DOI:10.1145/3460227

Simba: Scaling Deep-Learning Inference with Chiplet-Based Architecture

By Yakun Sophia Shao, Jason Cemons, Rangharajan Venkatesan, Brian Zimmer, Matthew Fojtik, Nan Jiang, Ben Keller, Alicia Klinefelter, Nathaniel Pinckney, Priyanka Raina, Stephen G. Tell, Yanqing Zhang, William J. Dally, Joel Emer, C. Thomas Gray, Brucek Khailany, and Stephen W. Keckler

Abstract
Package-level integration using multi-chip-modules (MCMs) is a promising approach for building large-scale systems. Compared to a large monolithic die, an MCM combines many smaller chiplets into a larger system, substantially reducing fabrication and design costs. Current MCMs typically only contain a handful of coarse-grained large chiplets due to the high area, performance, and energy overheads associated with inter-chiplet communication. This work investigates and quantifies the costs and benefits of using MCMs with fine-grained chiplets for deep learning inference, an application domain with large compute and on-chip storage requirements. To evaluate the approach, we architected, implemented, fabricated, and tested Simba, a 36-chiplet prototype MCM system for deep-learning inference. Each chiplet achieves 4 TOPS peak performance, and the 36-chiplet MCM package achieves up to 128 TOPS and up to 6.1 TOPS/W. The MCM is configurable to support a flexible mapping of DNN layers to the distributed compute and storage units. To mitigate inter-chiplet communication overheads, we introduce three tiling optimizations that improve data locality. These optimizations achieve up to 16% speedup compared to the baseline layer mapping. Our evaluation shows that Simba can process 1988 images/s running ResNet-50 with a batch size of one, delivering an inference latency of 0.50 ms.

1. INTRODUCTION

Deep learning (DL) has become critical for addressing complex real-world problems. In particular, deep neural networks (DNNs) have demonstrated their effectiveness across a wide-range of applications. State-of-the-art DNNs[12] require billions of operations and hundreds of megabytes to store activations and weights. Given the trend toward even larger and deeper networks, the ensuing compute and storage requirements motivate the large-scale compute capability in DL hardware, which is currently addressed by a combination of large monolithic chips and homogeneous multi-chip board designs.[9, 15] Previously proposed multi-chip DL accelerators have focused on improving total compute throughput and on-chip storage size but have not addressed the scalability challenges associated with building a large-scale system with multiple discrete components.

Recently, the need for high compute throughput in an era of slowing transistor scaling has motivated advances in multi-chip-module (MCM) integration to build large-scale CPUs[3] and GPUs.[1] MCM packaging approaches can also reduce cost by employing smaller chiplets connected together postfabrication, as yield losses cause fabrication cost to grow super-linearly with die size. Packaging technologies such as organic substrates[13] and silicon interposers[11] can be used to assemble a large-scale MCM system. In addition, the recent advances in package-level signaling offer the necessary high-speed, high-bandwidth signaling needed for a chiplet-based system.[24] As a result, chiplet-based MCM systems can provide improved performance more efficiently than the board-level integration but with lower cost than monolithic chips. Although MCMs have been used for general compute systems, applying MCMs to high-performance DNN inference algorithms has not been previously examined. Specific challenges stem from the natural nonuniformity between on-chip and on-package bandwidth and latency. Although multi-chip systems also exhibit similar forms of nonuniformity, this paper focuses on the specific characteristics of MCM-based systems as they provide a natural progression beyond monolithic single-chip inference accelerators.

This paper presents Simba, a scalable deep-learning inference accelerator employing multi-chip-module-based integration. Each of the Simba chiplets can be used as a standalone, edge-scale inference accelerator, whereas multiple Simba chiplets can be packaged together to deliver a data-center-scale compute throughput. To explore the challenges and evaluate the benefits of MCM-based inference accelerator architectures, we designed, implemented, and fabricated a prototype of Simba, consisting of 36 chiplets connected via a mesh network in an MCM.[25] We specifically examine the implications of the nonuniform network access (NUNA) architecture with nonuniform latency and bandwidth for on-chip and on-package communication that lead to significant latency variability across chiplets. Such latency variability results in a long tail latency during the execution of individual inference layers. As a result, the overall performance for each layer is restricted by the slowest chiplet in the system, limiting scalability. To address these challenges, we propose three tail-latency-aware, nonuniform tiling optimizations targeted at improving locality and minimizing inter-chiplet communication: (1) nonuniform work partitioning to balance

The original version of this paper appeared in the *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture (Columbus, OH, USA, Oct. 2019).*

compute latency with communication latency; (2) communication-aware data placement to minimize inter-chiplet traffic; and (3) cross-layer pipelining to improve resource utilization.

2. BACKGROUND

Package-level MCM integration is a promising alternative for assembling large-scale systems out of small building blocks known as *chiplets*. Such systems consist of multiple chiplets connected together via on-package links using a silicon interposer or an organic substrate and employing efficient intra-package signaling circuits.[3, 24] Compared to a large monolithic die, MCMs can reduce (1) design costs, because logic design, verification, and physical design are all easier on a small chip than a large chip; and (2) fabrication costs, as the much lower manufacturing yield of large chips makes them far more expensive than small chips. In addition, different scales of systems can be created merely by adjusting the number of chiplets placed in a package, without requiring a different chip tapeout for each market segment. MCMs have been recently applied to a general-purpose CPU design[3] as an alternative to building multi-core CPUs on reticle-limited large die. They have also been an active research area for scaling of multi-CPU[16] and multi-GPU systems.[1] However, package-level wires do not provide the same communication density or energy/bit as on-chip wires. Consequently, MCM architects and software developers must still consider the nonuniform bandwidth, latency, and energy present in these systems to achieve an efficient application performance.

An MCM-based system has a heterogeneous interconnect architecture, as the available intra-chiplet bandwidth is expected to be significantly higher than available inter-chiplet bandwidth. In addition, sending data to remote chiplets incurs additional latency. This latency may include on-chip wire delays to move data to the edge of the chiplet, synchronizer delays for crossing clock domains, serialization and deserialization latency in high-speed communication links, and the on-package wire delays of inter-chiplet links. As a result, the communication latency between two elements in an MCM depends heavily on their spatial locality on the package.

Mapping DNN layers to a tile-based architecture is a well-studied research problem.[6, 19] The state-of-the-art DNN tiling typically assumes a flat architecture with uniform latency and bandwidth across processing elements (PEs) and focuses on data reuse for reducing global bandwidth demands. This assumption is acceptable for small-scale systems, as the communication latency variability is small and the computation is often tolerant of communication latencies. However, as the DNN inference performance is scaled-up to larger systems, the execution time decreases and latency-related effects become more important. Furthermore, in the large-scale systems with heterogeneous interconnect architectures such as MCMs, the assumptions of uniform latency and bandwidth in selecting DNN tiling can degrade the performance and energy efficiency. Simba is the first work that quantitatively highlights the challenge of mapping DNN layers to nonuniform, MCM-based DNN accelerators and proposes communication-aware tiling strategies to address the challenge.

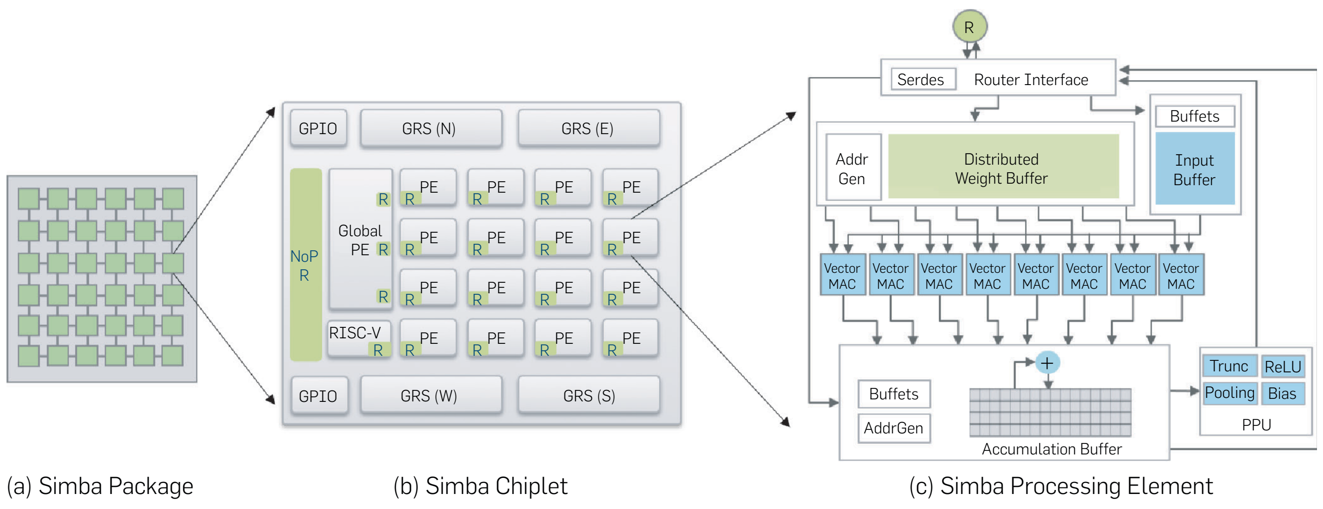
3. SIMBA ARCHITECTURE AND SYSTEM

To understand the challenges and opportunities of using MCMs for building large-scale, deep-learning systems, we designed, implemented, fabricated, and characterized Simba, the first chiplet-based deep-learning system. This section first presents an overview of the Simba architecture and its default uniform tiling strategy. We then describe Simba's silicon prototype and present a detailed characterization of the Simba system in Section 4.

3.1. Simba architecture

Tile-based architectures have been frequently proposed for deep-learning accelerator designs.[6, 5, 20] Our design target is an accelerator scalable to data center inference, where state-of-the-art data center accelerators deliver around 100 tera-operations-per-second (TOPS). For example, the first generation of the tensor processing unit (TPU) delivers 92 TOPS[15] and is designed for inference applications. One simple approach to achieve this design goal is to increase the number of tiles in a monolithic single chip. However,

Figure 1. Simba architecture from package to processing element (PE).



building a flat network with hundreds of tiles would lead to high tile-to-tile communication latency, as examined in both multi-core CPU[7] and accelerator[10] research.

Simba adopts a hierarchical interconnect to efficiently connect different processing elements (PEs). This hierarchical interconnect consists of a network-on-chip (NoC) that connects PEs on the same chiplet and a network-on-package (NoP) that connects chiplets together on the same package. Figure 1 illustrates the three-level hierarchy of the Simba architecture: package, chiplet, and PE. Figure 1(a) shows a Simba package consisting of a 6×6 array of Simba chiplets connected via a mesh interconnect. Each Simba chiplet, as shown in Figure 1(b), contains an array of PEs, a global PE, a NoP router, and a controller, all connected by a chiplet-level interconnect. To enable the design of a large-scale system, all communication between the PEs, Global PEs, and controller is designed to be latency-insensitive[4] and is sent across the interconnection network through the NoC/NoP routers.

Simba PE. Figure 1(c) shows the microarchitecture of the Simba PE, which includes a distributed weight buffer, an input buffer, parallel vector MAC units, an accumulation buffer, and a postprocessing unit. Each Simba PE is similar to a scaled-down version of NVDLA, a state-of-the-art DL accelerator product.[22] The heart of the Simba PE is an array of parallel vector multiply-and-add (MAC) units that are optimized for efficiency and flexibility. The Simba PE uses a weight-stationary dataflow[6]: weights remain in the vector MAC registers and are reused across iterations, although new inputs are read every cycle. Each vector MAC performs an 8:1 dot-product along the input channel dimension C to exploit an efficient spatial reduction. To provide flexible tiling options, the Simba PE also supports cross-PE reduction with configurable producers and consumers. If the current PE is the last PE on the reduction chain, it first sends partial sums to its local postprocessing unit that performs ReLU, truncation and scaling, pooling, and bias addition. The final output activation is sent to the target Global PE for computation of the next layer.

Simba global PE. The Global PE serves as a second-level storage for input/output activation data to be processed by the PEs. To support flexible partitioning of the computation, the Global PE can either unicast data to one PE or multicast to multiple PEs, even across chiplet boundaries. The Global PE has a multicast manager that oversees these producer-consumer relationships. The Global PE also serves as a platform for near-memory computation. Many DNNs feature some computation that has low data reuse, such as element-wise multiply/add or depth-wise convolution. The Global PE can perform such computations locally to reduce communication overhead for these types of operations.

Simba controller. Each Simba chiplet contains a RISC-V processor core[2] that is responsible for configuring and managing the chiplet's PEs and Global PE states via memory-mapped registers using an AXI-based communication protocol. After all states are configured, the RISC-V triggers the execution in the active PEs and Global PEs and waits for these blocks to send *done* notifications via interrupts. Synchronization of chiplet control processors across the package is implemented via memory-mapped interrupts.

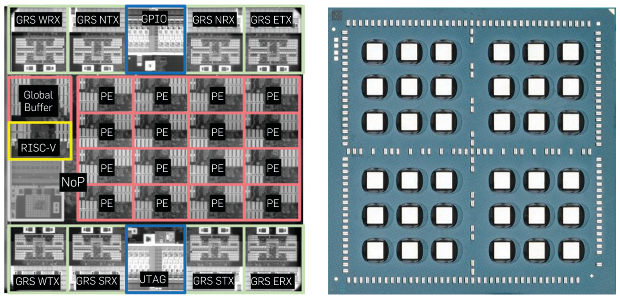
Simba interconnect. To efficiently execute different neural networks with diverse layer dimensions, Simba supports flexible communication patterns across the NoC and NoP. Both NoC and NoP use a mesh topology with a hybrid wormhole/cut-through flow control. Specifically, unicast packets use wormhole flow control for large packet size, whereas multicast packets are cut-through to avoid wormhole deadlocks. Each Simba PE can unicast to any local or remote PE for cross-PE partial-sum reduction, to any local or remote Global PE to transmit output activation values, and to any local or remote chiplet controller to signal execution completion. A PE does not need to send multicast packets as its computation requires only point-to-point communication. In addition to unicast communication, a Global PE can also send multicast packets to local and remote PEs for flexible data tiling.

3.2. Simba silicon prototype

We implemented, fabricated, and tested a silicon prototype of the Simba system, as shown in Figure 2, with the microarchitecture parameters listed in the original Simba paper.[21] We chose parameters so that a Simba chiplet has area and power similar to an efficient edge system, such as DianNao[5] or Eyeriss,[6] whereas a full Simba package is comparable to a data-center-scale system such as TPU.[15] Table 1 shows the synthesis area breakdown of key components in the Simba chiplet architecture.

```
//Package level
for p3 = [0: P3):
  for q3 = [0: Q3):
    parallel_for k3 = [0: K3):
      parallel_for c3 = [0: C3):
```

Figure 2. Simba silicon prototype.



(a) Simba chiplet (b) Simba package

Table 1. Area breakdown of the Simba system.

Partition	Component	Area $(\lambda m)^2$
PE	Vector MACs	12K
	Weight Buffer	41K
	Input Buffer	11K
	Accumulation Buffer	24K
	NoC Router	19K
Global PE	Distributed Buffer	125K
	NoC Routers	27K
RISC-V	Processor	109K
NoP	NoP Router	42K

```
// Chiplet level
for p2 = [0: P2):
  for q2 = [0: Q2):
    parallel_for k2 = [0: K2):
      parallel_for c2 = [0: C2):
// PE level
for r = [0: R):
  for s = [0: S):
    for k1 = [0: K1):
      for c1 = [0: C1):
        for p1 = [0: P1):
          for q1 = [0: Q1):
// Vector-MAC level
parallel_for k0 = [0: K0):
  parallel_for c0 = [0: C0):
   p = (p3 * P2 + p2) * P1 + p1;
   q = (q3 * Q2 + q2) * Q1 + q1;
   k = ((k3 * K2 + k2) * K1 + k1) * K0 + k0;
   c = ((c3 * C2 + c2) * C1 + c1) * C0 + c0;
  OA[p,q,k] += IA[p-1+r,q-1+s,c] * W[r,s,c,k];
```

Listing 1. Simba baseline dataflow.

As shown in Figure 2a, the 2.5 × 2.4 Simba chiplets were implemented in a TSMC 16 nm FinFET process technology.[25] Each Simba package (Figure 2b) contains an array 6×6 of chiplets connected on an organic package substrate using a ground-referenced signaling (GRS) technology for intra-package communication.[24] The top and bottom rows of each chiplet include eight chiplet-to-chiplet GRS transceiver macros. Four macros are configured as receivers and four as transmitters. Each transceiver macro has four data lanes and a clock lane with configurable speed from 11 Gbps/pin to 25 Gbps/pin, consuming 0.82–1.75 pJ/bit, with a total peak chiplet bandwidth of 100 GB/s. We chose GRS as our communication mechanism because it delivers 3.5× higher bandwidth per unit area and lower energy per bit compared to other MCM interconnects.[3]

The prototype chiplets were implemented using a globally asynchronous, locally synchronous (GALS) clocking methodology,[8] allowing independent clock rates for individual PEs, Global PEs, RISC-V processors, and NoP routers. Running in a single-chiplet configuration, Simba prototypes have been measured to operate correctly in the lab at a minimum voltage of 0.42 V with a 161 MHz PE frequency, achieving 0.11 pJ/Op (9.1 TOPS/W) core power efficiency on a peak-utilization convolution micro-benchmark. At 1.2 V, each chiplet operates with a 2 GHz PE frequency for a peak throughput of 4 TOPS. The 36-chiplet Simba system is functional over a slightly narrower voltage range, from 0.52 to 1.1 V, achieving 0.16 pJ/op at 0.52 V and 484 MHz; at 1.1 V, the 36-chiplet system achieves a 1.8 GHz PE frequency and 128 TOPS.

3.3. Simba baseline tiling

To map the DNN layers onto the hierarchical tile-based architecture, we first use a state-of-the-art DNN tiling strategy that uniformly partitions weights spatially, leveraging model parallelism.[22, 15, 5, 20] Listing 1 shows the default tiling in a loop-nest form. Each dimension of a DNN layer can be tiled temporally, spatially, or both at each level of the system hierarchy: package, chiplet, PE, and vector MAC. The loop bounds and orderings in Listing 1 are configurable in Simba so that users can flexibly map computation to the Simba system. In particular, the default dataflow uniformly partitions weights along the input channel (C) and the output channel (K) dimensions, as noted in the `parallel_for` loops. In addition, Simba can also uniformly partition along the height (P) and width (Q) dimensions of an output activation across chiplets and PEs to support flexible tiling. Section 5 highlights the limitations of this approach when mapping networks onto a large-scale, nonuniform network access architecture with an MCM-based integration.

We developed a flow that uses Caffe[14] to map a DNN inference application to the Simba system, which primarily determines an efficient tiling strategy for the dataflow that best exploits data reuse in the memory hierarchy. To facilitate the evaluation of different mapping alternatives, we also developed a fast, analytical energy model for Simba that quantifies the energy cost of a particular mapping, similar to the methodology discussed in prior work.[6, 19] The compilation process starts with a *mapper* that is provided with data regarding available system resources (such as the number of PEs, the number of Global PEs, and the sizes of buffers in the system) and the parameters of a given layer from the Caffe specification. The mapper determines which PE will run each portion of the loop nest and in which buffers the activations and weights are stored. As this mapping is a logical one, the mapper is followed by a *binder* which decides in which physical resource in the Simba topology the loop nests and data structures are placed. We use a random search algorithm to sample the mapping space and use the energy and performance models to select good mappings and placements. Finally, the flow generates the configuration binaries for each chiplet that implement the execution created by the mapper and binder.

4. SIMBA CHARACTERIZATION

This section details the performance characterization of Simba, focusing on achieved scalability using the uniform-tiling baseline. All evaluation results are measured using the prototype system.

4.1. Methodology

Figure 3 shows the experimental setup for measuring the performance and power of the Simba prototype system. The silicon prototype test board is attached to an x86 host through PCI-E using a Xilinx FPGA. To measure the performance of the Simba prototype system, we use software running on the RISC-V to query cycle counters built into the RISC-V microcontrollers. Power and performance measurements begin after the weights have been loaded into each PE's weight buffer and the inputs have been loaded into the Global PE buffers. Unless otherwise noted, the chiplets operate at a core voltage of 0.72 V, a PE frequency of 1.03 GHz, and GRS bandwidth of 11 Gbps. We use sense resistors on the board power supplies and a digital acquisition module to measure energy during experiment execution. As the chiplets support independent clock frequencies for different units (PEs, Global PEs, RISC-V, and NoP routers), we can vary these frequencies to change the compute-to-bandwidth ratios for our experiments. The NoC and NoP routing tables use dimension-ordered X-Y routing for all inter-chiplet communication. Although all 36 Simba chiplets are functional, our evaluation uses 32 chiplets, as it is easier to partition computation by powers of two because the number of input channels (C) and output channels (K) are typically powers of two.

We focus our application measurements on ResNet-50,[12] a state-of-the-art, representative deep-learning network, and evaluate its layers running on the Simba system with a batch size of one, as low-latency inference is a highly critical deployment scenario for data center inferencing.[15] We compile and run each layer independently, except when we map multiple layers to different physical partitions of Simba and execute them in a pipelined manner. Networks are pretrained and quantized to 8-bit using TensorRT without accuracy loss. We believe that the diversity of layers in ResNet-50 provides a sufficient breadth to cover a wide range of behaviors across different convolutional networks.

4.2. Performance/energy overview

Figure 4 summarizes the performance and energy measurements across all ResNet-50 layers. Each point represents a unique mapping for that layer, whereas different colors show

Figure 3. Bench measurement setup for the Simba prototype.

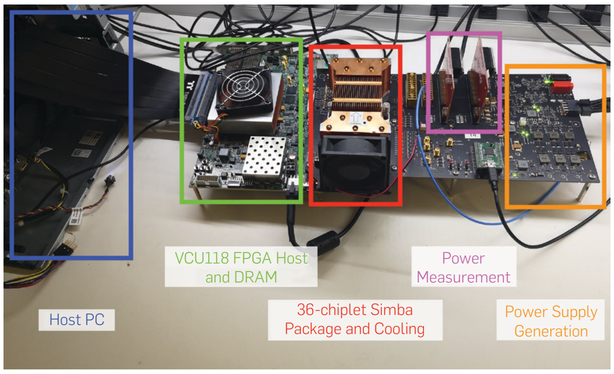
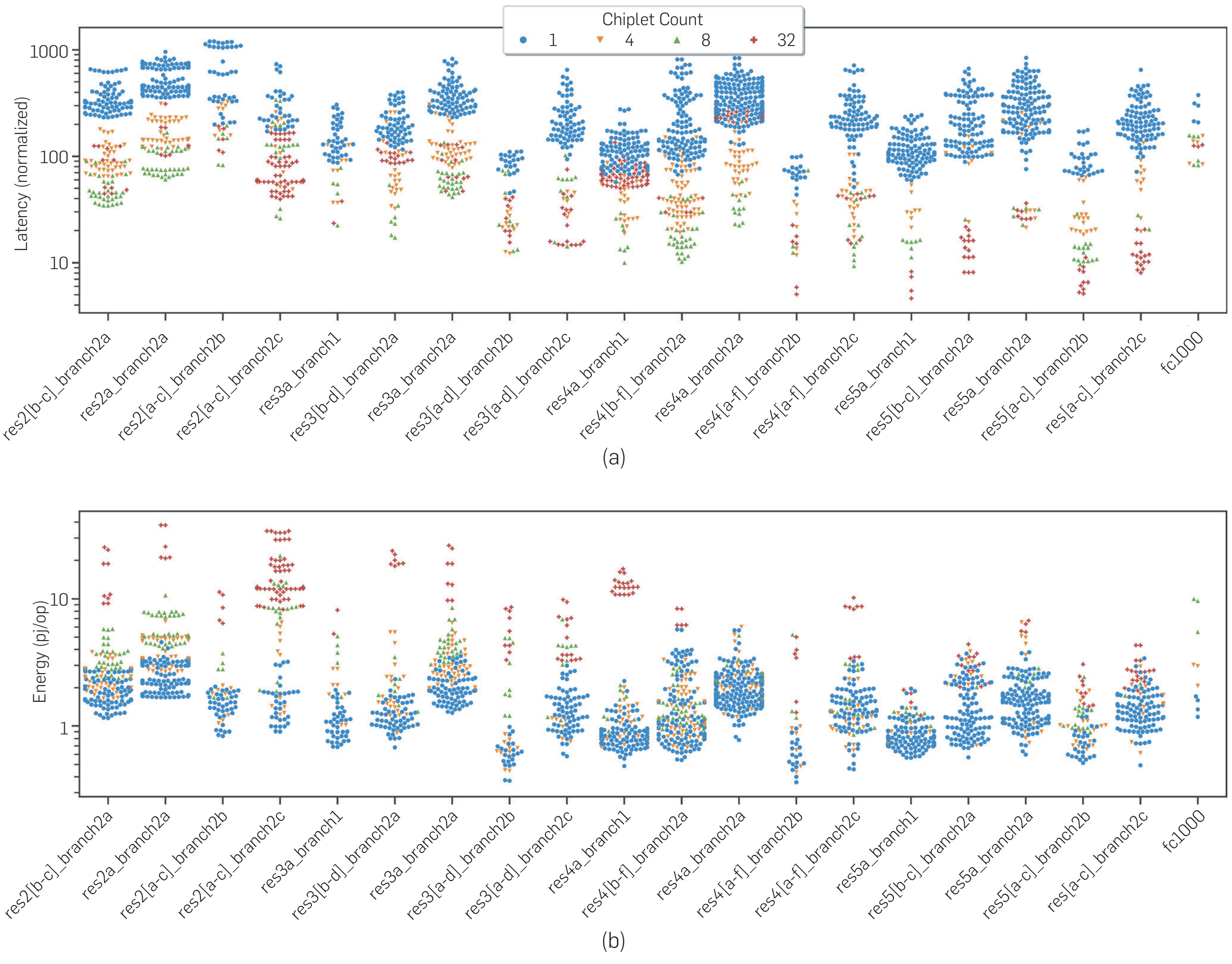

VCU118 FPGA Host and DRAM
Power Measurement
Host PC
36-chiplet Simba Package and Cooling
Power Supply Generation

Figure 4. Measured performance and energy and running ResNet-50 on the fabricated Simba prototype. Each point is a valid workload mapping onto the system; each column cluster shows the different performance and energy achieved by different mappings of the same workload. Each symbol shape represents a different number of active chiplets used for the mapping.



the number of chiplets active for that mapping. Latency is normalized to a hypothetical best-achievable latency that would be realized if each of the 576 PEs of the system operated with 100% utilization and no communication or synchronization overheads. Simba provides a large number of mapping options with drastically different performance and energy profiles, highlighting the importance of strategies for efficiently mapping DNNs to hardware. The figure also demonstrates the highly variable behavior of different layers. For example, the most energy-efficient configurations of layer `res3[b-d]_branch2b` achieve almost an order of magnitude better efficiency than those of `res3a_branch2a`. The degree of data reuse highly influences the efficiency; layers with high reuse factors, for example, 3×3 the convolution in `3[b-d]_branch2b`, tend to perform computation more efficiently than layers that require more data movement. Finally, although increasing the number of chiplets used in the system improves performance, it also leads to increased energy cost for chiplet-to-chiplet communication and synchronization. Efficiency can drop by nearly an order of magnitude for some layers, which further emphasizes the effect of data movement on overall efficiency. To better understand the system-level trade-offs, the remainder of this section characterizes the sensitivity of Simba to mapping alternatives, layer parameters, bandwidth, latency, and weak scaling, and includes a comparison to modern GPUs.

4.3. Layer sensitivity

Figure 5 shows the performance scalability of running three different layers in ResNet-50 across different numbers of chiplets. Although the performance of `res2[a-c]_branch2b` initially improves with increased chiplet count, the performance gains cease beyond eight chiplets. As one of the early layers of the network, the number of weights in this layer is so small that it cannot fully utilize the compute throughput of Simba. The performance degrades with 32 chiplets because the inter-PE communication costs overwhelm the limited parallelism. By contrast, the performance of the `res2a_branch1` layer improves when increasing the number of chiplets from 1 to 8, though it stops improving from 8 to 32 chiplets. This layer has more compute parallelism than `res2[a-c]_branch2b`, but it still does not have enough to fully overcome the overheads of inter-chiplet communication.

The `res5[a-c]_branch2b` layer demonstrates the best performance scaling, with improvements observed up to 32 chiplets. However, the performance scaling slows down significantly past eight chiplets due to communication overheads. Of the 53 layers of ResNet-50, 12 follow the behavior of `res3a_branch1`, 24 follow that of `res5[a-c]_branch2b`, and the remaining 17 have behavior similar to `res2[a-c]_branch2b`. These measurements demonstrate that the amount of compute parallelism that an MCM can leverage varies from layer to layer, and that the cost of communication can hinder the ability to exploit that parallelism, even on a single chiplet.

4.4. NoP bandwidth sensitivity

Figure 6 shows how execution time is affected by NoP bandwidth for two representative ResNet layers when mapped to 32 chiplets. We adjusted the bandwidth of the NoP relative to the intra-chiplet compute performance to measure the network bandwidth sensitivity. For `res3[a-d]_branch2b`, the increased bandwidth between chiplets results in only a 5% decrease in execution time, indicating that this layer is not bound by the NoP bandwidth or inter-chiplet communication latency. However, for `res3a_branch1`, the increased

Figure 5. Simba scalability across different layers from ResNet-50. Latency is normalized to the latency of the best-performing tiling with one chiplet.

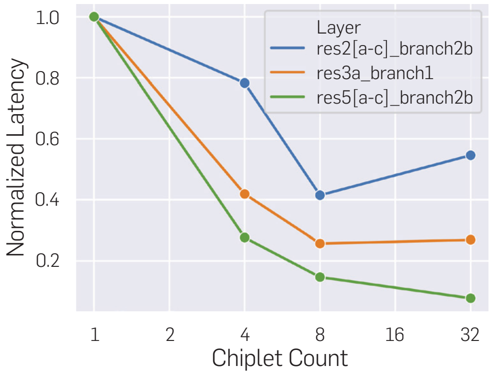


Figure 6. Simba scalability with different chiplet-to-chiplet communication bandwidths.

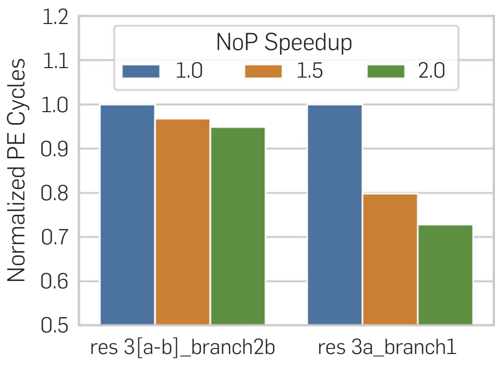
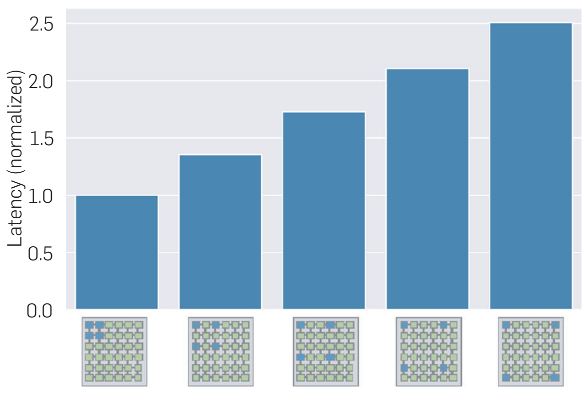


Figure 7. Simba scalability with different chiplet-to-chiplet communication latencies running `re+s4a_branch1` with four chiplets (using the same tiling). Different bars represent different selections of the active four chiplets, as shown under the *X*-axis; the active chiplets are highlighted in blue.



bandwidth decreases execution time by 27%, indicating that this layer is bottlenecked by communication between chiplets. Because an MCM-based system intrinsically has a nonuniform network architecture between intra-chiplet and inter-chiplet PEs, mapping policies must consider the different latency and bandwidth parameters to deliver good performance and efficiency.

4.5. NoP latency sensitivity

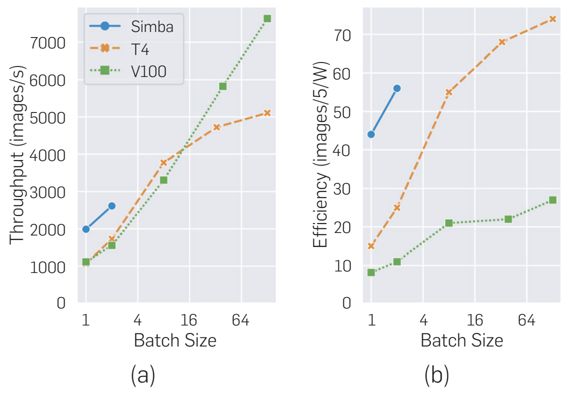
In addition to lower bandwidth, the NoP has higher latency than the NoC due to inter-chiplet signaling overheads. To isolate the effect of NoP latency, we ran experiments mapping layers to four chiplets, but adjusted the locations of the selected chiplets in the package to modulate latency. Figure 7 shows the effect of increasing the longest inter-chiplet latency from 2 hops to 12 hops for `res4a_branch1`. The figure shows the execution time normalized to a configuration of adjacent chiplets, with the chiplet selection shown under each bar. With active chiplets further apart, the overall execution time increases by up to 2.5× compared to execution on adjacent chiplets. Communication latency is typically less pronounced for small-scale systems but plays a significant role in achieving good performance and energy efficiency for a large-scale, MCM-based system like Simba.

4.6. Comparisons with GPUs

Figure 8 compares Simba to NVIDIA's V100 and T4 GPUs. We run ResNet-50 with different batch sizes and compare to the GPU results that are published.[18] Due to Simba's limited on-package storage capacity for input activations, we only run Simba at batch size one and two. Unlike GPUs, Simba is designed for low-latency inference with a small batch size, which motivates the use of distributed and persistent weight storage to reduce data movement. The Simba package, such as the MCM interface, has substantially a smaller total silicon area (216) than T4 (525) or V100 (815), due to differences in math precision, on-chip storage, DRAM interface, and types of computation supported in these architectures.

Figure 8a shows the throughput of Simba, V100, and T4 running ResNet-50. Simba delivers 1.8× and 1.9× better throughput at batch size one compared to V100 and T4, respectively. Figure 8b illustrates the corresponding energy efficiency

Figure 8. Throughput and efficiency of Simba, V100, and T4 running ResNet-50 with different batch sizes.



improvement of Simba compared to V100 (5.4×) and T4 (2.9×). When running ResNet-50 with a larger batch size, instead of exploiting the batch-level parallelism like GPUs, Simba would run each batch sequentially. As a result, we expect that the throughput of Simba is close to that of running with a batch size of one.

5. SIMBA NONUNIFORM TILING

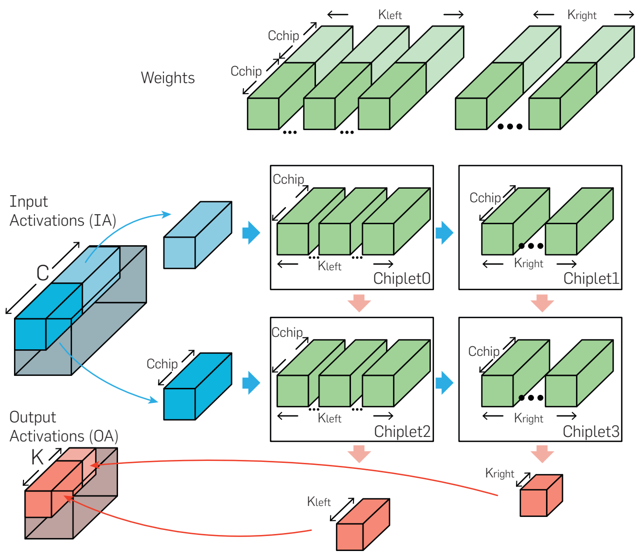
This section presents the details of two novel DNN workload tiling techniques that target the nonuniform latency and bandwidth presented by large-scale MCM-based systems, nonuniform work partitioning and communication-aware data placement. Our results indicate the importance of communication-latency-aware tiling when mapping DNN workloads to large-scale, hierarchical systems.

5.1. Nonuniform work partitioning

Efficient use of parallel systems requires proper load balancing among the components in the system. Failure to properly balance the load on the system leads to higher latency and energy caused by resources waiting for the slowest unit to complete, that is, increased tail latency. The total execution time can be subdivided into two major components: communication latency and compute latency. The state-of-the-art DNN tiling strategies typically assign the same amount of the work to each of the available resources.[23] However, this approach breaks down for large-scale systems, especially when the PEs are spatially distributed with different communication latencies among them.

To address this limitation, we propose a nonuniform work partitioning strategy that considers communication latencies. Instead of uniformly assigning the same amount of work to each PE, we nonuniformly partition the work across the PEs. PEs closer to the data producers will perform more work to

Figure 9. Illustration of communication-aware, nonuniform work partitioning. The top green tensors represent weights (W), the left blue tensors represent input activations (IA), and the bottom red tensors represent the output activation (OA). In this example, IA is stored in Chiplet0 and Chiplet2.



maximize physical data locality, whereas PEs that are further away will do less work to decrease the tail latency effects. Figure 9 illustrates an example of nonuniform work partitioning using a 4-chiplet system. In this example, we assume that the input activation (IA) is physically stored in the Global PEs of `Chiplet0` and `Chiplet2`, whereas the weights (W) and the work are partitioned across all the four chiplets. During execution, the Global PE of `Chiplet0` will multicast a slice of IA to PEs in both `Chiplet0` and `Chiplet1`. Because the communication latencies from the `Chiplet0` Global PE to the PEs in `Chiplet0` and `Chiplet1` are different, `Chiplet1` will fall behind. To prevent the longer communication to `Chiplet1` from increasing the tail latency of the execution, we can adjust the amount of computation that each chiplet is assigned in a manner inversely proportional to its communication distance from the source. In the example as shown in Figure 9, `Chiplet0` and `Chiplet2` are provided with larger chunks of work (K_{left}), whereas `Chiplet1` and `Chiplet3` get the smaller chunks (K_{right}). This work schedule evens out the completion time across the chiplets, thereby improving overall system performance. For simplicity, this example only shows nonuniform partitioning with respect to input activations. A similar technique can be used to mitigate the communication latency for output activations to the destination chiplets by using nonuniform partitioning along the C dimension.

The variation in communication latency is quite pronounced in large-scale systems such as Simba, with hundreds of spatially distributed PEs. To dynamically adjust the amount of work that each PE performs, we use the performance counters within each PE to collect accurate latency and utilization information during the initial execution of a layer. We then adjust the work distribution for the subsequent executions of each layer based on the latency variation across PEs.

Figure 10 illustrates the measured performance improvement using nonuniform work partitioning. For each of the layers, we pick the highest performance uniform tiling from Figure 5 as the baseline. We then measure the execution time of different chiplets and identify layers with a large tail latency. For these layers, we use nonuniform work partitioning to shift the computation from the tail PEs to the PEs that are closer to the data. Depending on layer dimensions, we achieve up to 15% performance improvement compared to the best uniform tiling for a given layer. The achievable performance improvement is highly sensitive to the compute-and-communication ratio in a given mapping. For example, when either compute or communication is significantly dominating the overall execution latency, as in the case of `res5a_branch1`, incrementally modulating the amount of work each PE performs provides little performance improvement. However, when compute and communication latencies are more comparable, which are typically desired to achieve good mapping, the performance improvement is more pronounced, as in the case of `res2[b-c]_branch2c`.

5.2. Communication-aware data placement

The communication latency in a parallel system can have a large effect on the overall system performance, as observed in multi-core and multi-GPU characterizations.[17] Due to the limited scale of today's deep-learning accelerators, most have one unified global buffer that supplies data to all of the PEs.[6, 5, 22] However, in large-scale MCM systems where on-chip buffers are spatially distributed among the chiplets, the communication latency becomes highly sensitive to the physical location of data. Figure 11 illustrates how data placement affects communication distances and latencies. For example, if the `Src` chiplet in Figure 11(a) broadcasts data to the other chiplets, the arrival time of the data will vary greatly depending the distance of the receiving chiplets from the `Src`. Depending on the amount of computation each

Figure 10. Nonuniform work partition for ResNet-50 with speedup normalized to the best-performing tiling.

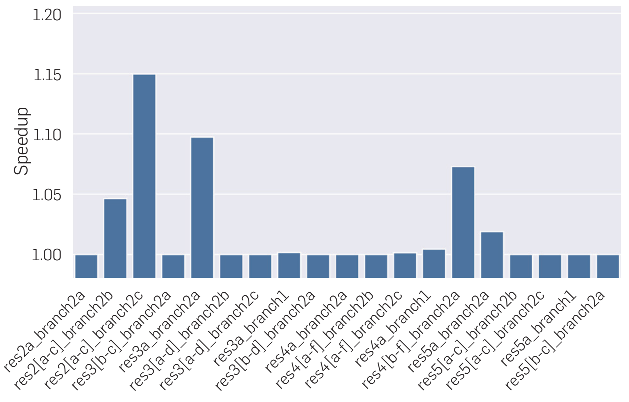
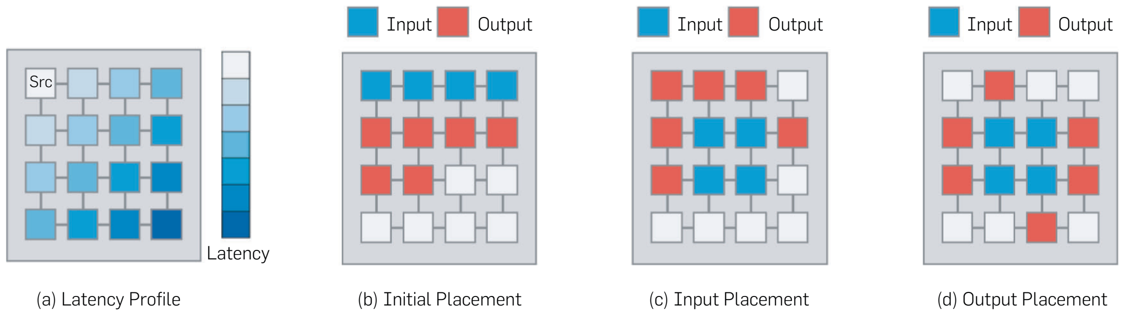


Figure 11. Data placement on the Simba system. (a) Assessment of the relative latency to different chiplets that receive data from Src. (b) Default input activation (IA) and output activation (OA) placement where data is sequentially placed from the Global PE of the first chiplet. (c) An improved IA placement at the center of the package so that data can be multicast to all chiplets. (d) OA placement with even distribution along the periphery of the package to minimize OA communication latency.



chiplet performs, such variations in communication distance could significantly limit the achievable speedup in a distributed, tile-based system like Simba, motivating the need for data placement optimizations.

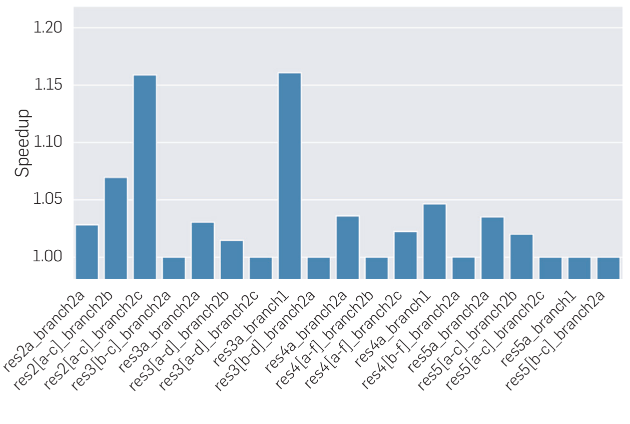
Although optimal data placement is an NP-hard problem, we use a practical greedy algorithm to iteratively determine where input and output activation data should be placed in the Simba system. The algorithm starts by performing a placement of the input activation data blocks. Once the input activations are placed, the same greedy algorithm is executed for tiles of output activations. Because a previous stage of the mapping process has already determined the data tiling, this stage need only focus on data placement and not re-tiling. Figure 11(b) shows a naive data placement with a sequential allocation of input activations to the chiplets on the top row and output activations in the next six chiplets. Figure 11(c) shows a better assignment of IAs to chiplets, selecting the four in the middle that minimize aggregate multicast hop-count to all chiplets. Finally, Figure 11(d) shows a placement of output activations on chiplets in regions where OA accumulation can occur.

Figure 12 shows the performance improvement of ResNet-50 layers with optimized data placement. Although all of the layers use 32 chiplets, many of them have different communication patterns. For example, layers like `res2[a-c]_branch2c` communicate frequently within a group of eight chiplets. In this case, it is better to group those chiplets together to minimize communication cost. In contrast, layer `res4a_branch1` must broadcast from a single chiplet to all 32 chiplets. In this case, instead of placing the source chiplet sequentially at the upper left corner, placing it at the center of the package leads to a 5% performance improvement. Data placement optimization results in up to 15% improved performance compared to the best achieved baseline.

6. CONCLUSION

This work presented Simba, a scalable MCM-based deep-learning inference accelerator architecture. Simba is a heterogeneous tile-based architecture with a hierarchical interconnect. We developed a silicon prototype system consisting of 36 chiplets that achieves up to 128 TOPS at high energy efficiency. We used the prototype to characterize the overheads of the nonuniform network of an MCM-based architecture, observing that load imbalance and communication latencies contribute to noticeable tail-latency effects. We then showed how considering the nonuniform nature of system can help improve performance through techniques such as nonuniform work partitioning, communication-aware data placement, and cross-layer pipelining. Applying these optimizations results in performance increases of up to 16% compared to naive mappings.

Figure 12. Data placement for ResNet-50 layers with speedup normalized to the best performing tiling.



Acknowledgments

The authors would like to thank Frans Sijstermans, Dan Smith, Don Templeton, Guy Peled, Jim Dobbins, Ben Boudaoud, Randall Laperriere, Borhan Moghadam, Sunil Sudhakaran, Zuhair Bokharey, Sankara Rajapandian, James Chen, John Hu, and Vighnesh Iyer for package, PCB, signal integrity, fabrication, and prototyping support. This research was, in part, funded by the U.S. government under the DARPA CRAFT program. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited). C

References

1. Arunkumar, A., Bolotin, E., Cho, B., Milic, U., Ebrahimi, E., Villa, O., Jaleel, A., Wu, C.-J., Nellans, D. MCM-GPU: Multi-chip-module GPUs for continued performance scalability. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (Toronto, ON, Canada, 2017), Association for Computing Machinery, New York, NY, USA.
2. Asanovic, K., Avizienis, R., Bachrach, J., Beamer, S., Biancolin, D., Celio, C., Cook, H., Dabbelt, D., Hauser, J., Izraelevitz, A., Karandikar, S., Keller, B., Kim, D., Koenig, J., Lee, Y., Love, E., Maas, M., Magyar, A., Mao, H., Moreto, M., Ou, A., Patterson, D.A., Richards, B., Schmidt, C., Twigg, S., Vo, H., Waterman, A. *The Rocket Chip Generator*. Technical Report, EECS Department, University of California, Berkeley, 2016.
3. Beck, N., White, S., Paraschou, M., Naffziger, S. Zeppelin: An SoC for multichip architectures. In *Proceedings of the International Solid State Circuits Conference (ISSCC)* (2018), IEEE, San Francisco, CA, USA.
4. Carloni, L.P., McMillan, K.L., Saldanha, A., Sangiovanni-Vincentelli, A.L. A methodology for correct-by-construction latency insensitive design. In *Design Automation Conference* (1999), IEEE, San Jose, CA, USA.
5. Chen, T., Du, Z., Sun, N., Wang, J., Wu, C., Chen, Y., Temam, O. DianNao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)* (Salt Lake City, Utah, USA, 2014), Association for Computing Machinery, New York, NY, USA.
6. Chen, Y.-H., Emer, J., Sze, V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (2016), IEEE, Seoul, South Korea.
7. Das, R., Eachempati, S., Mishra, A.K., Narayanan, V., Das, C.R. Design and evaluation of a hierarchical on-chip interconnect for next-generation CMPs. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)* (2009), IEEE, Raleigh, NC, USA.
8. Fojtik, M., Keller, B., Klinefelter, A., Pinckney, N., Tell, S.G., Zimmer, B., Raja, T., Zhou, K., Dally, W.J., Khailany, B. A fine-grained GALS SoC with pausible adaptive clocking in 16nm FinFET. In *International Symposium on Asynchronous Circuits and Systems (ASYNC)* (2019), IEEE, Hirosaki, Japan.
9. Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., Heil, S., Patel, P., Sapek, A., Weisz, G., Woods, L., Lanka, S., Reinhardt, S., Caulfield, A., Chung, E., Burger, D. A configurable cloud-scale DNN processor for real-time AI. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (2018), IEEE, Los Angeles, CA, USA.
10. Gao, M., Yang, X., Pu, J., Horowitz, M., Kozyrakis, C. Tangram: Optimized coarse-grained dataflow for scalable NN accelerators. In *Proceedings of the International Conference on Architectural Support for Programming Languages and Operation Systems (ASPLOS)* (2019), Association for Computing Machinery, New York, NY, USA.
11. Greenhill, D., Ho, R., Lewis, D., Schmit, H., Chan, K.H., Tong, A., Atsatt, S., How, D., McElheny, P., Duwel, K., Schulz, J., Faulkner, D., Iyer, G., Chen, G., Phoon, H.K., Lim, H.W., Koay, W.-Y., Garibay, T. A 14nm 1GHz FPGA with 2.5D transceiver integration. In *Proceedings of the International Solid State Circuits Conference (ISSCC)* (2017), IEEE, San Francisco, CA, USA.
12. He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition. In *Proceedings of the Conference on Computer Vision and Pattern Recognition (CVPR)* (2016), IEEE, Las Vegas, NV, USA.
13. Iyer, S.S. Heterogeneous integration

for performance and scaling. *IEEE Transactions on Components, Packaging and Manufacturing Technology* (2016), IEEE.
14. Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R.B., Guadarrama, S., Darrell, T. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM International Conference on Multimedia* (2014), Association for Computing Machinery, New York, NY, USA.
15. Jouppi, N.P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., luc Cantin. P., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T.V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C.R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., Yoon, D.H. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (Toronto, ON, Canada, 2017), Association for Computing Machinery, New York, NY, USA.
16. Loh, G.H., Jerger, N.E., Kannan, A., Eckert, Y. Interconnect-memory challenges for multi-chip, silicon interposer systems. In *Proceedings of the International Symposium on Memory System (MEMSYS)* (Washington DC, USA, 2015), Association for Computing Machinery, New York, NY, USA.
17. Mirhoseini, A., Pham, H., Le, Q.V., Steiner, B., Larsen, R., Zhou, Y., Kumar, N., Norouzi, M., Bengio, S., Dean, J. Device placement optimization with reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)* (2017), JMLR.org, Sydney, NSW, Australia.
18. NVIDIA. NVIDIA Tesla deep learning product performance. https://developer.nvidia.com/deep-learning-performance-training-inference, 2019.
19. Parashar, A., Raina, P., Shao, Y.S., Chen, Y.-H., Ying, V.A., Mukkara, A., Venkatesan, R., Khailany, B., Keckler, S.W., Emer, J. Timeloop: A systematic approach to DNN accelerator evaluation. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)* (2019), IEEE, Madison, WI, USA.
20. Parashar, A., Rhu, M., Mukkara, A., Puglielli, A., Venkatesan, R., Khailany, B., Emer, J., Keckler, S.W., Dally, W.J. SCNN: An accelerator for compressed-sparse convolutional neural networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (Toronto, ON, Canada, 2017), Association for Computing Machinery, New York, NY, USA.
21. Shao, Y.S., Clemons, J., Venkatesan, R., Zimmer, B., Fojtik, M., Jiang, N., Keller, B., Klinefelter, A., Pinckney, N., Raina, P., Tell, S.G., Zhang, Y., Dally, W.J., Emer, J.S., Gray, C.T., Keckler, S.W., Khailany, B. Simba: Scaling deep-learning inference with multi-chip-module-based architecture. In *Proceedings of the International Symposium on Microarchitecture (MICRO)* (Columbus, OH, USA, 2019), Association for Computing Machinery, New York, NY, USA.
22. Sijstermans, F. The NVIDIA deep learning accelerator. In *Hot Chips* (2018).
23. Venkataramani, S., Ranjan, A., Banerjee, S., Das, D., Avancha, S., Jagannathan, A., Durg, A., Nagaraj, D., Kaul, B., Dubey, P., Raghunathan, A. ScaleDeep: A scalable compute architecture for learning and evaluating deep networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA)* (Toronto, ON, Canada, 2017), Association for Computing Machinery, New York, NY, USA.
24. Wilson, J.M., Turner, W.J., Poulton, J.W., Zimmer, B., Chen, X., Kudva, S.S., Song, S., Tell, S.G., Nedovic, N., Zhao, W., Sudhakaran, S.R., Gray, C.T., Dally, W.J. A 1.17pJ/b 25Gb/s/pin ground-referenced single-ended serial link for off- and on-package communication in 16nm CMOS using a process- and temperature-adaptive voltage regulator. In *Proceedings of the International Solid State Circuits Conference (ISSCC)* (2018), IEEE, San Francisco, CA, USA.
25. Zimmer, B., Venkatesan, R., Shao, Y.S., Clemons, J., Fojtik, M., Jiang, N., Keller, B., Klinefelter, A., Pinckney, N., Raina, P., Tell, S.G., Zhang, Y., Dally, W.J., Emer J.S., Gray, C.T., Keckler, S.W., Khailany, B. A 0.11 pJ/Op, 0.32–128 TOPS, scalable multi-chip-module-based deep neural network accelerator with ground-reference signaling in 16nm. In *Proceedings of the International Symposia on VLSI Technology and Circuits (VLSI)* (2019), IEEE, Kyoto, Japan, Japan.

Yakun Sophia Shao* ([ysshao]@berkeley.edu), UC Berkeley, CA, USA.

Jason Cemons, Nathaniel Pinckney, Brucek Khailany, and Stephen W. Keckler ([jclemons, npinckney, bkhailany, skeckler]@nvidia.com), NVIDIA, Austin, TX, USA.

Rangharajan Venkatesan, Brian Zimmer, Ben Keller, and Yanqing Zhang ([rangharajanv, bzimmer, benk, yanqingz]@nvidia.com), NVIDIA, Santa Clara, CA, USA.

Matthew Fojtik, Alicia Klinefelter, Stephen G. Tell, and Tom Gray ([mfojtik, aklinefelter, stell, tgray]@nvidia.com), NVIDIA, Durham, NC, USA.

Nan Jiang ([tedj]@nvidia.com), NVIDIA, Westford, MA, USA.

Priyanka Raina ([praina]@stanford.edu), Stanford University, Stanford, CA, USA.

William J. Dally ([bdally]@nvidia.com), NVIDIA, Incline Village, NV, USA/Stanford University, Stanford, CA, USA.

Joel Emer ([jemer]@nvidia.com), Massachusetts Institute of Technology, Cambridge, MA, USA/NVIDIA, Westford, TX, USA.

*Work done at NVIDIA