP-I MLPerf Inference." [Online]. Available: https://www.intel.ai/mlperf-nov2019/#gs.wp7eto

[16] "Intel(R) Threading Building Blocks Documentation." [Online]. Available: https://software.intel.com/en-us/tbb-reference-manual

[17] "MLPerf Inference Results." [Online]. Available: https://mlperf.org/inference-results/

[18] "NVIDIA cuDNN." [Online]. Available: https://developer.nvidia.com/cudnn

[19] "oneAPI Deep Neural Network Library (oneDNN)." [Online]. Available: https://https://github.com/oneapi-src/oneDNN

[20] "OpenMP Application Programming Interface." [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMPApplicatioopenMP-API-Specification-5.0.pdf

[21] "OpenVINO Toolkit." [Online]. Available: https://software.intel.com/en-us/openvino-toolkit

[22] "PyTorch." [Online]. Available: https://pytorch.org/docs/stable/index.html

[23] "Skylake Server Microarchitecture." [Online]. Available: https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(server)

[24] "Support for the OpenMP Language." [Online]. Available: https://openmp.llvm.org/index.html

[25] "Synopsys Design Compiler Ultra." [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html

[26] "TensorFlow Optimizations on Modern Intel Architecture." [Online]. Available: https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture

[27] "The Apple iPhone 11, 11 Pro & 11 Pro Max Review." [Online]. Available: https://www.anandtech.com/show/14892/the-apple-iphone-11-pro-and-max-review/2

[28] "Understanding Memory Formats." [Online]. Available: https://oneapi-src.github.io/oneDNN/dev_guide_understanding_memory_formats.html

[29] "Vector Neural Network Instructions Enable Int8 AI Inference on Intel Architecture." [Online]. Available: https://www.intel.ai/vnni-enables-inference/

[30] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah,

M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[31] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. T. Blaauw, and R. Das, "Compute caches," in *2017 IEEE International Symposium on High Performance Computer Architecture, HPCA 2017, Austin, TX, USA, February 4-8, 2017*, 2017, pp. 481–492. [Online]. Available: https://doi.org/10.1109/HPCA.2017.21

[32] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A scalable processing-in-memory accelerator for parallel graph processing," *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3, pp. 105–117, 2016.

[33] A. Boroumand, S. Ghose, Y. Kim, R. Ausavarungnirun, E. Shiu, R. Thakur, D. Kim, A. Kuusela, A. Knies, P. Ranganathan, and O. Mutlu, "Google workloads for consumer devices: Mitigating data movement bottlenecks," in *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2018, Williamsburg, VA, USA, March 24-28, 2018*, 2018, pp. 316–331. [Online]. Available: https://doi.org/10.1145/3173162.3173177

[34] T. E. Carlson, W. Heirman, S. Eyerman, I. Hur, and L. Eeckhout, "An evaluation of high-level mechanistic core models," *ACM Transactions on Architecture and Code Optimization (TACO)*, 2014.

[35] B. Chen, T. Medini, J. Farwell, S. Gobriel, T. C. Tai, and A. Shrivastava, "SLIDE : In defense of smart algorithms over hardware acceleration for large-scale deep learning systems," in *Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020*, I. S. Dhillon, D. S. Papailiopoulos, and V. Sze, Eds. mlsys.org, 2020. [Online]. Available: https://proceedings.mlsys.org/book/306.pdf

[36] G. Dai, T. Huang, Y. Chi, J. Zhao, G. Sun, Y. Liu, Y. Wang, Y. Xie, and H. Yang, "Graphh: A processing-in-memory architecture for large-scale graph processing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 4, pp. 640–653, 2018.

[37] Q. Deng, L. Jiang, Y. Zhang, M. Zhang, and J. Yang, "Dracc: a DRAM based accelerator for accurate CNN inference," in *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*, 2018, pp. 168:1–168:6. [Online]. Available: https://doi.org/10.1145/3195970.3196029

[38] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, D. Sylvester, D. T. Blaauw, R. Das, and R. R. Iyer, "Neural cache: Bit-serial in-cache acceleration of deep neural networks," *IEEE Micro*, vol. 39, no. 3, pp. 11–19, 2019. [Online]. Available: https://doi.org/10.1109/MM.2019.2908101

[39] D. Ernst, "Competing in artificial intelligence chips: ChinaâĂŹs challenge amid technology war"." [Online]. Available: https://www.cigionline.org/sites/default/files/documents/CompetinginArtificalIntelligenceChips-DieterErnst_web.pdf

[40] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 1933–1941. [Online]. Available: https://doi.org/10.1109/CVPR.2016.213

[41] D. Fujiki, S. A. Mahlke, and R. Das, "Duality cache for data parallel acceleration," in *Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, Phoenix, AZ, USA, June 22-26, 2019*, 2019, pp. 397–410. [Online]. Available: https://doi.org/10.1145/3307650.3322257

[42] E. Georganas, S. Avancha, K. Banerjee, D. D. Kalamkar, G. Henry, H. Pabst, and A. Heinecke, "Anatomy of high-performance deep learning convolutions on SIMD architectures," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, TX, USA, November 11-16, 2018*, 2018, pp. 66:1–66:12. [Online]. Available: http://dl.acm.org/citation.cfm?id=3291744

[43] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, 2015, pp. 1737–1746. [Online]. Available: http://proceedings.mlr.press/v37/gupta15.html

[44] J. Hanhirova, T. Kämäräinen, S. Seppälä, M. Siekkinen, V. Hirvisalo, and A. Ylä-Jääski, "Latency and throughput characterization of convolutional neural networks for mobile computer vision," in *Proceedings of the 9th ACM Multimedia Systems Conference, MMSys 2018, Amsterdam, The Netherlands, June 12-15, 2018*, 2018, pp. 204–215. [Online]. Available: https://doi.org/10.1145/3204949.3204975

[45] M. Hayenga, V. R. K. Naresh, and M. H. Lipasti, "Revolver: Processor architecture for power efficient loop execution," in *20th IEEE International Symposium on High Performance Computer Architecture, HPCA 2014, Orlando, FL, USA, February 15-19, 2014*. IEEE Computer Society, 2014, pp. 591–602. [Online]. Available: https://doi.org/10.1109/HPCA.2014.6835968

[46] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[47] G. Henry, P. Palangpour, M. Thomson, J. S. Gardner, B. Arden, J. Donahue, K. Houck, J. Johnson, K. OâĂŹBrien, S. Petersen, B. Seroussi, and T. Walker, "High-performance deep-learning coprocessor integrated into x86 soc with server-class cpus," in *47th ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2020, Worldwide Event, May 29-June 3, 2020*, 2020, pp. 15–26.

[48] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *CoRR*, vol. abs/1704.04861, 2017. [Online]. Available: http://arxiv.org/abs/1704.04861

[49] K. Hsieh, S. M. Khan, N. Vijaykumar, K. K. Chang, A. Boroumand, S. Ghose, and O. Mutlu, "Accelerating pointer chasing in 3d-stacked memory: Challenges, mechanisms, evaluation," in *34th IEEE International Conference on Computer Design, ICCD 2016, Scottsdale, AZ, USA, October 2-5, 2016*, 2016, pp. 25–32. [Online]. Available: https://doi.org/10.1109/ICCD.2016.7753257

[50] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 2261–2269. [Online]. Available: https://doi.org/10.1109/CVPR.2017.243

[51] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. B. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the ACM International Conference on Multimedia, MM '14, Orlando, FL, USA, November 03 - 07, 2014*, 2014, pp. 675–678. [Online]. Available: https://doi.org/10.1145/2647868.2654889

[52] N. P. Jouppi, C. Young, N. Patil, D. A. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snelham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, "In-datacenter performance analysis of a tensor processing unit," in *Proceedings of the 44th Annual*

International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017, 2017, pp. 1–12. [Online]. Available: https://doi.org/10.1145/3079856.3080246

[53] A. Krause, T. Kissinger, D. Habich, and W. Lehner, "Nemesys-a showcase of data oriented near memory graph processing," in *Proceedings of the 2019 International Conference on Management of Data*. ACM, 2019, pp. 1945–1948.

[54] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "The mcpat framework for multicore and manycore architectures: Simultaneously modeling power, area, and timing," *TACO*, vol. 10, no. 1, pp. 5:1–5:29, 2013. [Online]. Available: https://doi.org/10.1145/2445572.2445577

[55] E. Lockerman, A. Feldmann, M. Bakhshalipour, A. Stanescu, S. Gupta, D. Sánchez, and N. Beckmann, "Livia: Data-centric computing throughout the memory hierarchy," in *ASPLOS '20: Architectural Support for Programming Languages and Operating Systems, Lausanne, Switzerland, March 16-20, 2020*, J. R. Larus, L. Ceze, and K. Strauss, Eds. ACM, 2020, pp. 417–433. [Online]. Available: https://doi.org/10.1145/3373376.3378497

[56] K. K. Matam, G. Koo, H. Zha, H. Tseng, and M. Annavaram, "Graphssd: graph semantics aware SSD," in *Proceedings of the 46th International Symposium on Computer Architecture, ISCA 2019, Phoenix, AZ, USA, June 22-26, 2019*, 2019, pp. 116–128. [Online]. Available: https://doi.org/10.1145/3307650.3322275

[57] N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI : A Tool to Model Caches/Memories, 3D stacking, and off-chip IO." [Online]. Available: https://github.com/HewlettPackard/cacti

[58] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Enabling practical processing in and near memory for data-intensive computing," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC 2019, Las Vegas, NV, USA, June 02-06, 2019*. ACM, 2019, p. 21. [Online]. Available: https://doi.org/10.1145/3316781.3323476

[59] O. Mutlu, S. Ghose, J. Gómez-Luna, and R. Ausavarungnirun, "Processing data where it makes sense: Enabling in-memory computation," *Microprocess. Microsystems*, vol. 67, pp. 28–41, 2019. [Online]. Available: https://doi.org/10.1016/j.micpro.2019.01.009

[60] J. Park, M. Naumov, P. Basu, S. Deng, A. Kalaiah, D. S. Khudia, J. Law, P. Malani, A. Malevich, N. Satish, J. Pino, M. Schatz, A. Sidorov, V. Sivakumar, A. Tulloch, X. Wang, Y. Wu, H. Yuen, U. Diril, D. Dzhulgakov, K. M. Hazelwood, B. Jia, Y. Jia, L. Qiao, V. Rao, N. Rotem, S. Yoo, and M. Smelyanskiy, "Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications," *CoRR*, vol. abs/1811.09886, 2018. [Online]. Available: http://arxiv.org/abs/1811.09886

[61] J. T. Pawlowski, "Hybrid memory cube (hmc)," in *2011 IEEE Hot Chips 23 Symposium (HCS)*, Aug 2011, pp. 1–24.

[62] V. J. Reddi, C. Cheng, D. Kanter, P. Mattson, G. Schmuelling, C. Wu, B. Anderson, M. Breughe, M. Charlebois, W. Chou, R. Chukka, C. Coleman, S. Davis, P. Deng, G. Diamos, J. Duke, D. Fick, J. S. Gardner, I. Hubara, S. Idgunji, T. B. Jablin, J. Jiao, T. S. John, P. Kanwar, D. Lee, J. Liao, A. Lokhmotov, F. Massa, P. Meng, P. Micikevicius, C. Osborne, G. Pekhimenko, A. T. R. Rajan, D. Sequeira, A. Sirasao, F. Sun, H. Tang, M. Thomson, F. Wei, E. Wu, L. Xu, K. Yamada, B. Yu, G. Yuan, A. Zhong, P. Zhang, and Y. Zhou, "Mlperf inference benchmark," *CoRR*, vol. abs/1911.02549, 2019. [Online]. Available: http://arxiv.org/abs/1911.02549

[63] A. Reuther, P. Michaleas, M. Jones, V. Gadepally, S. Samsi, and J. Kepner, "Survey and benchmarking of machine learning accelerators," in *2019 IEEE High Performance Extreme Computing Conference, HPEC 2019, Waltham, MA, USA, September 24-26, 2019*. IEEE, 2019, pp. 1–9. [Online]. Available: https://doi.org/10.1109/HPEC.2019.8916327

[64] A. Rodriguez, E. Segal, E. Meiri, E. Fomenko, Y. J. Kim, H. Shen, and B. Ziv, "Lower numerical precision deep learning inference and training," *Intel White Paper*, 2018.

[65] V. Seshadri, Y. Kim, C. Fallin, D. Lee, R. Ausavarungnirun, G. Pekhimenko, Y. Luo, O. Mutlu, P. B. Gibbons, M. A. Kozuch, and T. C. Mowry, "Rowclone: fast and energy-efficient in-dram bulk data copy and initialization," in *The 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, Davis, CA, USA, December 7-11, 2013*, 2013, pp. 185–197. [Online]. Available: https://doi.org/10.1145/2540708.2540725

[66] V. Seshadri, D. Lee, T. Mullins, H. Hassan, A. Boroumand, J. Kim, M. A. Kozuch, O. Mutlu, P. B. Gibbons, and T. C. Mowry, "Ambit: in-memory accelerator for bulk bitwise operations using commodity DRAM technology," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2017, Cambridge, MA, USA, October 14-18, 2017*, 2017, pp. 273–287. [Online]. Available: https://doi.org/10.1145/3123939.3124544

[67] A. Shafaei, Y. Wang, X. Lin, and M. Pedram, "Fincacti: Architectural analysis and modeling of caches with deeply-scaled finfet devices," in *IEEE Computer Society Annual Symposium on VLSI, ISVLSI 2014, Tampa, FL, USA, July 9-11, 2014*. IEEE Computer Society, 2014, pp. 290–295. [Online]. Available: https://doi.org/10.1109/ISVLSI.2014.94

[68] A. Stillmaker and B. M. Baas, "Corrigendum to "scaling equations for the accurate prediction of CMOS device performance from 180nm to 7nm" [integr. VLSI j. 58. (2017) 74-81]," *Integr.*, vol. 67, p. 170, 2019. [Online]. Available: https://doi.org/10.1016/j.vlsi.2019.04.006

[69] S. M. Tam, H. Muljono, M. Huang, S. Iyer, K. Royneogi, N. Satti, R. Qureshi, W. Chen, T. Wang, H. Hsieh, S. Vora, and E. Wang, "Skylake-sp: A 14nm 28-core xeon® processor," in *2018 IEEE International Solid-State Circuits Conference, ISSCC 2018, San Francisco, CA, USA, February 11-15, 2018*. IEEE, 2018, pp. 34–36. [Online]. Available: https://doi.org/10.1109/ISSCC.2018.8310170

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, 2017, pp. 5998–6008. [Online]. Available: http://papers.nips.cc/paper/7181-attention-is-all-you-need

[71] C. Wu, D. Brooks, K. Chen, D. Chen, S. Choudhury, M. Dukhan, K. M. Hazelwood, E. Isaac, Y. Jia, B. Jia, T. Leyvand, H. Lu, Y. Lu, L. Qiao, B. Reagen, J. Spisak, F. Sun, A. Tulloch, P. Vajda, X. Wang, Y. Wang, B. Wasti, Y. Wu, R. Xian, S. Yoo, and P. Zhang, "Machine Learning at Facebook: Understanding Inference at the Edge," in *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2019, pp. 331–344.

[72] S. Xie, R. B. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, 2017, pp. 5987–5995. [Online]. Available: https://doi.org/10.1109/CVPR.2017.634

[73] M. Xu, J. Liu, Y. Liu, F. X. Lin, Y. Liu, and X. Liu, "A first look at deep learning apps on smartphones," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, 2019, pp. 2125–2136. [Online]. Available: https://doi.org/10.1145/3308558.3313591

[74] C. Zhang, M. Yu, W. Wang, and F. Yan, "Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving," in *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, 2019, pp. 1049–1062. [Online]. Available: https://www.usenix.org/conference/atc19/presentation/zhang-chengliang

[75] M. Zhang, Y. Zhuo, C. Wang, M. Gao, Y. Wu, K. Chen, C. Kozyrakis, and X. Qian, "Graphp: Reducing communication for pim-based graph processing with efficient data partition," in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2018, pp. 544–557.

[76] M. Zhang, S. Rajbhandari, W. Wang, and Y. He, "Deepcpu: Serving rnn-based deep learning models 10x faster," in *2018 USENIX Annual Technical Conference, USENIX ATC 2018, Boston, MA, USA, July 11-13, 2018*, 2018, pp. 951–965. [Online]. Available: https://www.usenix.org/conference/atc18/presentation/zhang-minjia