# EITF20: Computer Architecture

## Application-Specific-Instruction-Set Processors

Steffen Malkowsky
steffen.malkowsky@eit.lth.se

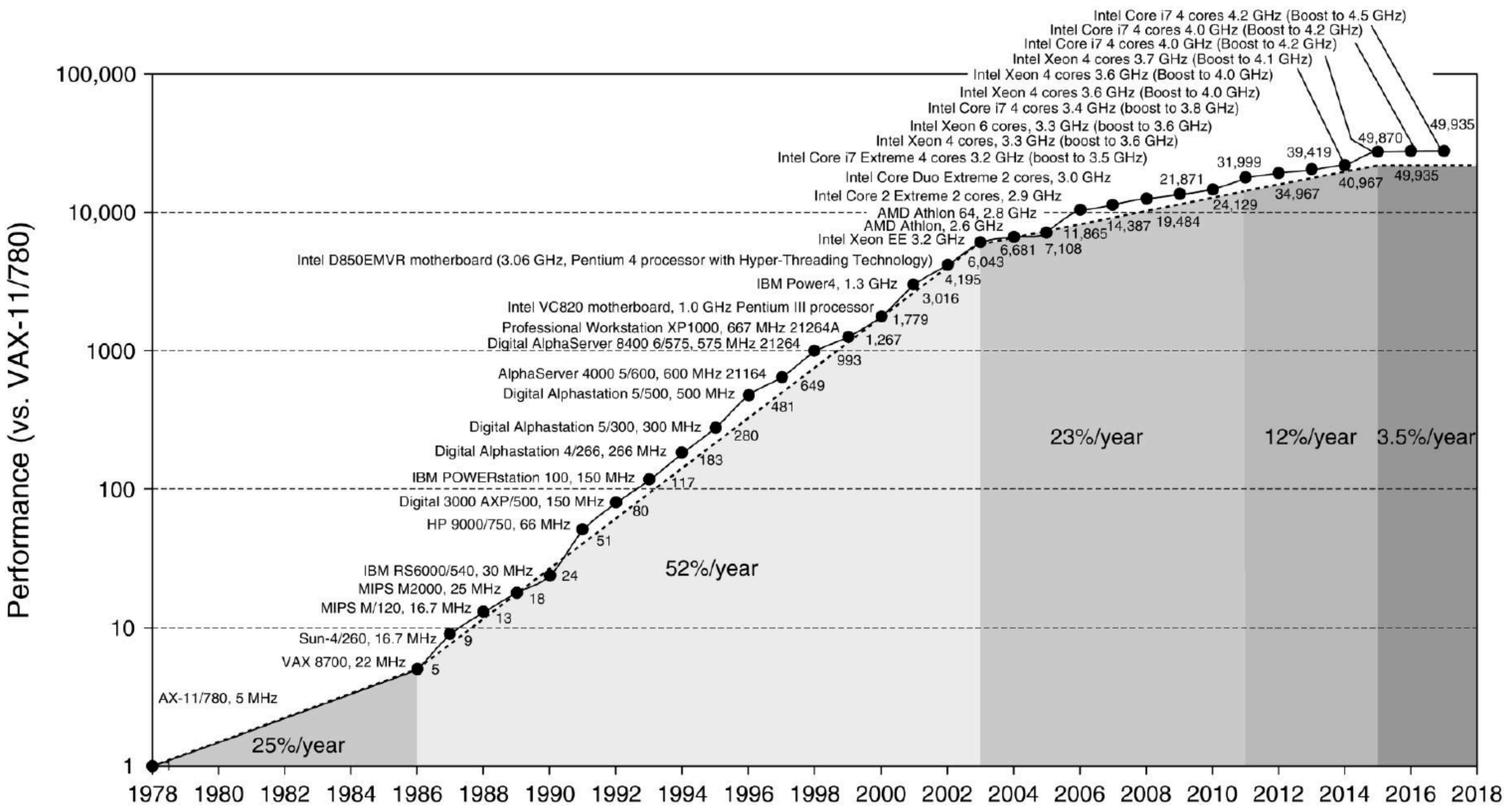
## Outline

- Part I: Performance Limits of GPPs
- Part II: Opportunities of ASIP
- Part III: Case Study
    - Google TPU
- Part IV: Some Remarks

## Outline

- Part I: Performance Limits of GPPs
- Part II: Opportunities of ASIP
- Part III: Case Study
    - Google TPU
- Part IV: Some Remarks

# Moore's Law



- The number of transistors doubles approximately every two years

## Moore's law enabled

- Deep Memory hierarchies
  - Up to four level of caches
- Wide SIMD units
  - Used in GPPs to accelerate vector processing
- Deep Pipelines
  - Higher Execution speed
- Branch Prediction
  - Reduce the number of pipeline flushes
- Out-of-Order Execution
- Speculative Pre-fetching
- Multithreading
- Multicore Processing

## Dennard Scaling I

- Dennard observed that as technology scales, the power consumption can remain the same.
- Example:

Transistor dimension scales by 30%

→ area reduces by 50%

Circuit delay scales by 30%

→ frequency increases approx. 40%

Voltage is reduced by 30%

→ power reduces by 50%

**Lennard Scaling failed around 2005**

## Dennard Scaling II

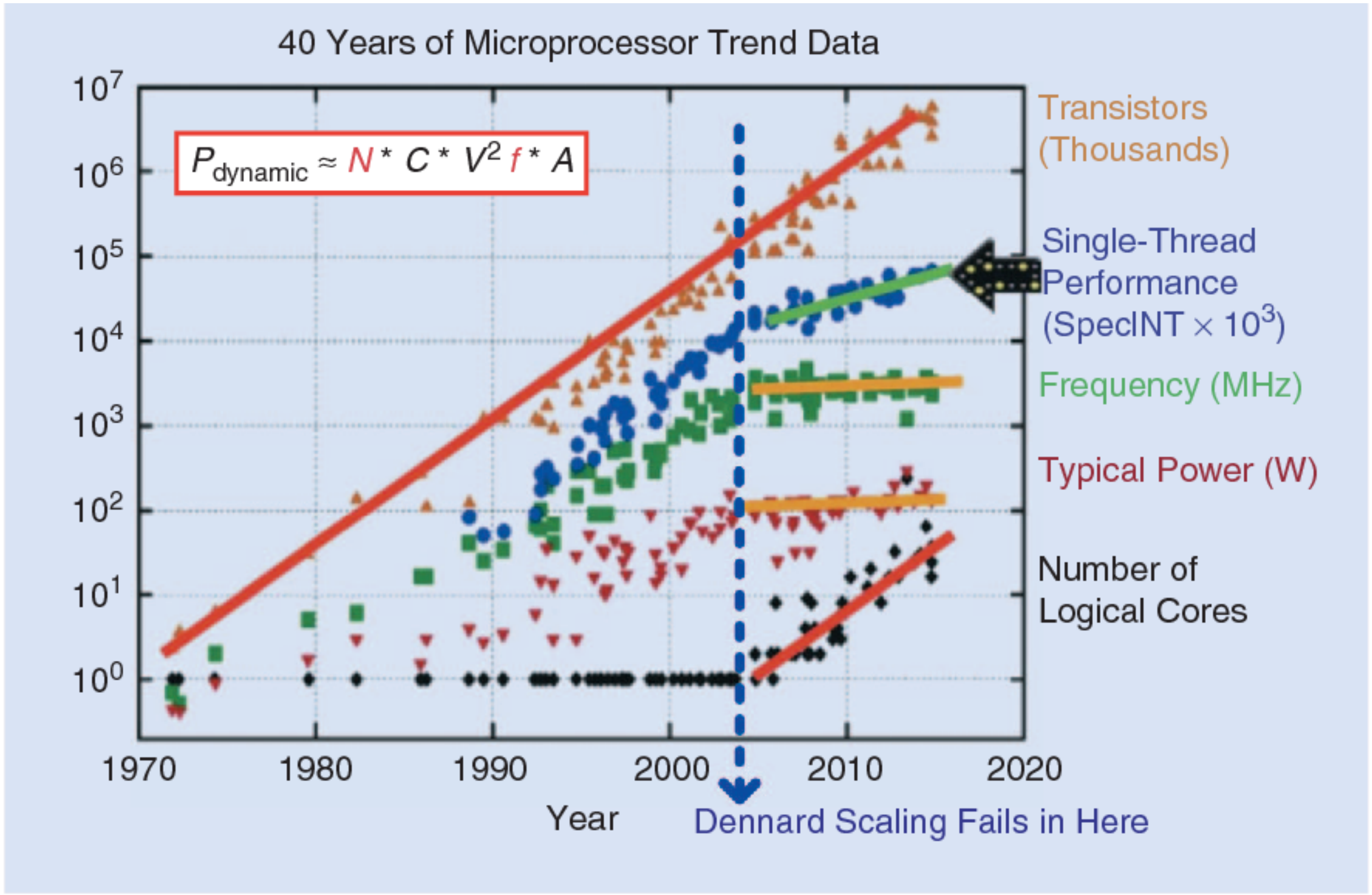
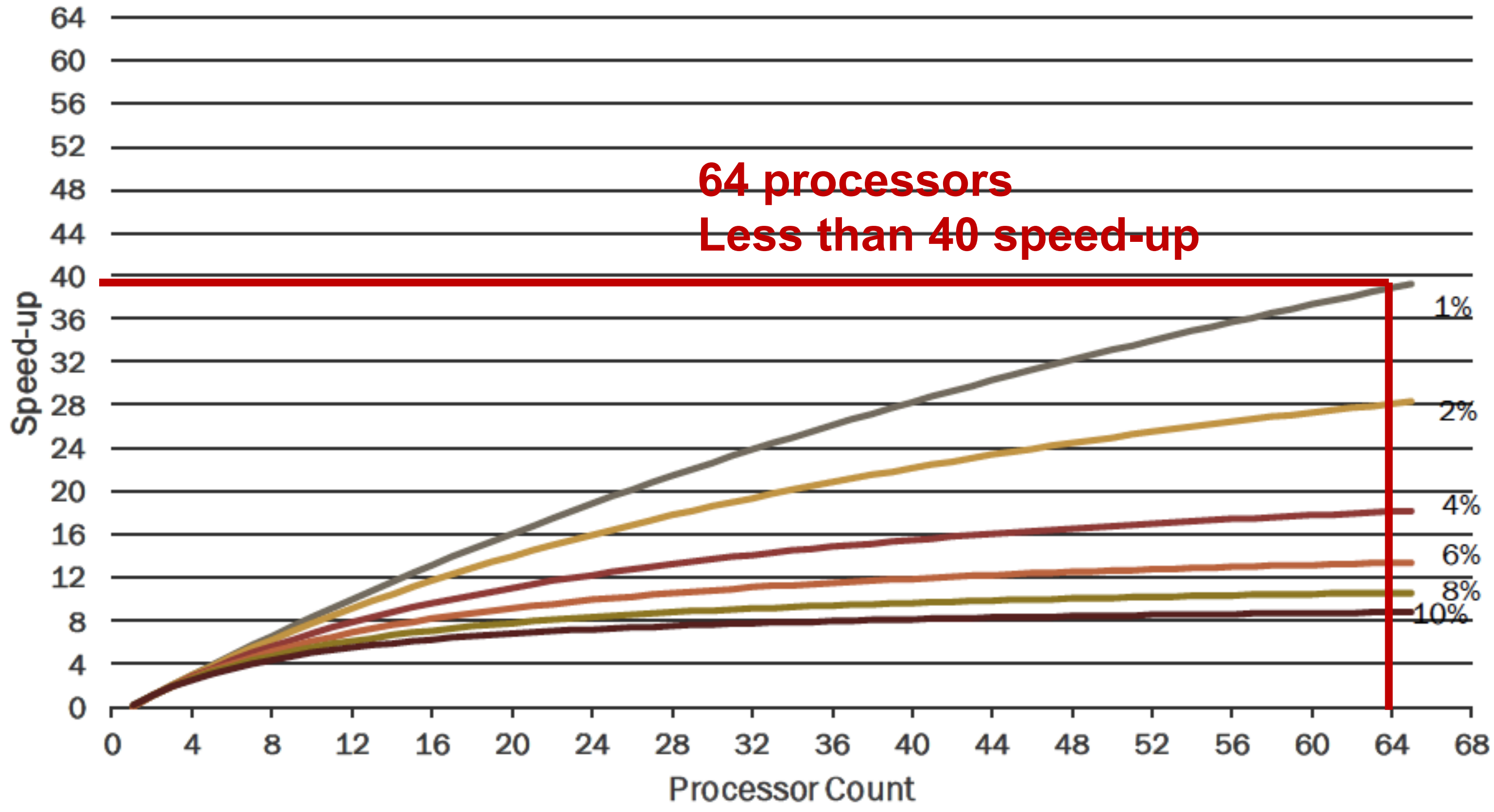


**FIGURE 1:** The Dennard scaling failed around the middle of the 2000s [24].

**Source**: https://www.semanticscholar.org/paper/Time-Moore%3A-Exploiting-Moore%27s-Law-From-The-of-Time-Xiu/7ab3f9215c7d8369a6acdf9a3546138a71ac9fb5/figure/0

## Amdahl's Law

- The fundamental limit to gains of parallelization
- Only 1% serial code fragment has significant impact on achievable many core performance



## Summary Part I

- Advances in processors supported by Moore's Law
  - 1980 RISC: 25k transistors
  - Today: Up to 2.5 billion transistors

- Lennard scaling supported theses advances as well
  - Power remained similar with more and more transistors
  - Lennard scaling failed in 2005

- Energy per operation needs to be minimized
  - Utilize specific features of application domain
  - Tailor architecture to the actual domain

- New opportunities for engineers
  - Application Specific Instruction Processors
  - Domain specific architectures

## Outline

- ☐ Part I: Performance Limits of GPPs
- ☐ Part II: Opportunities of ASIP
- ☐ Part III: Case Study
  - ☐ Google TPU
- ☐ Part IV: Some Remarks

## The Opportunities

- SW-centric
  - Modern scripting languages are interpreted, dynamically typed and encourage reuse
  - Efficient for the programmer but not for execution
- HW-centric
  - Application-Specific architectures
  - Design to perform extraordinary well in one domain / doing a few tasks
- HW-/SW-centric
  - Combine both
  - Application-specific features but yet programmable
  - Use existing languages working well or even introduce new ones for specific domain

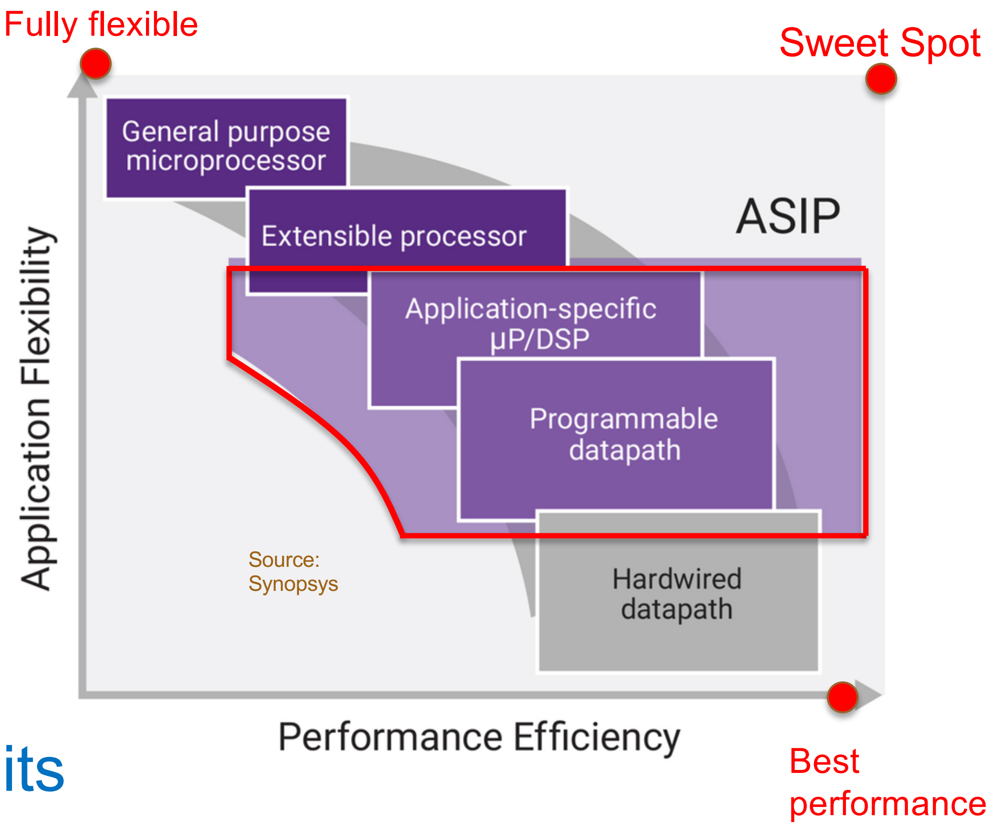
## Platforms to target applications

- SW-centric
  - Standard processors (GPP)
    - Flexible and short design time
    - Lack of computational capacity
- HW-centric
  - Specialized Hardware (ASIC)
    - Real-time performance, small size and low power
    - High non-recurring engineering (NRE) costs
  - Fine-grained reconfigurable architectures (FPGA)
    - High calculation capacity and flexible
    - Routing overhead, high power consumption
    - HW oriented design approach
- HW-/SW-centric:
  - Application-Specific Instruction-Set Processors
    - High performance in dedicated application domain
    - Short design time, SW + HW oriented design approach

# Application-Specific Instruction-Set Processor

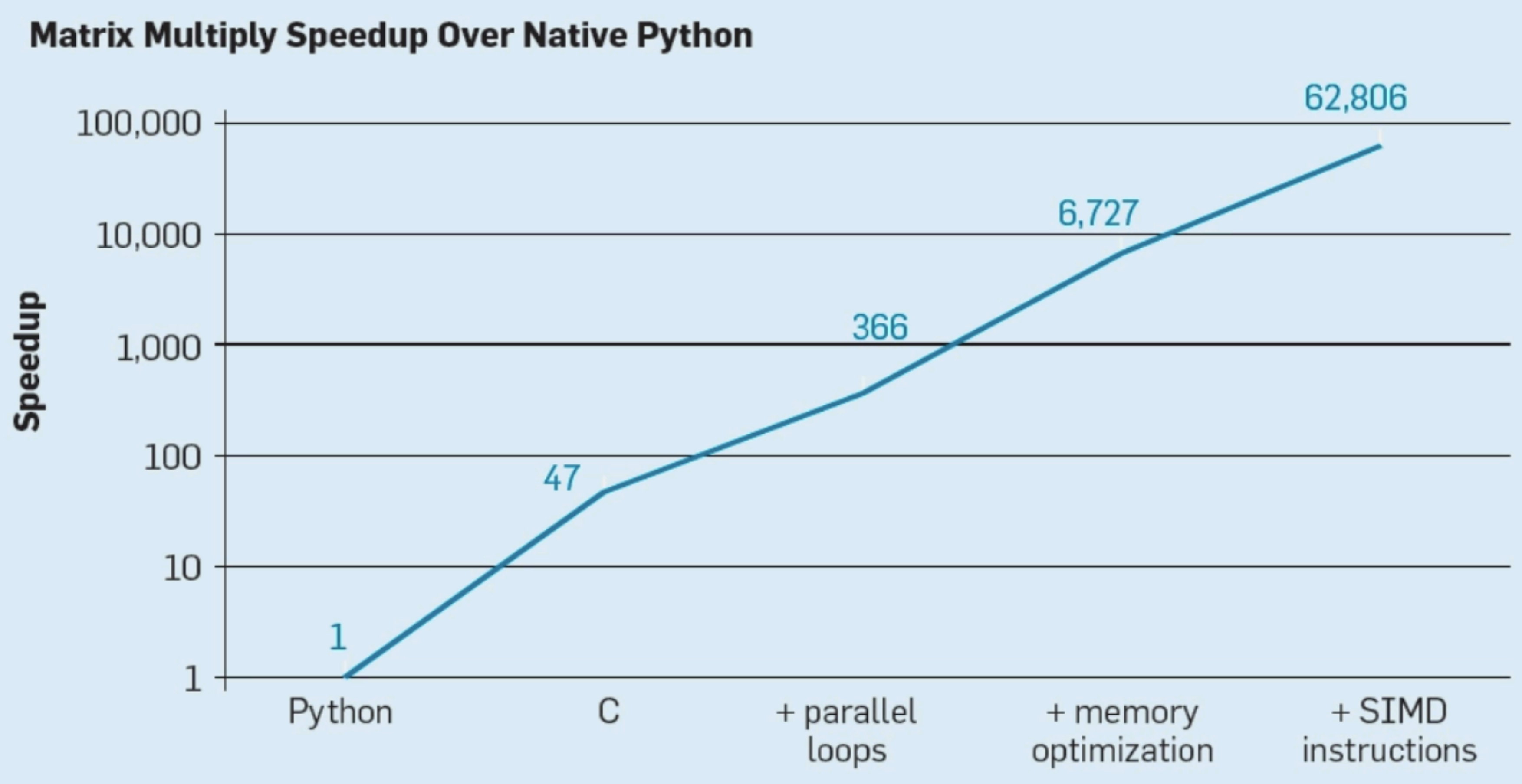
- Flexibility and Efficiency
  - 2 contradicting design goals

- Optimize architecture for specific domain
- Keep it flexible
- Keep it efficient



The big challenge:
Find the architecture that suits the application domain best while still being programmable

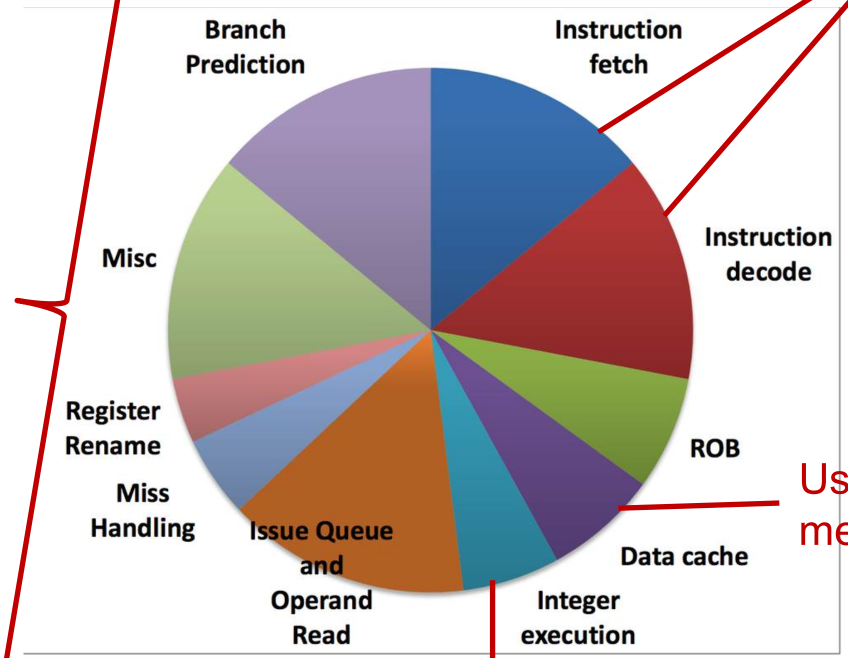
# Example: Matrix-Multiplication



# Guidelines for ASIPs / DSAs

Use the easiest form of parallelism

Invest the resources saved from dropping advances microarchitectural optimizations into more arithmetic units or bigger memories

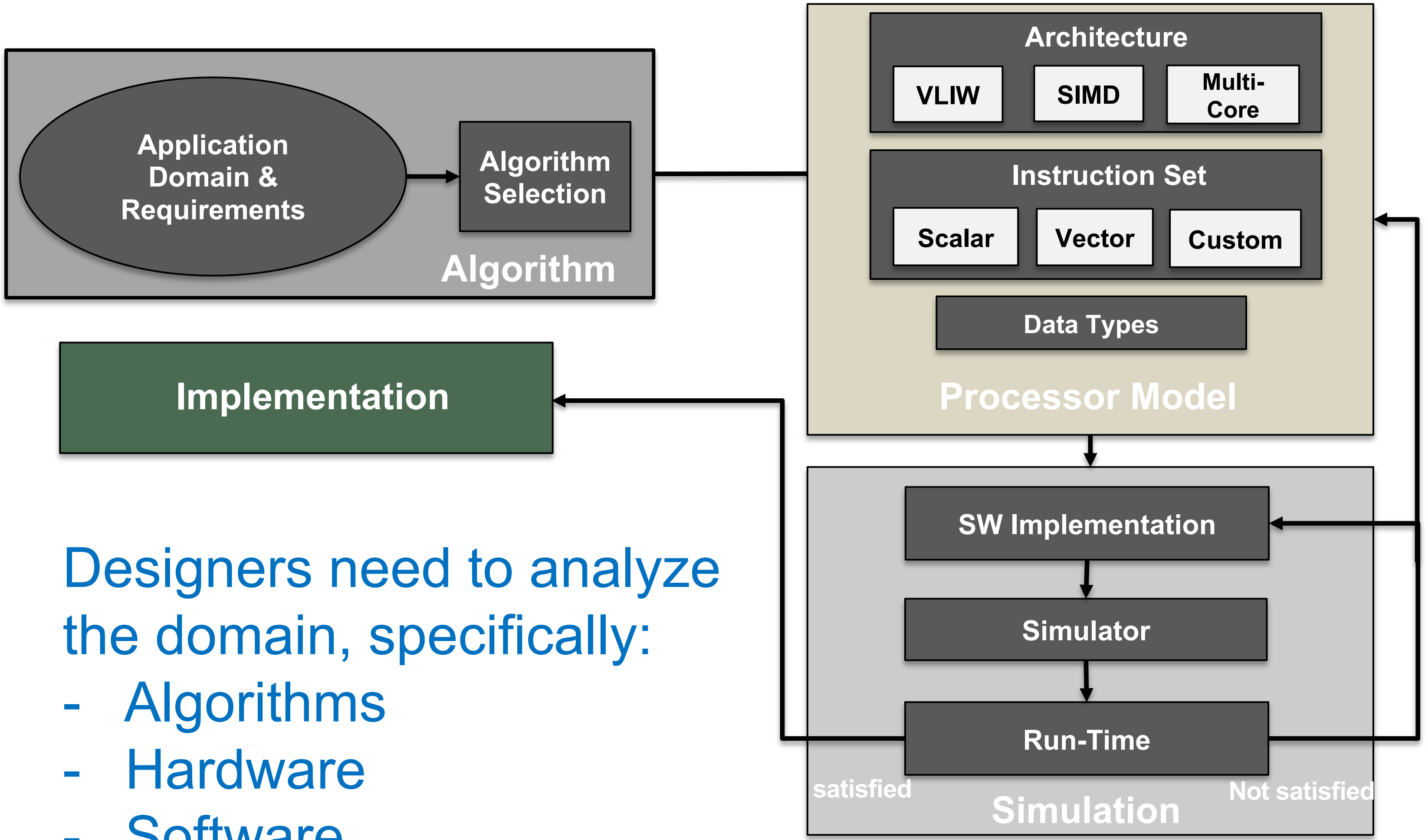


Use dedicated memories

Reduce data size and type to the simplest needed for the domain

Use domain specific language whenever possible

# Algorithm / Architecture Co-Design



Designers need to analyze the domain, specifically:

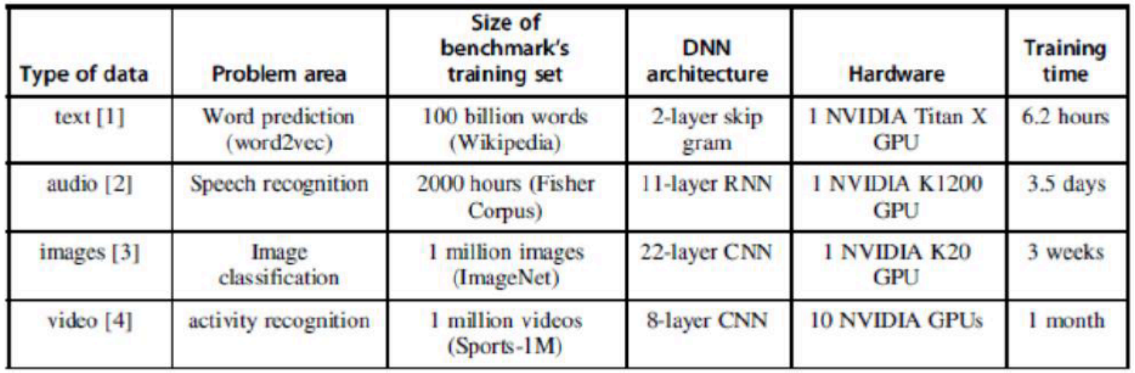
- Algorithms
- Hardware
- Software

## Outline

- Part I: Performance Limits of GPPs
- Part II: Opportunities of ASIP
- **Part III: Case Study**
  - **Google TPU**
- Part IV: Some Remarks

## Example: Deep Neural Networks

- Inspired by neurons of the brain
- Computes the non-linear "activation" function of the weighted sum of input values
- In general practitioners select existing designs that showed to work well
- Training (learning)
  - Calculate weights using backpropagation of the error
  - Training may take an extremely long

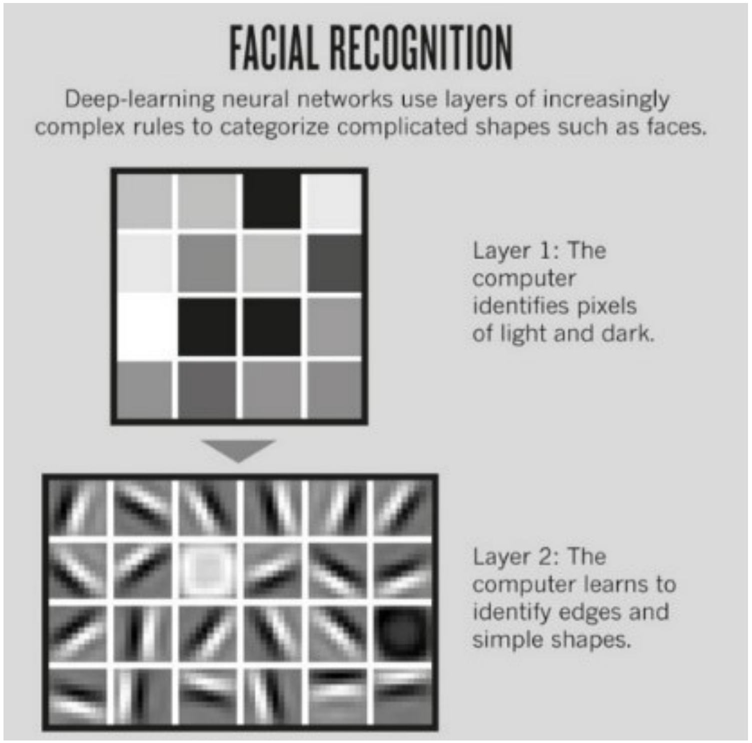


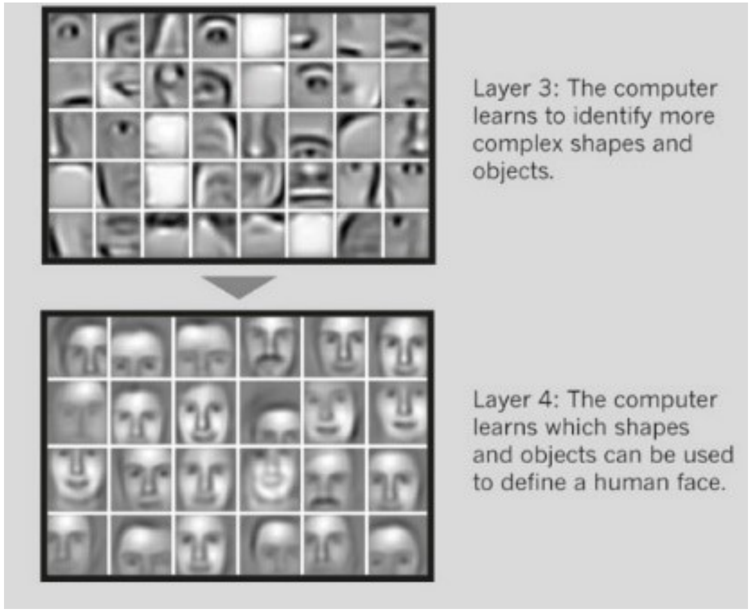
| Type of data | Problem area | Size of benchmark's training set | DNN architecture | Hardware | Training time |
|---|---|---|---|---|---|
| text [1] | Word prediction (word2vec) | 100 billion words (Wikipedia) | 2-layer skip gram | 1 NVIDIA Titan X GPU | 6.2 hours |
| audio [2] | Speech recognition | 2000 hours (Fisher Corpus) | 11-layer RNN | 1 NVIDIA K1200 GPU | 3.5 days |
| images [3] | Image classification | 1 million images (ImageNet) | 22-layer CNN | 1 NVIDIA K20 GPU | 3 weeks |
| video [4] | activity recognition | 1 million videos (Sports-1M) | 8-layer CNN | 10 NVIDIA GPUs | 1 month |

- **Inference**: Use the neural network for classification

# Convolutional Neural Networks

- Widely used in computer vision
- Each layer raises the level of abstraction
  - From detecting a line, to an edge, to a car etc.

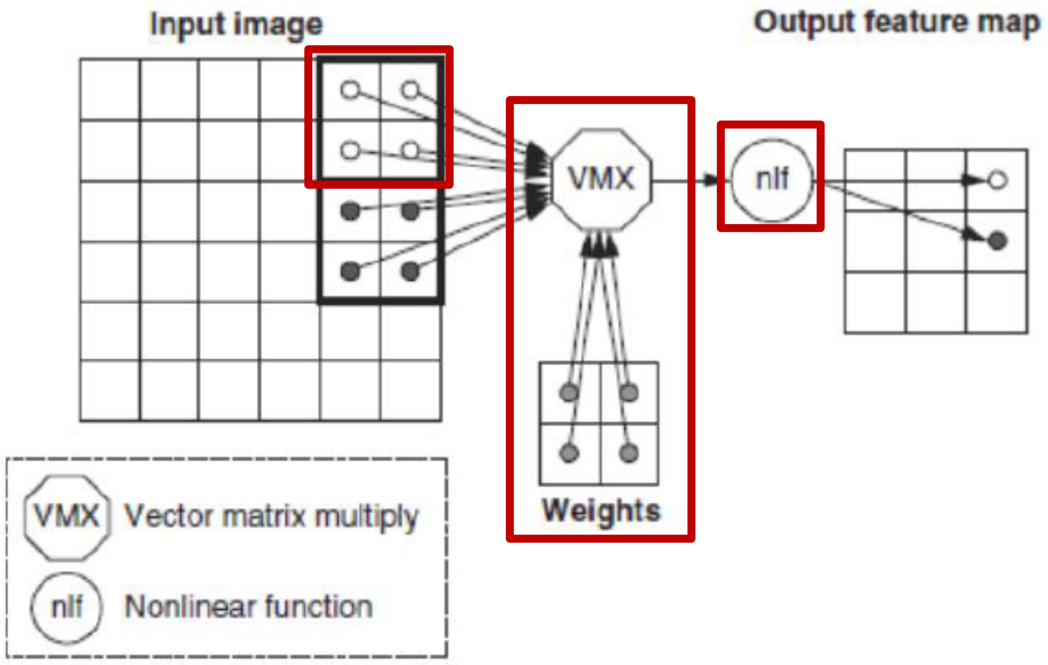

FACIAL RECOGNITION
Deep-learning neural networks use layers of increasingly complex rules to categorize complicated shapes such as faces.
Layer 1: The computer identifies pixels of light and dark.
Layer 2: The computer learns to identify edges and simple shapes.


Layer 3: The computer learns to identify more complex shapes and objects.
Layer 4: The computer learns which shapes and objects can be used to define a human face.

Source:
https://medium.com/dataminingapps-articles/understanding-deep-learning-in-5-minutes-cc140a10078b

# Convolutional Neural Networks

- Input Image is transformed to an output feature map



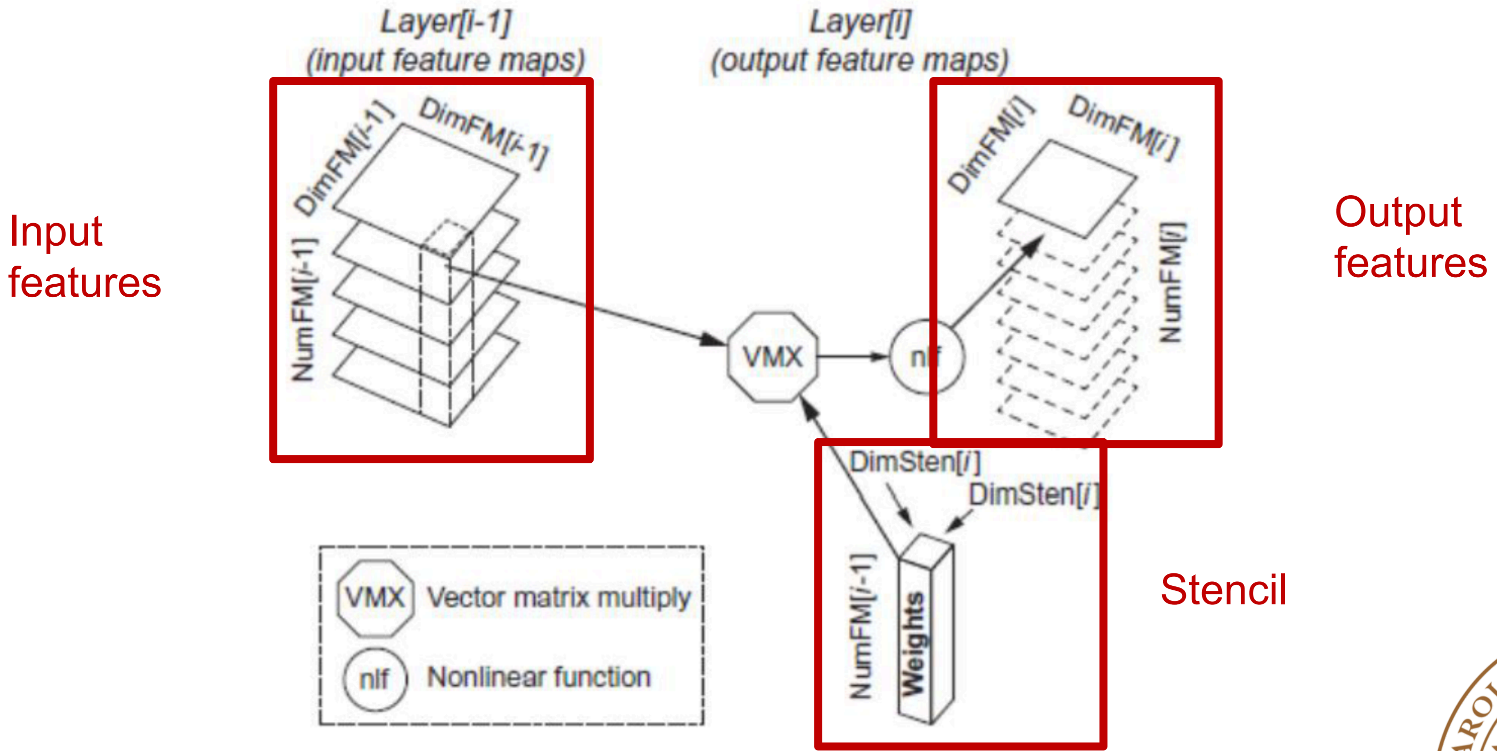
Fetch input image or output from previous layer

Convolution of the input vector with the (learned) weights

Apply non-linear function

## Convolutional Neural Networks

- Each layer may consist of several kernel, each having different weights and producing a different output feature map
- E.g. colored pixel with red, green and blue



## Convolutional Neural Networks

- Total number of weights
  - NumFM[i] x NumFM[i-1] x DimSten[i]$^2$
- Total number of operations
  - 2 x DimFM[i]$^2$ x # of weights
- Operation per weight
  - 2 x DimFM[i]$^2$
- Example

CNN with

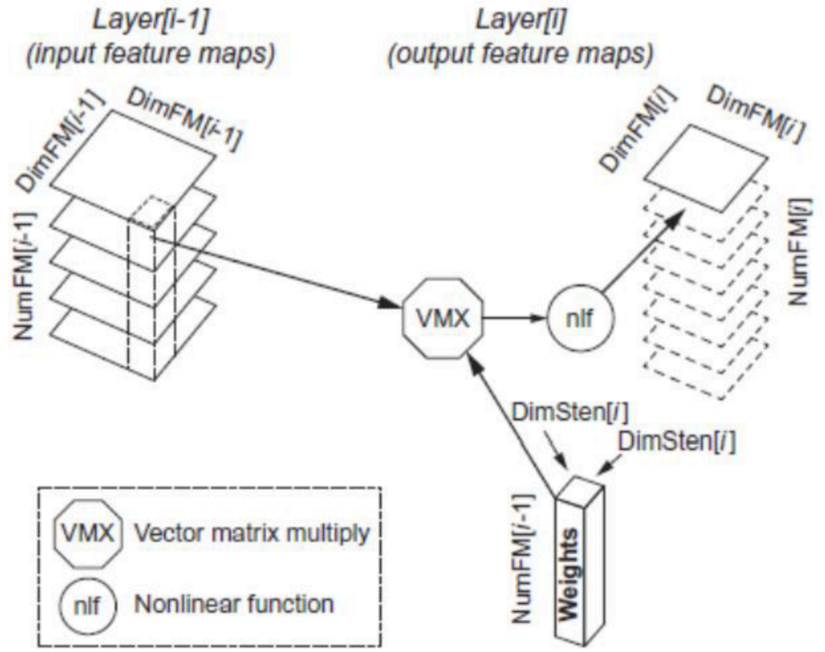
DimFM[i-1] = 28

DimFM[i] = 14

DimSten[i] = 3

NumFM[i-1] = 128

NumFM[i] = 64



**#weights = 73 k**
**#operations = 28 M**

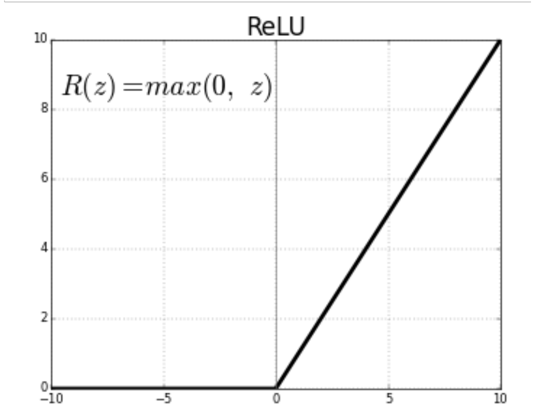
# Convolutional Neural Networks

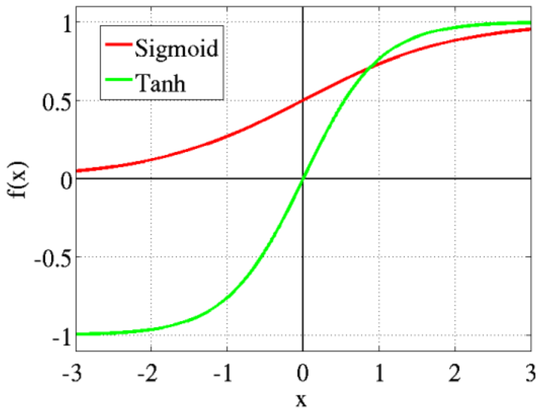
- Efficiently implemented kernels needed

- Arithmetic units
  - Matrix-Matrix multiplication
  - Matrix-Vector Multiplication

- Memory
  - Stencil operation
  - Efficient access

- Accelerated functionalities
  - Non-linear functions
    - ReLU
    - Sigmoid
    - Hyperbolic tangent





## Google's Tensor Processing Unit (TPU)

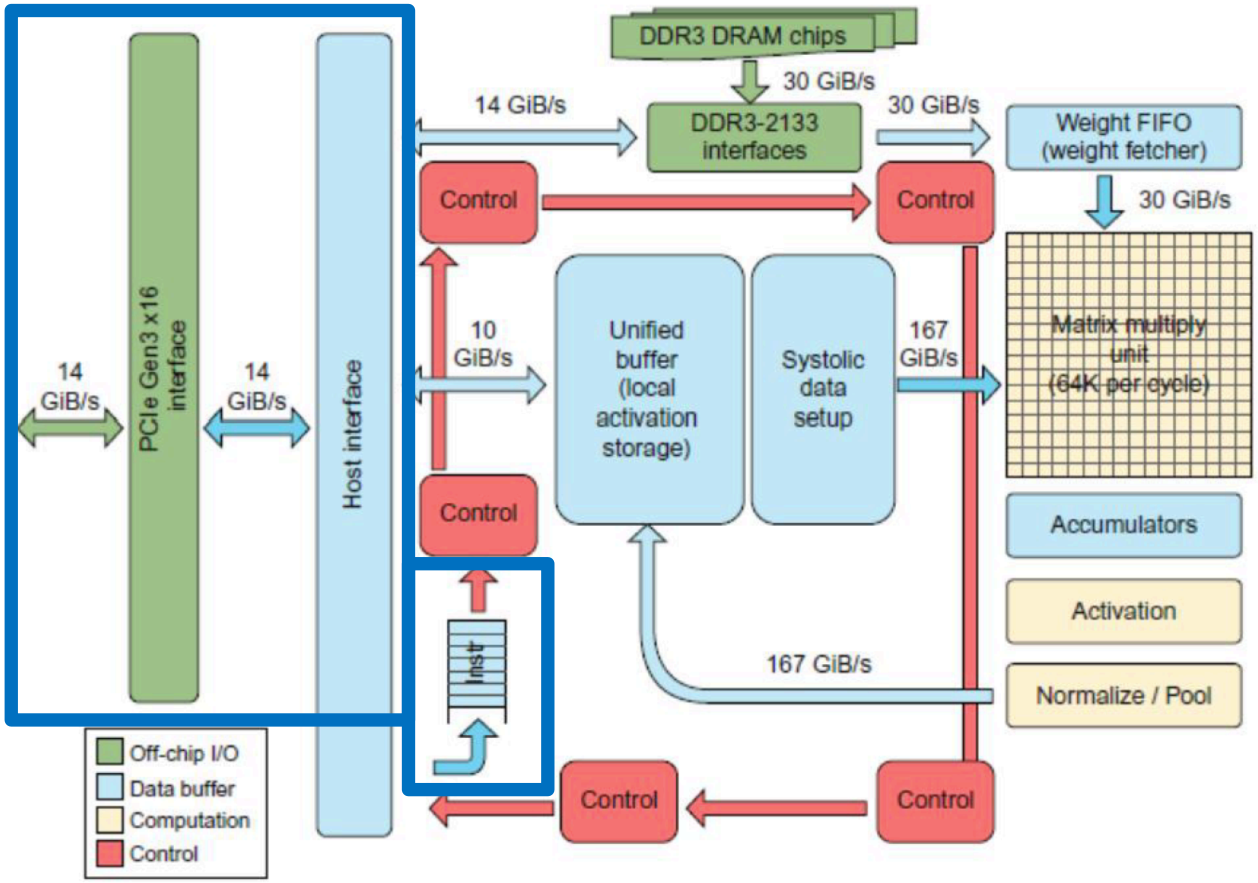
- Google's DNN accelerator
- Used for inference

- Large Arithmetic Unit
  - 256x256 8-bit multiplication unit

- Memory
  - Large software managed on-chip memory

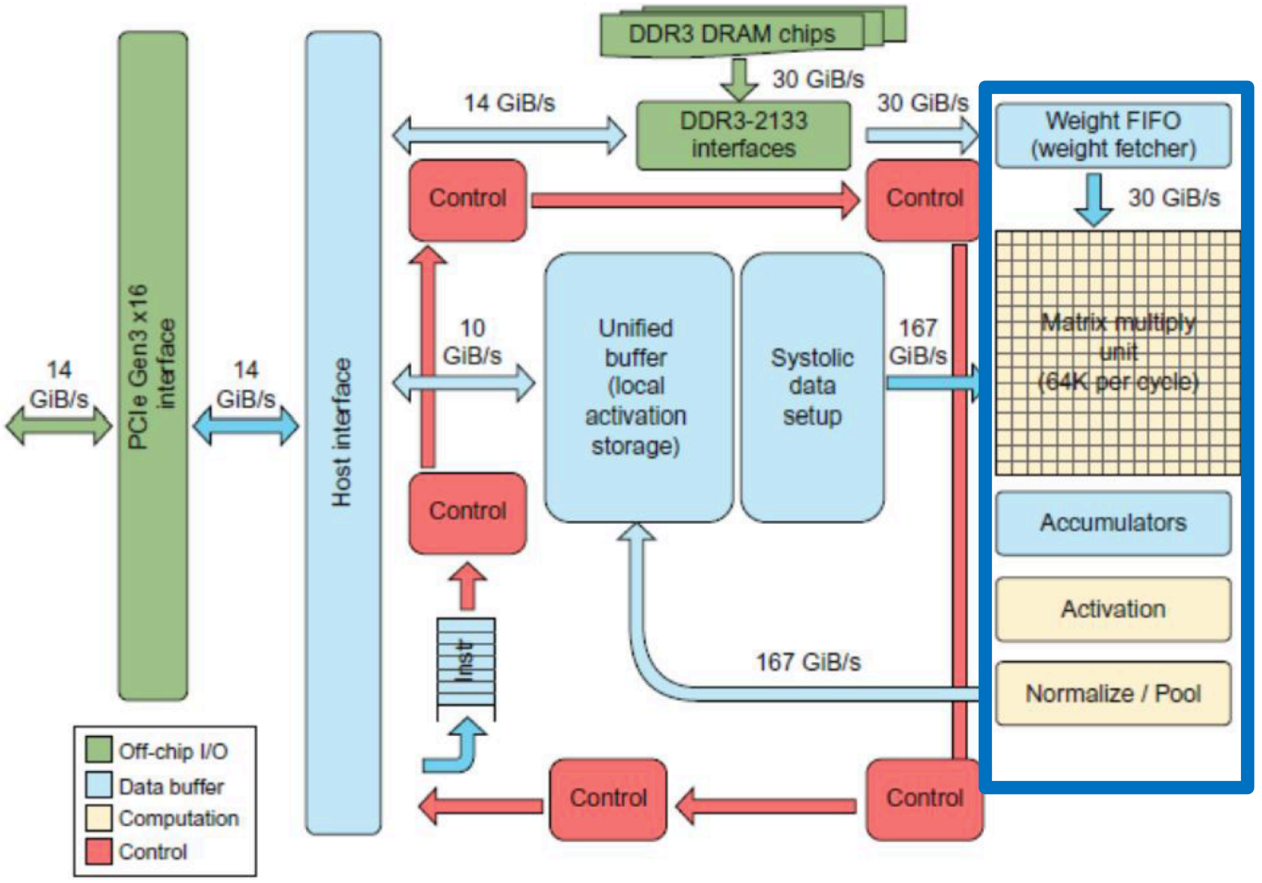
- Utilized as co-processor attached on PCIe-bus

## Google TPU Architecture



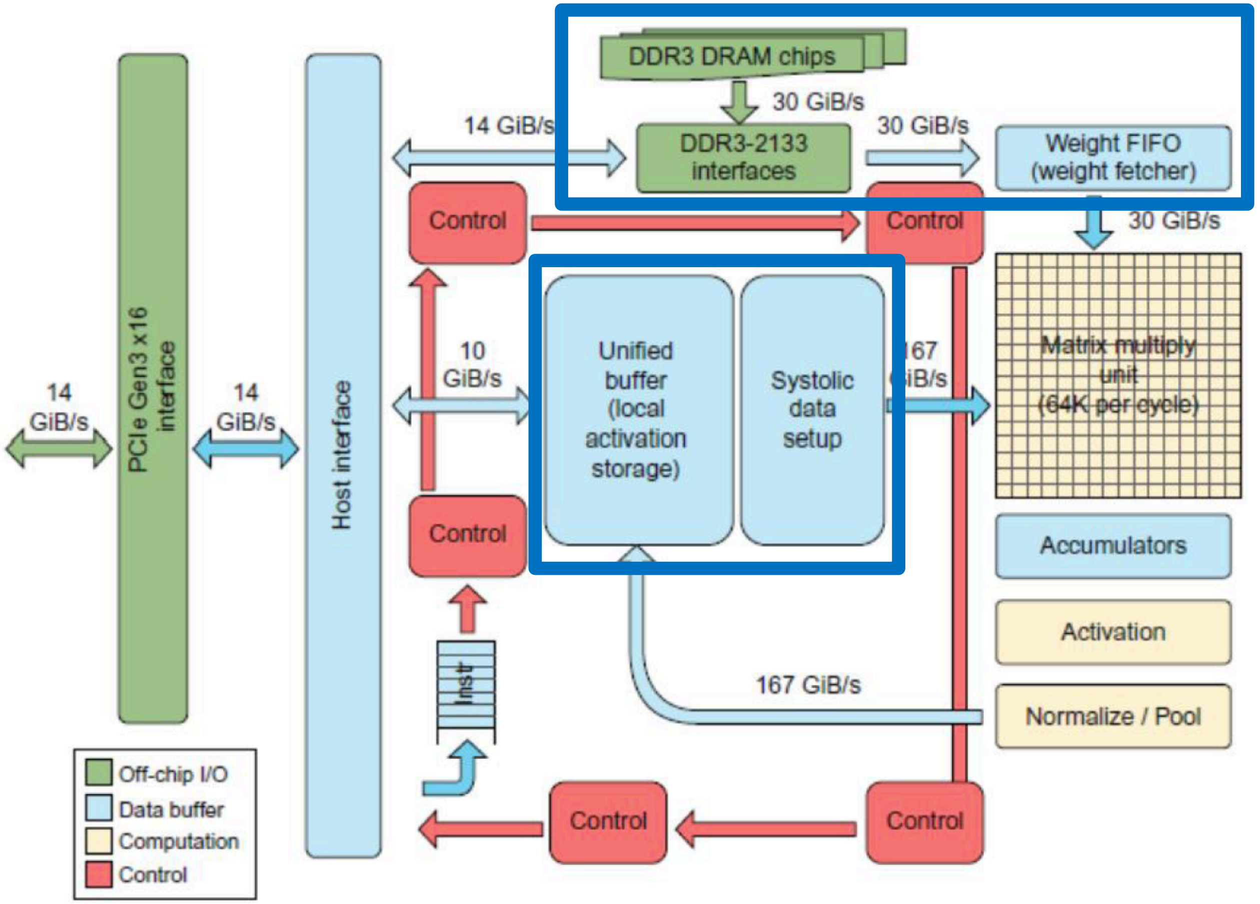
- Attached to server via PCIe Interface
- Instructions send from server to TPU and locally stored in instruction buffer (no instruction fetch)

# Google TPU Architecture



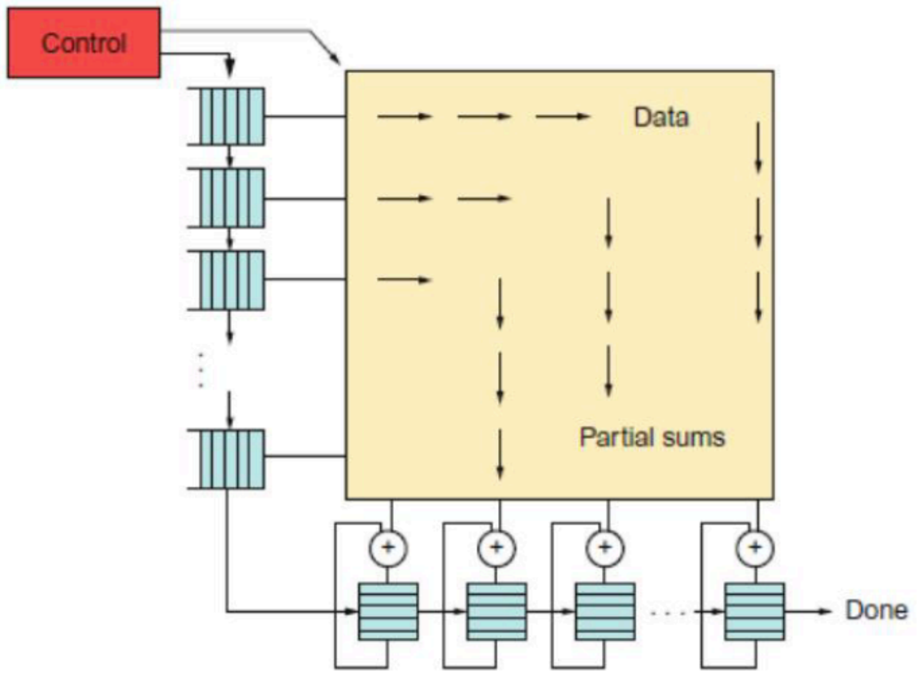
- Heart is the matrix multiply unit (multiplication or convolution)
  - Reads 256 values per clock cycle
- Results are stored in accumulators
- Non-linear functions calculated in activation hardware

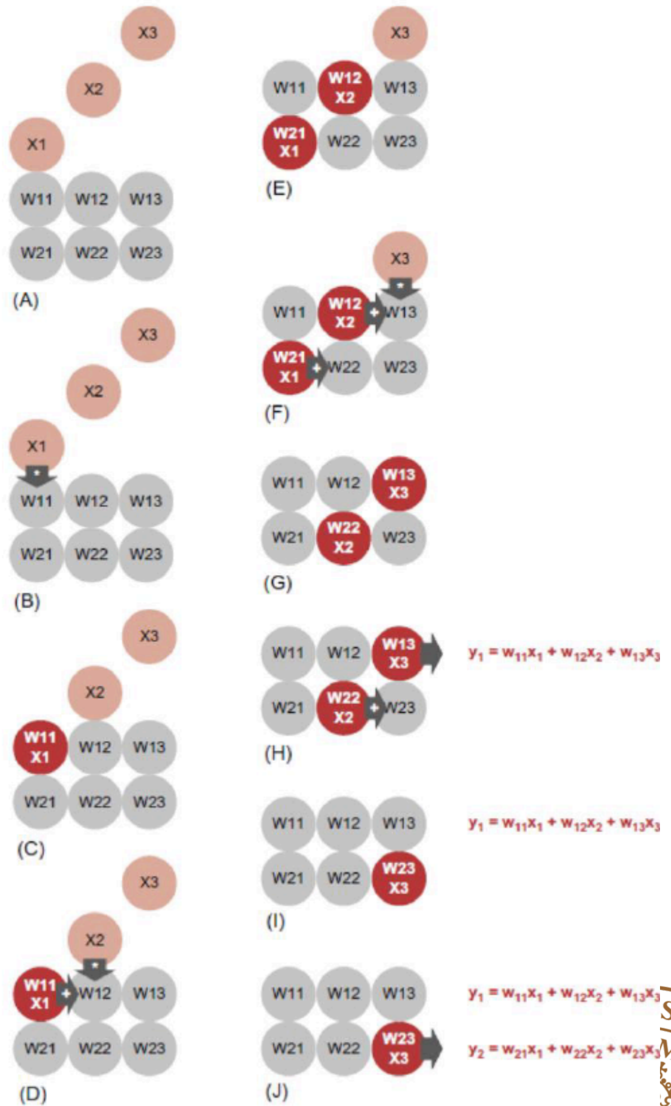
## Google TPU Architecture



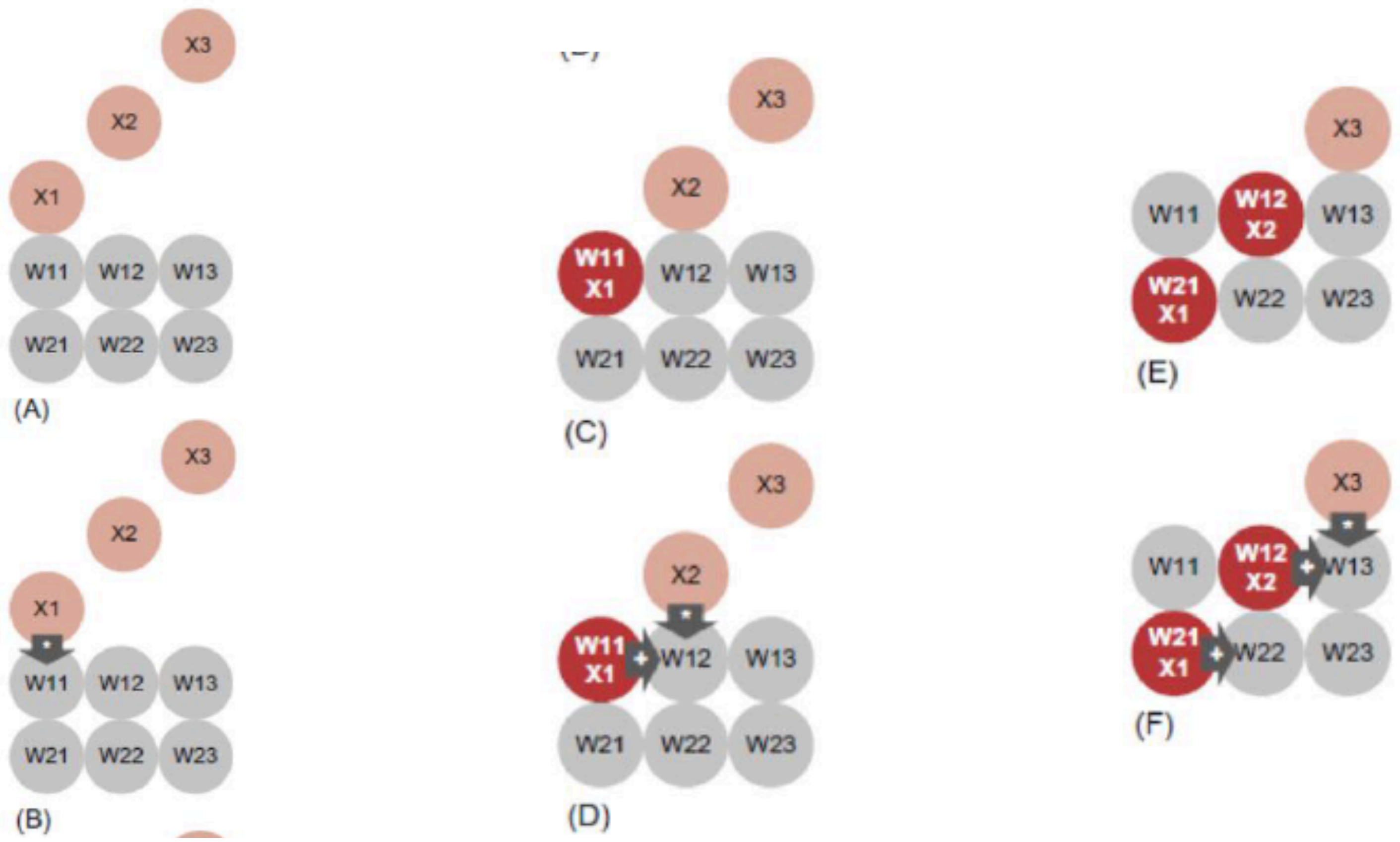
- Weights fetched via on-chip weight FIFO reading from an 8GiB off-chip DRAM
- Input data and intermediate results in 24 MiB unified buffer
- DMA controller handles transfer of data

# Google TPU Matrix Multiply Unit

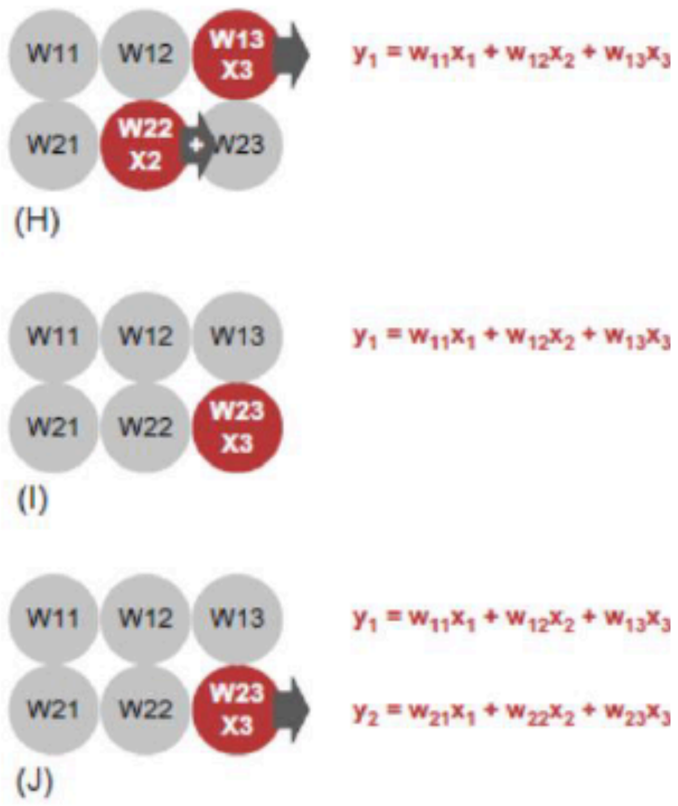



X1
X2
X3
W11
W12
W13
W21
W22
W23
(A)
(B)
(C)
(D)
(E)
(F)
(G)
(H)
(I)
(J)
$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$
$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$

# Google TPU Matrix Multiply Unit

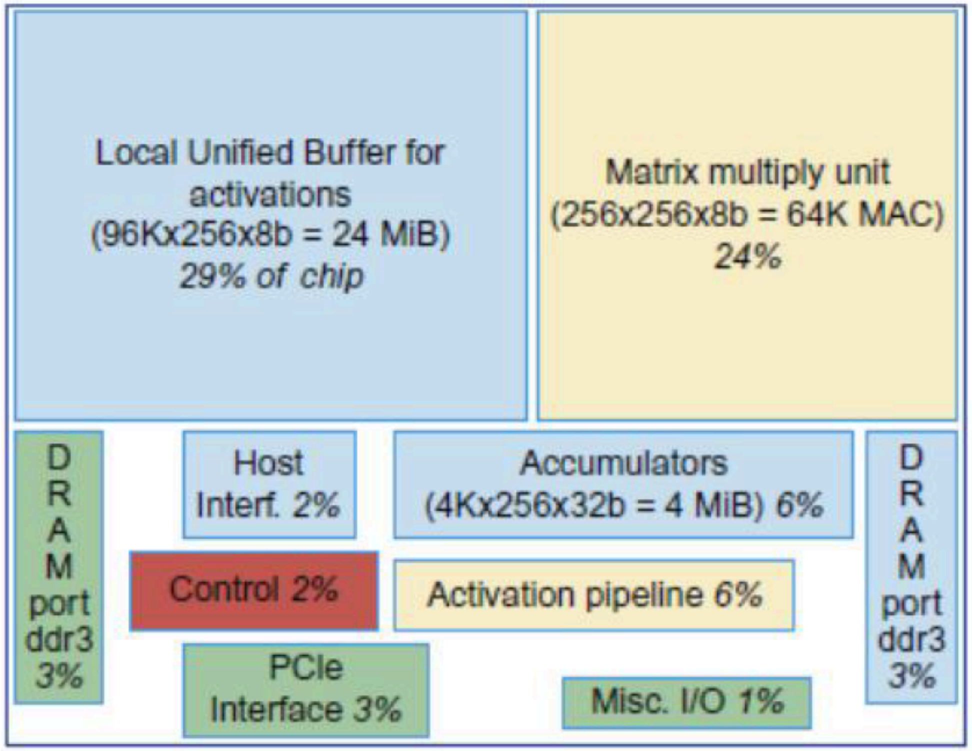


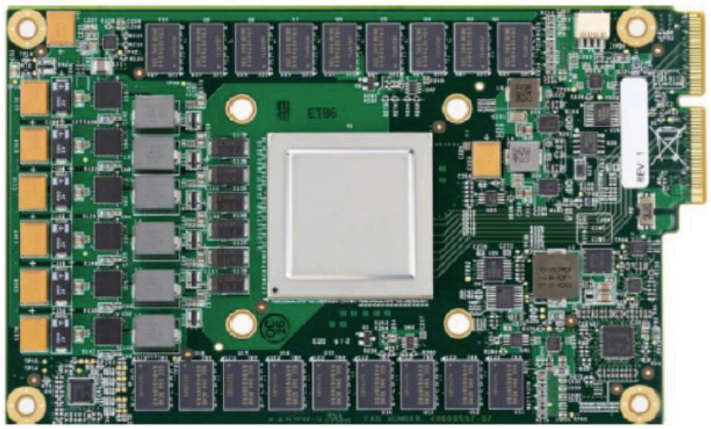
# Google TPU Matrix Multiply Unit


W11
W12
W13
X3
$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$
W21
W22
X2
+
W23
(H)
W11
W12
W13
$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$
W21
W22
W23
X3
(I)
W11
W12
W13
$y_1 = w_{11}x_1 + w_{12}x_2 + w_{13}x_3$
W21
W22
W23
X3
$y_2 = w_{21}x_1 + w_{22}x_2 + w_{23}x_3$
(J)

# Google TPU Area Distribution

- First version TPU
  - 25% for matrix multiply unit
  - 30% for unified buffer

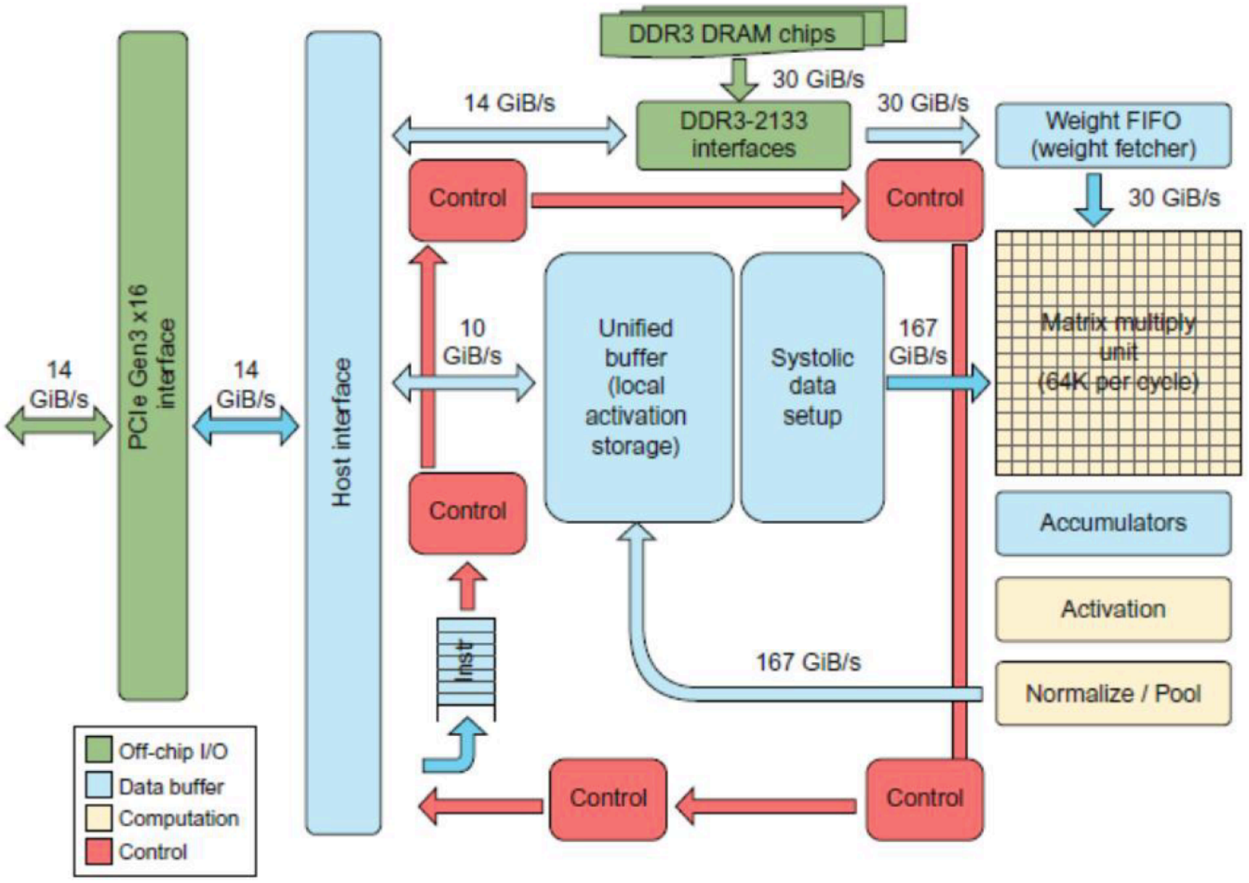

Local Unified Buffer for activations (96Kx256x8b = 24 MiB) 29% of chip
Matrix multiply unit (256x256x8b = 64K MAC) 24%
DRAM port ddr3 3%
Host Interf. 2%
Accumulators (4Kx256x32b = 4 MiB) 6%
DRAM port ddr3 3%
Control 2%
Activation pipeline 6%
PCIe Interface 3%
Misc. I/O 1%



## Google TPU ISA

- CISC instruction set with repeat field
  - No program counter
  - No branch instructions
  - CPI between 10 and 20
- Instructions
  - READ_HOST_MEMORY
    - Read memory from CPU into the unified memory
  - READ_WEIGHTS
    - Read weights from weight memory into the weight FIFO
  - MatrixMatrixMultiply/Convolve
    - Perform matrix-matrix, matrix-vector, elementwise matrix-multiply, an elementwise vector multiply or convolution
    - Utilize repeat field to multiply a B x 256 with a 256x256 to receive a B x 256 output
    - Store result in unified buffer
  - Activate
    - Compute the activation functions output
  - WRITE_HOST_MEMORY
    - Write data from unified buffer into host memory

# Google TPU Architecture



# Google TPU Performance

| | | | | | Measured | | TOPS/s | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Chip model | $mm^2$ | nm | MHz | TDP | Idle | Busy | 8b | FP | GB/s | On-chip memory |
| Intel Haswell | 662 | 22 | 2300 | 145W | 41W | 145W | 2.6 | 1.3 | 51 | 51 MiB |
| NVIDIA K80 | 561 | 28 | 560 | 150W | 25W | 98W | – | 2.8 | 160 | 8 MiB |
| TPU | <331* | 28 | 700 | 75W | 28W | 40W | 92 | – | 34 | 28 MiB |

*The TPU die size is less than half of the Haswell die size.

**Figure 7.42** **The chips used by the benchmarked servers are Haswell CPUs, K80 GPUs, and TPUs. Haswell has 18 cores, and the K80 has 13 SMX processors.**

| | | | | Measured power | |
|---|---|---|---|---|---|
| Server | Dies/Server | DRAM | TDP | Idle | Busy |
| Intel Haswell | 2 | 256 GiB | 504W | 159W | 455W |
| NVIDIA K80 (2 dies/card) | 8 | 256 GiB (host) + 12 GiB × 8 | 1838W | 357W | 991W |
| TPU | 4 | 256 GiB (host) + 8 GiB × 4 | 861W | 290W | 384W |

**Figure 7.43** **Benchmarked servers that use the chips in Figure 7.42.** The low-power TPU allows for better rack-level density than the high-power GPU. The 8 GiB DRAM per TPU is Weight Memory.

# Google TPU Performance

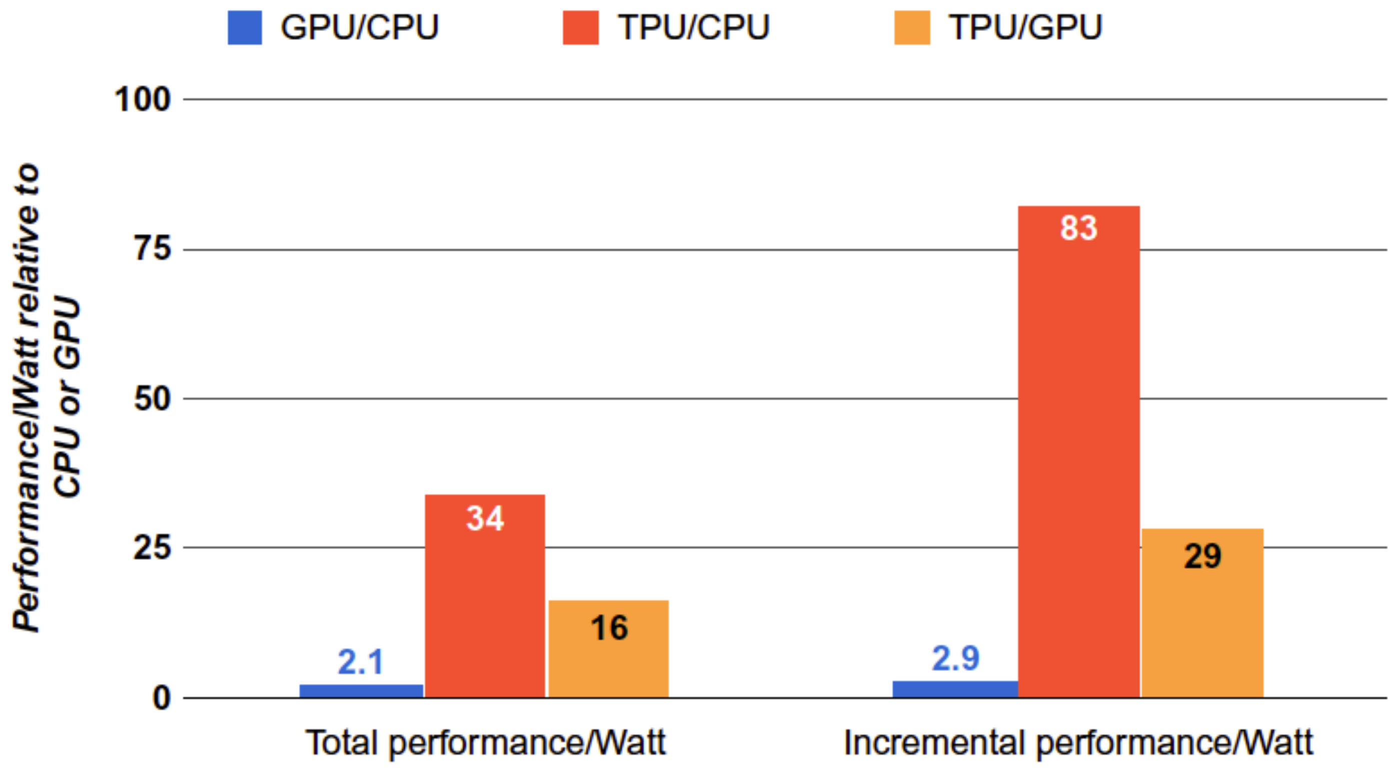


**Figure 7.50 Relative performance/watt of GPU and TPU servers to CPU or GPU servers.** Total performance/watt includes host server power, but incremental doesn't. It is a widely quoted metric, but we use it as a proxy for performance/TCO in the data center.

## Google TPU Deployed

Up to 64 TPUs may be combined to achieve
11.5 petaops
of performance

## Guidelines for ASIPs / DSAs

- How TPU follows the guidlines
  - Use dedicated memories to minimize the distance over which data is moved
    - 24 MiB unified buffer, optimized for access of 256 bytes at a time
    - 4 MiB accumulators each 32-bit wide
    - 8-bit weights stored in off-chip DDR3 RAM
  - Invest the resources saved from dropping advanced microarchitectural optimizations into more arithmetic units or bigger memories
    - 28 MiB dedicated memory (60% compared to server CPU)
    - 65,538 8-bit ALUs (250 times as many as server CPU)
  - Use the easiest form of parallelism that matches the domain
    - Two-dimensional SIMD pipelined with systolic structure

## Guidelines for ASIPs / DSAs

- How TPU follows the guidelines
  - Reduce data size and type to the simplest needed for the domain
    - TPU is mainly optimized for 8-bit computations
    - Supports up to 16-bit by running multiply unit in quarter rate configuration
    - No support for 64-bit or floating point
  - Use a domain specific programming language
    - Programmed using TensorFlow programming framework

## Outline

- Part I: Performance Limits of GPPs
- Part II: Opportunities of ASIP
- Part III: Case Study
  - Google TPU
- Part IV: Some Remarks

## Take a Look at History

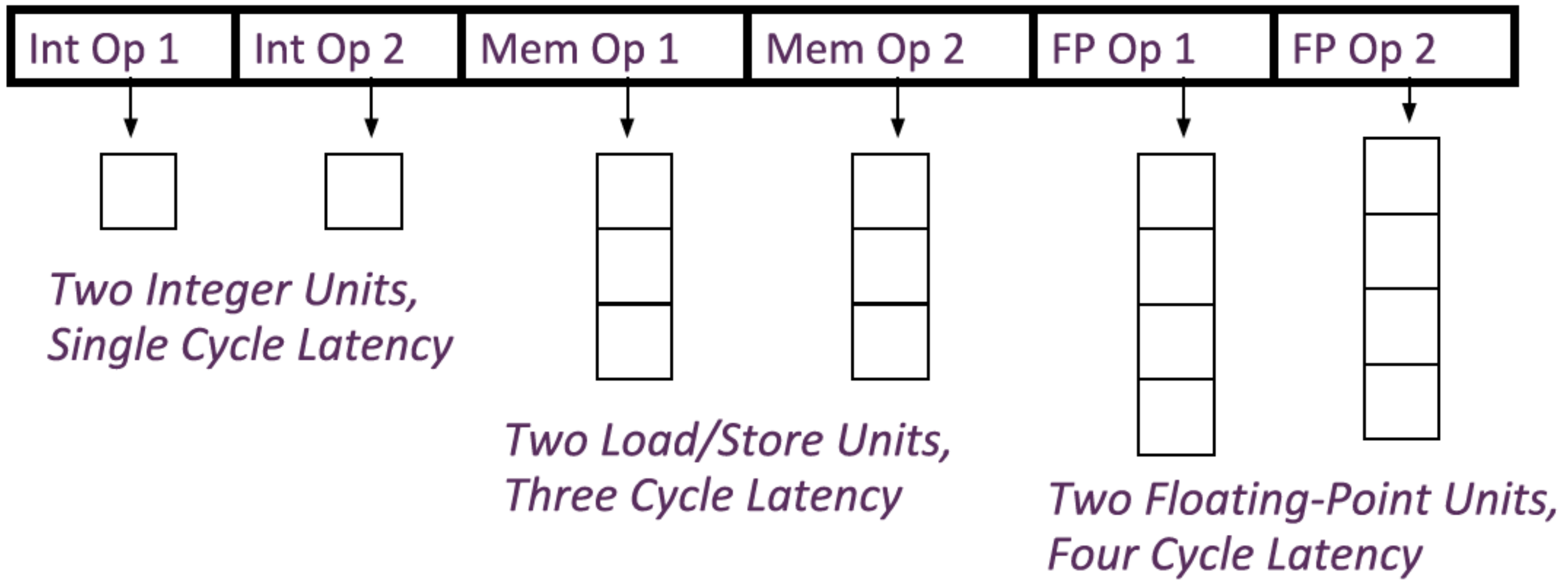
- GPPs and ASIPs are very different
- Don’t be ignorant of research and architectures in past
- Something that did not work for GPPs may still be a good design choices for ASIPs

- Example: Google TPU
  - Utilizes CISC instructions
    - Small instruction set
    - Specialized instruction set that benefits from complex instructions
  - Nowadays all popular processors utilize RISC instructions

- Another Example:

  VLIW – Very Long Instruction Word

## Very Long Instruction Word (VLIW)


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP Op 1
FP Op 2
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

- Multiple instructions packed into one instruction
- Each slot for a dedicated functional unit
- Latency of each unit fixed
- Compiler schedules the slots during compile time

## From RISC to Intel/HP Itanium

- Intel developed jointly with HP starting in 1994
- EPIC – “Explicitly Parallel Instruction Computing”
- Many companies gave up RISC for Itanium
  - Microsoft
  - SGI
  - Itachi
- VLIW for GPPs ended in a failure
- Why? Any ideas?

# VLIW issues

- “The Itanium approach…was supposed to be so terrific – until it turned out that the wished-for compilers were basically impossible to write”

  - Donald Knuth, Stanford

- Compiler could not handle dependencies
- Code size explosion (also due to NOPs)
- Cannot predict complex branches offline

- VLIW are a failure for GPPs, but could they be an option for ASIPs?

## Tools for ASIP Design

- Cadence Tensilica Platform
  - Modifiable architectures with pre-configured architecture and selectable accelerators
  - Fixed pipeline stages
  - Add customized instructions and hardware
- Synopsys ASIP Designer
  - Design a fully custom ASIP from scratch using nML processor description language
  - Full flexibility including pipeline stages
  - Add own peripherals designed in HDL or HLS
  - Compiler is automatically generated based on processor description
- RISC-V
  - Can serve as a good basis for further domain specific acceleration

## Summary

- Simply scaling will not provide the required performance boost for future applications
  - Energy per operation becomes more and more important
- Processor have to be ”specialized” for the application domain
  - Designers need to understand hardware, software and algorithms to develop cutting edge designs
  - Tailor architecture to specific needs
  - Co-processors, accelerators or complete architectures help solbing bottlenecks
- A look in past research may help you boost domain specific performance
  - CISC
  - VLIW
  - Systolic arrays
  - …

## References

- Some interesting presentations
  - A New Golden Age for Computer Architecture History, Challenges, and Opportunities
  - DARPA ERI Summit 2018: The End of Moore's Law & Faster General Purpose Computing, & a New Golden Age

- Papers
  - 50 years of Computer Architecture: From the Mainframe CPU to the Domain-Specific TPU and the Open RISC-V Instruction Set

- Online Sources
  - Slides to The End of Moore's Law & Faster General Purpose Computing, & a New Golden Age
  - DNNs Tutorial
  - CNN to try yourself

- Course Book
  - Computer Architecture, 6th edition, chapter 7, Domain-Specific Architectures