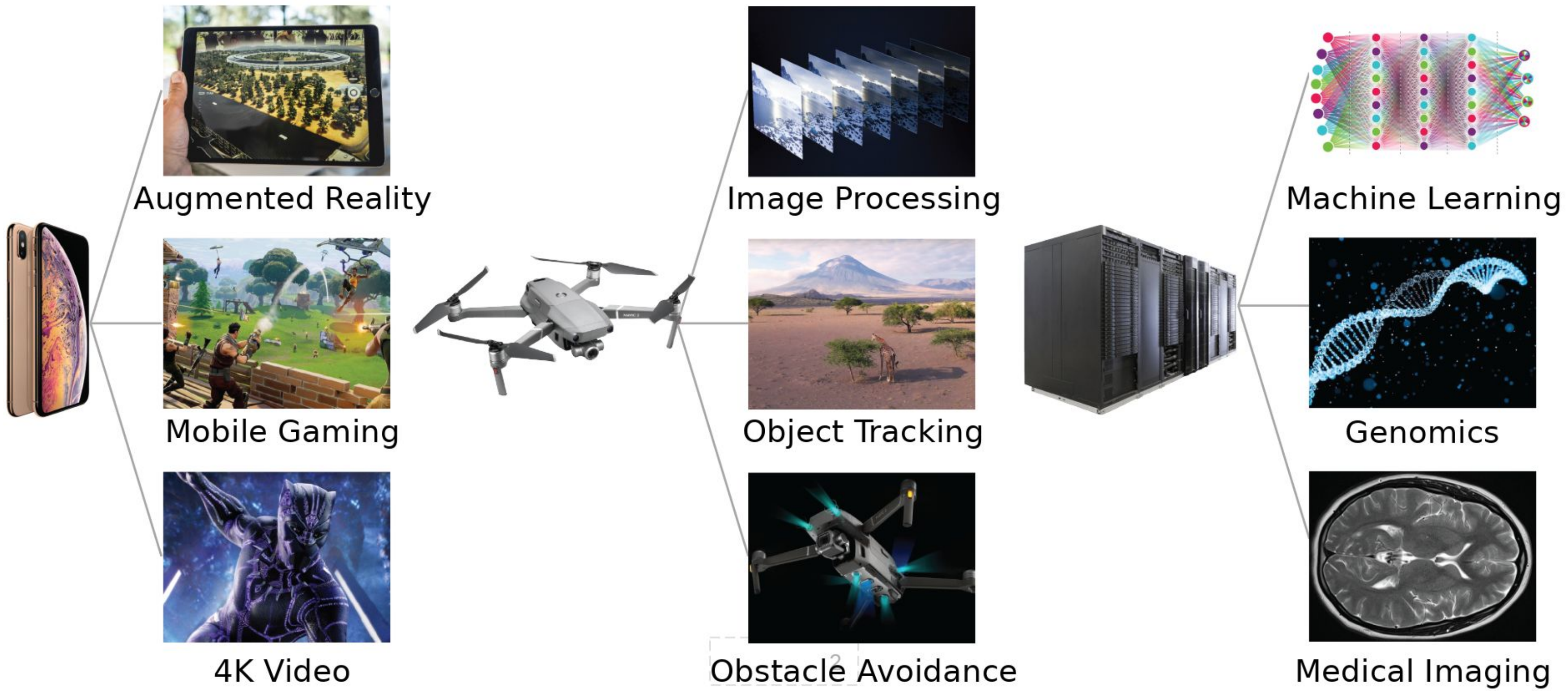
Simba: Scaling Deep Learning Inference with MCM Architecture

Presentation By: Stephen More, Braam Beresford, Faaiq Waqar

Y. S. Shao, J. Clemons, R. Venkatesan, B. Zimmer, M. Fojtik, N. Jiang, B. Keller, A. Klinefelter, N. Pinckney, P. Raina, S. G. Tell, Y. Zhang, W. J. Dally, J. Emer, C. T. Gray, B. Khailany, and S. W. Keckler, “Simba,” *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019.

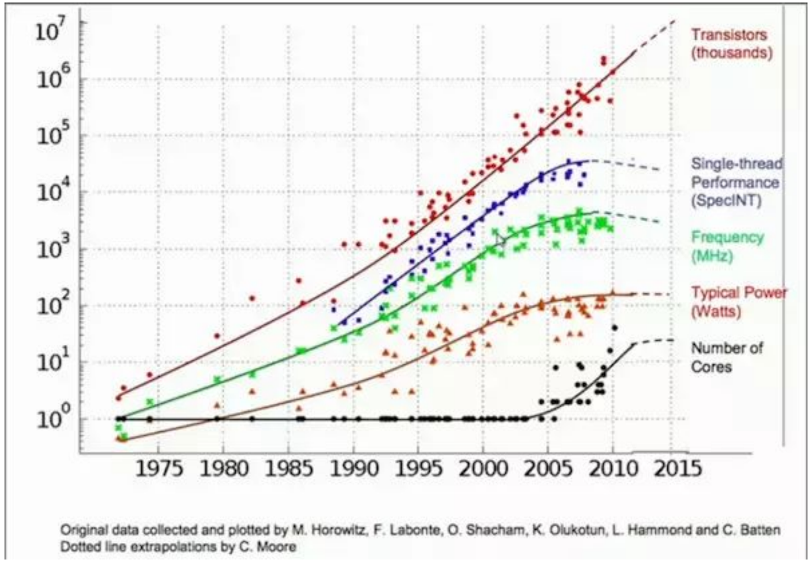
Background and Motivation

Increasing Demand for Computing



Motivation: Goodbye Moore, Hello Powerwall

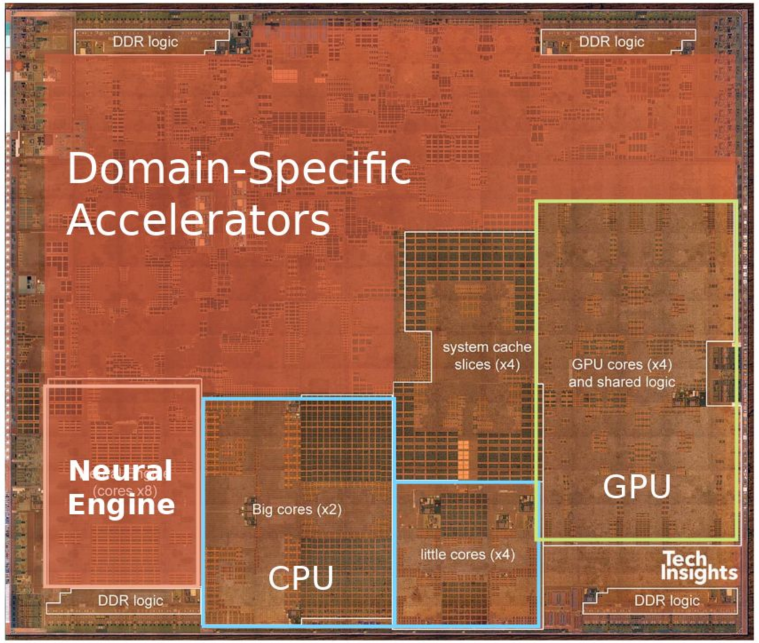
- The size of transistors is tapering off
- The cost of advanced nodes is very high
- Chips are already pushed to their thermal limits in terms of clock speed

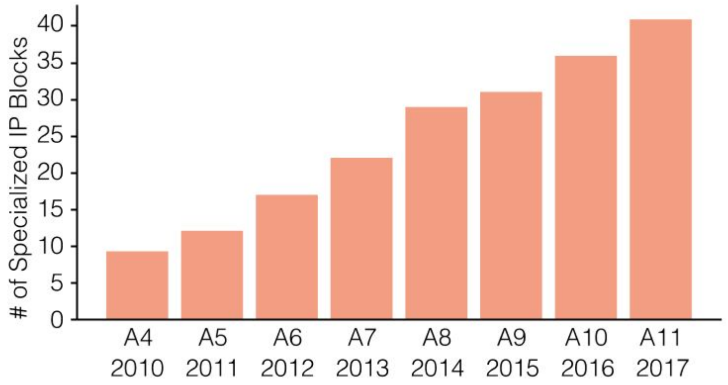


Source: https://en.wikipedia.org/wiki/Deep_learning

How did we address this challenge in the past? Build outward!

Bigger Die; Domain Specific Accelerators

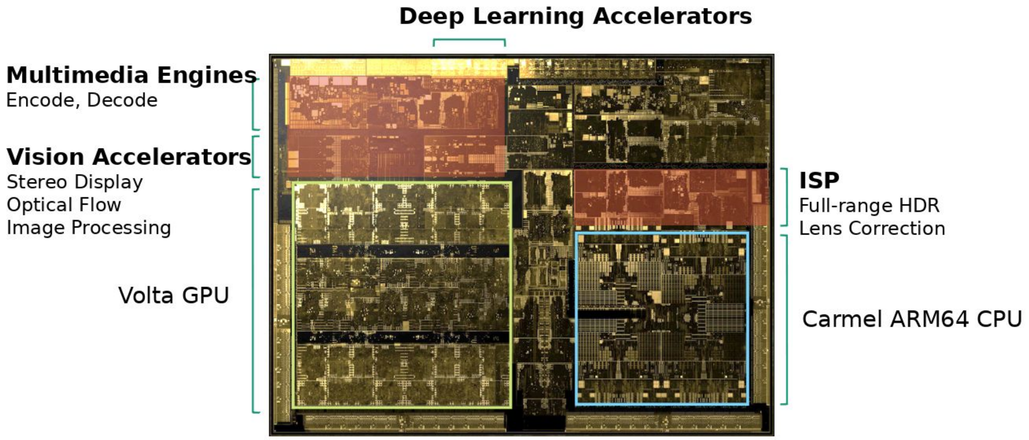




https://www.techinsights.com/blog/apple-iphone-xs-max-teardown

Problem Presented

- Increasing demand for computational resources
- Slowing Moore's law means we can not longer keep packing more logic into same area
- Increasing die size effective at introducing more domain specific logic but:
    - Lower yields
    - Design and implementation are longer and more expensive
    - Slower time to market, important to keep up with business cycles

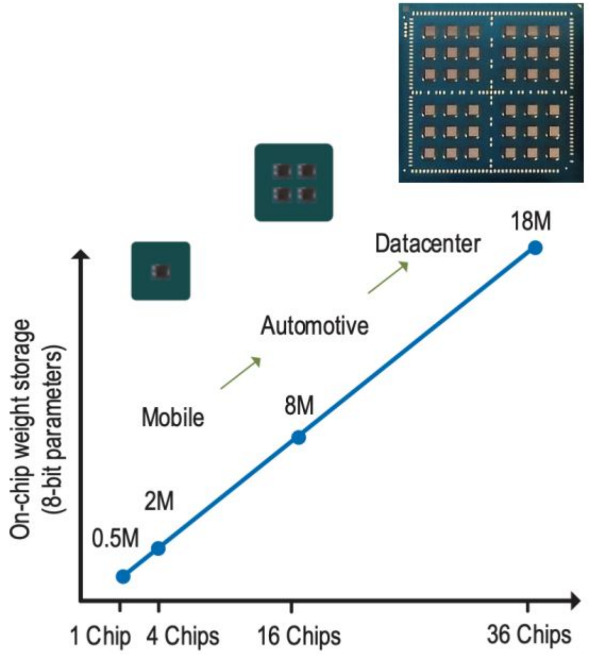


~8,000 Engineering Years

Source:
NVIDIA Xavier Announcement, CES'2018

Solution: Multi-Chip-Module Approach

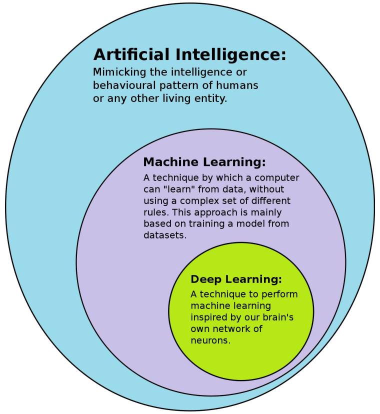
- MCM involves systems out of small building blocks known as chiplets
- Advantages:
  - Smaller chips are cheaper to design
  - Smaller chips have higher yield
  - Faster time-to-market
- Obstacles:
  - Power, latency, inter-chip communication
  - Load balancing
- This method has been used for CPU and GPU design already

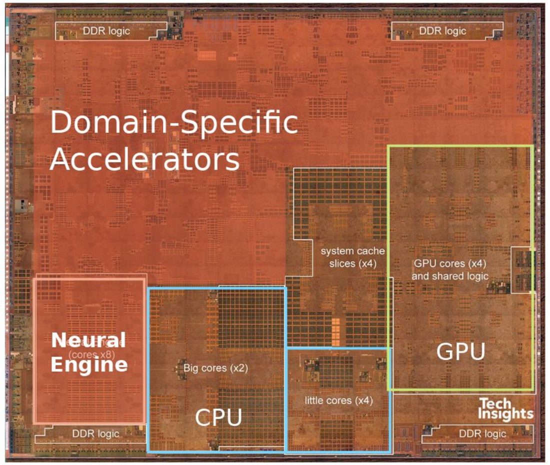


Ref: Zimmer et al., VLSI 2019

Deep Learning Takes a Front Seat

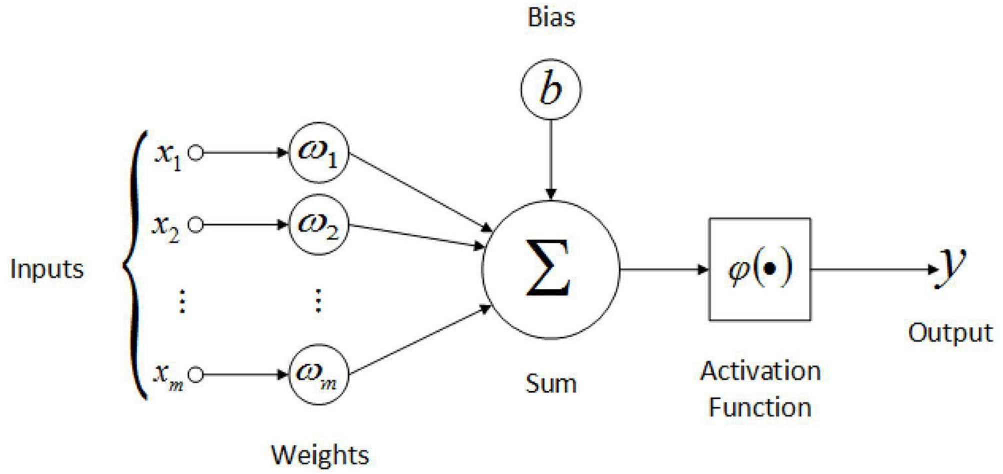
- A type of machine learning
  - Machine learning: computer learns from data
  - Applies this new knowledge to unseen data
- Deep learning architecturally mimics organic brains with neurons
- The driver of many AI improvements in recent years
- Very powerful tool for computer vision, decision making, and more
- **Computationally intensive**


Artificial Intelligence:
Mimicking the intelligence or behavioural pattern of humans or any other living entity.
Machine Learning:
A technique by which a computer can "learn" from data, without using a complex set of different rules. This approach is mainly based on training a model from datasets.
Deep Learning:
A technique to perform machine learning inspired by our brain's own network of neurons.



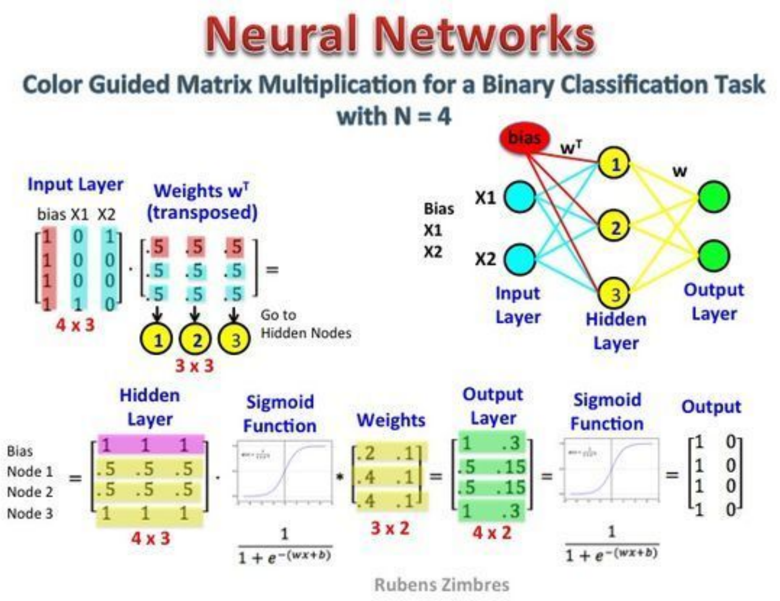
Source: https://en.wikipedia.org/wiki/Deep_learning

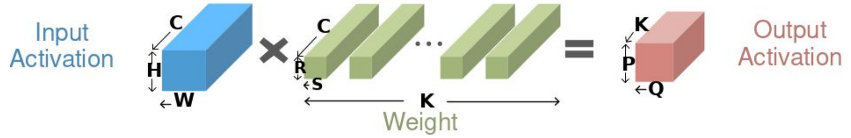
Deep Neural Networks

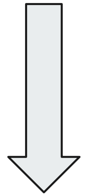


https://www.pnas.org/content/116/4/1074

Deep Neural Networks cont.



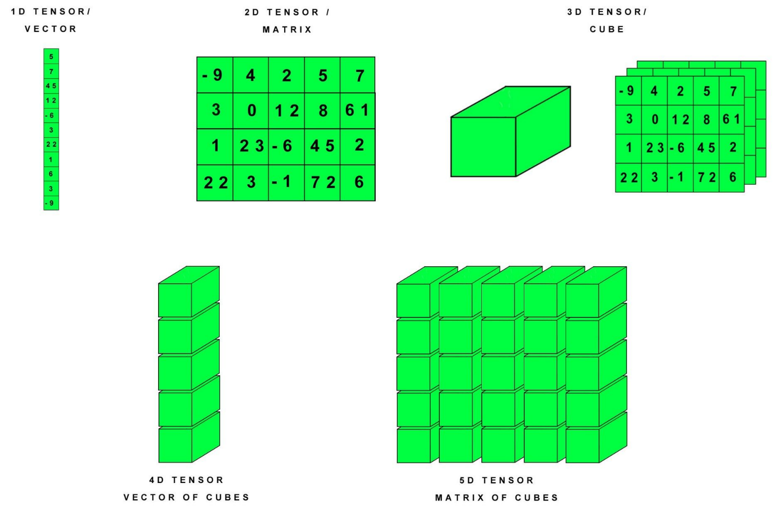




Screams parallelization

Source: https://www.datasciencecentral.com/profiles/blogs/matrix-multiplication-in-neural-networks

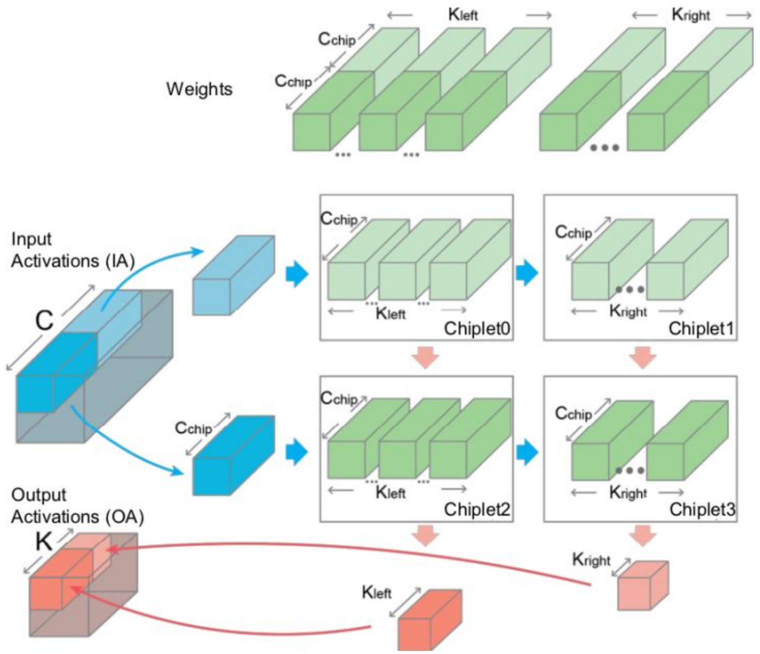
Deep Neural Networks cont



Screams parallelization

https://www.datacamp.com/community/tutorials/investigating-tensors-pytorch

Applying Neural Nets To MCM



Large MCMs Means Large Latency

- Smaller neural nets easily applied uniformly to chiplet design
- Inference performance scales with larger systems
    - Execution time decreases and latency-related effects become more important
- Assumptions of uniform latency and bandwidth in selecting DNN tiling can degrade performance and energy efficiency

This Paper’s Contribution

- The first to quantitatively highlight the challenge of mapping DNN layers to non-uniform MCM-based DNN accelerators
- Proposes communication-aware tiling strategies to address the challenge

Introducing the Simba Architecture

Simba Hierarchical Architecture

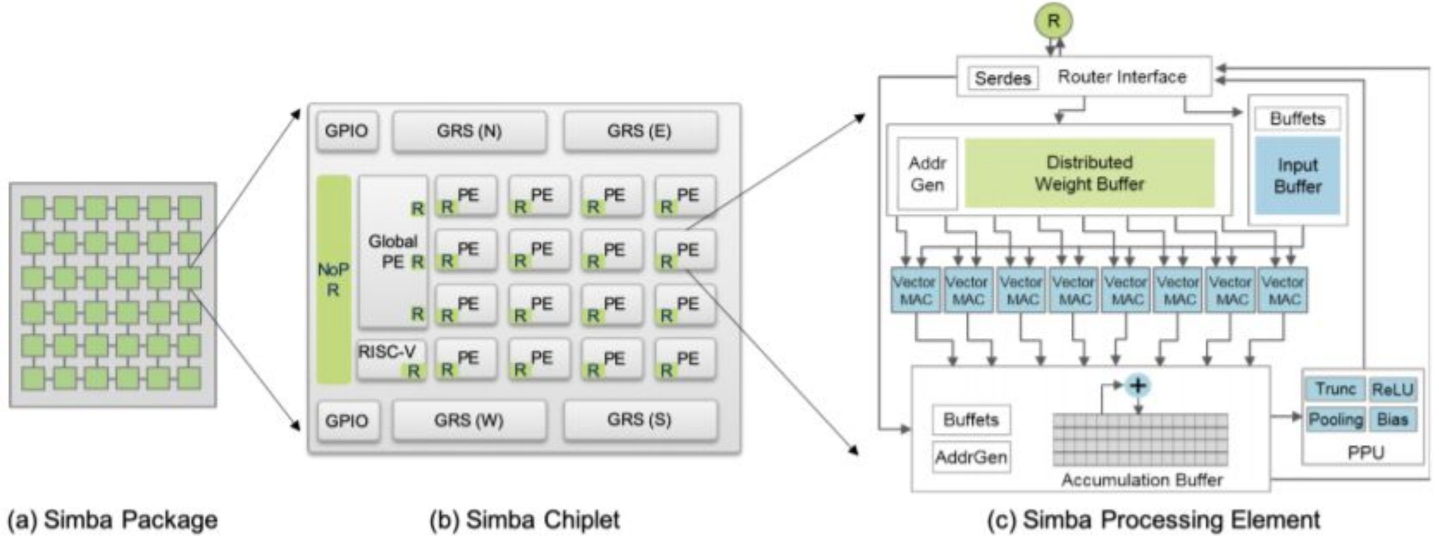
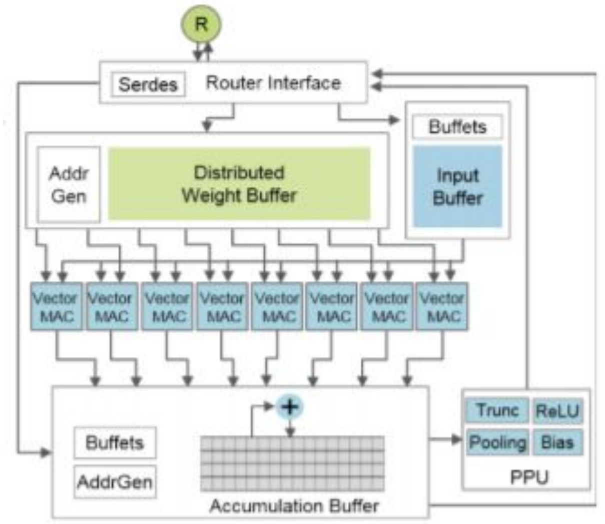


Figure 2: Simba architecture from package to processing element (PE).

Processing Element (PE)

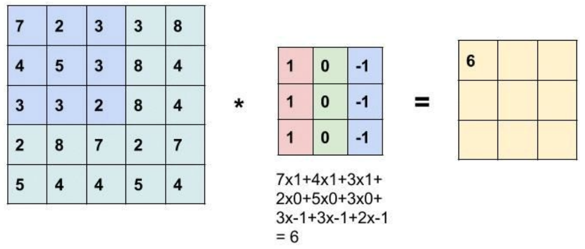
- The lowest building block of simba is the Processing Element
  - The heart of operations in each processing element is an array of vector multiply-accumulate units (MACs)
- MACs allows the intense matrix multiplication and add operations to be computed with a high degree of parallelism
- PEs can compute partial sums as well as activation values



Common Problems in DNN processors

- Loading weights during processing
  - Many of the weights in a layer will be reused across operations, so it will be inefficient to re-read data, especially from slower memory
- Frequent changes in DNN architectures will require hardware to be flexible to support next generation of networks
  - Flexibility in dataflow is required

Why might we reuse so many weights?



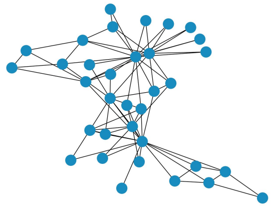
Sources:
https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/

Solutions used by Simba Processing Elements

- Simba PEs use weight-stationary dataflow
  - Weights stay in MAC registers and are reused across iterations
- To account for rapid change in topology of DNNs, Simba PEs are able to be produce or consume partial sums, and can be configured as inputs or outputs
  - This enables PEs to be used for a wide variety of DNN operations.

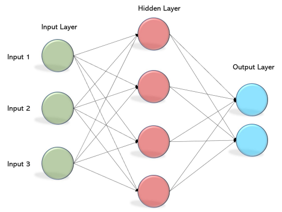
Examples of variable topologies in modern DNNs:

GNN topology:



vs

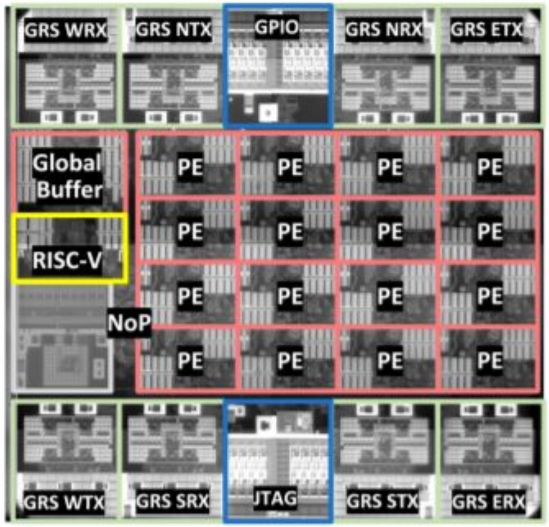
MLP topology:



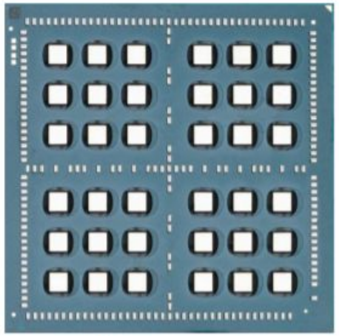
Sources:
https://pennylane.ai/qml/demos/qgrnn.html
https://becominghuman.ai/multi-layer-perceptron-mlp-models-on-real-world-banking-data-f6dd3d7e998f

Simba Chiplet

- Simba Chiplets are made up of many "latency insensitive" blocks such as PEs, control, networking, and transmit/receive building blocks.
- Latency insensitivity is important when activation inputs/outputs can be cast to PEs across chiplet boundaries



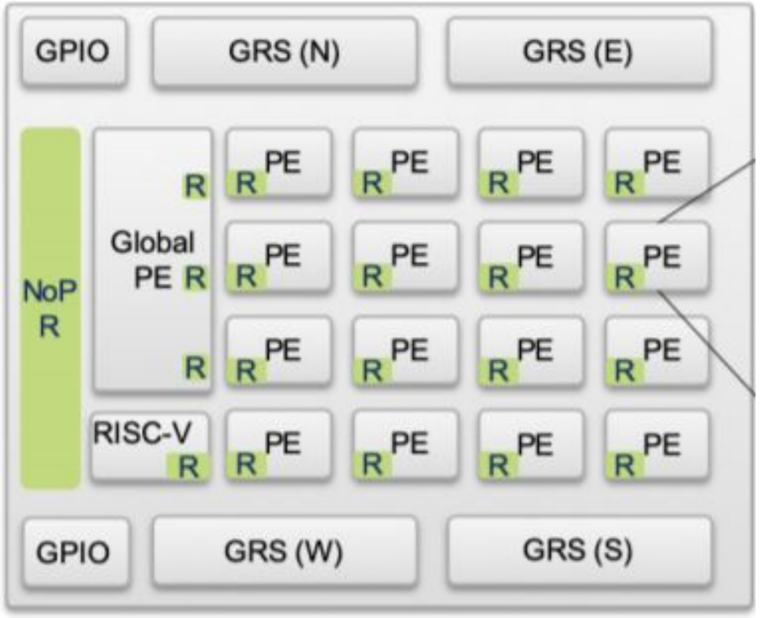
(a) Simba chiplet


(b) Simba package

Simba Global PE

- Serves as a second level storage medium for input/output activation data, routed to PEs (possibly across chiplets)

- For DNN operations which have limited data reuse, global PE can perform such operations to reduce communication overhead



Simba Controller

- To manage various states of PEs and global PE in chiplet, a RISC-V processor is used.
  - Execution is triggered by RISC-V core.
  - After execution PEs and Global PE will send "done" notifications via interrupts

- Memory mapped registers control synchronization of all RISC-V cores across the package

RISC-V®

RISC-V: The Free and Open RISC Instruction Set Architecture

Source: https://riscv.org/

Simba Interconnect:

- To handle connections between PEs, Global PEs, Controller, within the same chiplet and across chiplets a network on chip (NoC) and network on package (NoP) are used, continuing the theme of hierarchical design

- To achieve flexible dataflow, there are a wide range of communication options across levels of processing hierarchy

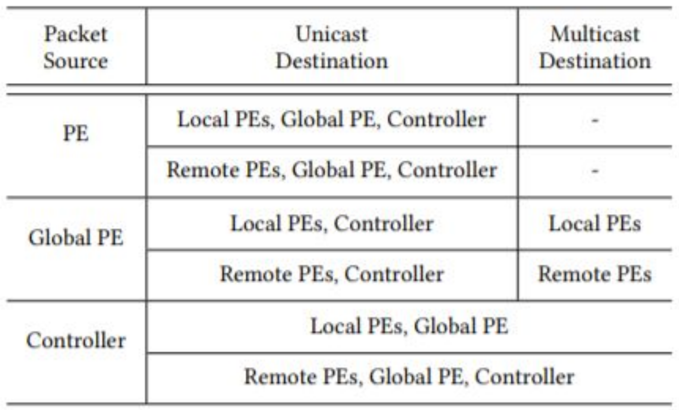


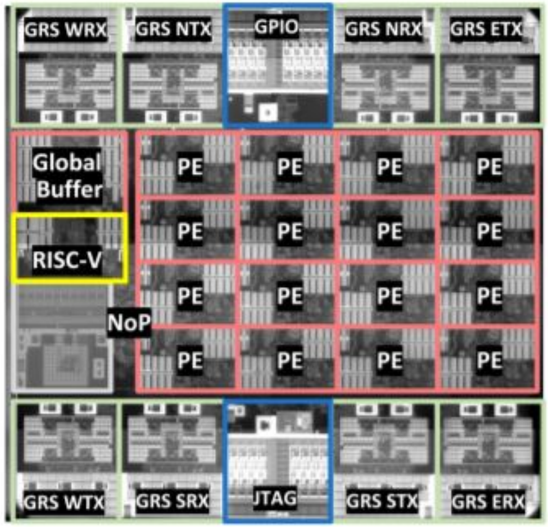
Table 1: Simba system communication capability.

Packet Source	Unicast Destination	Multicast Destination
PE	Local PEs, Global PE, Controller	-
	Remote PEs, Global PE, Controller	-
Global PE	Local PEs, Controller	Local PEs
	Remote PEs, Controller	Remote PEs
Controller	Local PEs, Global PE	
	Remote PEs, Global PE, Controller	

Getting Data to/from Simba Chiplets

- Each chiplet has 4 transmitters and 4 receivers, placed in strategic locations on the chiplet as to accomodate for intra-package communication

- Chiplets use globally asynchronous, locally synchronous clocking (GALS), meaning PEs, Global PEs, RISC-V cores, and NoP routers can all have independant clock frequency

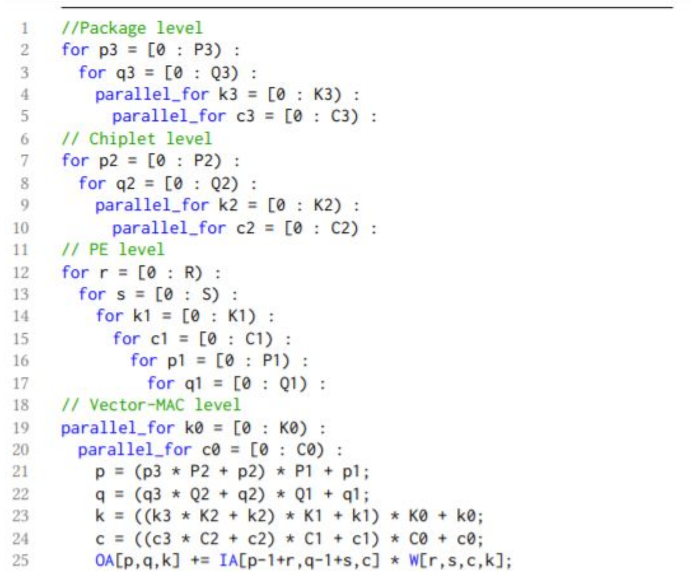


(a) Simba chiplet

Mapping DNN to Simba

- Listing 2 shows a default tiling for mapping computation to multiple layers
  - Users can configure the bounds of loops to allow for flexible mapping.

- Authors also developed tool, using Caffe, which allows users to map DNN inference to Simba, while maintaining efficient data reuse.



```
//Package level
for p3 = [0 : P3) :
  for q3 = [0 : Q3) :
    parallel_for k3 = [0 : K3) :
      parallel_for c3 = [0 : C3) :
// 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 2: Simba baseline dataflow.

Performance of Simba

Performance Characterization Setup

- To see how the architecture performs the authors used the following for testing:
    - Input DNN: ResNet-50
        - ResNet is a very commonly used image recognition network with RNN and CNN elements which demonstrates Simbas performance across a diverse network
    - Variables:
        - Layer of network
        - Mapping of network to Simba
        - Number of PEs used
        - Distance from Chiplet to Chiplet
    - Measured Outputs:
        - Energy
        - Latency
        - Throughput

Mapping Sensitivity

- The figure demonstrates how mapping of PEs across chiplets can have a large effect on latency.
- This large latency is due to synchronization latency and inter chiplet communication latency
- Latency flattens out on one chiplet due to Global PE SRAM contention
- Latency of multi chiplet increases linearly as communication overhead increases

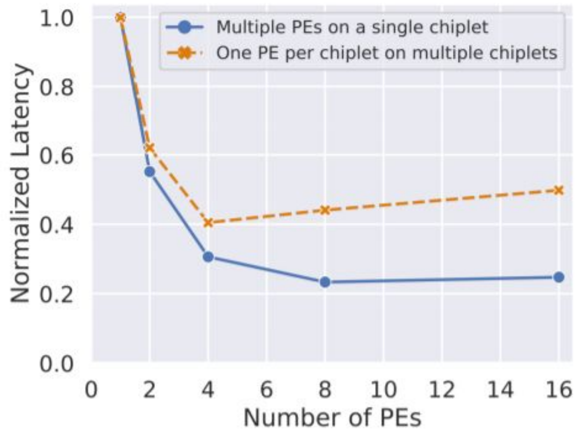


Figure 6: Performance comparison of on-chip and on-package communication and synchronization in Simba. Latency is normalized to single-PE execution latency.

Differences in Computation across Layers

- Certain layers of resnet are more ripe for parallelism than others
  - res2[a-c]_branch2b is early layer, has less weights, so as chip count increases from 8 to 32 communication overhead overwhelms parallelism - 17 layers in ResNet-50
  - res3a_branch1 flattens after 8 chiplets, higher parallelism than res2, not enough to overcome communication overhead - 12 layers in ResNet-50
  - res5[a-c] has most abundant parallelism -24 layers in ResNet-50

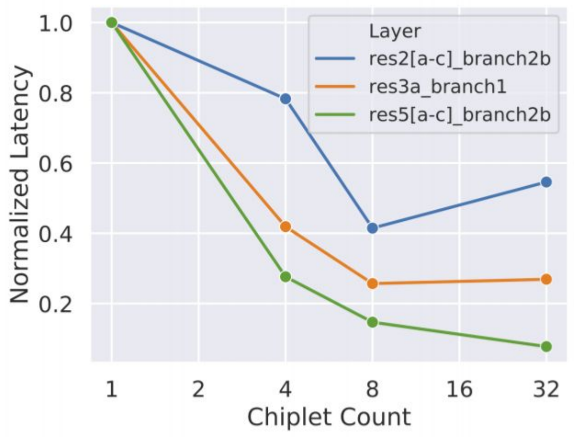
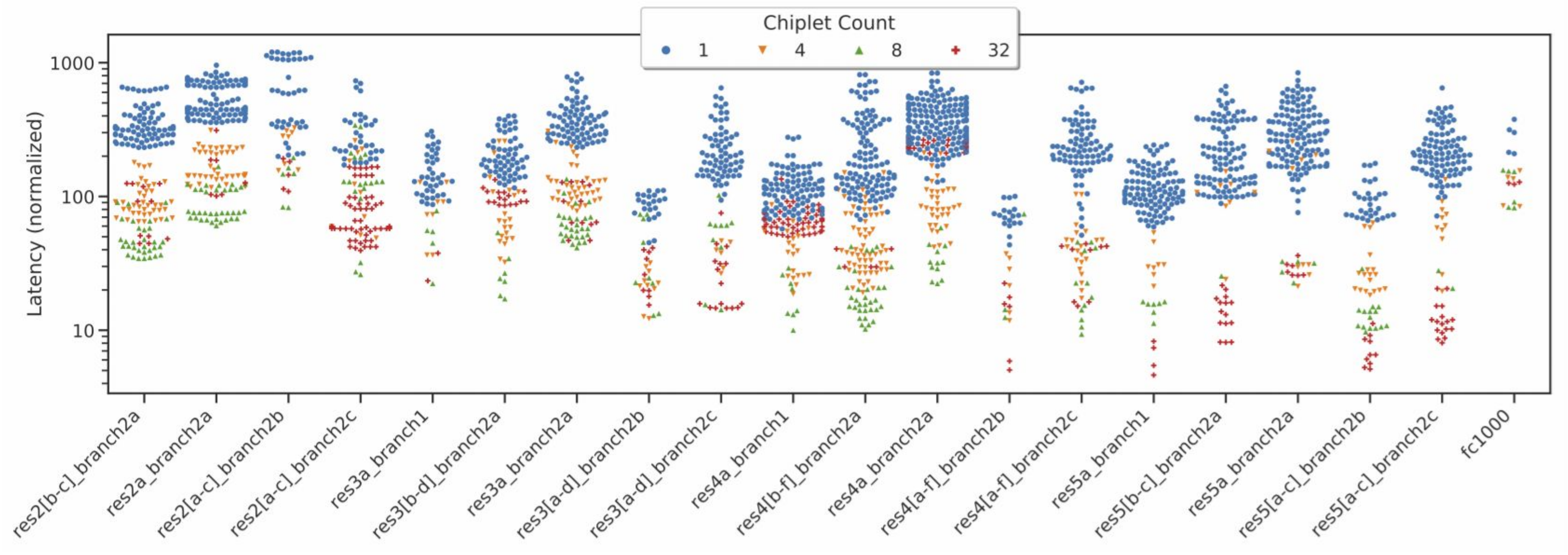


Figure 7: Simba scalability across different layers from ResNet-50. Latency is normalized to the latency of the best-performing tiling with one chiplet.

Layer Sensitivity



(a)

NoP Latency Sensitivity

- From 2 hops on package to 12 hops
- Execution time increases to 2.5x on comparing closest to farthest chip configs
- Latency is much more important on large scale systems!

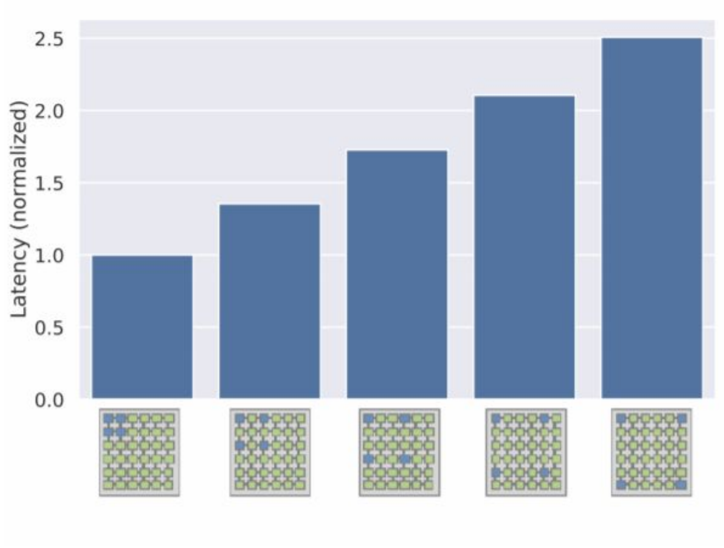


Figure 9: Simba scalability with different chiplet-to-chiplet communication latencies running res4a_branch1 with four chiplets (using the same tiling). Different bars represent different selections of the active four chiplets, as shown under the X-axis; the active chiplets are highlighted in blue.

Comparing Simba against GPU hardware

- 1.8x and 1.9x speedup compared to V100 and T4
- 5.4x and 2.9x better energy efficiency vs V100 and T4

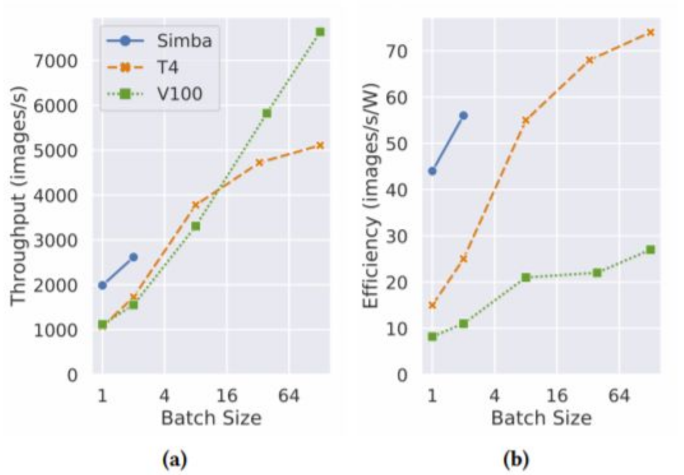


Figure 11: Throughput and efficiency of Simba, V100, and T4 running ResNet-50 with different batch sizes.

Tiling Strategies for Scalability

Tiling for More Scalable Systems

- With the goal of scalability in mind, we must overcome a few challenges …
  - Chiplets do not complete synchronously
  - Communication overhead isn't uniform

- How can we approach such challenges …
  - Distribute loads that improve synchronization
  - Map communication with locality in mind

Slowest Stage

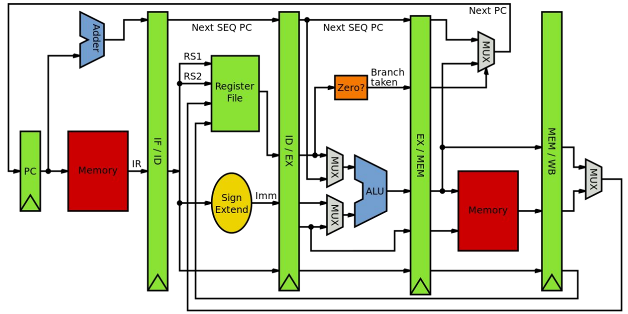


Figure: MIPS Pipeline*

Slowest Chip
(Due to Instr.)

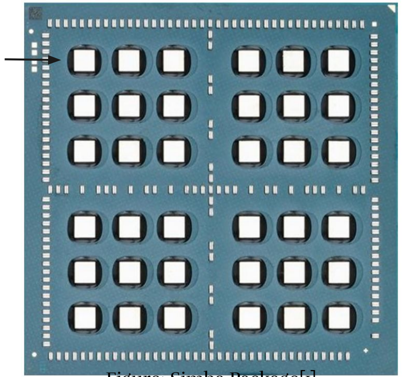


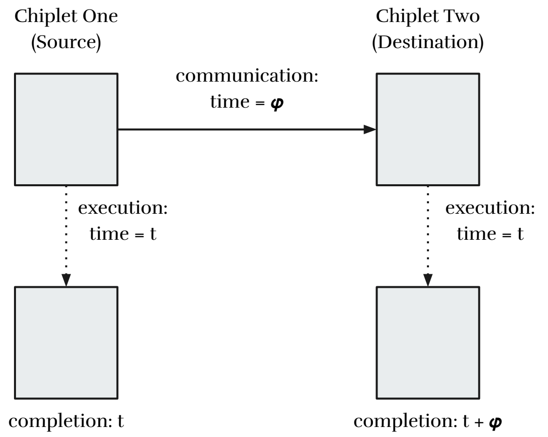
Figure: Simba Package[1]

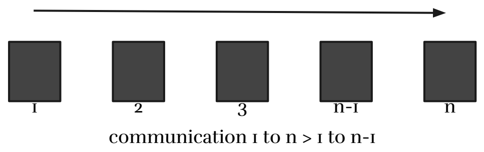
*https://commons.wikimedia.org/wiki/File:MIPS_Architecture_(Pipelined).svg

Non-Uniform Work Partitioning : Intro

- Parallel processing → Better Performance!
- How do we effectively balance processes?

- Some State-of-Art systems, like ScaleDeep (Venkataramani et al.) distribute loads evenly
  - However, this doesn't work well as scale increases
  - Communication latencies increase as a function of size





Non-Uniform Work Partitioning : Proposition

What if we partitioned load non-uniformly, with communication delays in mind?

Non-Uniform Work Partitioning : Visualization

- Input activations (IA) start in chiplet 0 & 2
- Weights (W) distributed across all 4 chiplets
- Loads balanced based on communication!

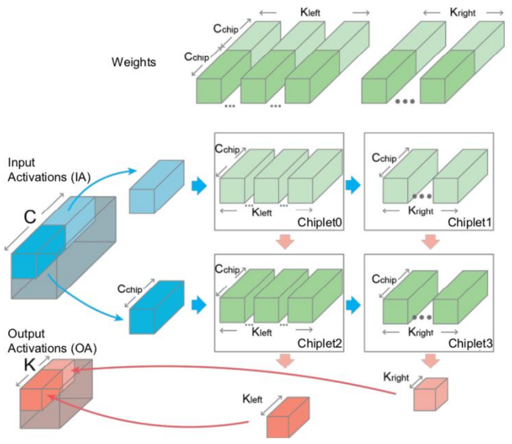
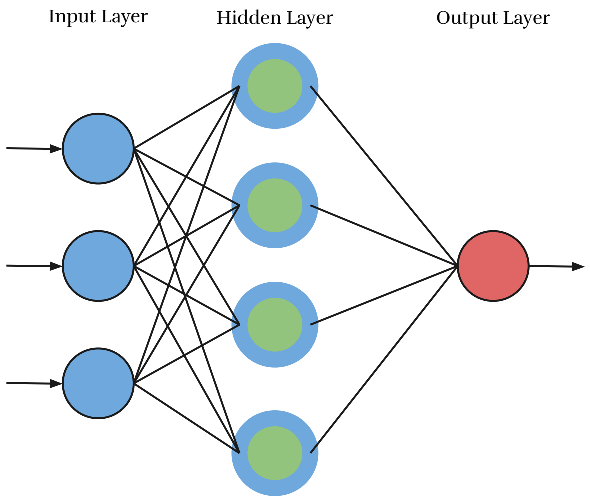


Figure: Illustration of Non-Uniform work partitioning[1]

Multi-Layer Perceptron Visualizer



Non-Uniform Work Partitioning : Algorithm

- Communication Latency is pronounced
- Performance Counters in PEs for quantification of tail latency
    - Track initial execution performance
    - Dynamically adjust the workload in the following layers!

Non-Uniform Work Partitioning : Results

- Obtaining a maximum 15% speedup [compared to uniform tiling]

- Layers with compute latency >> or << communication latency → less improvement
- Layers with closer latencies → more pronounced speedup

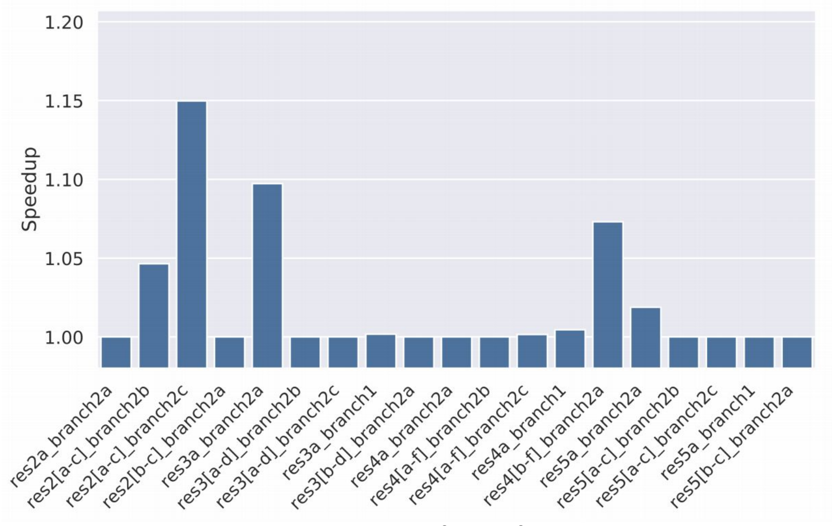


Figure: Non-Uniform Performance Comparison for ResNet-50[1]

Communication-Aware Data Placement : Intro

- Data locality also impacts communication latency
- Large MCM systems have on chip buffers spatially distributed among the chiplets
  - This is unlike other accelerators with a global buffer

- Communication latency → n-hop > n-1-hop

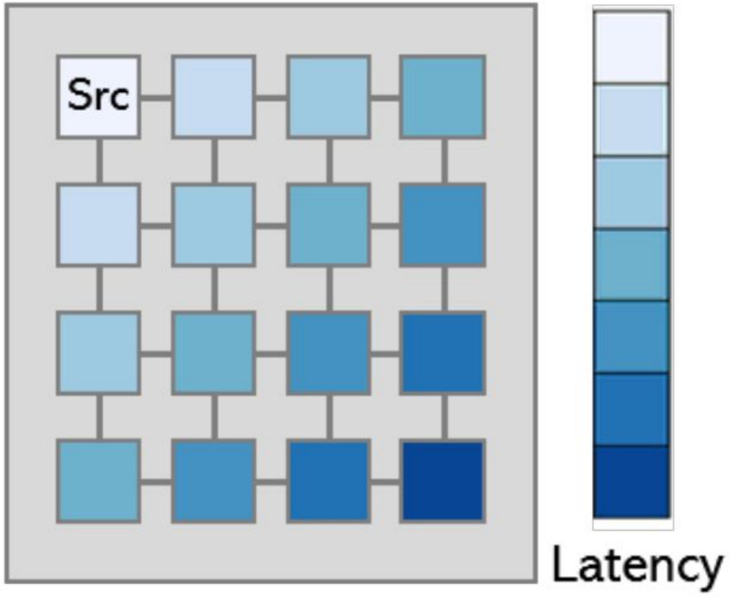
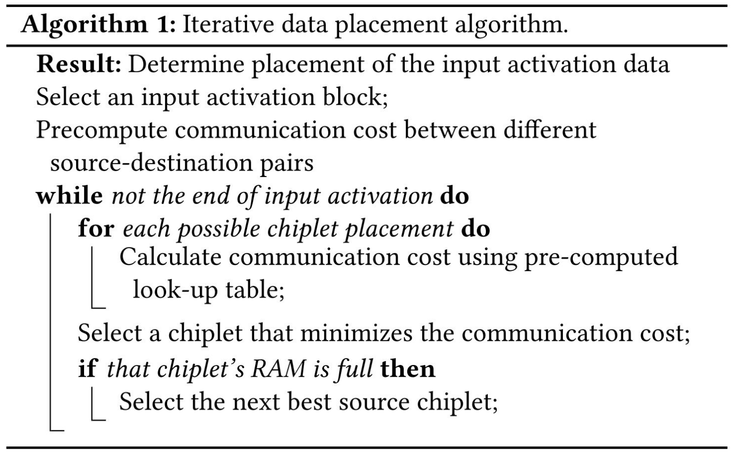


Figure: Latency Profile[1]

Communication-Aware Data Placement : Proposition

- Finding the best optimization is NP-Hard …
    - But we can find something solid using a greedy algo!

- Start with placement of input activations
- Then execute for output activations placement
    - Mapping these outputs helps with next stage!



Algorithm 1: Iterative data placement algorithm.

```
Result: Determine placement of the input activation data
Select an input activation block;
Precompute communication cost between different
  source-destination pairs
while not the end of input activation do
    for each possible chiplet placement do
        Calculate communication cost using pre-computed
          look-up table;
    Select a chiplet that minimizes the communication cost;
    if that chiplet's RAM is full then
        Select the next best source chiplet;
```

Figure: Data Placement Algorithm[1]

Communication-Aware Data Placement : Visualized

```
Algorithm 1: Iterative data placement algorithm.
Result: Determine placement of the input activation data
Select an input activation block;
Precompute communication cost between different
  source-destination pairs
while not the end of input activation do
    for each possible chiplet placement do
        Calculate communication cost using pre-computed
          look-up table;
    Select a chiplet that minimizes the communication cost;
    if that chiplet's RAM is full then
        Select the next best source chiplet;
```

Figure: Data Placement Algorithm[1]

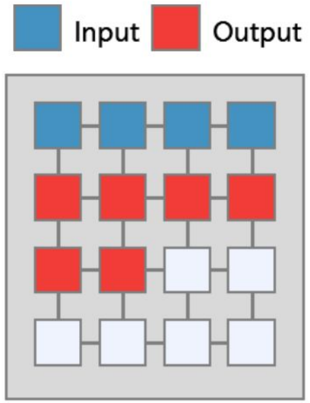


Figure: Initial Placement[1]

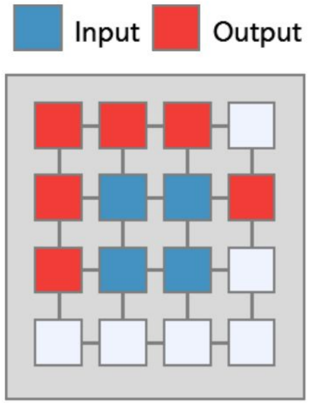


Figure: Input Placement[1]

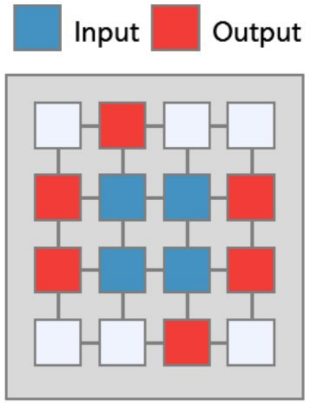


Figure: Output Placement[1]

Communication-Aware Data Placement : Results

- Obtaining a maximum 16% speedup [compared to uniform tiling]

- Locality of inter-chiplet communication in a layer leads to higher boost, chiplets broadcasting had smaller boost

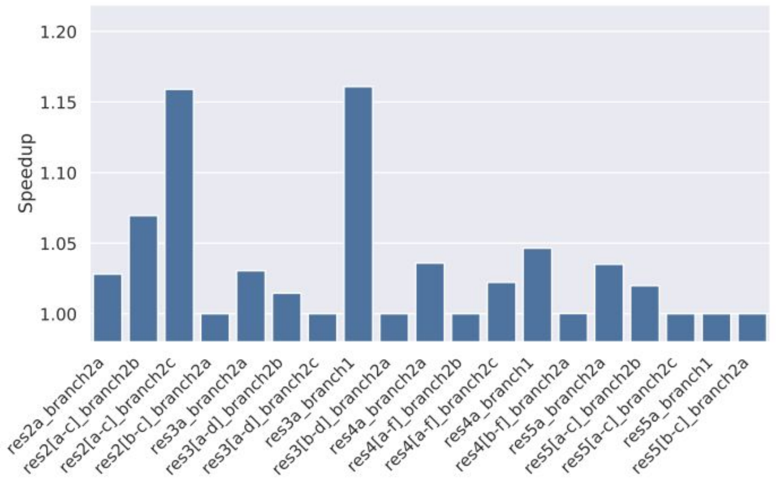


Figure: Communication-Aware Comparison for ResNet-50[1]

Cross-Layer Pipelining : Intro

- Exploiting Leveled parallelism can increase throughput, such as ILP
- How can we implement pipelining a way that is flexible and effective?

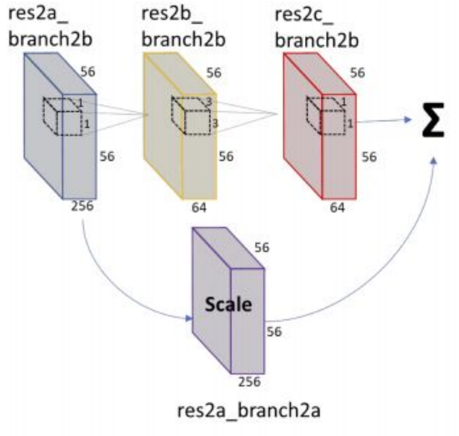
- Considerations
  - Think about how we may exploit stages in a Deep Neural Network ...
  - Think about we may turn chiplets into a pipeline ...

Cross Layer Pipelining : Proposition

What if we partitioned the package into clusters and pipelined DNN Layers as stages?

Cross Layer Pipelining : By Example & Result

- Pipelining on Simba improves throughput by 2.3X



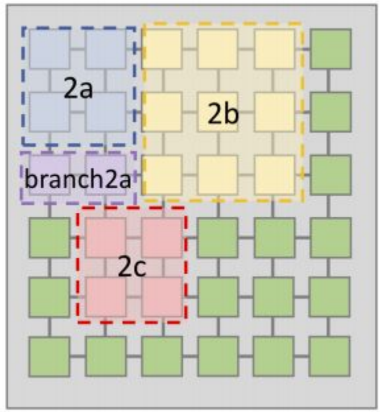
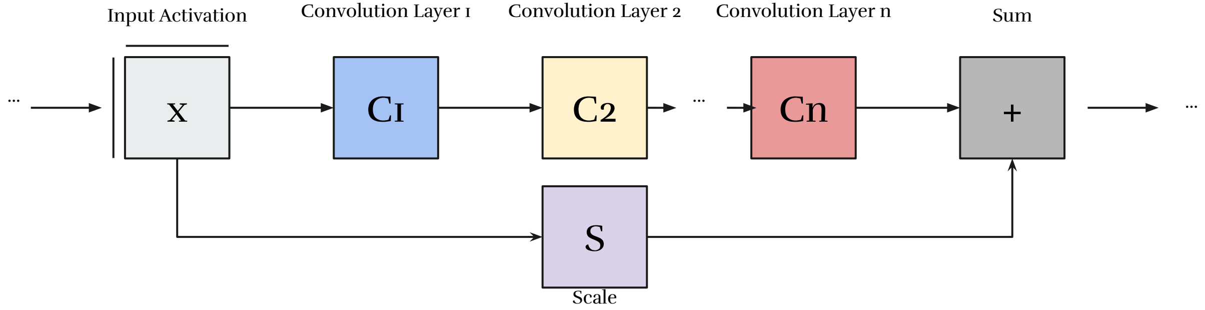


Figure: Pipelining of ResNet-50 Residual Block[1]



Summary/Conclusion

- Most DNN inference applications run on CPUs, GPGPUs, or DNN Inference Accelerators
    - General purpose hardware isn't optimal for DNN performance
    - Extended ISAs & Accelerators on efficiency, not many discuss scalability

- Simba is the first to explore use of MCM Accelerators for DNN Inference
    - Opens the door for exploration of scaling challenges and opportunities
    - Capable of high energy efficiency per operation
    - Considers non-uniform nature of system to propose tiling strategies

Thanks for Listening!