Sven Gregorio
Seminar on Computer Architecture

Why systolic architectures?

Hsiang-Tsung Kung
Carnegie Mellon University

IEEE computer, 1982

Background, Problem & Goal

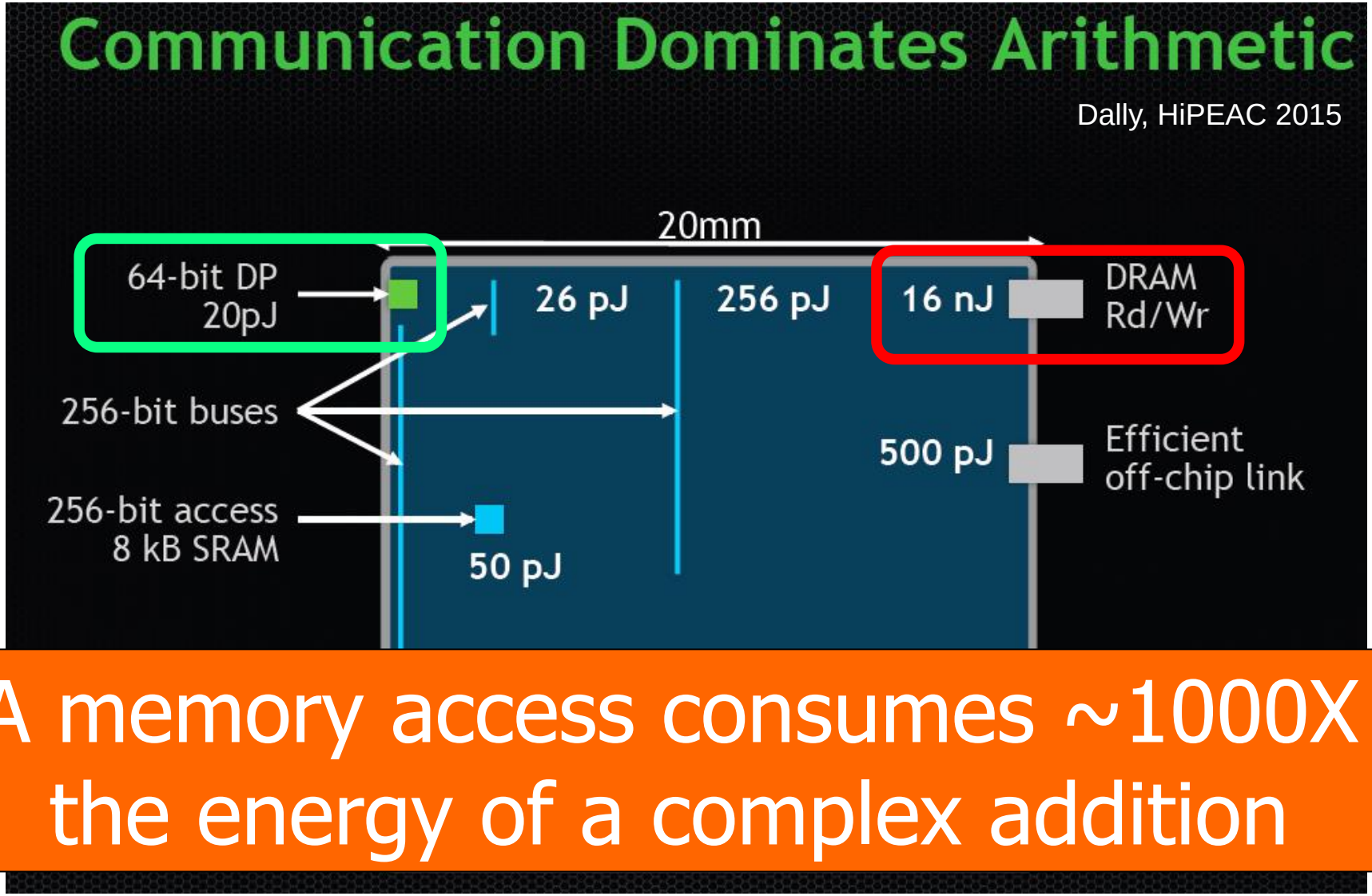
Special-purpose systems and their cost

- Many high-performance special-purpose systems are produced
  - General-purpose systems aren't always able to meet performance constraints
- Their cost is composed of **design** and **parts** cost
- Design cost tends to dominate the parts cost
  - Special-purpose systems usually produced in small quantities
- Special-purpose system are often design ad hoc
  - The designs solve one task only and aren’t generalizable
- The same errors are often repeated
  - Most notably: I/O imbalance

Why special-purpose systems?

- There is an interested in speeding up compute-bound computations
  - **Compute-bound**: #operations > #inputs + #outputs
    - E.g. matrix multiplication
  - Non compute-bound computations are **I/O bound**
    - E.g. matrix addition
- These computations tend to be too taxing for CPUs
  - **Von Neumann bottleneck**: for each operation at least an operand has to be fetched
    - Compute-bound computation become I/O bound
  - Memory bandwidth often isn't enough to keep the CPU pipeline filled
  - Memory accesses are costly in term of energy

Memory access energy cost



A memory access consumes ~1000X the energy of a complex addition

Adapted from Prof. Onur Mutlu's slides (Computer Architecture FS2018)

The key architectural requirements

1. **Simple** and **regular**
   - Decrease the design cost
   - Modular
   - Adjustable to performance goal
2. **High concurrency**
   - The main way to build faster computer systems
3. **Simple communication**
   - Tends to get more complex as concurrency increases
4. Balance of **computation** with **I/O**
   - The system shouldn't spend its time waiting for I/O operations

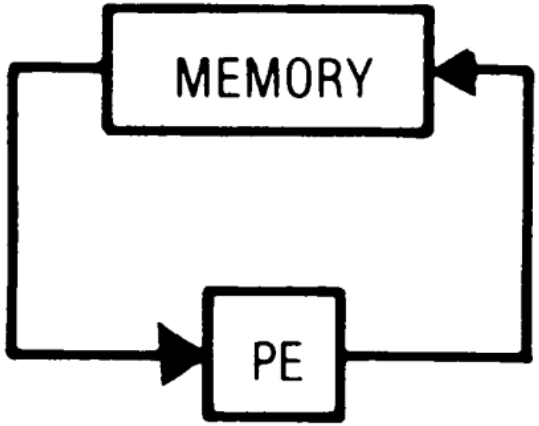
The goal

1. Accumulate the ideas of the author's previous work
   - Kung had already published multiple papers on systolic architectures

2. Correct the ad hoc approach by providing a general guideline
   - How to map high-level computations to hardware
   - The designs should respect the given requirements
   - Easy to use guideline

Novelty

The conventional approach



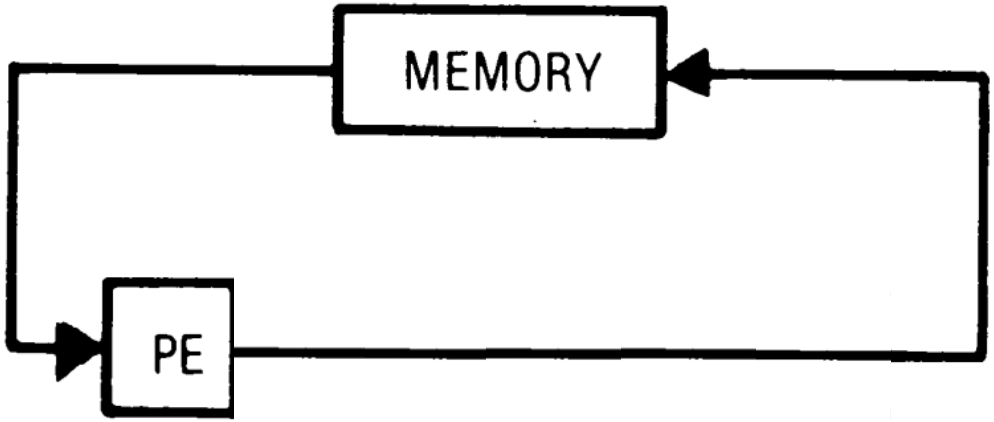
- I/O bandwidth: 10 MB/s
- Each operation uses 2 bytes

- At most 5 million operations per second

The systolic approach

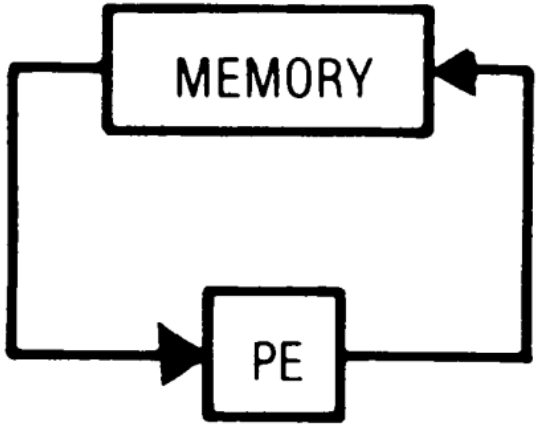
- Same conditions as before
  - Up to 6x improvements

- Systolic:
  - The memory "**pumps**" data to the **processing elements**
  - Like the heart **pumps** blood to the **body cells**



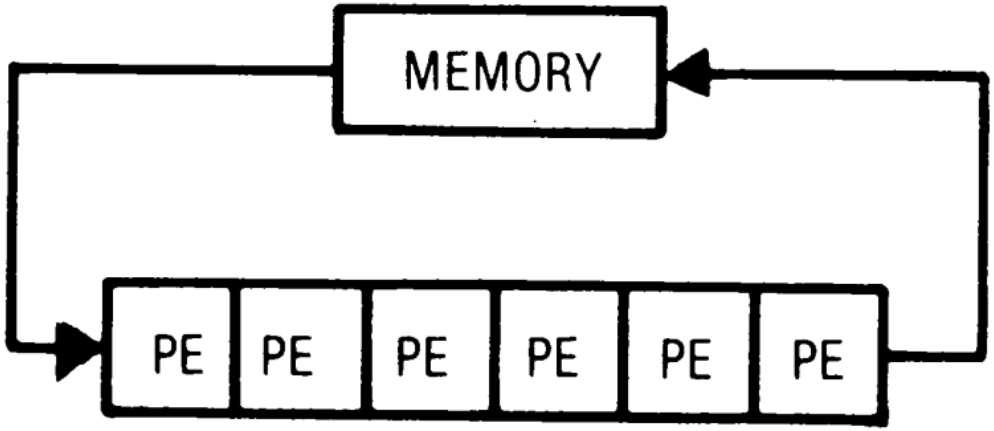
Both approaches visualized

INSTEAD OF:



5 MILLION
OPERATIONS
PER SECOND
AT MOST

WE HAVE:



30 MOPS
POSSIBLE

THE SYSTOLIC ARRAY

Key Approach and Ideas

The structure of a systolic architecture

- A systolic architecture is composed of multiple processing elements (**cells**)
- Only cells at the **boundary** can be I/O ports of the system
- Partial results and inputs flow **inside** the system
- Cells are **interconnected** to form simple and regular structures:
  - Trees
  - Arrays
  - Grids

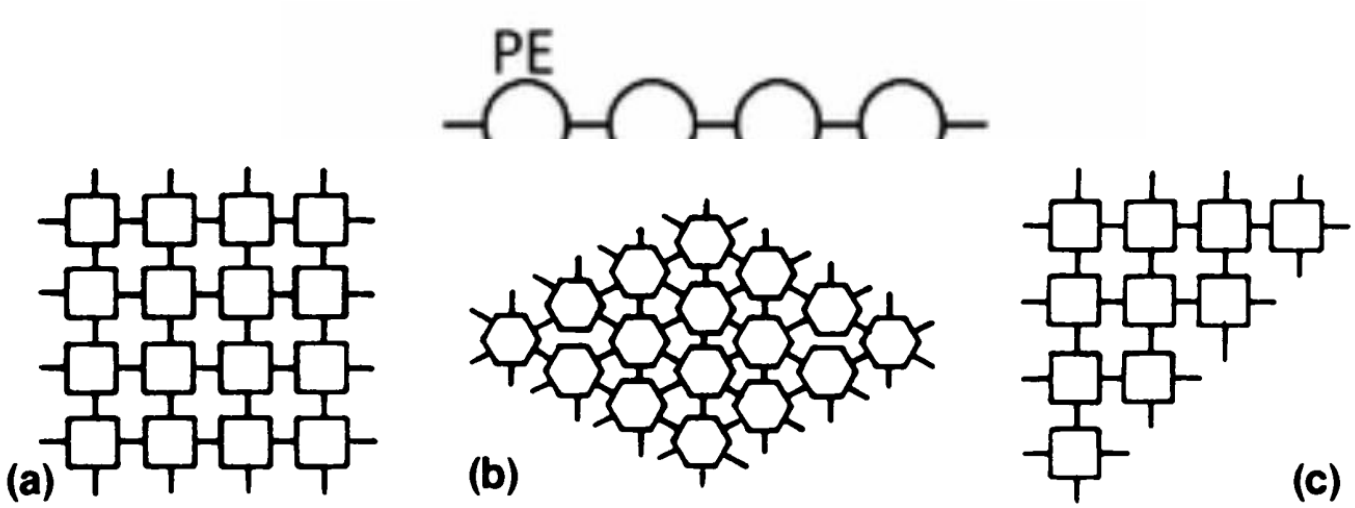


Image source: Sano K., Nakahara H. (2018) Hardware Algorithms.
In: Amano H. (eds) Principles and Structures of FPGAs. Springer, Singapore

Mechanisms

Problems solvable by systolic architectures

- A sample of problems with known systolic solution:
  - Signal and image processing:
    - **Convolution**
    - Discrete Fourier transform
    - Interpolation
  - Matrix arithmetic:
    - Matrix multiplication
    - QR decomposition of matrixes
    - Linear systems of equation
  - Non-numeric applications:
    - Regular expressions
    - Dynamic programming
    - Encoders (polynomial division)

An exemplar compute-bound problem

- The convolution problem
- Given:
  - The sequence of weights $\{w_1, w_2, \ldots, w_k\}$
  - The sequence of inputs $\{x_1, x_2, \ldots, x_n\}$
- Compute:
  - The sequence $\{y_1, y_2, \ldots, y_{n+1-k}\}$
  - Defined by

$$y_i = w_1 x_i + w_2 x_{i+1} + \ldots w_k x_{i+k-1}$$

- This problem is regular and **compute-bound**
- There are many related problems, e.g. pattern matching

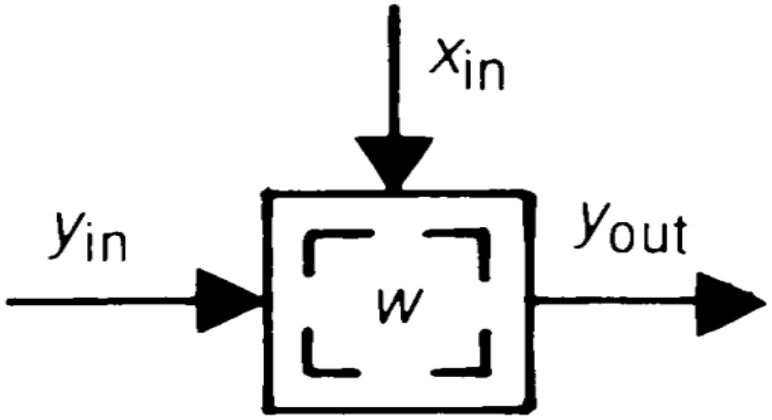
Example convolution problem instance

- Given:
  - The sequence of weights: {2, 1, 4}
  - The sequence of inputs: {5, 0, -7, 3, 1}
- The output sequence $\{y_1, y_2, y_3\}$ *is computed as follows*
  - $y_1 = 2*5 + 1*0 + 4*(-7) = -18$
  - $y_2 = 2*0 + 1*(-7) + 4*3 = 5$
  - $y_3 = 2*(-7) + 1*3 + 4*1 = -7$

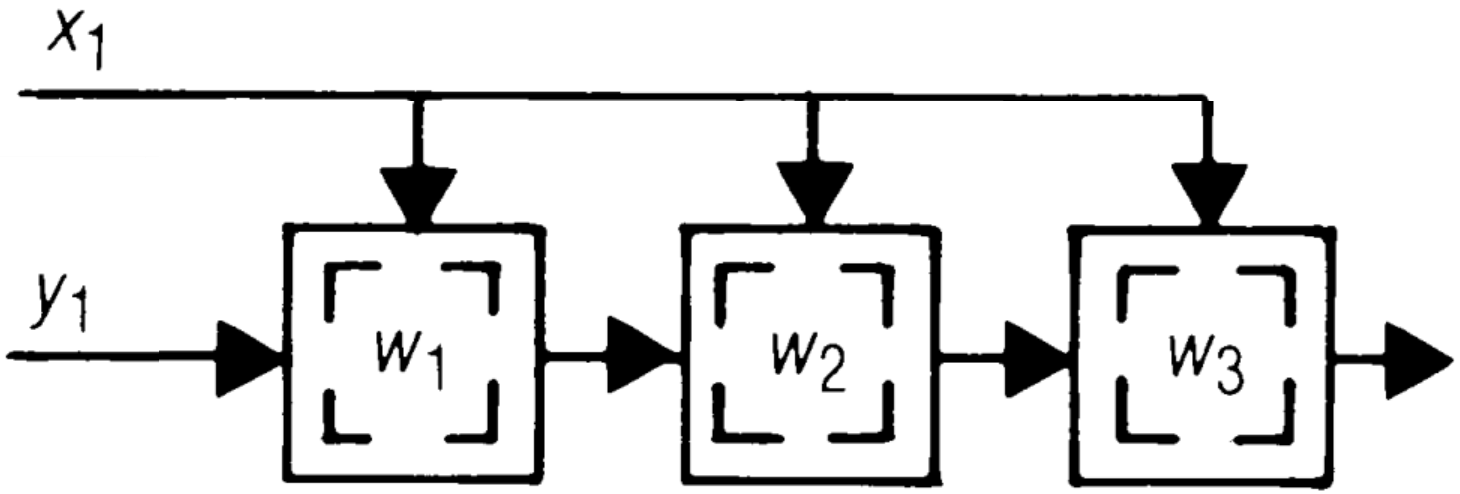
The proposed designs

- Three different systolic systems will be presented:
  1. **Broadcast**: A semi-systolic solution where the input sequence is broadcast to the cells
  2. **Low-latency**: A pure systolic solution with low output latency
  3. **High-throughput**: A pure systolic solution where no cell is idle during usage

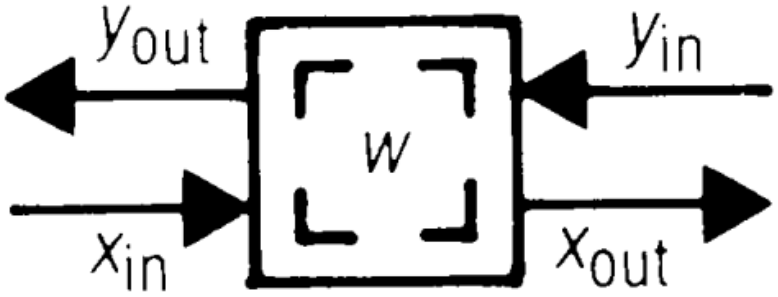
1. Broadcast



$$y_{out} \leftarrow y_{in} + w \cdot x_{in}$$

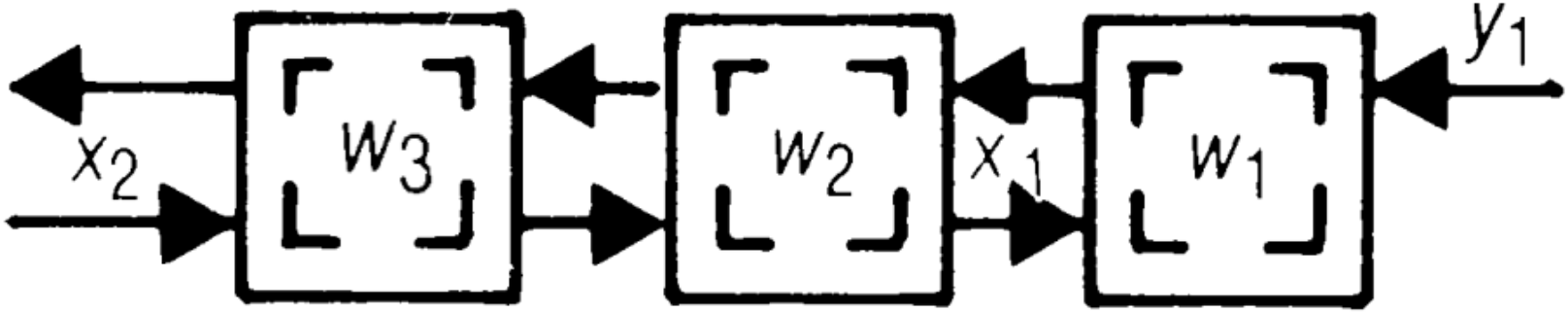


2. Low-latency

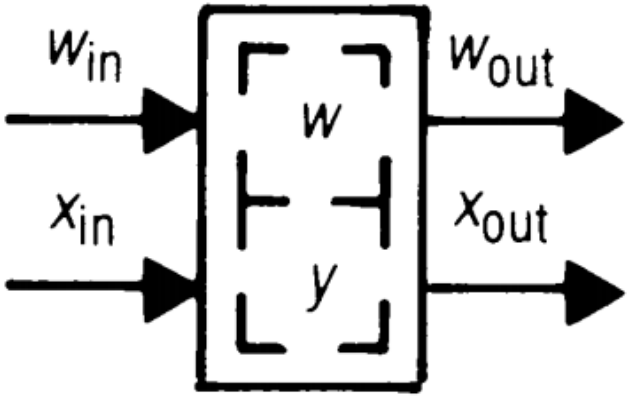


$$x_{out} \leftarrow x_{in}$$

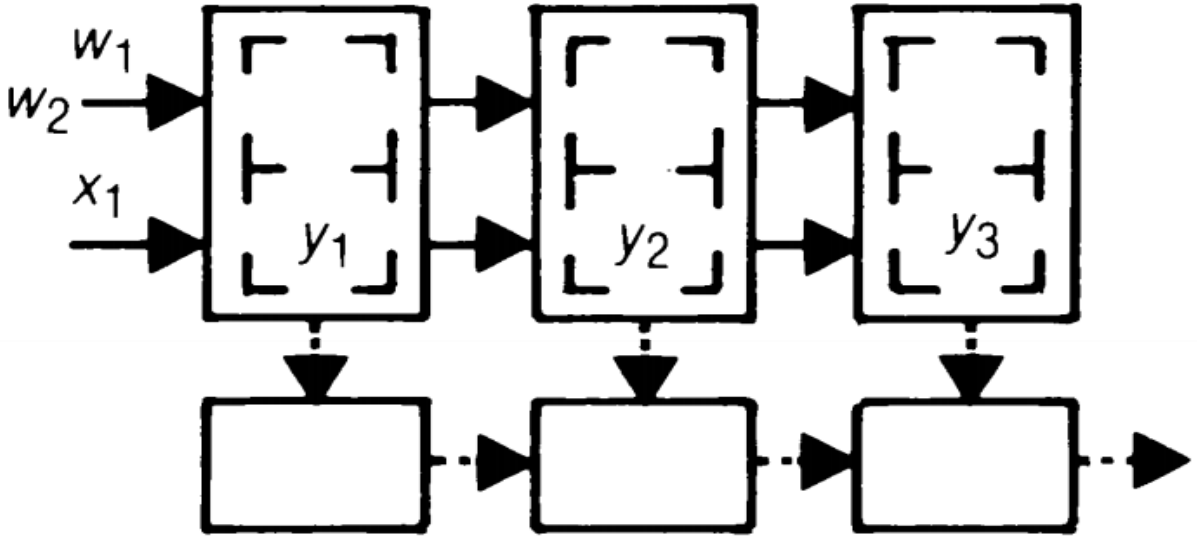
$$y_{out} \leftarrow y_{in} + w \cdot x_{in}$$



3. High-throughput



$$w \leftarrow w_{in}$$
$$w_{out} \leftarrow w$$
$$x_{out} \leftarrow x_{in}$$
$$y \leftarrow y + w_{in} \cdot x_{in}$$



Comparison of the designs

Nr	Design	Advantages	Disadvantages
1	Broadcast	• **Simplest** design • Cells use only 3 I/O ports	• Does **NOT** scale well
2	Low-latency	• Simplest **pure systolic** design	• Only **half** of the cells are used at any given time
3	High-throughput	• Works with **unbounded** amount of weights • The partial results stay in the cells*	• Requires a **bus** to collect results • More **complex** than 1 and 2 • Response time depends on the number of weights • Requires more I/O

*Partial results often carry more bits because of numerical accuracy

Key Results: Methodology and Evaluation

Key properties of systolic architectures

- Criteria of systolic designs and their effects:
  - They have **simple** and **regular** control flow
    - Simplicity, modularity, expandability, and high performance
  - They only use a few type of **simple cells**
    - Simplicity
  - They use each **input** data item **multiple times**
    - High performance
  - They are **highly concurrent** by design
    - High performance
- Highly **scalable**
  - Performance increases proportionally with number of cells

Summary

Summary

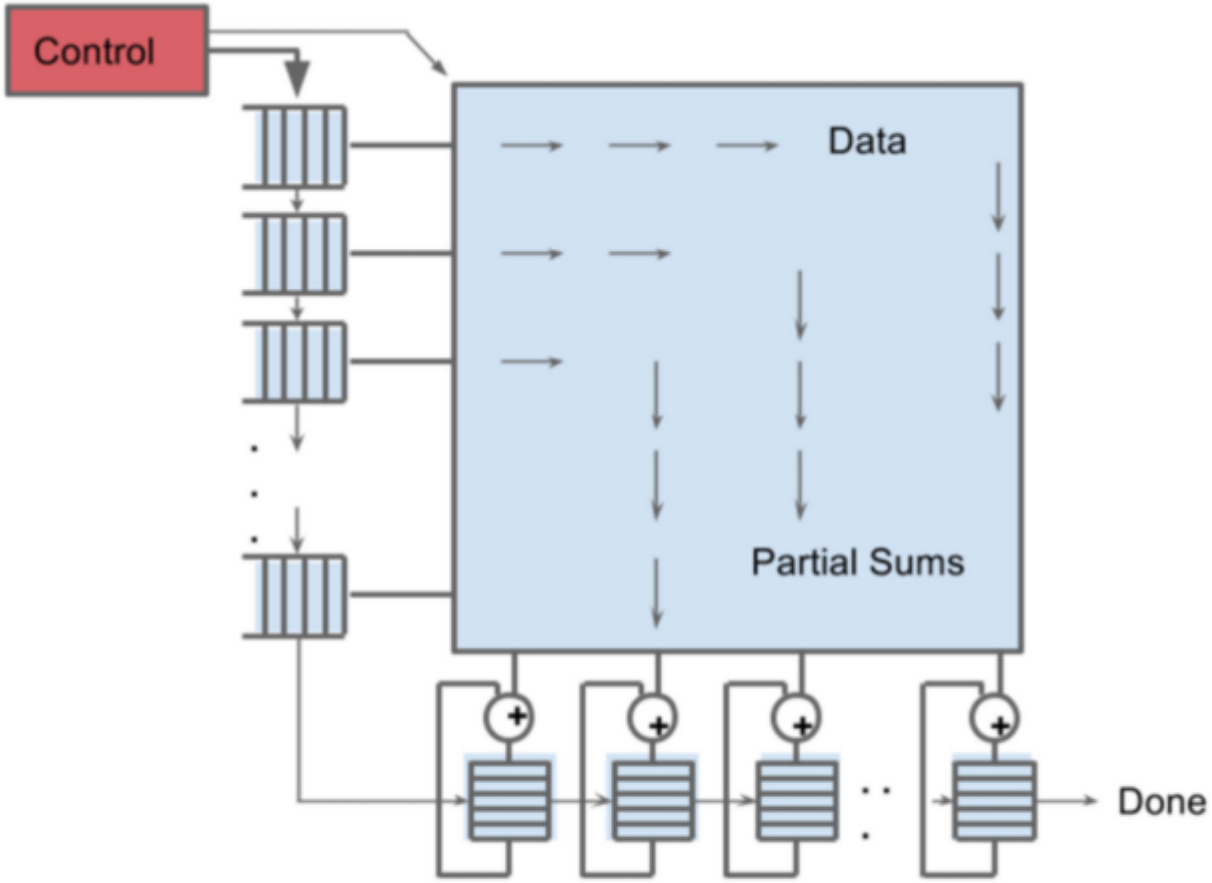
- Special-purpose system often
  - Have high design cost
  - Are designed ad hoc
  - Repeat known errors
- Systolic systems
  - Are simple and easy to design
  - Avoid the pitfalls of special-purpose systems designs
  - Modular, expandable, and high performance
  - Are applicable to many (if not all) problems where it makes sense to build special-purpose systems
- Systolic systems geared to different applications can be obtained with little effort

Strengths

Strengths of the paper

- **Intuitive** idea
- Well structured paper, with good flow
- Many different examples of systolic systems are presented
  - The tradeoffs between different designs are discussed
- General approach to common problems
  - Many compute-bound problems have a systolic solution
- **Scalable** and **adaptive** designs
  - Adaptable to different I/O bandwidth and problem size
- The paper is still relevant today! (36 years after)
  - More than 3000 citations, ~40 citations/year since 2000
  - **Google's TPU** is a systolic system at its heart

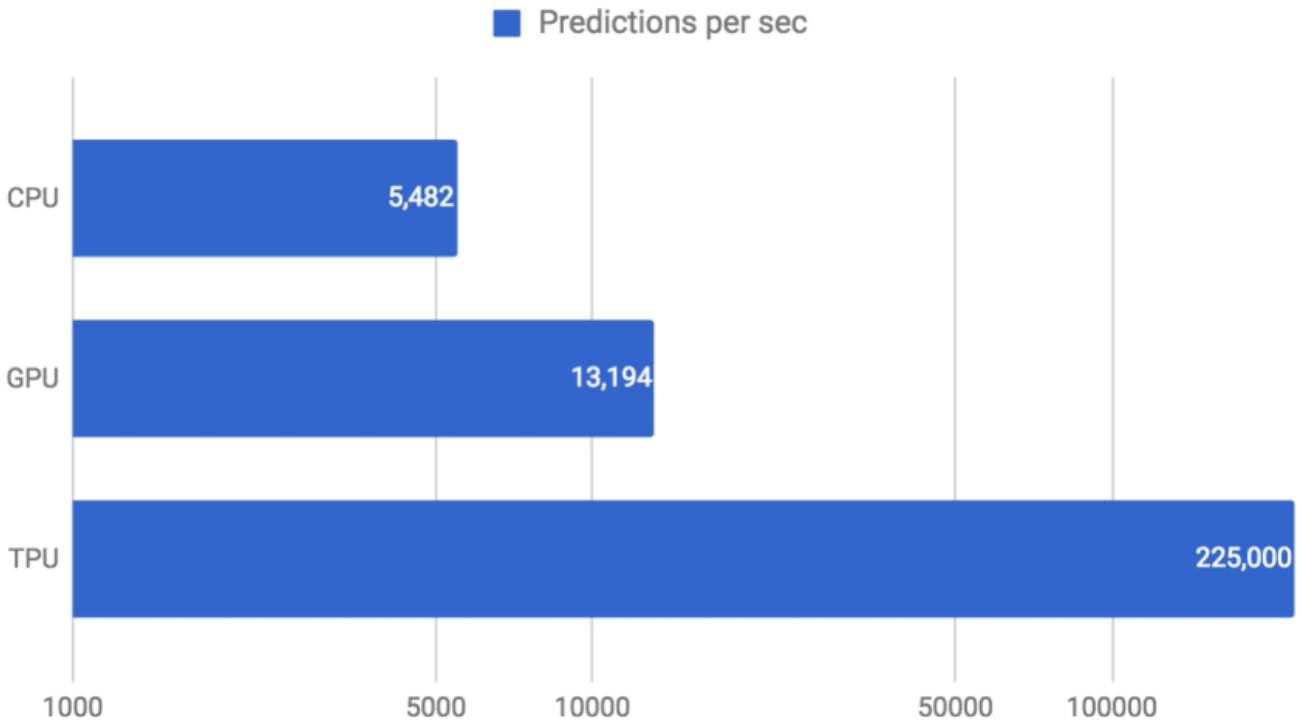
The heart of Google's first TPU



Matrix Multiplier Unit (MXU) of TPU

Source: Google product news (17.5.2017)

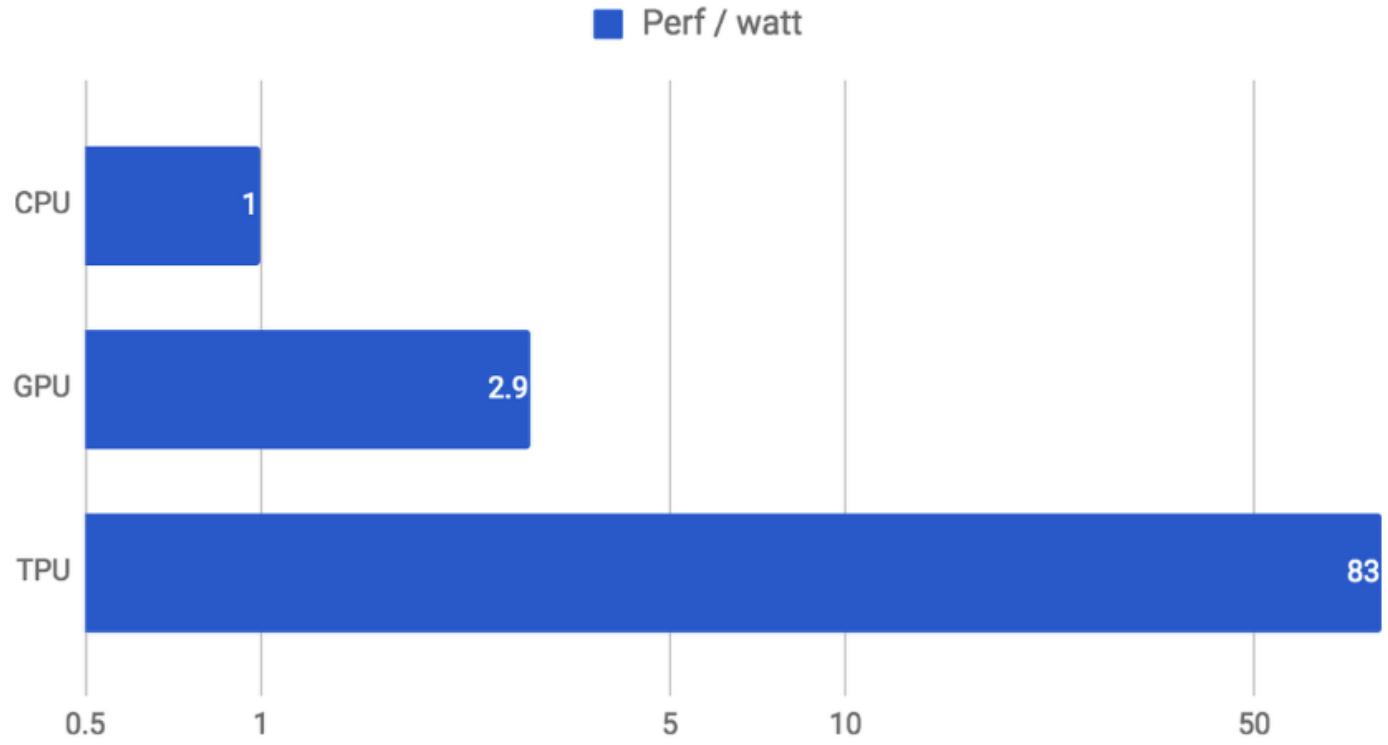
Performance of Google's first TPU



Throughput under 7 ms latency limit (in log scale)(99th% response with MLP0: CPU = 7.2 ms, GPU = 6.7 ms, TPU = 7.0 ms)

Source: Google product news (17.5.2017)

Performance of Google's first TPU



Performance / watt, relative to contemporary CPUs and GPUs (in log scale)(Incremental, weighted mean)

Source: Google product news (17.5.2017)

Weaknesses

Weaknesses of the paper

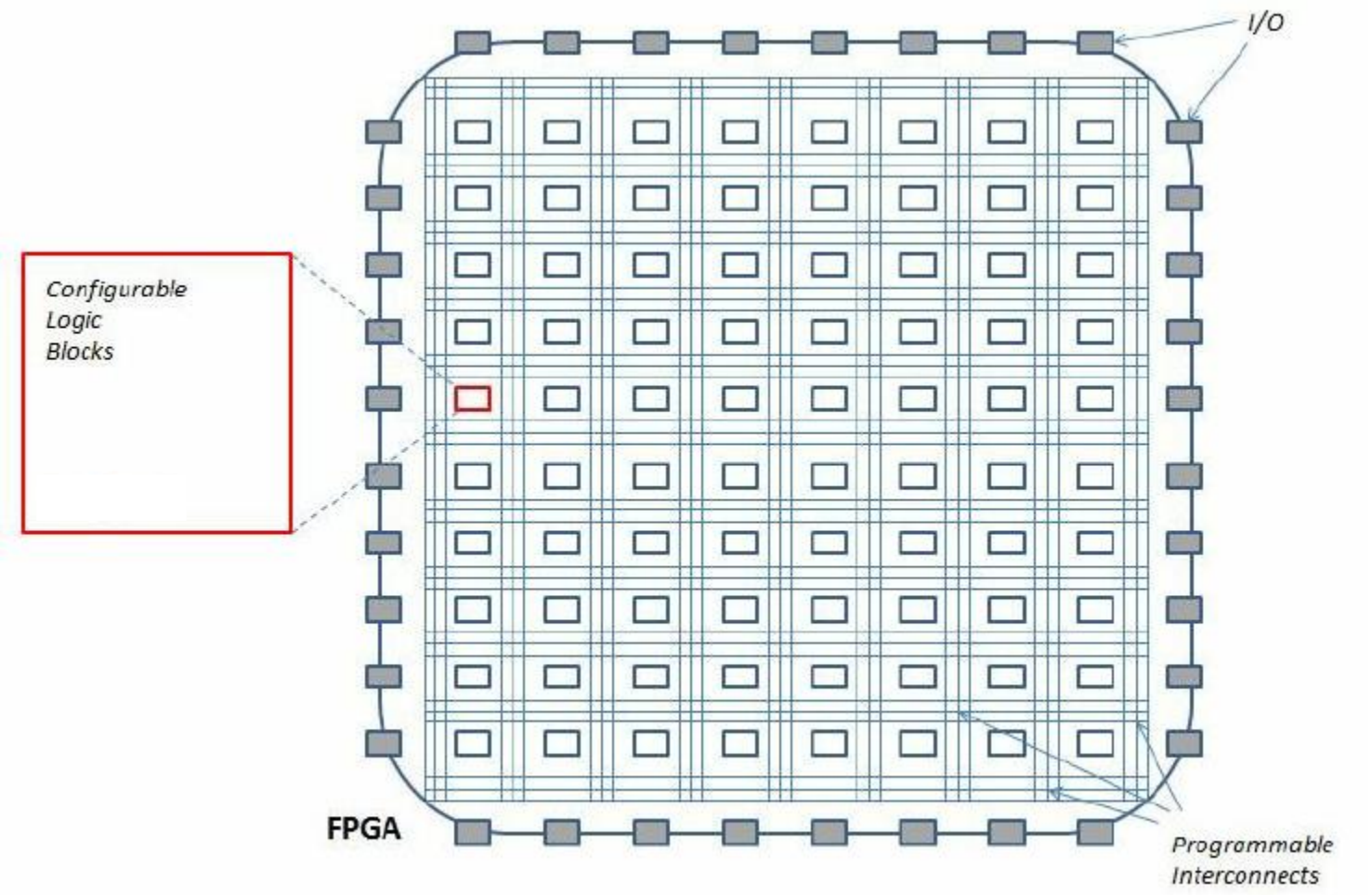
- No data to support the claim that systolic architectures are a viable alternative to ad hoc architectures
  - Design time
  - Energy efficiency
  - Performance
- Approach still limited by I/O bottleneck
  - In-memory accelerators don't share the same bottleneck
- It's difficult to design systolic systems for compute-bound problems which aren't inherently regular
  - Sparse matrix multiplication
- Difficult to **debug**
  - Partial result aren't exposed to the programmer

Thoughts and Ideas

Thoughts and ideas

- Can the design of systolic architectures be automated?
  - Still an open problem, but some can be designed automatically
- How can systolic architectures be specified and verified without building prototypes?
  - Are there simulation frameworks for systolic architectures?
- Systolic architectures map well to FPGAs

Simplified FPGA schematic



Source: David Norwood's master thesis

Thoughts and ideas

- Are there compute-bound problems with no systolic solution?
- Are there alternatives to systolic systems?
  - GPU, in-memory accelerators, …
- Can there be general-purpose systolic structures?
  - Yes, iWarp [1990, CMU & Intel]

A view of the iWarp

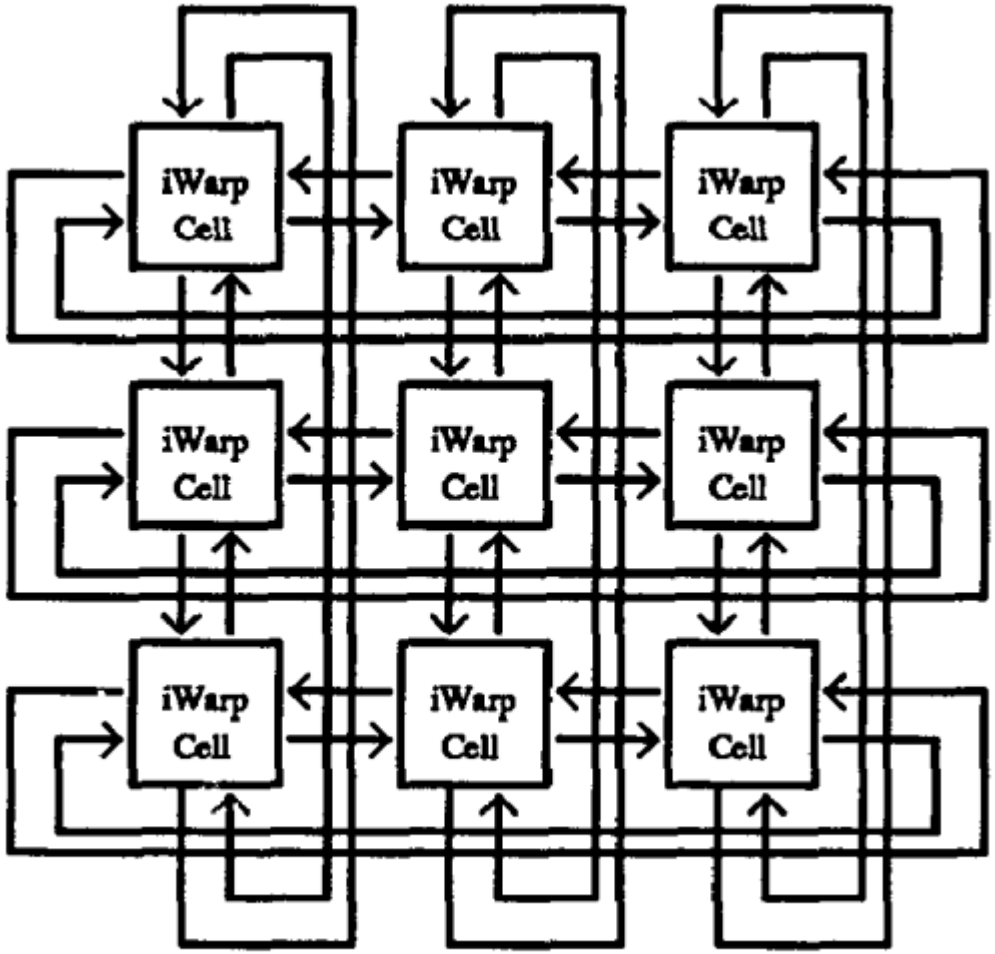
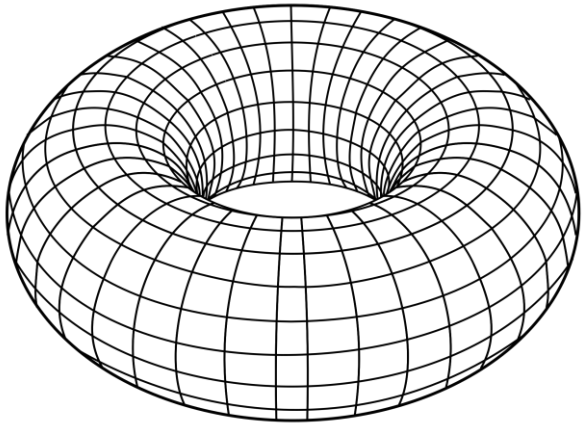


Figure 5. 3×3 torus

"The initial demonstration iWarp system in 1990 is an 8x8 torus"



Prof. Thomas Gross (ETHZ) participated in the design of the iWarp

Takeaways

Key takeaways

- General guideline to simple, efficient, and scalable designs
- Principled approach to the design of special-purpose systems
- Avoid designing ad hoc systems when possible
  - Avoid known pitfalls
- Less successful ideas may have an impact in the future
  - See Google's TPU

Open Discussion