Digital Design & Computer Arch.

Lecture 20: SIMD Processors

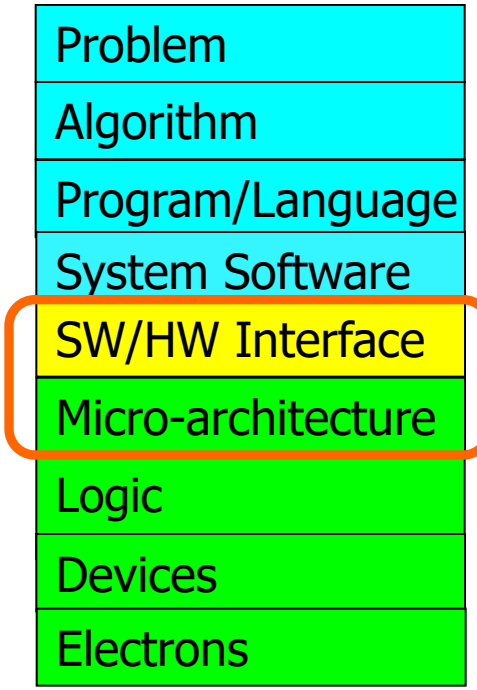
Dr. Juan Gómez Luna
Prof. Onur Mutlu
ETH Zürich

Spring 2022

12 May 2022

Other Execution Paradigms

- Dataflow (at the ISA level)
- Superscalar Execution
- VLIW
- Systolic Arrays
- Decoupled Access Execute
- SIMD Processing (Vector and Array processors)
- Graphics Processing Units (GPUs)



Readings for this Week

- **Required**
  - Lindholm et al., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," IEEE Micro 2008.

- **Recommended**
  - Peleg and Weiser, "MMX Technology Extension to the Intel Architecture," IEEE Micro 1996.

Exploiting Data Parallelism: SIMD Processors and GPUs

SIMD Processing:
Exploiting Regular (Data) Parallelism

Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

Very High-Speed Computing Systems

MICHAEL J. FLYNN, MEMBER, IEEE

Abstract—Very high-speed computers may be classified as follows:

1) Single Instruction Stream–Single Data Stream (SISD)
2) Single Instruction Stream–Multiple Data Stream (SIMD)
3) Multiple Instruction Stream–Single Data Stream (MISD)
4) Multiple Instruction Stream–Multiple Data Stream (MIMD).

"Stream," as used here, refers to the sequence of data or instructions as seen by the machine during the execution of a program.

The constituents of a system: storage, execution, and instruction handling (branching) are discussed with regard to recent developments and/or systems limitations. The constituents are discussed in terms of concurrent SISD systems (CDC 6600 series and, in particular, IBM Model 90 series), since multiple stream organizations usually do not require any more elaborate components.

Representative organizations are selected from each class and the arrangement of the constituents is shown.

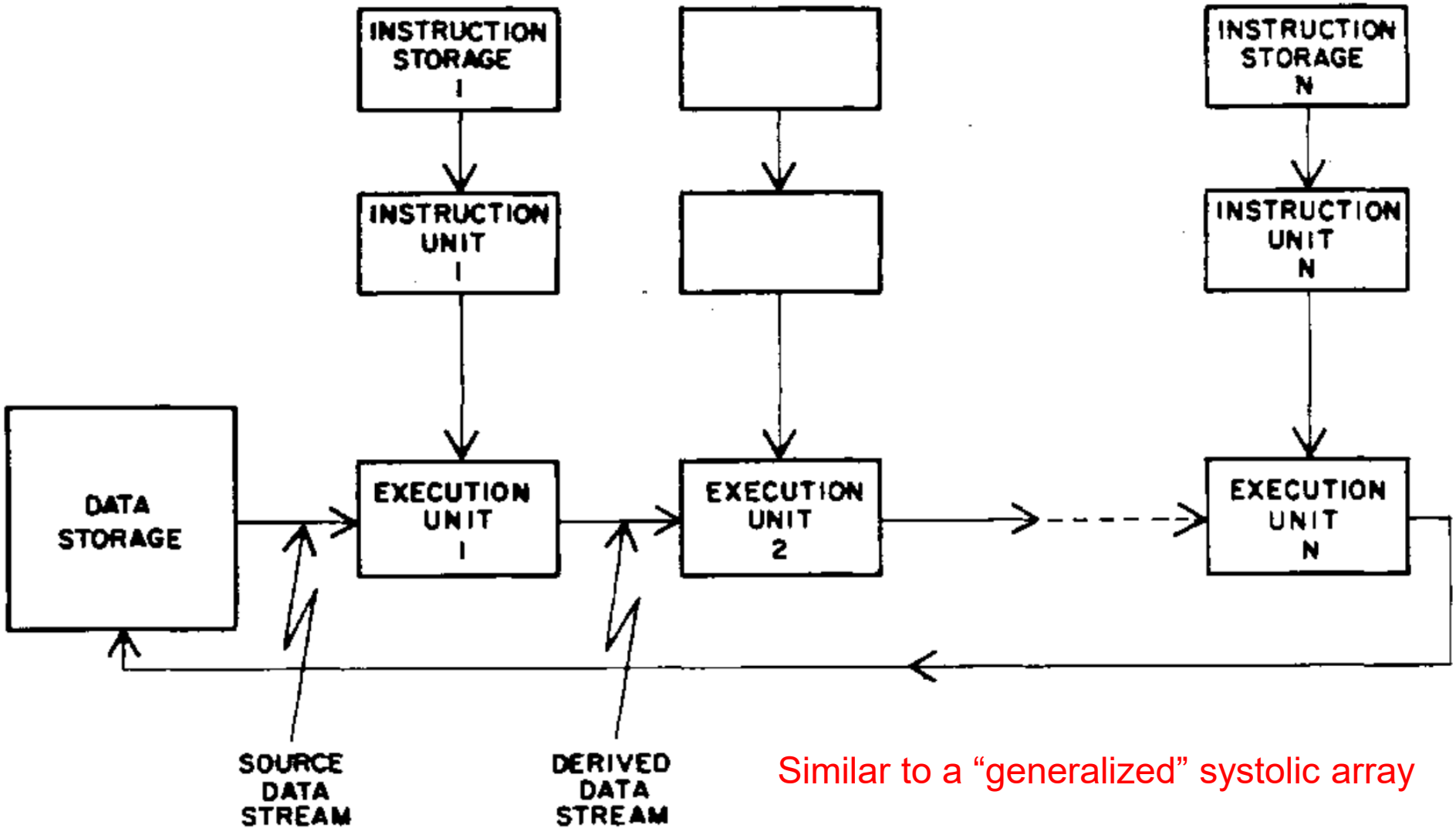
Manuscript received June 30, 1966; revised August 16, 1966. This work was performed under the auspices of the U. S. Atomic Energy Commission.

The author is with Northwestern University, Evanston, Ill., and Argonne National Laboratory, Argonne, Ill.

INTRODUCTION

MANY SIGNIFICANT scientific problems require the use of prodigious amounts of computing time. In order to handle these problems adequately, the large-scale scientific computer has been developed. This computer addresses itself to a class of problems characterized by having a high ratio of computing requirement to input/output requirements (a partially de facto situation

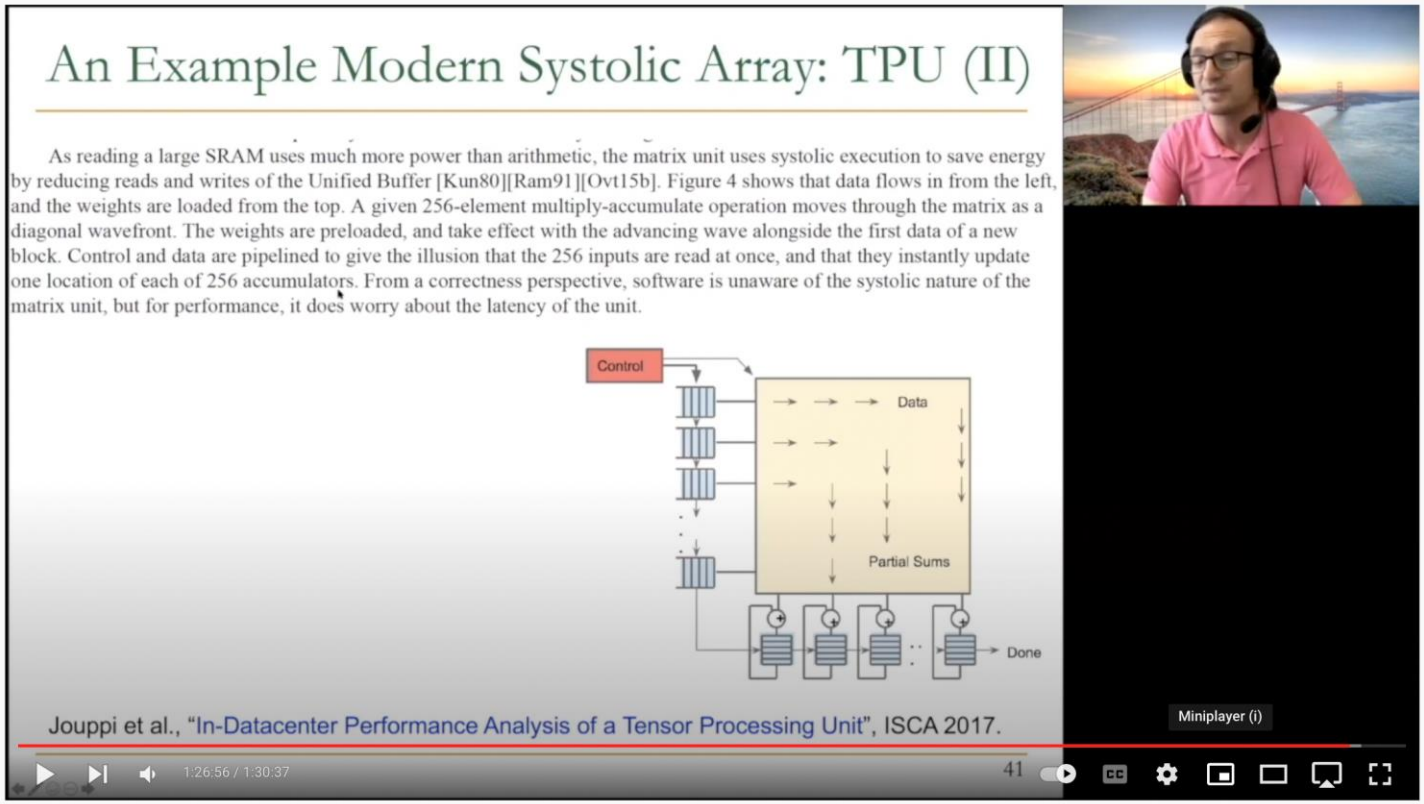
MISD Example from Flynn



Similar to a "generalized" systolic array

Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

Lecture 19b: Systolic Array Architectures



Digital Design & Computer Arch. - Lecture 19: VLIW and Systolic Array Architectures (Spring 2022)

842 views • Premiered May 6, 2022

35 DISLIKE SHARE CLIP SAVE ...

Onur Mutlu Lectures
24.5K subscribers

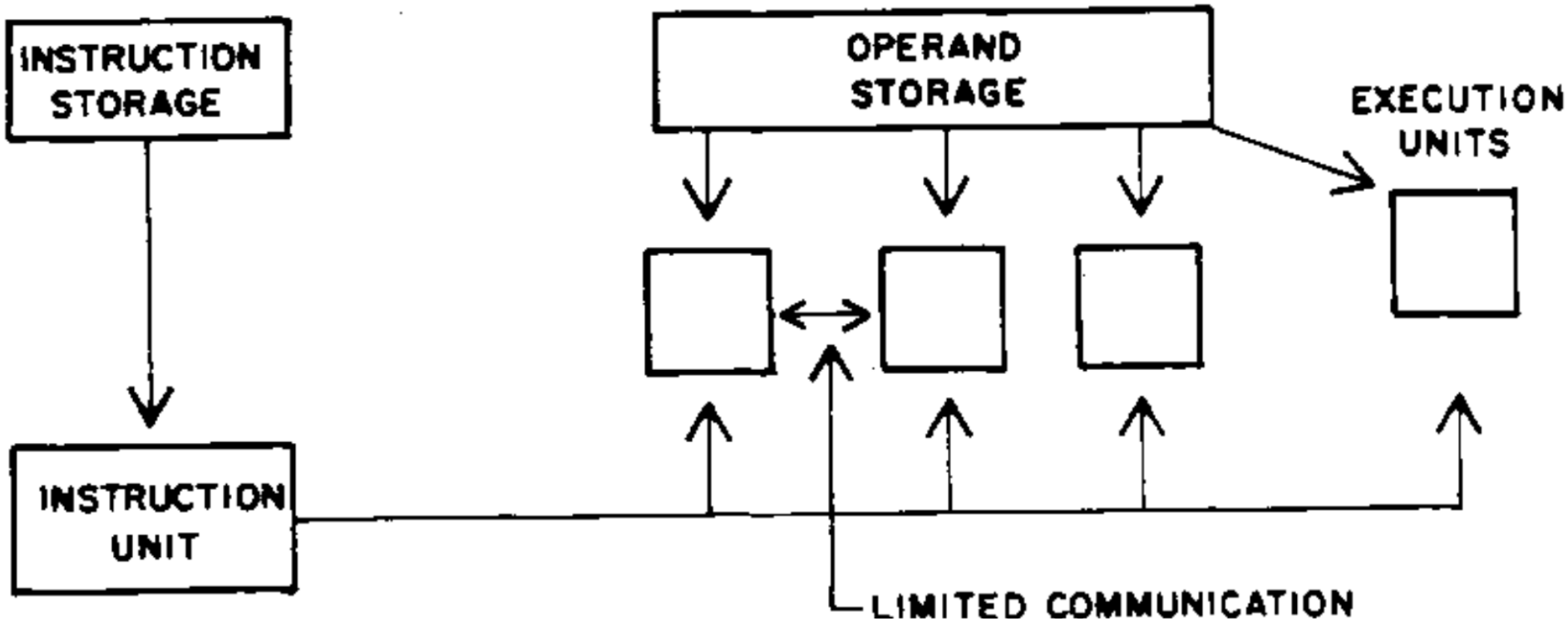
SUBSCRIBED

Digital Design and Computer Architecture, ETH Zürich, Spring 2022 (
https://safari.ethz.ch/digitaltechnik...)

Lecture 19a: VLIW Architectures
Lecture 19b: Systolic Array Architectures
Lecturer: Professor Onur Mutlu (https://people.inf.ethz.ch/omutlu/)
Date: May 6, 2022

https://youtu.be/1SSqV7Y75oU?t=2316

SIMD Example from Flynn



Similar to an "array processor"

Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

Data Parallelism

- Concurrency arises from performing the same operation on different pieces of data
  - Single instruction multiple data (SIMD)
  - E.g., dot product of two vectors
- Contrast with data flow
  - Concurrency arises from executing different operations in parallel (in a data driven manner)
- Contrast with thread ("control") parallelism
  - Concurrency arises from executing different threads of control in parallel
- SIMD exploits operation-level parallelism on different data
  - Same operation concurrently applied to different pieces of data
  - A form of ILP where instruction happens to be the same across data

SIMD Processing

- Single instruction operates on multiple data elements
  - In time or in space
- Multiple processing elements (PEs), i.e., execution units

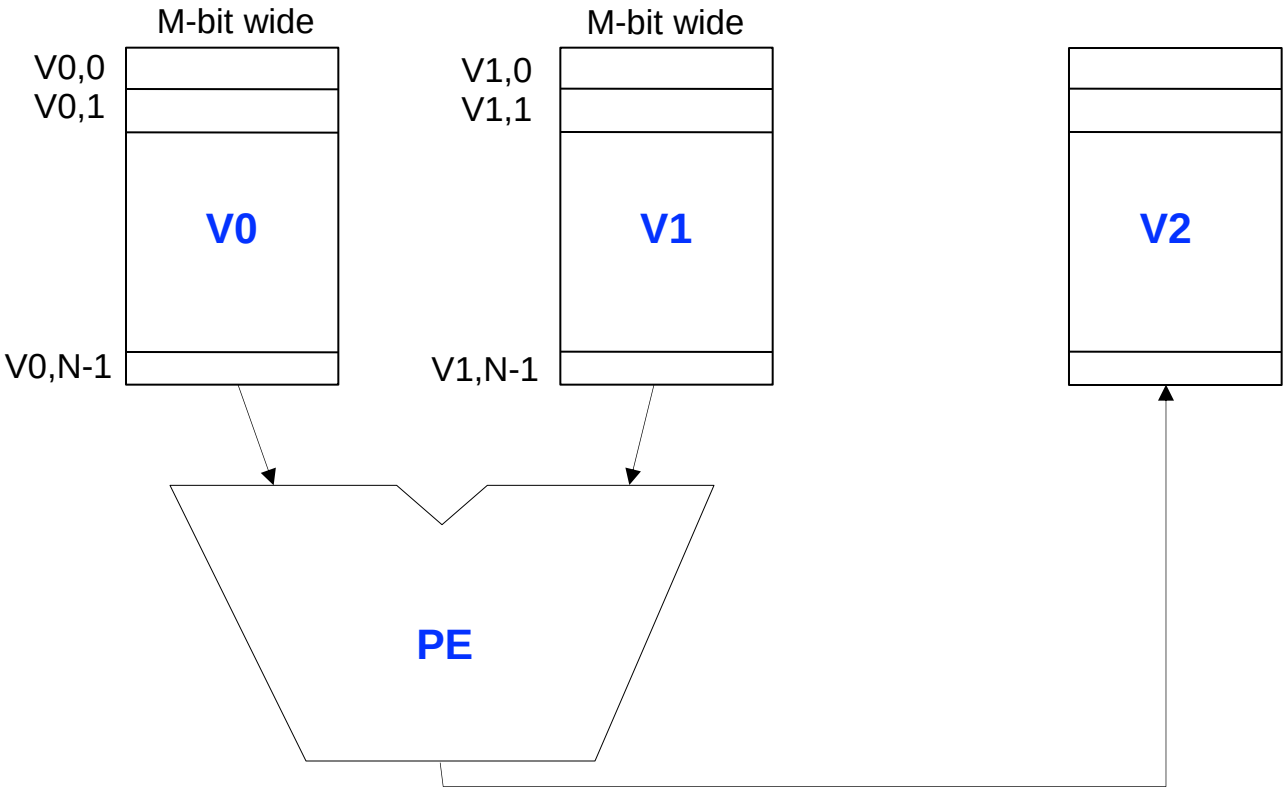
- Time-space duality

  - Array processor: Instruction operates on multiple data elements at the same time using different spaces (PEs)

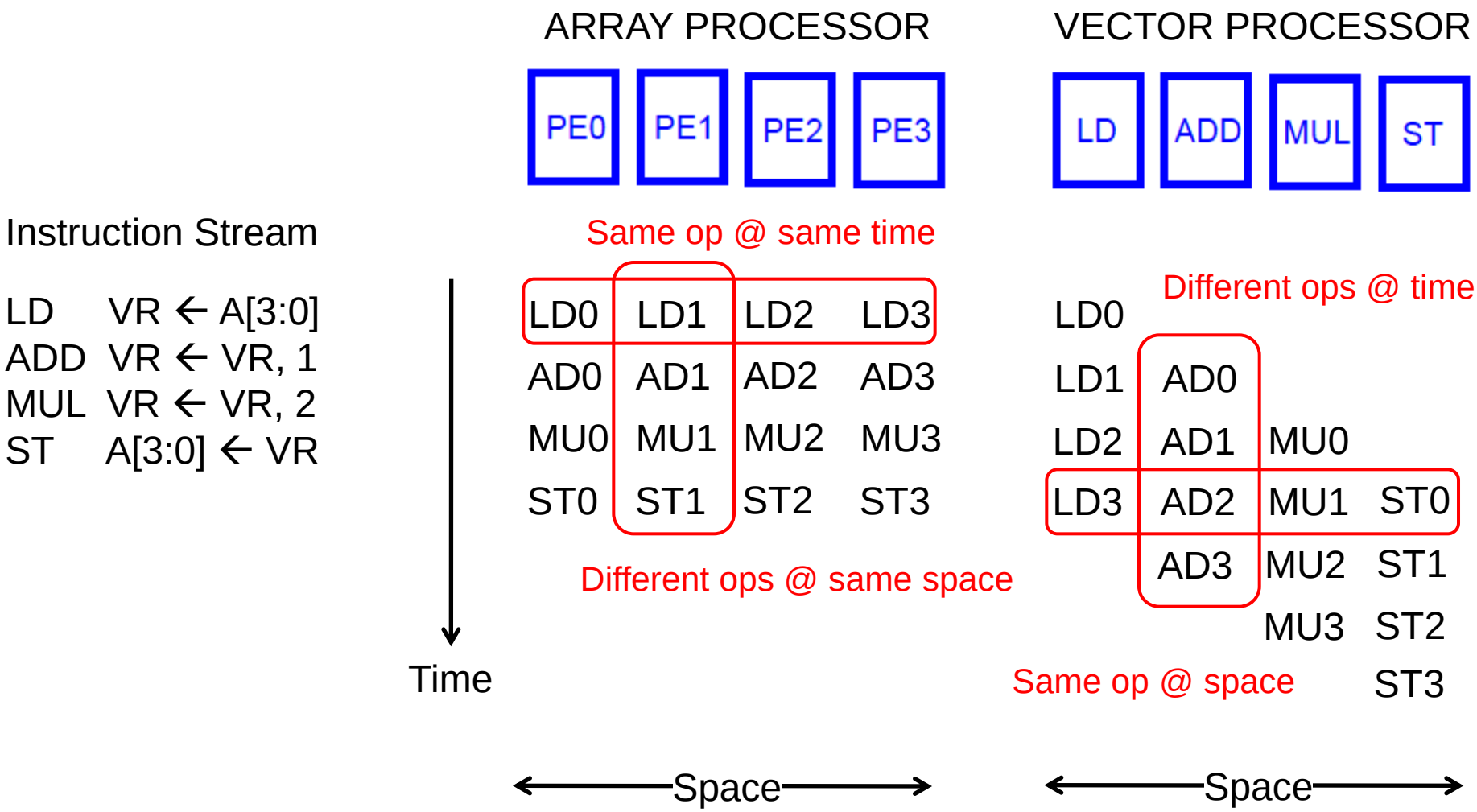
  - Vector processor: Instruction operates on multiple data elements in consecutive time steps using the same space (PE)

Storing Multiple Data Elements: Vector Registers

- Each vector data register holds N M-bit values
  - Each register stores a vector
  - Not a (single) scalar value as we saw before

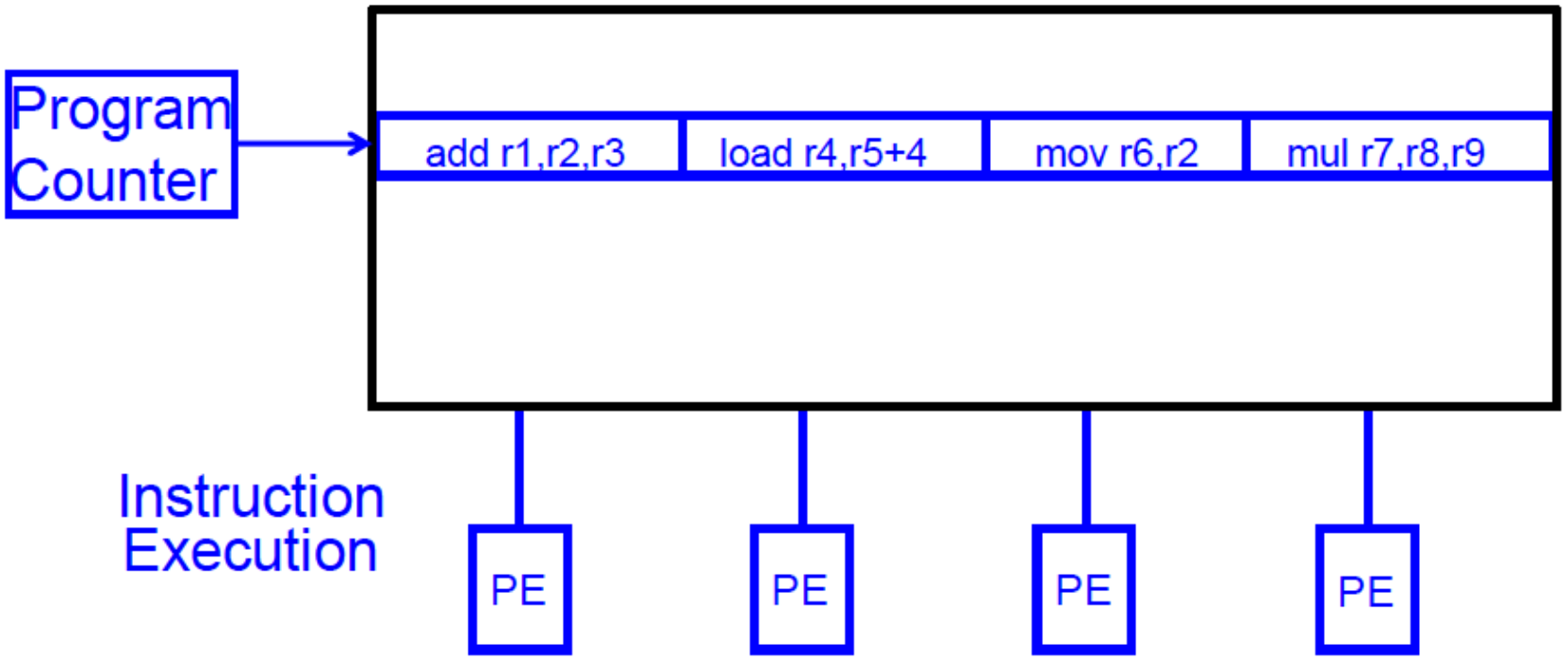


Array vs. Vector Processors



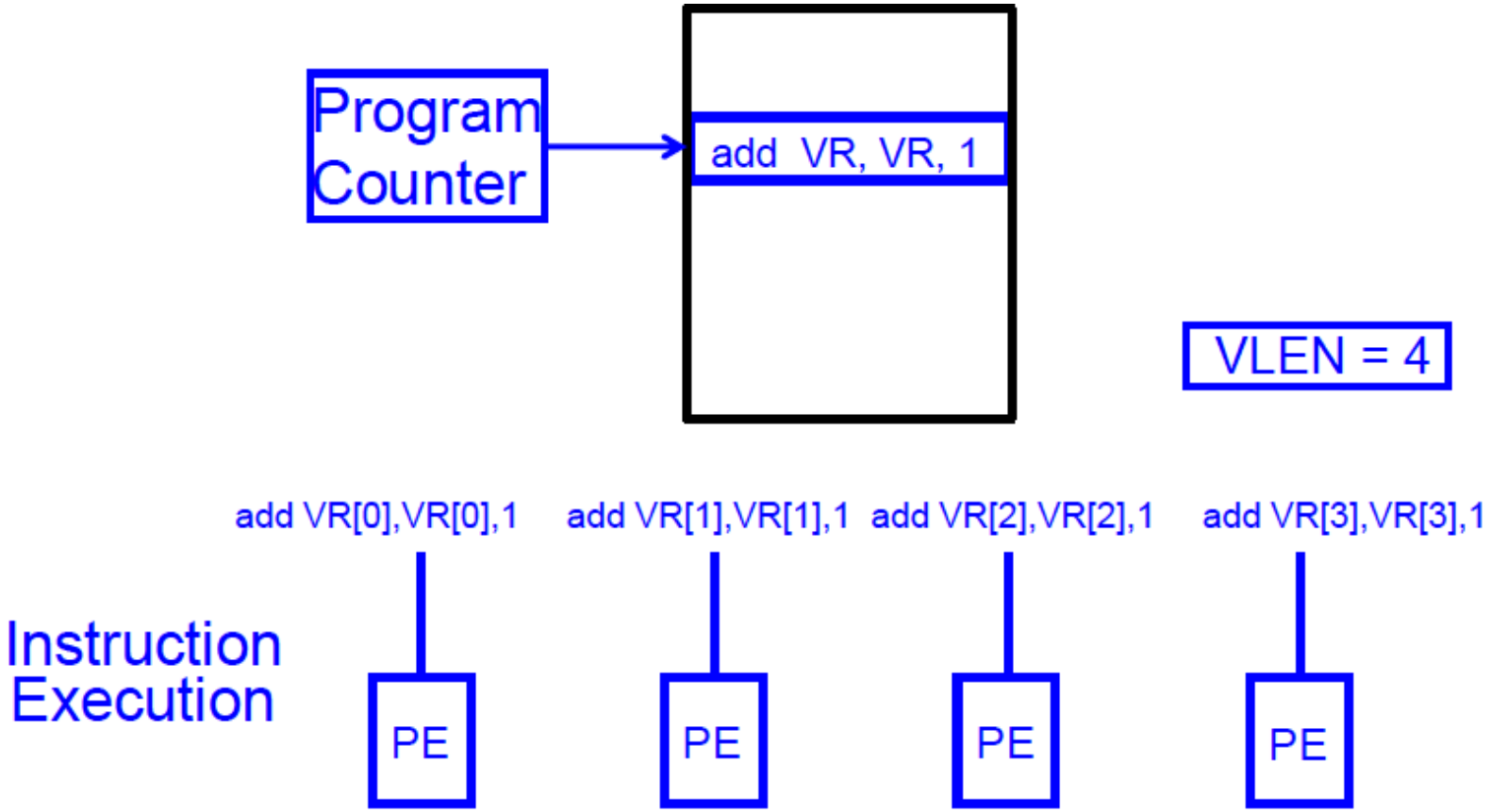
SIMD Array Processing vs. **VLIW**

- VLIW: **Multiple** independent **operations** packed together into a "long inst."

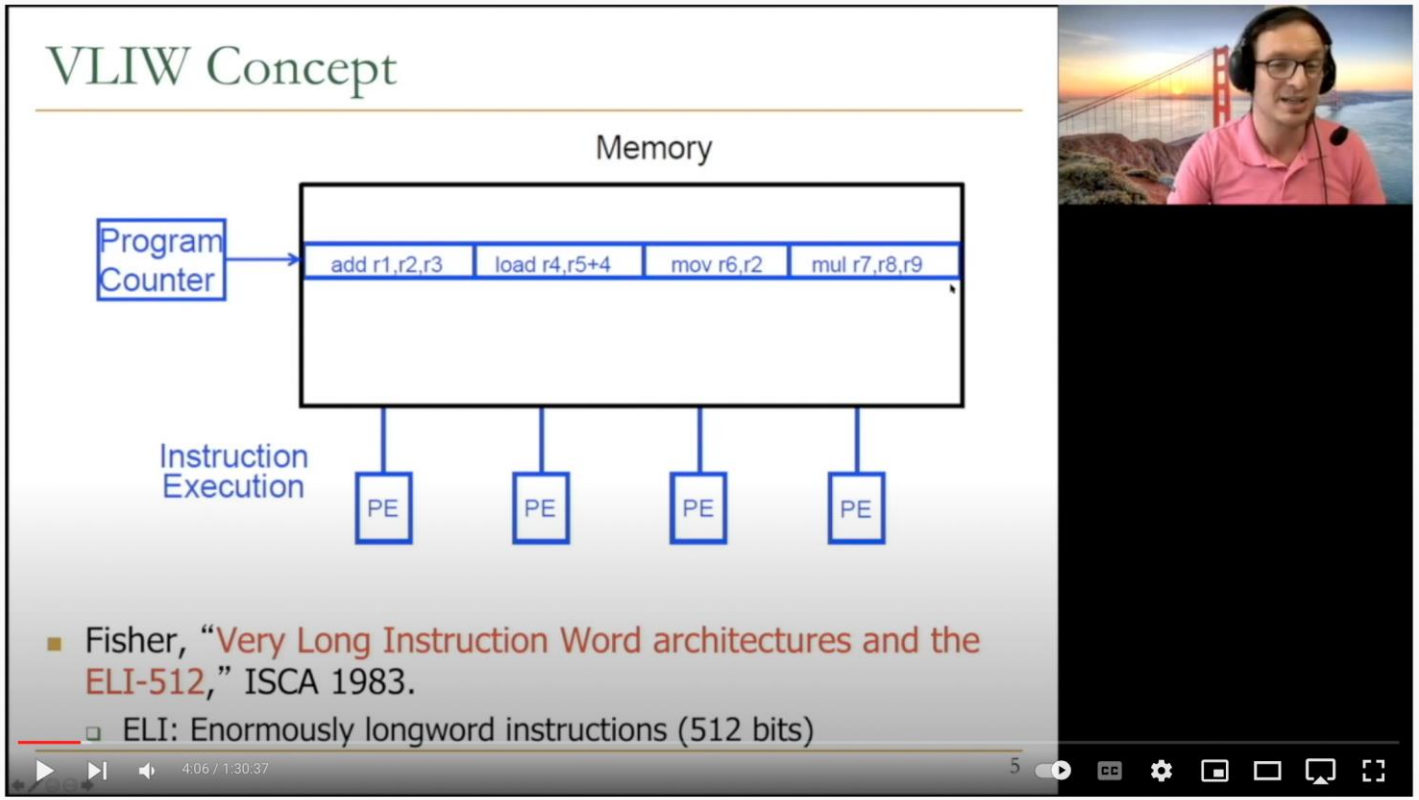


SIMD Array Processing vs. VLIW

- Array processor: **Single operation** on multiple (different) data elements



Lecture 19a: VLIW Architectures



Digital Design & Computer Arch. - Lecture 19: VLIW and Systolic Array Architectures (Spring 2022)

842 views • Premiered May 6, 2022

35 DISLIKE SHARE CLIP SAVE ...

Onur Mutlu Lectures
24.5K subscribers

SUBSCRIBED

Digital Design and Computer Architecture, ETH Zürich, Spring 2022 (
https://safari.ethz.ch/digitaltechnik...)

Lecture 19a: VLIW Architectures
Lecture 19b: Systolic Array Architectures
Lecturer: Professor Onur Mutlu (https://people.inf.ethz.ch/omutlu/)
Date: May 6, 2022

https://youtu.be/1SSqV7Y75oU

Vector Processors (I)

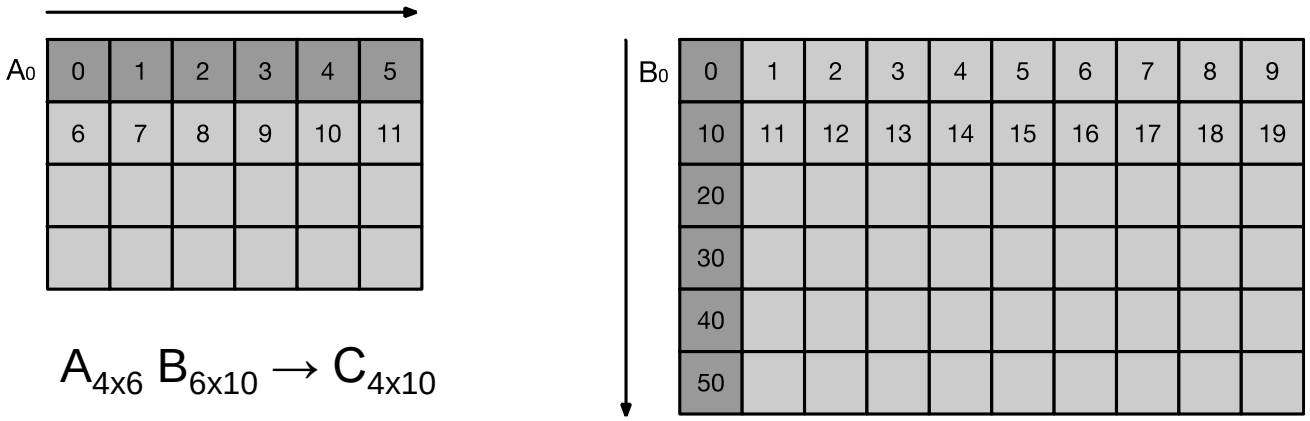
- A vector is a one-dimensional array of numbers
- Many scientific/commercial programs use vectors

```
for (i = 0; i<=49; i++)
    C[i] = (A[i] + B[i]) / 2
```

- A vector processor is one whose instructions operate on vectors rather than scalar (single data) values
- Basic requirements
  - Need to load/store vectors → vector registers (contain vectors)
  - Need to operate on vectors of different lengths → vector length register (VLEN)
  - Elements of a vector might be stored apart from each other in memory → vector stride register (VSTR)
    - Stride: distance in memory between two elements of a vector

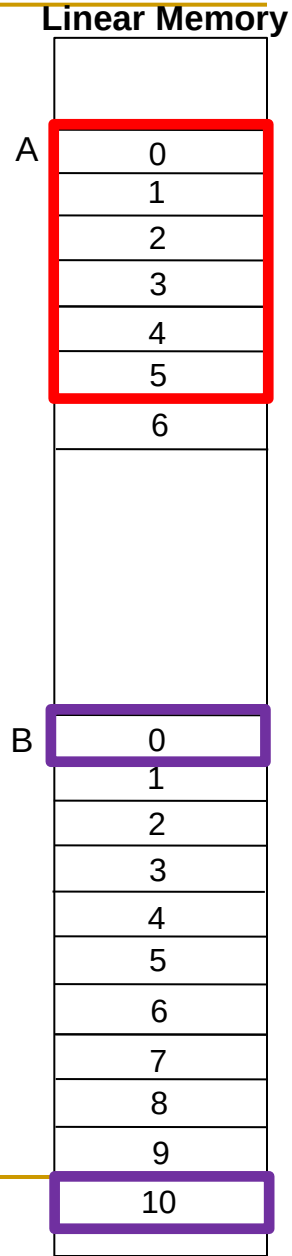
Vector Stride Example: Matrix Multiply

- A and B matrices, both stored in memory in row-major order



$A_{4x6}\ B_{6x10} \rightarrow C_{4x10}$

Dot product of each row vector of A with each column vector of B



- Load A's row 0 (A_{00} through A_{05}) into vector register V_1
  - Each time, increment address by 1 to access the next column
  - Accesses have a stride of 1
- Load B's column 0 (B_{00} through B_{50}) into vector register V_2
  - Each time, increment address by 10 to access the next row
  - Accesses have a stride of 10

Vector Processors (II)

- A vector instruction performs an operation on each element in consecutive cycles
  - Vector functional units are pipelined
  - Each pipeline stage operates on a different data element

- Vector instructions allow deeper pipelines
  - No intra-vector dependencies → no hardware interlocking needed within a vector
  - No control flow within a vector
  - Known stride allows easy address calculation for all vector elements
    - Enables easy loading (or even early loading, i.e., prefetching) of vectors into registers/cache/memory

Vector Processor Advantages

+ No dependencies within a vector
  - Pipelining & parallelization work really well
  - Can have very deep pipelines (without the penalty of deep pipelines)

+ Each instruction generates a lot of work (i.e., operations)
  - Reduces instruction fetch bandwidth requirements
  - Amortizes instruction fetch and control overhead over many data

    --> Leads to high energy efficiency per operation

+ No need to explicitly code loops
  - Fewer branches in the instruction sequence

+ Highly regular memory access pattern

Vector Processor Disadvantages

-- Works (only) if parallelism is regular (data/SIMD parallelism)

++ Vector operations

-- Very inefficient if parallelism is irregular

-- How about searching for a key in a linked list?

To program a vector machine, the compiler or hand coder must make the data structures in the code fit nearly exactly the regular structure built into the hardware. That's hard to do in first place, and just as hard to change. One tweak, and the low-level code has to be rewritten by a very smart and dedicated programmer who knows the hardware and often the subtleties of the application area. Often the rewriting is

Fisher, "Very Long Instruction Word architectures and the ELI-512," ISCA 1983.

Recommended Paper: VLIW

VERY LONG INSTRUCTION WORD
ARCHITECTURES
AND THE ELI-512

JOSEPH A. FISHER
YALE UNIVERSITY
NEW HAVEN, CONNECTICUT 06520

ABSTRACT

By compiling ordinary scientific applications programs with a radical technique called trace scheduling, we are generating code for a parallel machine that will run these programs faster than an equivalent sequential machine — we expect 10 to 30 times faster.

Trace scheduling generates code for machines called Very Long Instruction Word architectures. In Very Long Instruction Word machines, many statically scheduled, tightly coupled, fine-grained operations execute in parallel within a single instruction stream. VLIWs are more parallel extensions of several current architectures.

These current architectures have never cracked a fundamental barrier. The speedup they get from parallelism is never more than a factor of 2 to 3. Not that we couldn't build more parallel machines of this type; but until trace scheduling we didn't know how to generate code for them. Trace scheduling finds sufficient parallelism in ordinary code to justify thinking about a highly parallel VLIW.

At Yale we are actually building one. Our machine, the ELI-512, has a horizontal instruction word of over 500 bits and

are presented in this paper. How do we put enough tests in each cycle without making the machine too big? How do we put enough memory references in each cycle without making the machine too slow?

WHAT IS A VLIW?

Everyone wants to use cheap hardware in parallel to speed up computation. One obvious approach would be to take your favorite Reduced Instruction Set Computer, let it be capable of executing 10 to 30 RISC-level operations per cycle controlled by a very long instruction word. (In fact, call it a VLIW.) A VLIW looks like very parallel horizontal microcode.

More formally, VLIW architectures have the following properties:

There is one central control unit issuing a single long instruction per cycle.

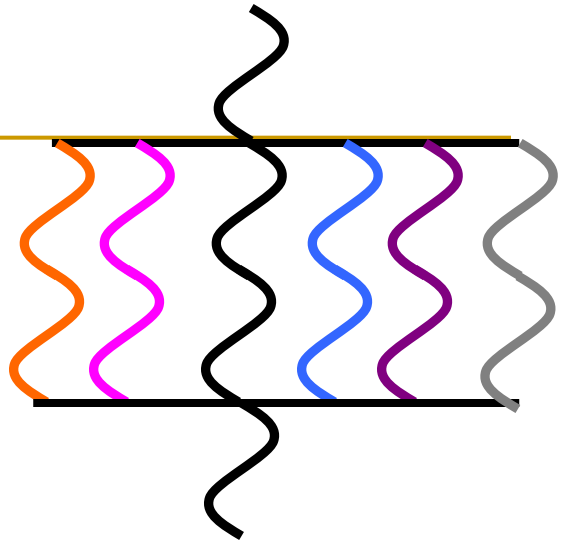
Each long instruction consists of many tightly coupled independent operations.

Each operation requires a small, statically predictable number of cycles to execute.

Operations can be pipelined. These properties distinguish

Fisher, "Very Long Instruction Word architectures and the ELI-512," ISCA 1983.

Amdahl's Law



- Amdahl's Law
  - f: Parallelizable fraction of a program
  - N: Number of processors

$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{N}}$$

  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.

- Maximum speedup limited by serial portion: Serial bottleneck

- All parallel machines "suffer from" the serial bottleneck

Recommended Paper: Amdahl's Law

Validity of the single processor approach to achieving large scale computing capabilities

by DR. GENE M. AMDAHL
International Business Machines Corporation
Sunnyvale, California

INTRODUCTION

For over a decade prophets have voiced the contention that the organization of a single computer has reached its limits and that truly significant advances can be made only by interconnection of a multiplicity of computers in such a manner as to permit cooperative solution. Variously the proper direction has been pointed out as general purpose computers with a generalized interconnection of memories, or as specialized computers with geometrically related memory interconnections and controlled by one or more instruction streams.

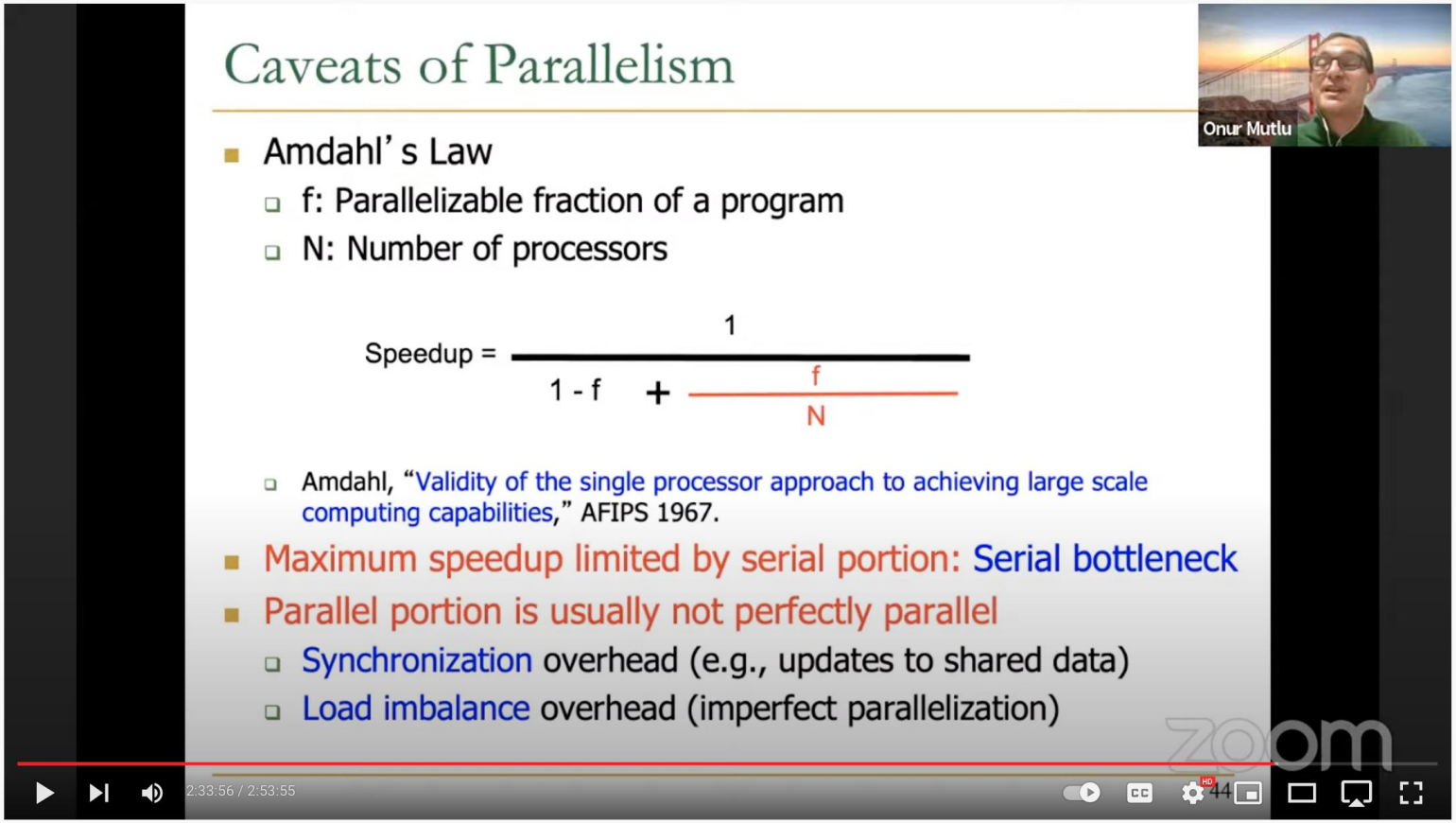
Demonstration is made of the continued validity of the single processor approach and of the weaknesses of the multiple processor approach in terms of application to real problems and their attendant irregularities.

The arguments presented are based on statistical characteristics of computation on computers over the last decade and upon the operational requirements within problems of physical interest. An additional

cessing rate, even if the housekeeping were done in a separate processor. The non-housekeeping part of the problem could exploit at most a processor of performance three to four times the performance of the housekeeping processor. A fairly obvious conclusion which can be drawn at this point is that the effort expended on achieving high parallel processing rates is wasted unless it is accompanied by achievements in sequential processing rates of very nearly the same magnitude.

Data management housekeeping is not the only problem to plague oversimplified approaches to high speed computation. The physical problems which are of practical interest tend to have rather significant complications. Examples of these complications are as follows: boundaries are likely to be irregular; interiors are likely to be inhomogeneous; computations required may be dependent on the states of the variables at each point; propagation rates of different physical effects may be quite different; the

Lecture on Parallelism & Heterogeneity I



Computer Architecture - Lecture 17: Parallelism & Heterogeneity (Fall 2021)

977 views • Streamed live on Nov 25, 2021

36 DISLIKE SHARE CLIP SAVE ...

Onur Mutlu Lectures
23.4K subscribers

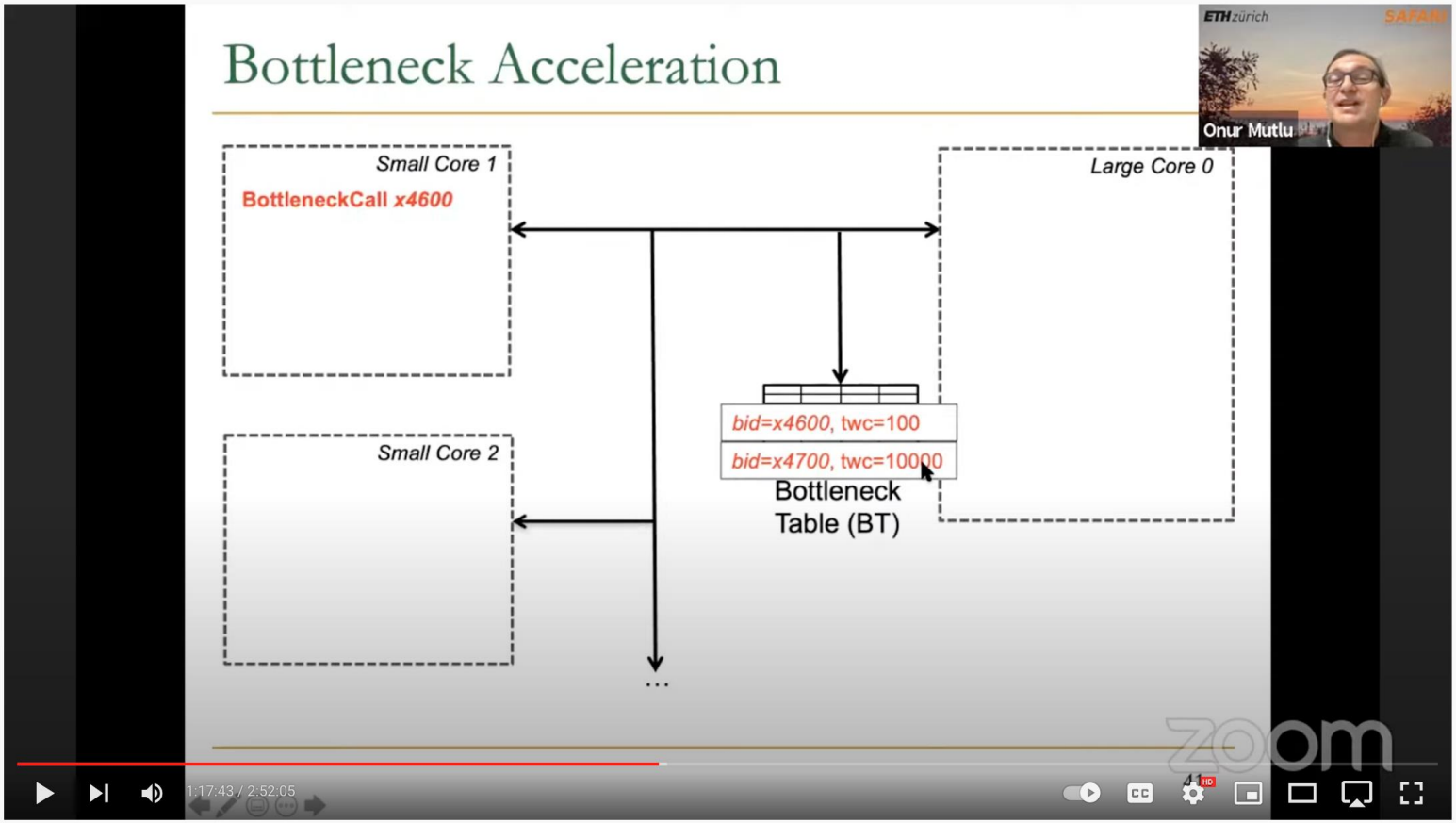
SUBSCRIBED

Computer Architecture, ETH Zürich, Fall 2021 (https://safari.ethz.ch/architecture/f...)

Lecture 17a: Emerging Memory Technologies II

https://youtu.be/GLzG_rEDn9A?t=7211

Lecture on Parallelism & Heterogeneity II



Computer Architecture - Lecture 18: Parallelism & Heterogeneity II (Fall 2021)

1,454 views • Streamed live on Nov 26, 2021

41 DISLIKE SHARE CLIP SAVE ...

Onur Mutlu Lectures
23.4K subscribers

SUBSCRIBED

Computer Architecture, ETH Zürich, Fall 2021 (https://safari.ethz.ch/architecture/f...)

Lecture 18: Parallelism & Heterogeneity II

https://youtu.be/P8l3SMAbyYw

Vector Processor Limitations

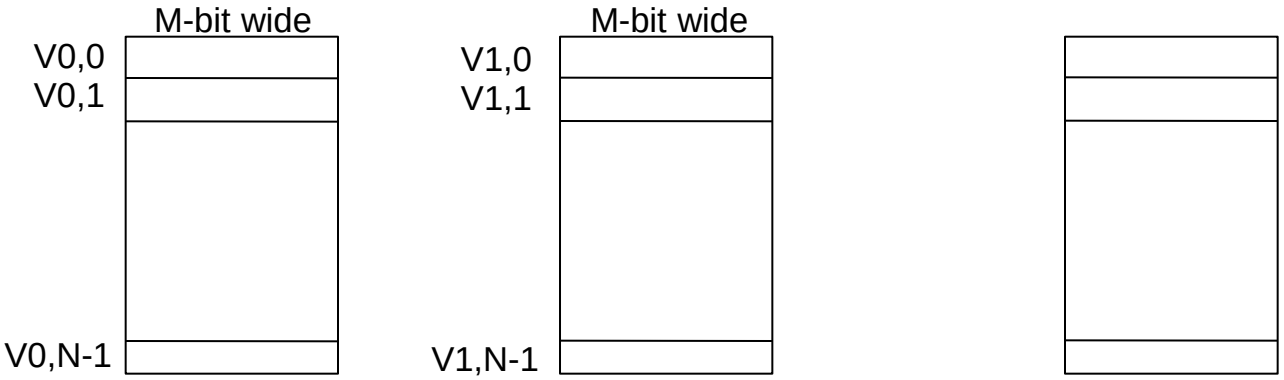
-- Memory (bandwidth) can easily become a bottleneck, especially if

1. compute/memory operation balance is not maintained
2. data is not mapped appropriately to memory banks

Vector Processing in More Depth

Vector Registers

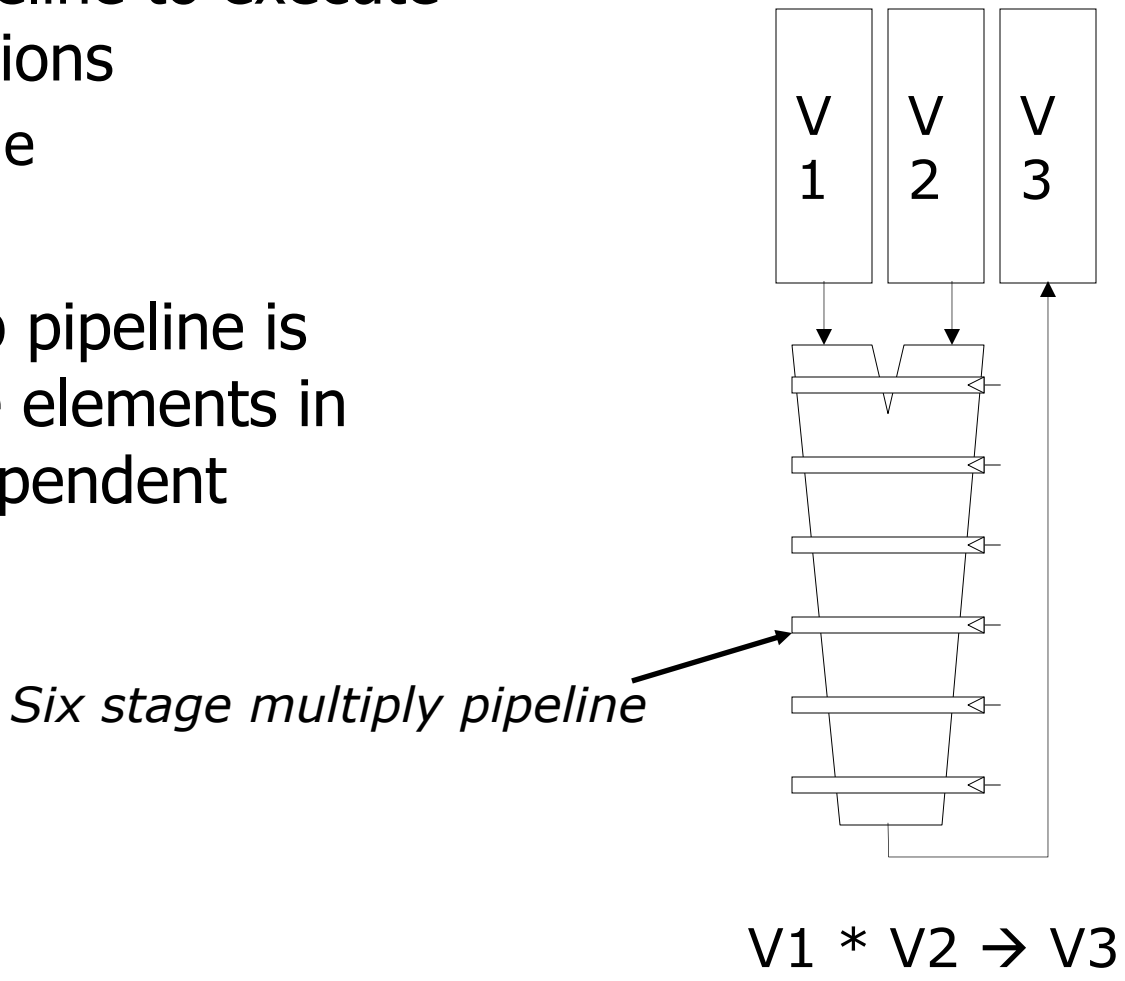
- Each vector data register holds N M-bit values
- Vector control registers: VLEN, VSTR, VMASK
- Maximum VLEN can be N
  - Maximum number of elements stored in a vector register
- Vector Mask Register (VMASK)
  - Indicates which elements of vector to operate on
  - Set by vector test instructions
    - e.g., VMASK[i] = (V_k[i] == 0)



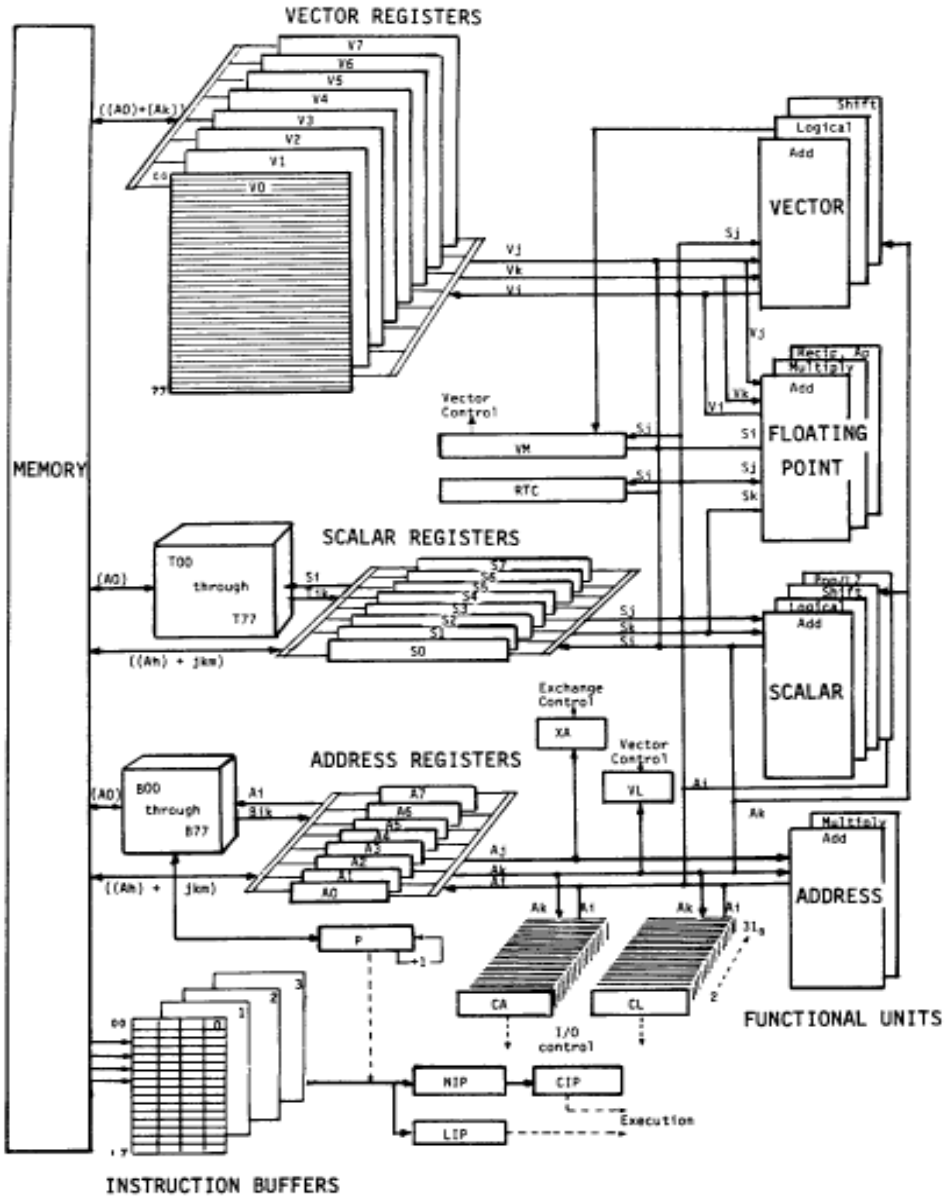
Vector Functional Units

- Use a deep pipeline to execute element operations
  → fast clock cycle

- Control of deep pipeline is simple because elements in vector are independent



Vector Machine Organization (CRAY-1)



- CRAY-1
- Russell, "The CRAY-1 computer system," CACM 1978.

- Scalar and vector modes
- 8 64-element vector registers
- 64 bits per element
- 16 memory banks
- 8 64-bit scalar registers
- 8 24-bit address registers

Recommended Paper

Russell,
"The CRAY-1 computer system,"
CACM 1978.

The CRAY-1 Computer System

Richard M. Russell
Cray Research, Inc.

This paper describes the CRAY-1, discusses the evolution of its architecture, and gives an account of some of the problems that were overcome during its manufacture.

The CRAY-1 is the only computer to have been built to date that satisfies ERDA's Class VI requirement (a computer capable of processing from 20 to 60 million floating point operations per second) [1].

The CRAY-1's Fortran compiler (CFT) is designed to give the scientific user immediate access to the benefits of the CRAY-1's vector processing architecture. An optimizing compiler, CFT, "vectorizes" innermost DO loops. Compatible with the ANSI 1966 Fortran Standard and with many commonly supported Fortran extensions, CFT does not require any source program modifications or the use of additional nonstandard Fortran statements to achieve vectorization. Thus the user's investment of hundreds of man months of effort to develop Fortran programs for other contemporary computers is protected.

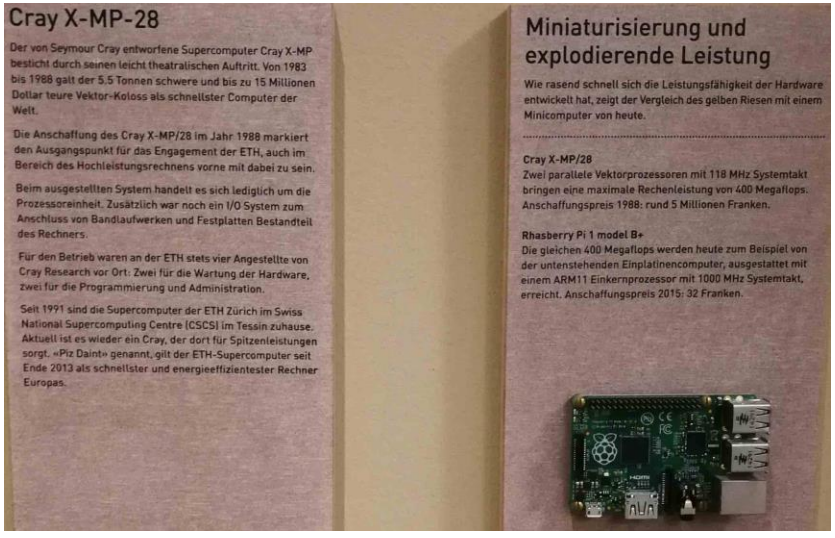
Key Words and Phrases: architecture, computer systems

CR Categories: 1.2, 6.2, 6.3

Introduction

Vector processors are not yet commonplace machines in the larger-scale computer market. At the time of this writing we know of only 12 non-CRAY-1 vector processor installations worldwide. Of these 12, the most powerful processor is the ILLIAC IV (1 installation), the most populous is the Texas Instruments Advanced Scientific Computer (7 installations) and the most publicized is Control Data's STAR 100

CRAY X-MP-28 @ ETH (CAB, E Floor)



Cray X-MP-28

Der von Seymour Cray entworfene Supercomputer Cray X-MP besticht durch seinen leicht theatralischen Auftritt. Von 1983 bis 1988 galt der 5.5 Tonnen schwere und bis zu 15 Millionen Dollar teure Vektor-Koloss als schnellster Computer der Welt.

Die Anschaffung des Cray X-MP/28 im Jahr 1988 markiert den Ausgangspunkt für das Engagement der ETH, auch im Bereich des Hochleistungsrechnens vorne mit dabei zu sein.

Beim ausgestellten System handelt es sich lediglich um die Prozessoreinheit. Zusätzlich war noch ein I/O System zum Anschluss von Bandlaufwerken und Festplatten Bestandteil des Rechners.

Für den Betrieb waren an der ETH stets vier Angestellte von Cray Research vor Ort: Zwei für die Wartung der Hardware, zwei für die Programmierung und Administration.

Seit 1991 sind die Supercomputer der ETH Zürich im Swiss National Supercomputing Centre (CSCS) im Tessin zuhause. Aktuell ist es wieder ein Cray, der dort für Spitzenleistungen sorgt. «Piz Daint» genannt, gilt der ETH-Supercomputer seit Ende 2013 als schnellster und energieeffizientester Rechner Europas.

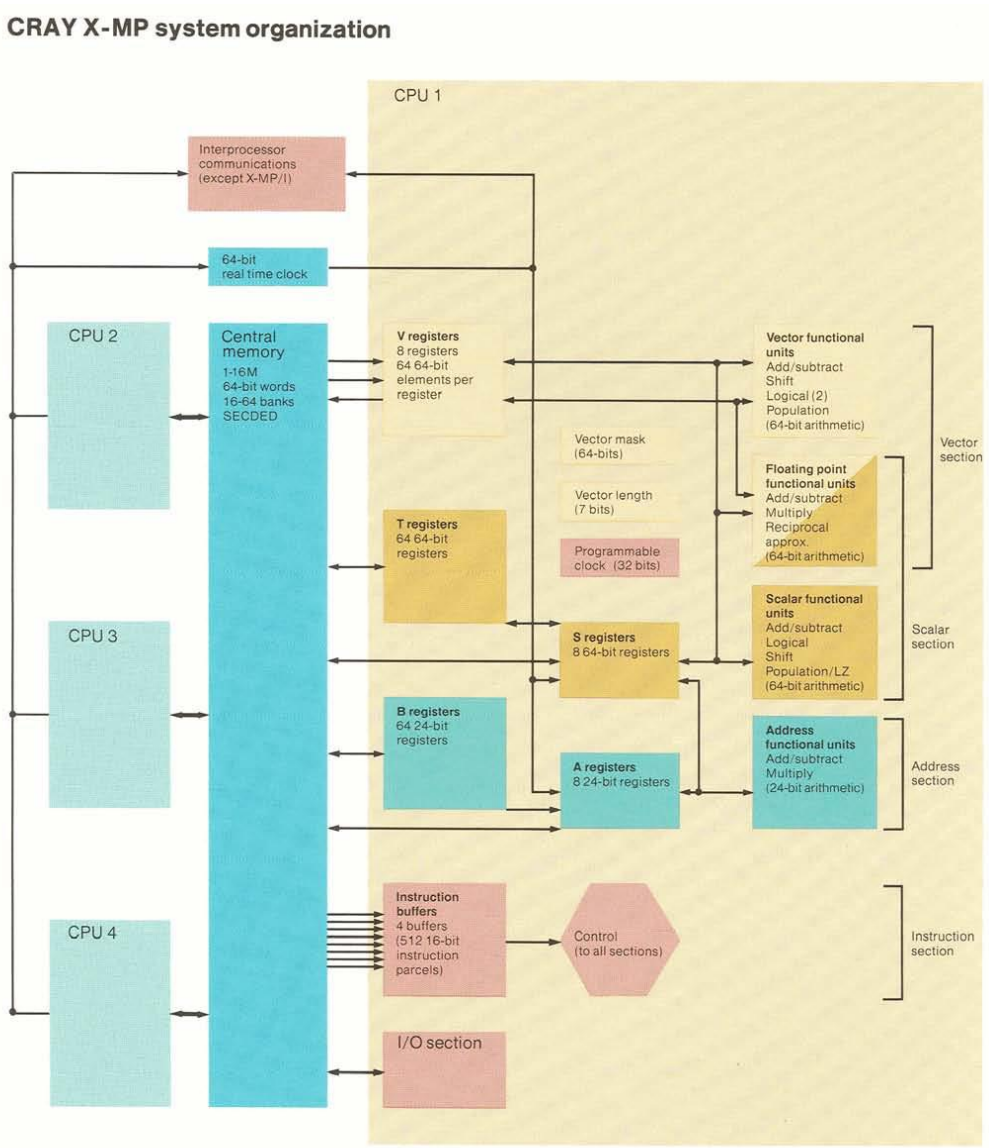
Miniaturisierung und explodierende Leistung

Wie rasend schnell sich die Leistungsfähigkeit der Hardware entwickelt hat, zeigt der Vergleich des gelben Riesen mit einem Minicomputer von heute.

Cray X-MP/28
Zwei parallele Vektorprozessoren mit 118 MHz Systemtakt bringen eine maximale Rechenleistung von 400 Megaflops. Anschaffungspreis 1988: rund 5 Millionen Franken.

Rhasberry Pi 1 model B+
Die gleichen 400 Megaflops werden heute zum Beispiel von der untenstehenden Einplatinencomputer, ausgestattet mit einem ARM11 Einkernprozessor mit 1000 MHz Systemtakt, erreicht. Anschaffungspreis 2015: 32 Franken.

CRAY X-MP System Organization



Cray Research Inc., "The CRAY X-MP Series of Computer Systems," 1985

CRAY X-MP Design Detail

CRAY X-MP design detail

Mainframe

CRAY X-MP single- and multiprocessor systems are designed to offer users outstanding performance on large-scale, compute-intensive and I/O-bound jobs.

CRAY X-MP mainframes consist of six (X-MP/1), eight (X-MP/2) or twelve (X-MP/4) vertical columns arranged in an arc. Power supplies and cooling are clustered around the base and extend outward.

Model	Number of CPUs	Memory size (millions of 64-bit words)	Number of banks
CRAY X-MP/416	4	16	64
CRAY X-MP/48	4	8	32
CRAY X-MP/216	2	16	32
CRAY X-MP/28	2	8	32
CRAY X-MP/24	2	4	16
CRAY X-MP/18	1	8	32
CRAY X-MP/14	1	4	16
CRAY X-MP/12	1	2	16
CRAY X-MP/11	1	1	16

Hardware features:

- 9.5 nsec clock
- One, two or four CPUs, each with its own computation and control sections
- Large multiport central memory
- Memory bank cycle time of 38 nsec on X-MP/4 systems, 76 nsec on X-MP/1 and X-MP/2 models
- Memory bandwidth of 25-100 gigabits, depending on model
- I/O section
- Proven cooling and packaging technologies

A description of the major system components and their functions follows.

CPU computation section

Within the computation section of each CPU are operating registers, functional units and an instruction control network — hardware elements that cooperate in executing sequences of instructions. The instruction control network makes all decisions related to instruction issue as well as coordinating the three types of processing within each CPU: vector, scalar and address. Each of the processing modes has its associated registers and functional units.

The block diagram of a CRAY X-MP/4 (opposite page) illustrates the relationship of the registers to the functional units, instruction buffers, I/O channel control registers, interprocessor communications section and memory. For multiple-processor CRAY X-MP models, the interprocessor communications section coordinates processing between CPUs, and central memory is shared.

Registers

The basic set of programmable registers is composed of:

Eight 24-bit address (A) registers
Sixty-four 24-bit intermediate address (B) registers
Eight 64-bit scalar (S) registers
Sixty-four 64-bit scalar-save (T) registers
Eight 64-element (4096-bit) vector (V) registers with 64 bits per element

The 24-bit A registers are generally used for addressing and counting operations. Associated with them are 64 B registers, also 24 bits wide. Since the transfer between an A and a B register takes only one clock period, the B registers assume the role of data cache, storing information for fast access without tying up the A registers for relatively long periods.

Cray Research Inc., "The CRAY X-MP Series of Computer Systems," 1985

CRAY X-MP CPU Functional Units

CRAY X-MP CPU functional units

	Register usage	Time in clock periods
Address functional units		
Addition	A	2
Multiplication	A	4
Scalar functional units		
Addition	S	3
Shift-single	S	2
Shift-double	S	3
Logical	S	1
Population, parity and leading zero	S	3 or 4
Vector functional units		
Addition	V	3
Shift	V	3 or 4
Full vector logical	V	2

Cray Research Inc., "The CRAY X-MP Series of Computer Systems," 1985

CRAY X-MP System Configuration

System configuration options

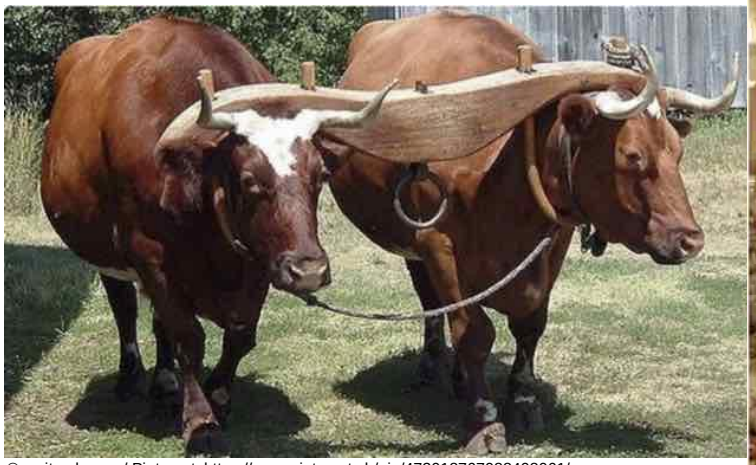
	X-MP/1	X-MP/2	X-MP/4
Mainframe			
CPUs	1	2	4
Bipolar memory (64-bit words)	N/A	N/A	8 or 16M
MOS memory (64-bit words)	1, 2, 4 or 8M	4, 8 or 16M	N/A
6-Mbyte channels	2 or 4	4	4
100-Mbyte channels	1 or 2	2	4
1000-Mbyte channels	1	1	2
I/O Subsystem			
I/O processors	2, 3 or 4	2, 3 or 4	4
Disk storage units	2-32	2-32	2-32
Magnetic tape channels	1-8	1-8	1-8
Front-end interfaces	1-7	1-7	1-7
Buffer memory (Mbytes)	8, 32 or 64	8, 32 or 64	64
Solid-state Storage Device			
Memory size (Mbytes)	256, 512 or 1024	256, 512 or 1024	256, 512 or 1024

N/A signifies option is not available on the model

Cray Research Inc., "The CRAY X-MP Series of Computer Systems," 1985

Seymour Cray, Leader in Supercomputer Design

"If you were plowing a field, which would you rather use: Two strong oxen or 1024 chickens?"

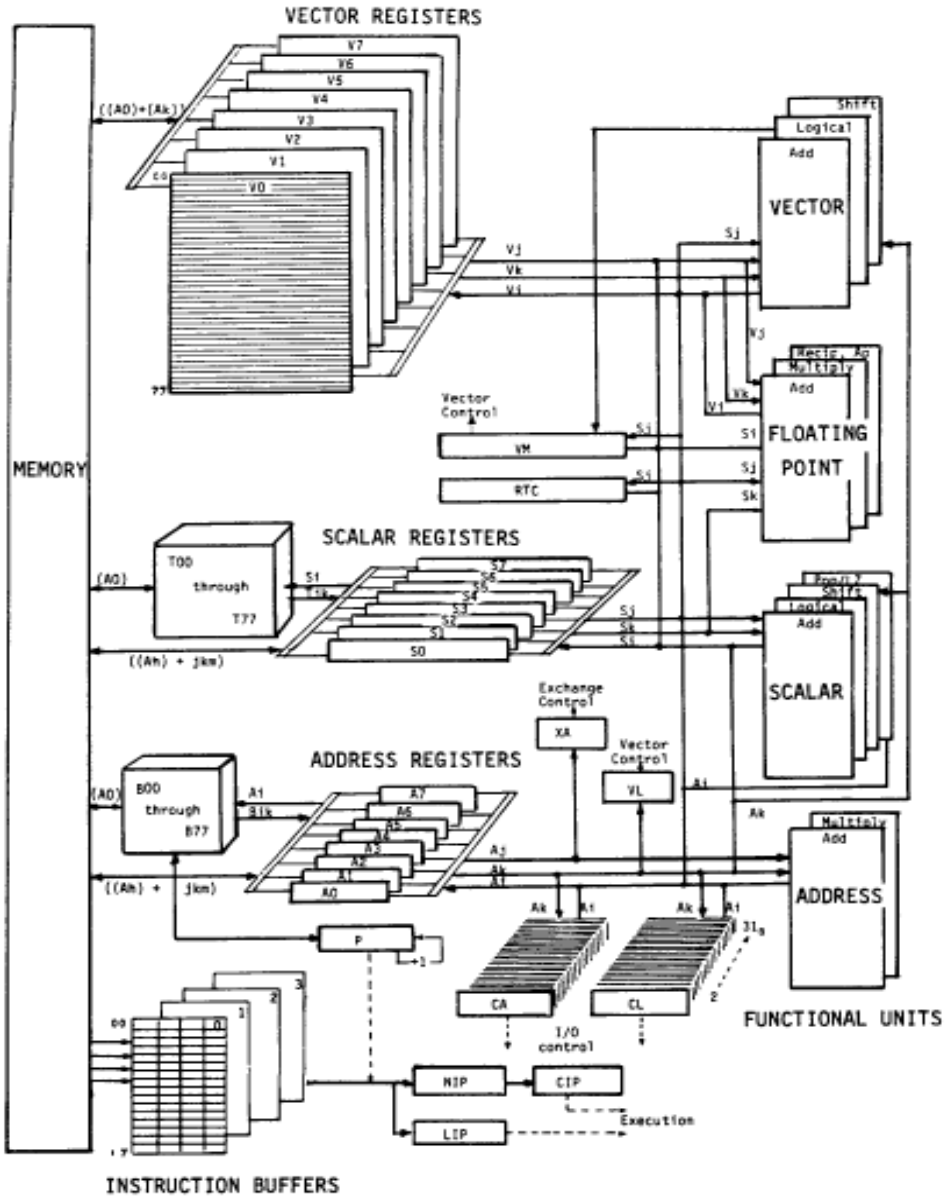


© amityrebecca / Pinterest. https://www.pinterest.ch/pin/473018767088408061/

© Scott Sinklier / Corbis. http://america.aljazeera.com/articles/2015/2/20/the-short-brutal-life-of-male-chickens.html

https://en.wikipedia.org/wiki/Seymour_Cray

Vector Machine Organization (CRAY-1)



- CRAY-1
- Russell, "The CRAY-1 computer system," CACM 1978.

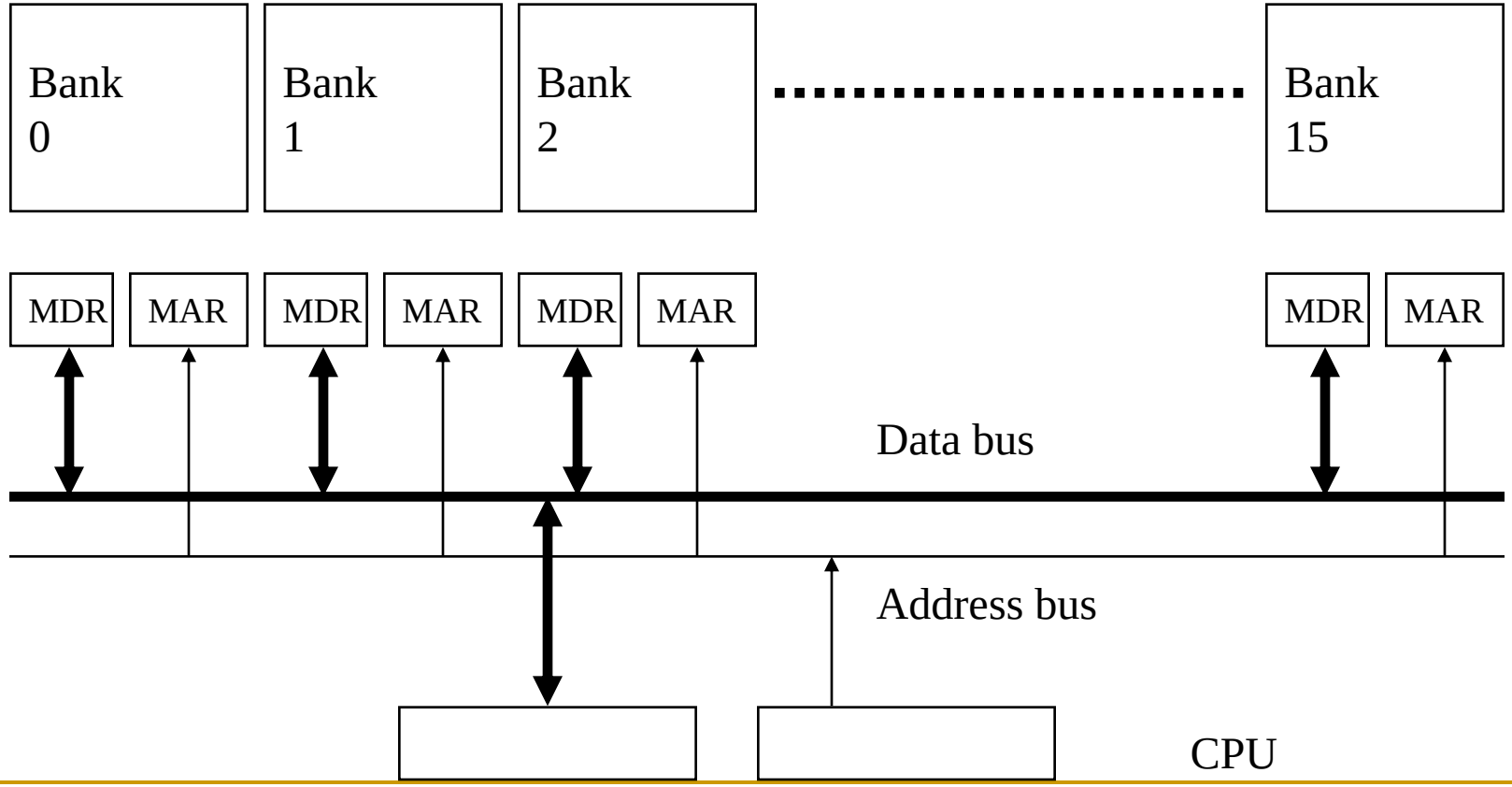
- Scalar and vector modes
- 8 64-element vector registers
- 64 bits per element
- 16 memory banks
- 8 64-bit scalar registers
- 8 24-bit address registers

Loading/Storing Vectors from/to Memory

- Requires loading/storing multiple elements
- Elements separated from each other by a constant distance (stride)
  - Assume stride = 1 for now
- Elements can be loaded in consecutive cycles if we can start the load of one element per cycle
  - Can sustain a throughput of one element per cycle
- Question: How do we achieve this with a memory that takes **more than 1 cycle to access**?
- Answer: Bank the memory; interleave the elements across banks

Memory Banking

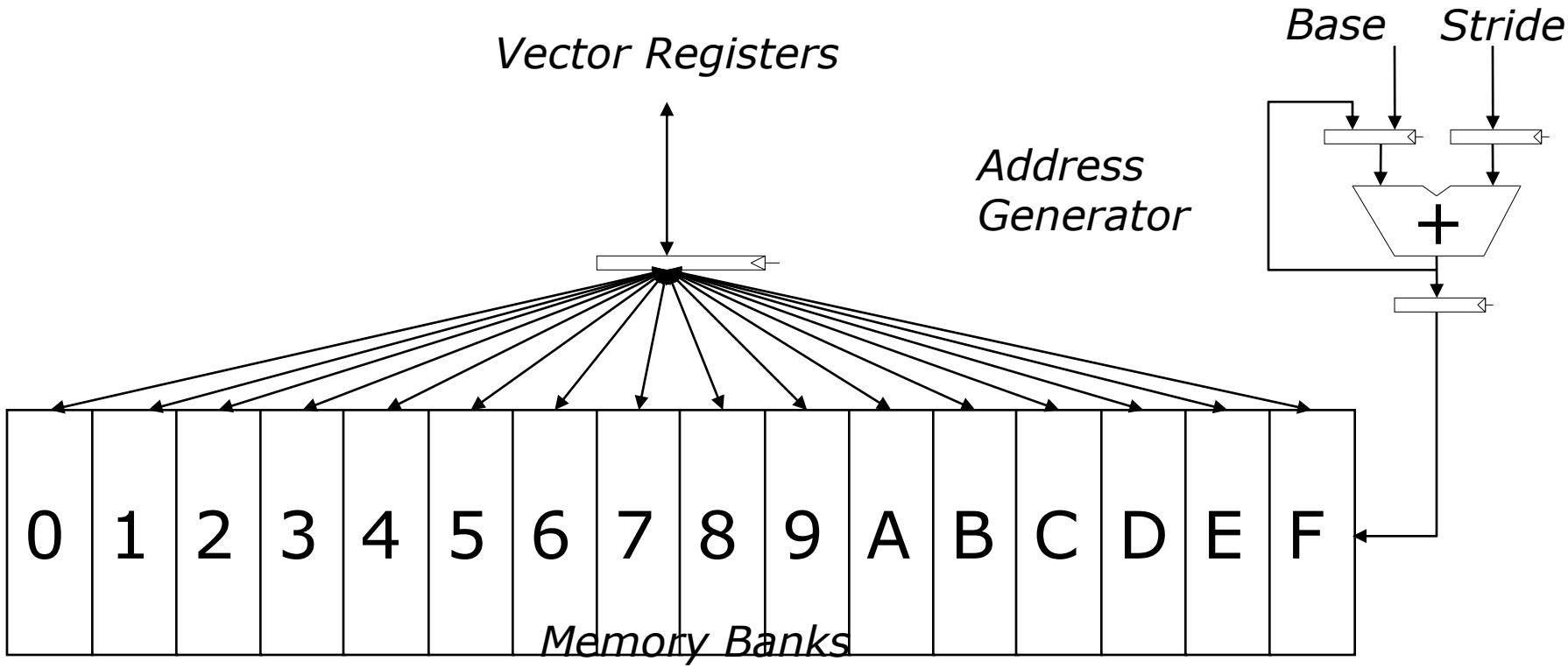
- Memory is divided into banks that can be accessed independently; banks share address and data buses (to minimize pin cost)
- Can start and complete one bank access per cycle
- Can sustain N concurrent accesses if all N go to different banks



Picture credit: Derek Chiou

Vector Memory System

- Next address = Previous address + Stride
- If (stride == 1) && (consecutive elements interleaved across banks) && (number of banks >= bank latency), then
  - we can sustain 1 element/cycle throughput



Picture credit: Krste Asanovic

Scalar Code Example: Element-Wise Avg.

- For I = 0 to 49
  - C[i] = (A[i] + B[i]) / 2

- Scalar code (instruction and its latency)

```
    MOVI R0 = 50                1
    MOVA R1 = A                 1
    MOVA R2 = B                 1
    MOVA R3 = C                 1
X:  LD R4 = MEM[R1++]           11  ;autoincrement addressing
    LD R5 = MEM[R2++]           11
    ADD R6 = R4 + R5            4
    SHFR R7 = R6 >> 1           1
    ST MEM[R3++] = R7           11
    DECBNZ R0, X                2   ;decrement and branch if NZ
```

304 dynamic instructions

Scalar Code Execution Time (In Order)

- Scalar execution time on an in-order processor with 1 bank
  - First two loads in the loop cannot be pipelined: 2*11 cycles
  - 4 + 50*40 = 2004 cycles

- Scalar execution time on an in-order processor with 16 banks (word-interleaved: consecutive words are stored in consecutive banks)
  - First two loads in the loop can be pipelined
  - 4 + 50*30 = 1504 cycles

- Why 16 banks?
  - 11-cycle memory access latency
  - Having 16 (>11) banks ensures there are enough banks to overlap enough memory operations to cover memory latency

Vectorizable Loops

- A loop is vectorizable if each iteration is independent of any other

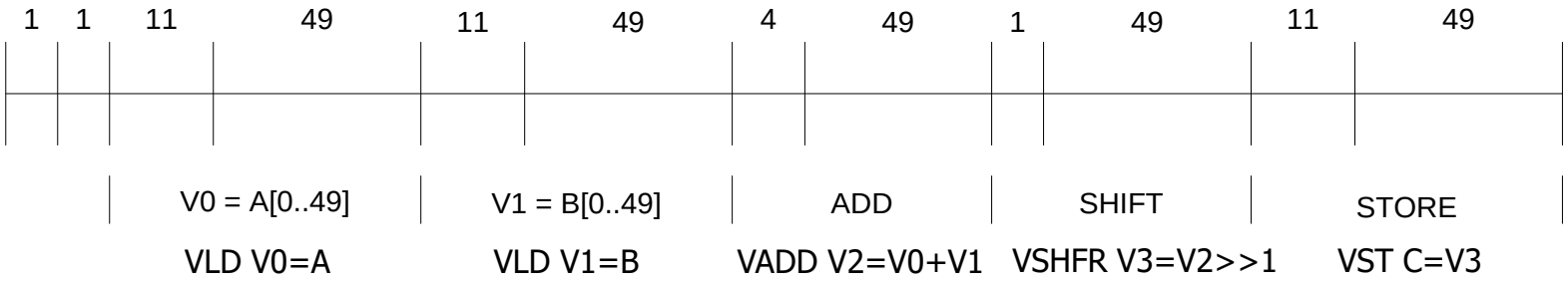
- For I = 0 to 49
  - C[i] = (A[i] + B[i]) / 2
- Vectorized loop (each instruction and its latency):

MOVI VLEN = 50	1
MOVI VSTR = 1	1
VLD V0 = A	11 + VLEN – 1
VLD V1 = B	11 + VLEN – 1
VADD V2 = V0 + V1	4 + VLEN – 1
VSHFR V3 = V2 >> 1	1 + VLEN – 1
VST C = V3	11 + VLEN – 1

7 dynamic instructions

Basic Vector Code Performance

- Assume no chaining (no vector data forwarding)
  - i.e., output of a vector functional unit cannot be used as the direct input of another
  - The entire vector register needs to be ready before any element of it can be used as part of another operation
- One memory port (one address generator)
- 16 memory banks (word-interleaved)

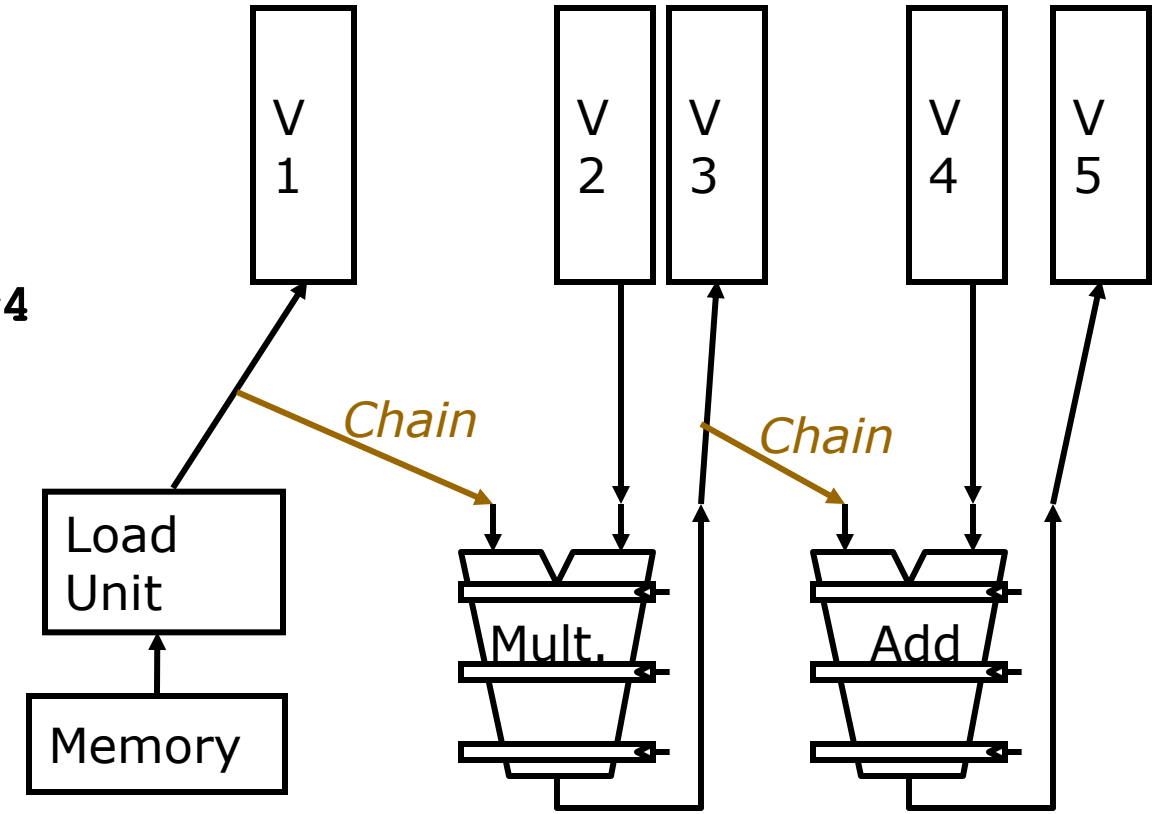


- 285 cycles

Vector Chaining

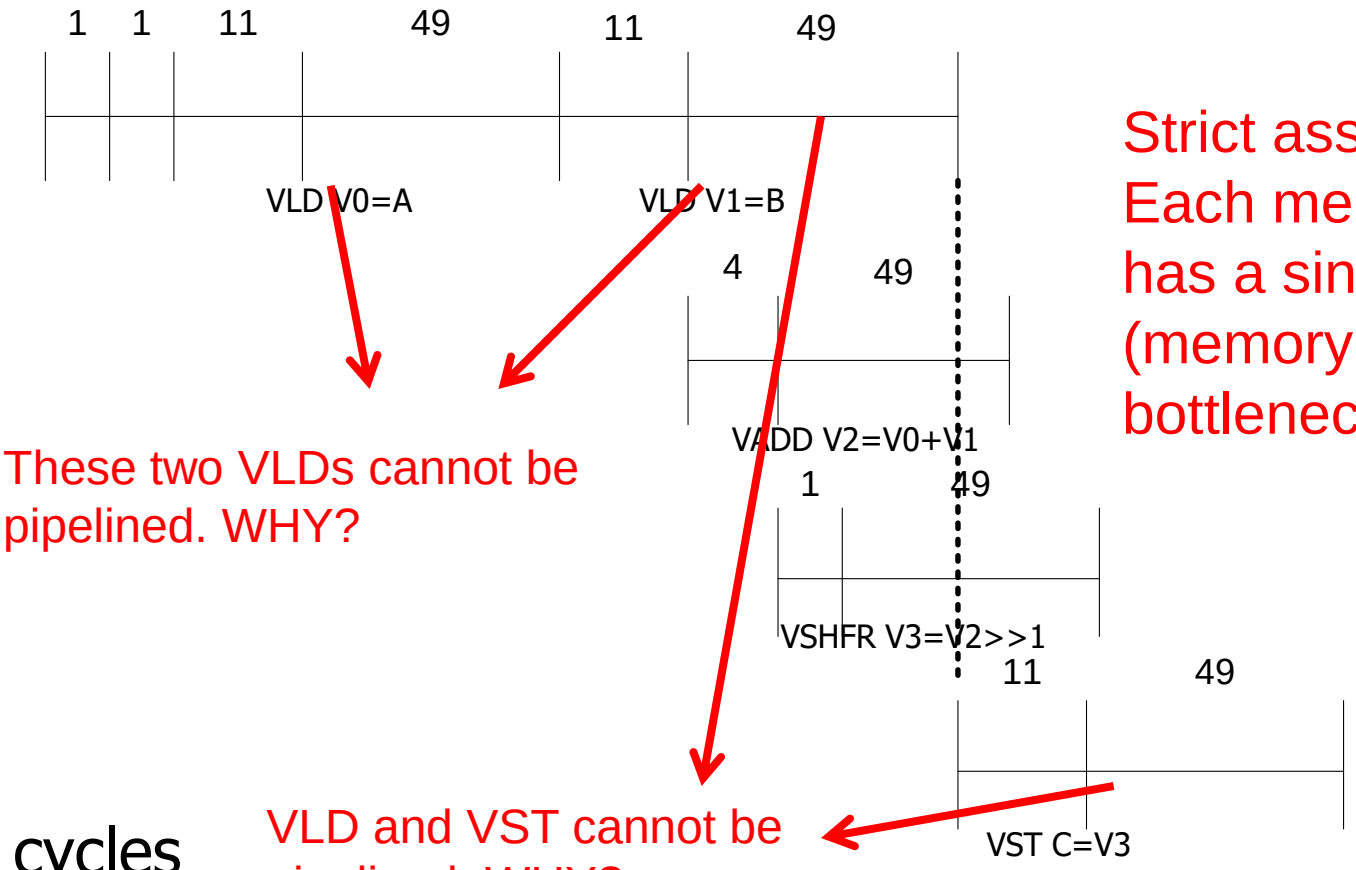
- Vector chaining: Data forwarding from one vector functional unit to another

```
LV   v1
MULV v3,v1,v2
ADDV v5,v3, v4
```



Vector Code Performance - Chaining

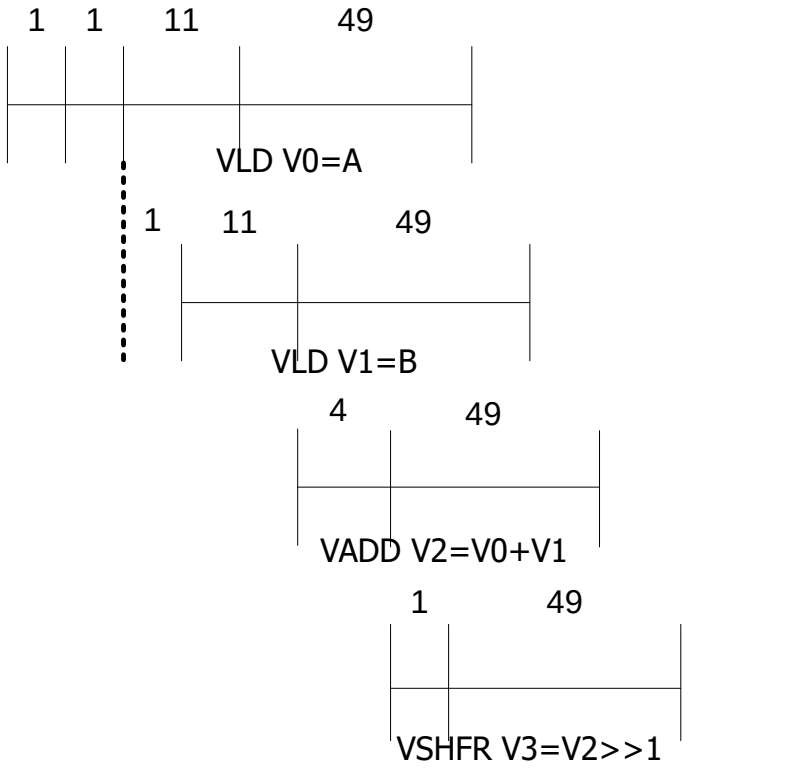
- Vector chaining: Data forwarding from one vector functional unit to another



- 182 cycles

Vector Code Performance – Multiple Memory Ports

- Chaining and 2 load ports, 1 store port in each bank



- 79 cycles
- 19X perf. improvement!

Questions (I)

- What if # data elements > # elements in a vector register?
  - Idea: Break loops so that each iteration operates on # elements in a vector register
    - E.g., 527 data elements, 64-element VREGs
    - 8 iterations where VLEN = 64
    - 1 iteration where VLEN = 15 (need to change value of VLEN)
  - Called vector stripmining

(Vector) Stripmining

Surface mining, including **strip mining**, open-pit mining and mountaintop removal mining, is a broad category of mining in which soil and rock overlying the mineral deposit (the overburden) are removed, in contrast to underground mining, in which the overlying rock is left in place, and the mineral removed through shafts or tunnels.

Surface mining began in the mid-sixteenth century[1] and is practiced throughout the world, although the majority of surface coal mining occurs in North America.[2] It gained

Coal strip mine in Wyoming

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

Questions (II)

- What if vector data is not stored in a strided fashion in memory? (irregular memory access to a vector)
  - Idea: Use indirection to combine/pack elements into vector registers
  - Called scatter/gather operations
  - Doing so also helps with avoiding useless computation on sparse vectors (i.e., vectors where many elements are 0)

Gather/Scatter Operations

Want to vectorize loops with indirect accesses:

```
for (i=0; i<N; i++)
    A[i] = B[i] + C[D[i]]
```

Indexed load instruction (*Gather*)

```
LV vD, rD         # Load indices in D vector
LVI vC, rC, vD    # Load indirect from rC base
LV vB, rB         # Load B vector
ADDV.D vA,vB,vC   # Do add
SV vA, rA         # Store result
```

Gather/Scatter Operations

- Gather/scatter operations often implemented in hardware to handle sparse vectors (matrices) or indirect indexing
- Vector loads and stores use an index vector which is added to the base register to generate the addresses

- *Scatter* example

Index Vector	Data Vector (to Store)	Stored Vector (in Memory)	
0	3.14	Base+0	3.14
2	6.5	Base+1	X
6	71.2	Base+2	6.5
7	2.71	Base+3	X
		Base+4	X
		Base+5	X
		Base+6	71.2
		Base+7	2.71

Conditional Operations in a Loop

- What if some operations should not be executed on a vector (based on a dynamically-determined condition)?

```
loop:       for (i=0; i<N; i++)
                if (a[i] != 0) then b[i]=a[i]*b[i]
```

- Idea: Masked operations
  - VMASK register is a bit mask determining which data element should not be acted upon

```
VLD V0 = A
VLD V1 = B
VMASK = (V0 != 0)
VMUL V1 = V0 * V1
VST B = V1
```

  - This is predicated execution. Execution is *predicated* on mask bit.

Another Example with Masking

```
for (i = 0; i < 64; ++i)
    if (a[i] >= b[i])
        c[i] = a[i]
    else
        c[i] = b[i]
```

A	B	VMASK
1	2	0
2	2	1
3	2	1
4	10	0
-5	-4	0
0	-3	1
6	5	1
-7	-8	1

Steps to execute the loop in SIMD code

1. Compare A, B to get VMASK

2. Masked store of A into C

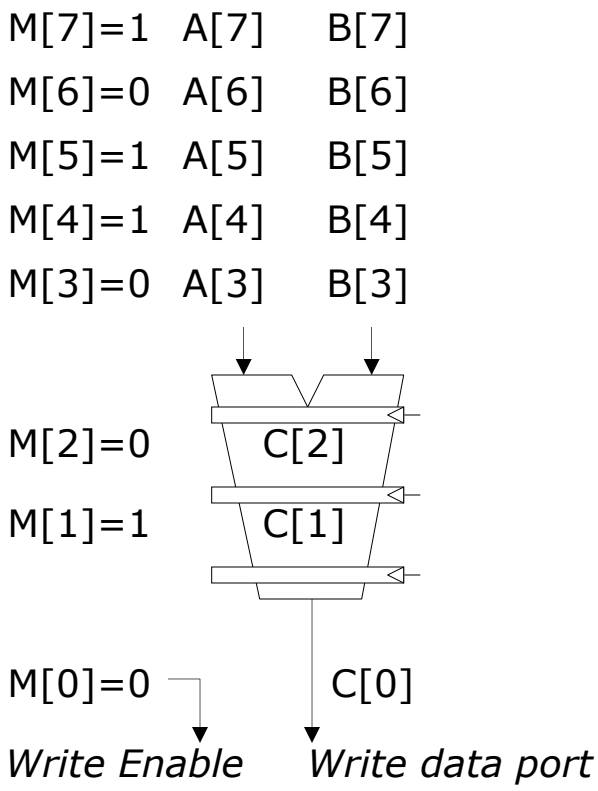
3. Complement VMASK

4. Masked store of B into C

Masked Vector Instructions

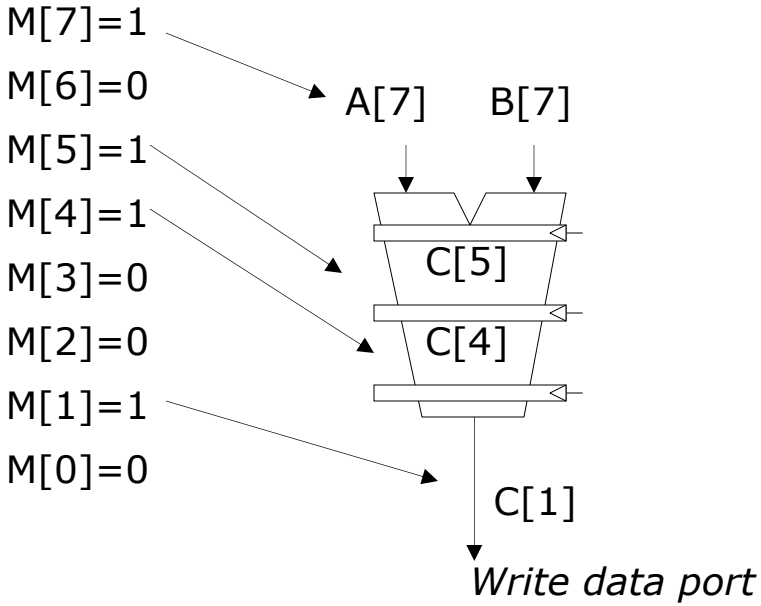
Simple Implementation

- execute all N operations, turn off result writeback according to mask



Density-Time Implementation

- scan mask vector and only execute elements with non-zero masks



Which one is better?

Tradeoffs?

Slide credit: Krste Asanovic

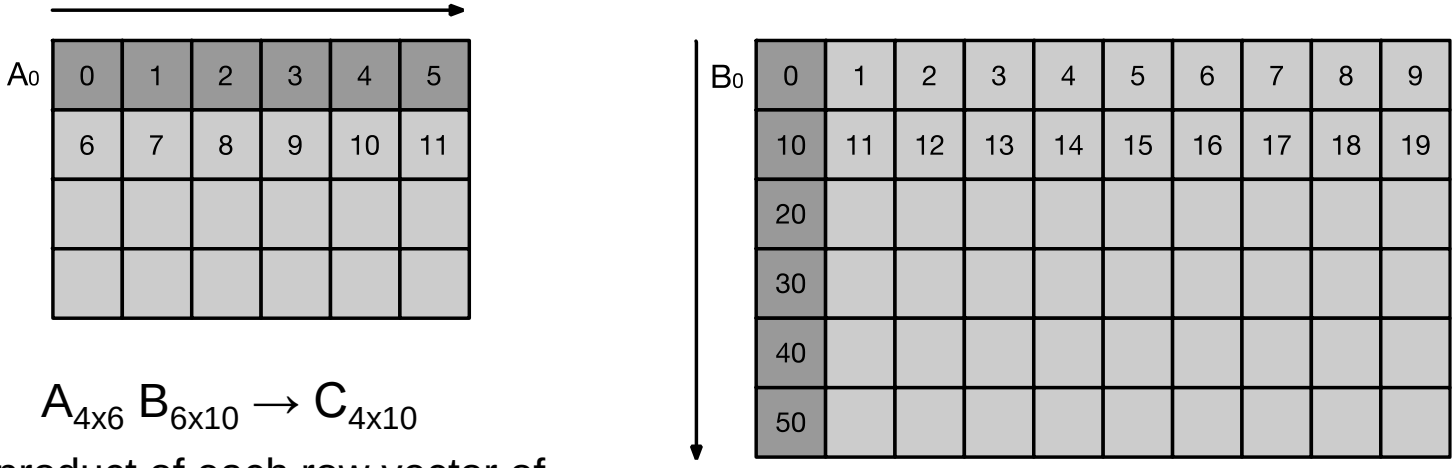
Some Issues

- Stride and banking
  - As long as they are *relatively prime* to each other and there are enough banks to cover bank access latency, we can sustain 1 element/cycle throughput

- Storage format of a matrix
  - Row major: Consecutive elements in a row are laid out consecutively in memory
  - Column major: Consecutive elements in a column are laid out consecutively in memory
  - You need to change the stride when accessing a row versus column

Bank Conflicts in Matrix Multiplication

- A and B matrices, both stored in memory in row-major order



$A_{4x6}\ B_{6x10} \rightarrow C_{4x10}$

Dot product of each row vector of A with each column vector of B

- Load A's row 0 into vector register V_1
  - Each time, increment address by 1 to access the next column
  - Accesses have a stride of 1
- Load B's column 0 into vector register V_2
  - Each time, increment address by 10
  - Accesses have a stride of 10

Different strides can lead to bank conflicts

How do we minimize them?

Minimizing Bank Conflicts

- More banks
- More ports in each bank
- Better data layout to match the access pattern
  - Is this always possible?
- Better mapping of address to bank
  - E.g., randomized mapping
  - Rau, "Pseudo-randomly interleaved memory," ISCA 1991.

Recommended Reading: Minimizing Bank Conflicts

PSEUDO-RANDOMLY INTERLEAVED MEMORY

B. Ramakrishna Rau
Hewlett Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94303

ABSTRACT

Interleaved memories are often used to provide the high bandwidth needed by multiprocessors and high performance uniprocessors such as vector and VLIW processors. The manner in which memory locations are distributed across the memory modules has a significant influence on whether, and for which types of reference patterns, the full bandwidth of the memory system is achieved. The most common interleaved memory architecture is the sequentially interleaved memory in which successive memory locations are assigned to successive memory modules. Although such an architecture is the simplest to implement and provides good performance with strides that are odd integers, it can degrade badly in the face of even strides, especially strides that are a power of two.

In a pseudo-randomly interleaved memory architecture, memory locations are assigned to the memory modules in some pseudo-random fashion in the hope that those sequences of references, which are likely to occur in practice, will end up being evenly distributed across the memory modules. The notion of polynomial interleaving modulo an irreducible polynomial is introduced as a way of achieving pseudo-random interleaving with certain attractive and provable properties. The theory behind this scheme is developed and the results of simulations are presented.

Keywords: supercomputer memory, parallel memory, interleaved memory, hashed memory, pseudo-random interleaving, memory buffering.

The conventional solution is to provide each processor with a data cache constructed out of SRAM. The problem is maintaining cache coherency, at high request rates, across multiple private caches in a multiprocessor system. The alternative is to use a shared cache if the additional delay incurred in going through the processor-cache interconnect is acceptable. The problem here is that the bandwidth, even with SRAM chips, is inadequate unless some form of interleaving is employed in the cache. So once again, the interleaving scheme used is an issue. Furthermore, data caches are susceptible to problems arising out of the lack of spatial and/or data locality in the data reference pattern of many applications. This phenomenon has been studied and reported elsewhere, e.g., in [4,5]. Since data caches are essential to achieving good performance on scalar computations with little parallelism, the right compromise is to provide a data cache that can be bypassed when referencing data structures with poor locality. This is the solution employed in various recent products such as the Convex C-1 and Intel's i860.

Interleaved memory systems. Whether or not a data cache is present, it is important to provide a memory system with bandwidth to match the processors. This is done by organizing the memory system as multiple memory modules which can operate in parallel. The manner in which memory locations are distributed across the memory modules has a significant influence on whether, and for which types of reference patterns, the full bandwidth of the memory system is achieved.

Engineering and scientific applications include

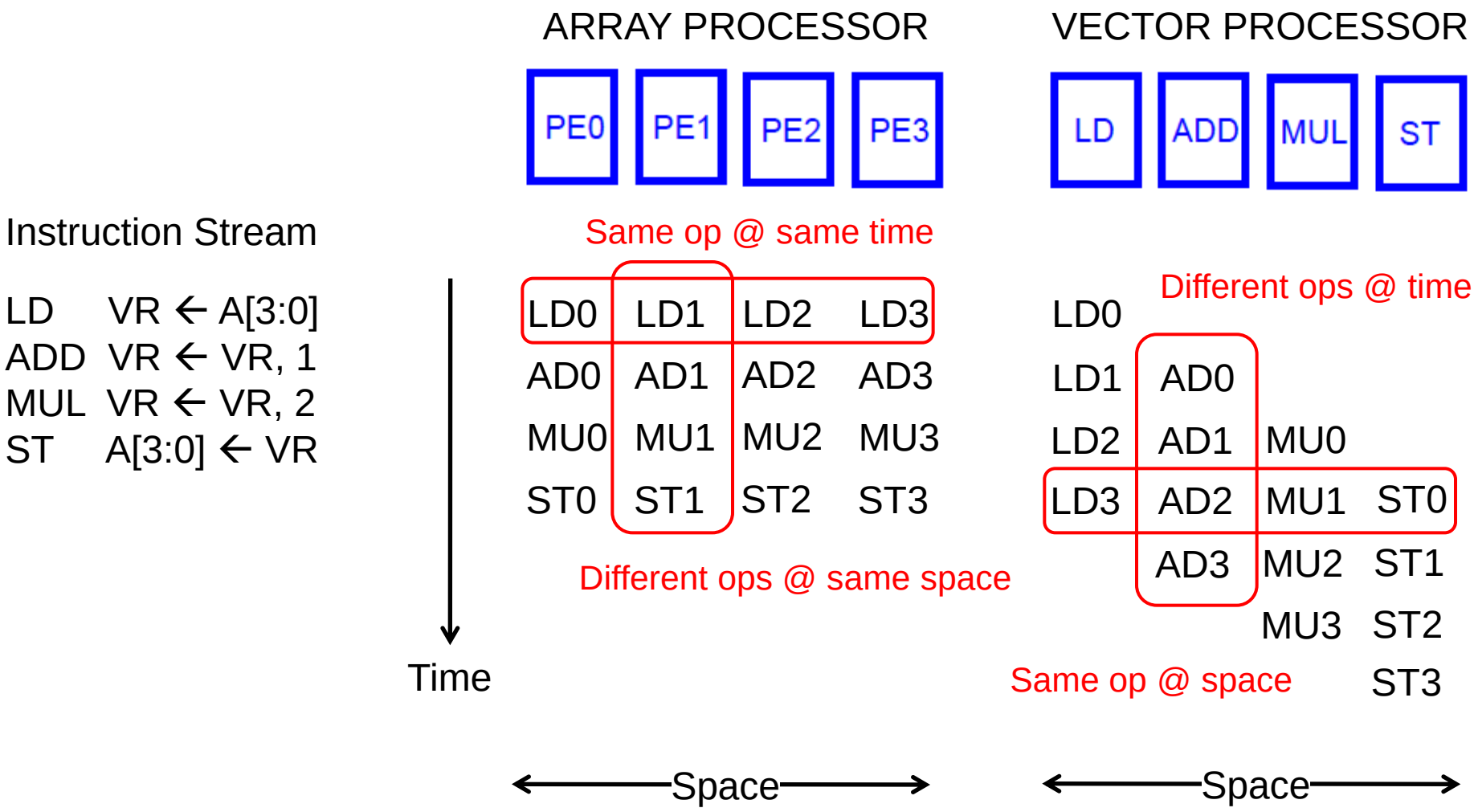
Rau, "Pseudo-randomly Interleaved Memory," ISCA 1991.

Array vs. Vector Processors, Revisited

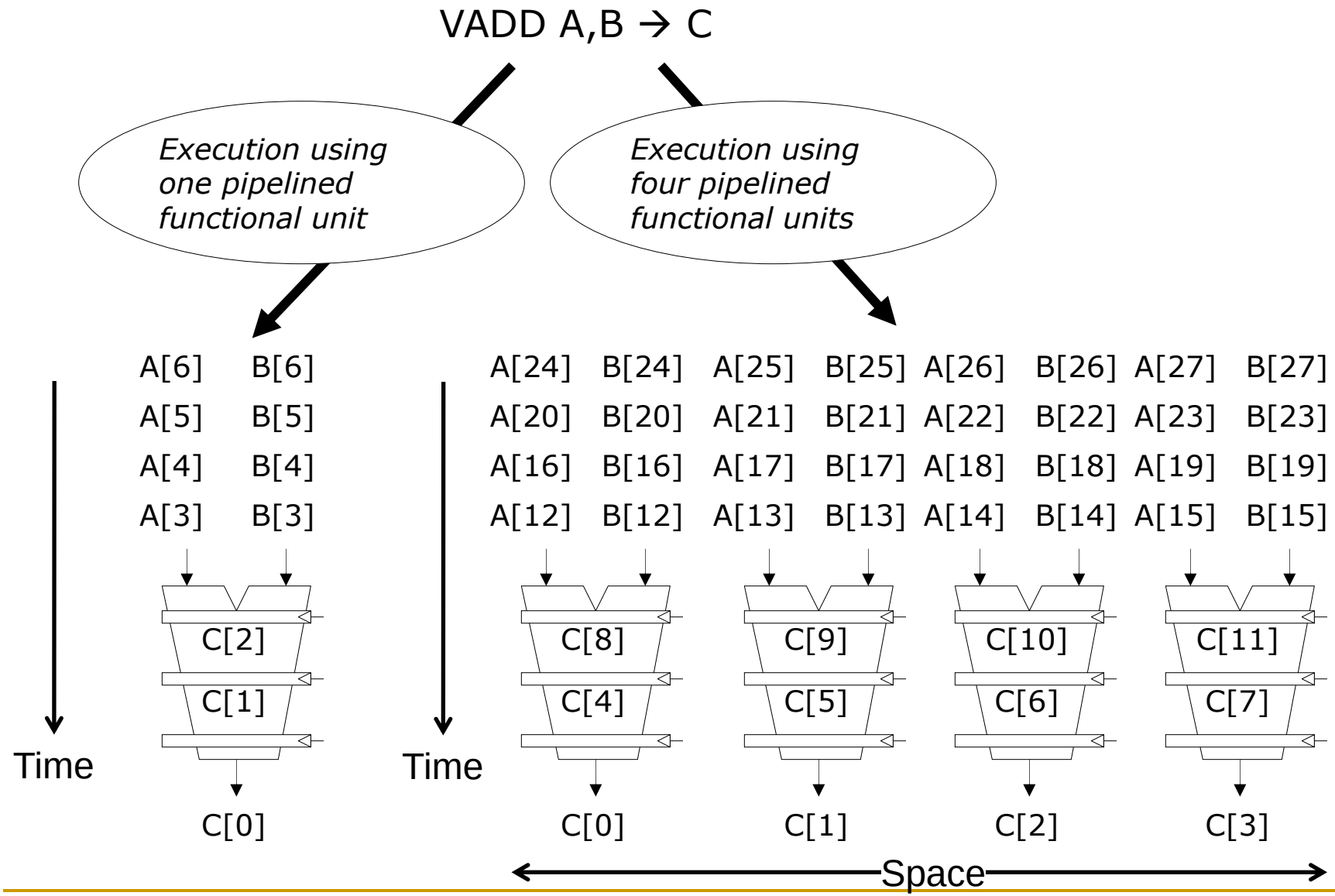
- Array vs. vector processor distinction is a "purist's" distinction

- Most "modern" SIMD processors are a combination of both
  - They exploit data parallelism in both time and space
  - GPUs are a prime example we will cover in a bit more detail

Recall: Array vs. Vector Processors

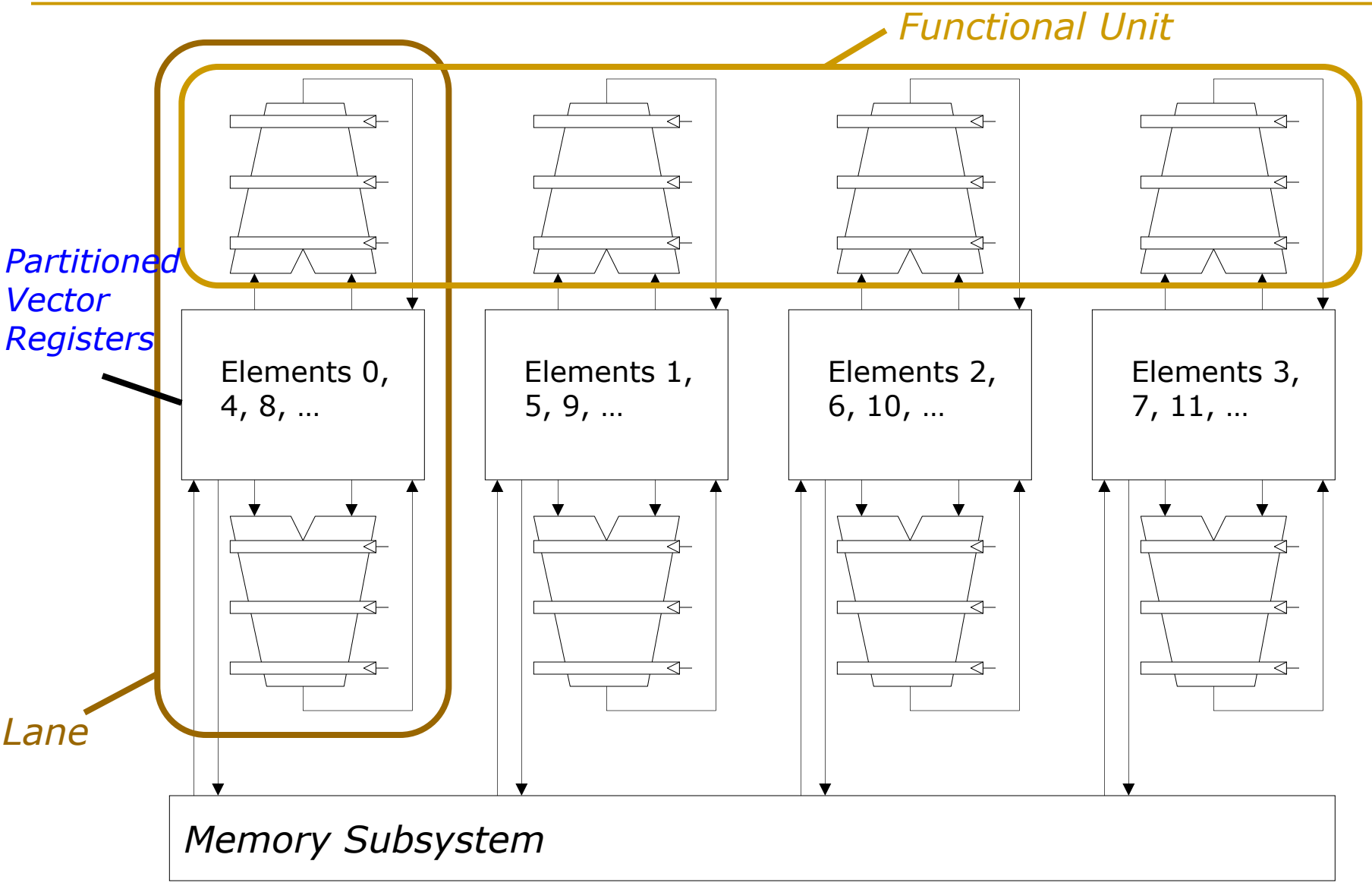


Vector Instruction Execution



Slide credit: Krste Asanovic

Vector Unit Structure

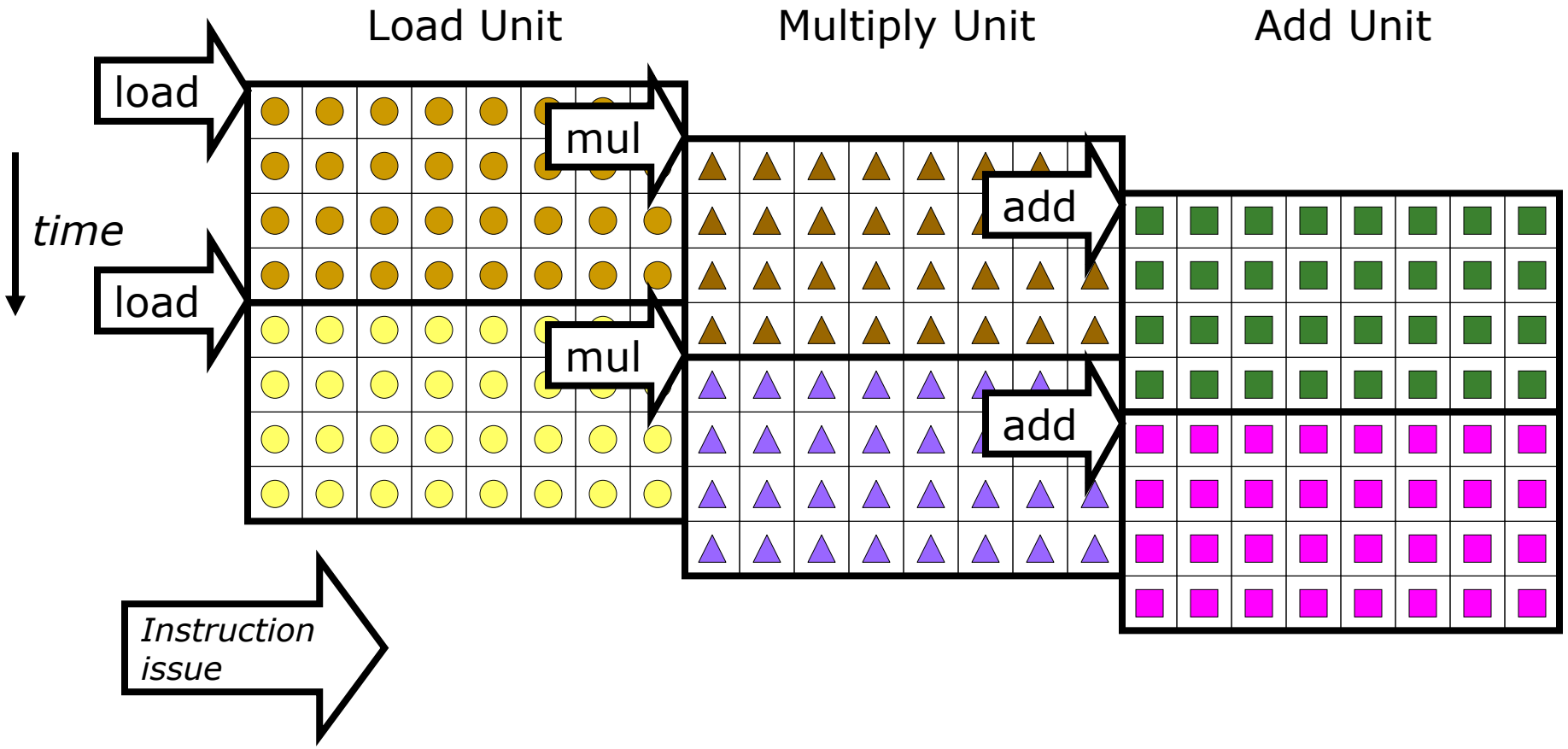


Slide credit: Krste Asanovic

Vector Instruction Level Parallelism

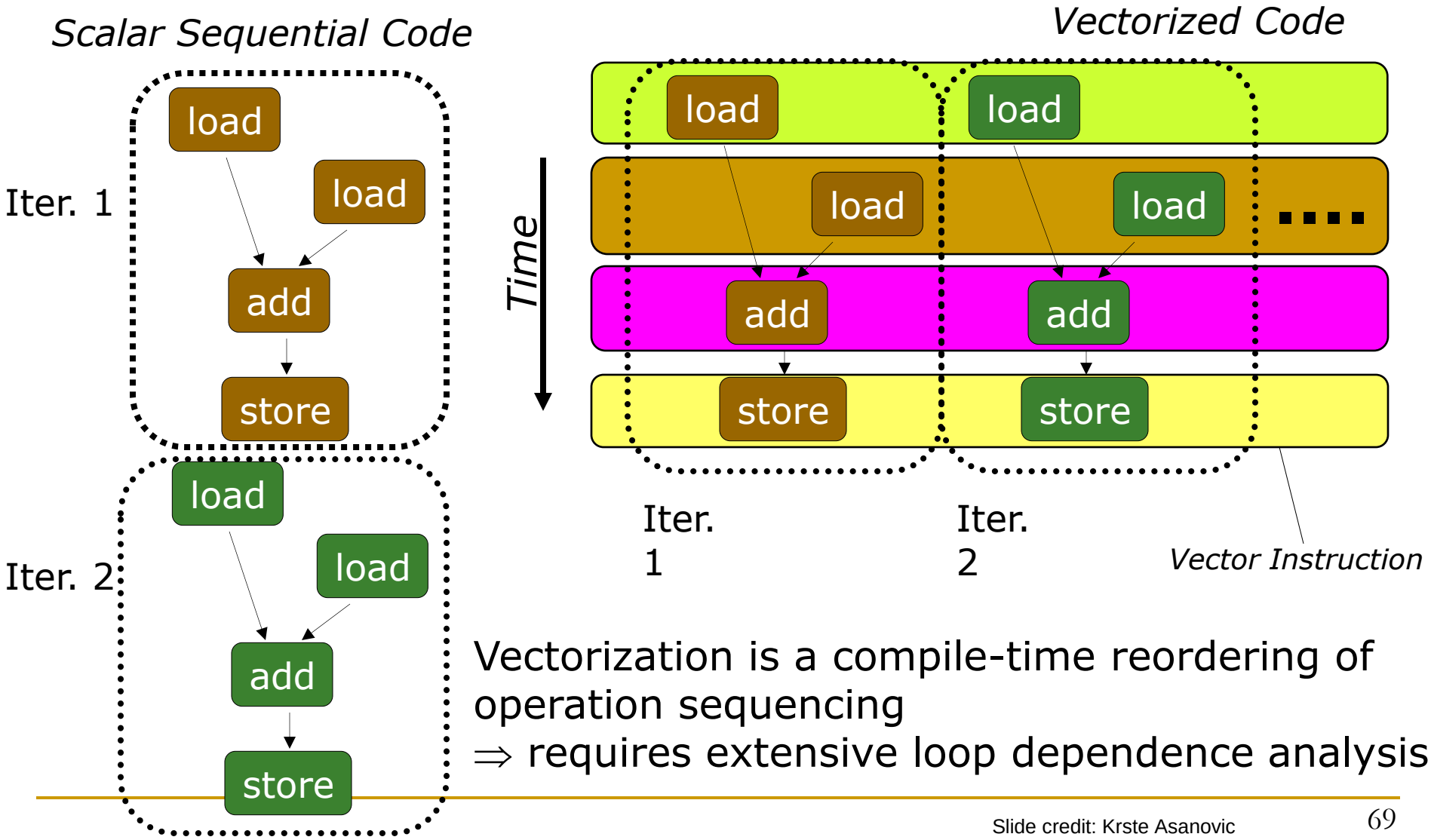
Can overlap execution of multiple vector instructions

- Example machine has 32 elements per vector register and 8 lanes
- Completes 24 operations/cycle while issuing 1 vector instruction/cycle



Automatic Code Vectorization

```
for (i=0; i < N; i++)
    C[i] = A[i] + B[i];
```



Vectorization is a compile-time reordering of operation sequencing

⇒ requires extensive loop dependence analysis

Slide credit: Krste Asanovic

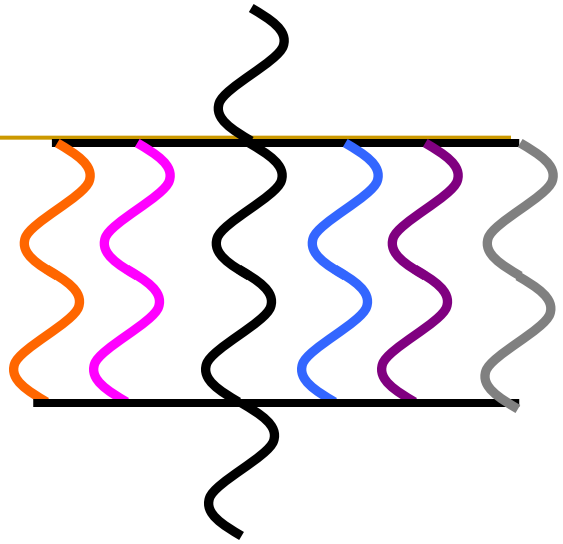
Vector/SIMD Processing Summary

- Vector/SIMD machines are good at exploiting regular data-level parallelism
  - Same operation performed on many data elements
  - Improve performance, simplify design (no intra-vector dependencies)

- Performance improvement limited by vectorizability of code
  - Scalar operations limit vector machine performance
  - Remember Amdahl's Law
  - CRAY-1 was the fastest SCALAR machine at its time!

- Many existing ISAs include (vector-like) SIMD operations
  - Intel MMX/SSEn/AVX, PowerPC AltiVec, ARM Advanced SIMD

Recall: Amdahl's Law



- Amdahl's Law
  - f: Parallelizable fraction of a program
  - N: Number of processors

$$\text{Speedup} = \frac{1}{1 - f + \frac{f}{N}}$$

  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.

- Maximum speedup limited by serial portion: Serial bottleneck

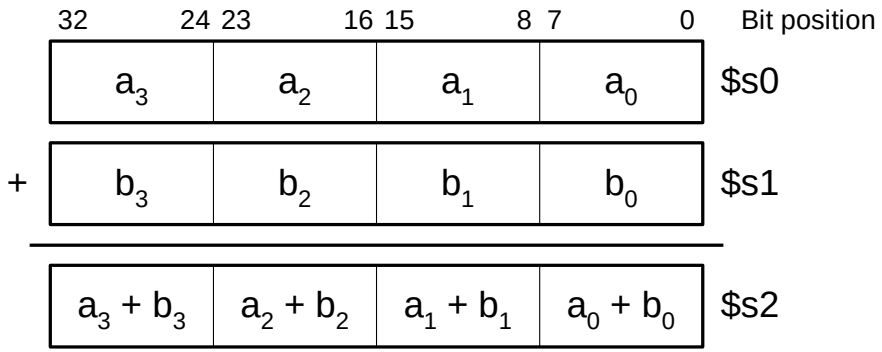
- All parallel machines "suffer from" the serial bottleneck

SIMD Operations in Modern ISAs

SIMD ISA Extensions

- Single Instruction Multiple Data (SIMD) extension instructions
  - Single instruction acts on multiple pieces of data at once
  - Common application: graphics
  - Perform short arithmetic operations (also called *packed arithmetic*)
- For example: add four 8-bit numbers
- Must modify ALU to eliminate carries between 8-bit values

```
padd8 $s2, $s0, $s1
```



Intel Pentium MMX Operations

- Idea: One instruction operates on multiple data elements simultaneously
  - *À la* array processing (yet much more limited)
  - Designed with multimedia (graphics) operations in mind

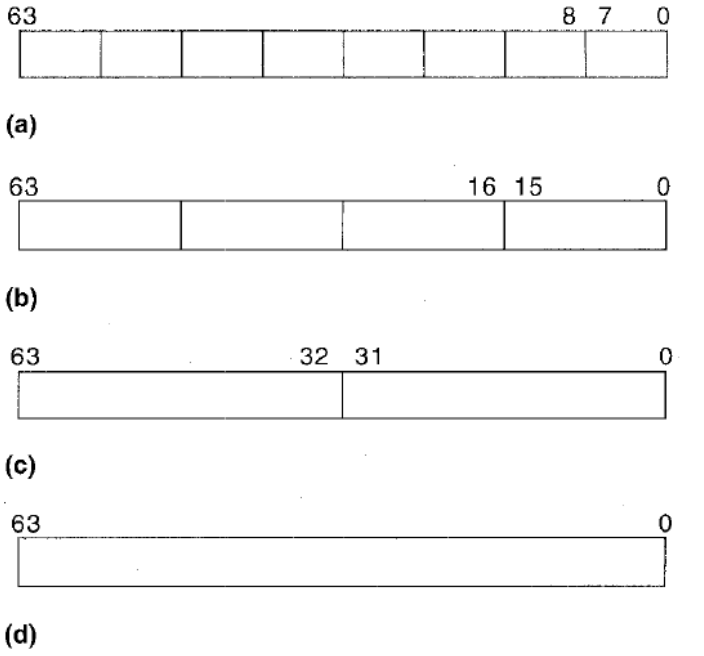


Figure 1. MMX technology data types: packed byte (a), packed word (b), packed doubleword (c), and quadword (d).

No VLEN register
Opcode determines data type:
8 8-bit bytes
4 16-bit words
2 32-bit doublewords
1 64-bit quadword

Stride is always equal to 1.

Peleg and Weiser, "MMX Technology Extension to the Intel Architecture," IEEE Micro, 1996.

MMX Example: Image Overlaying (I)

- Goal: Overlay the human in image x on top of the background in image y

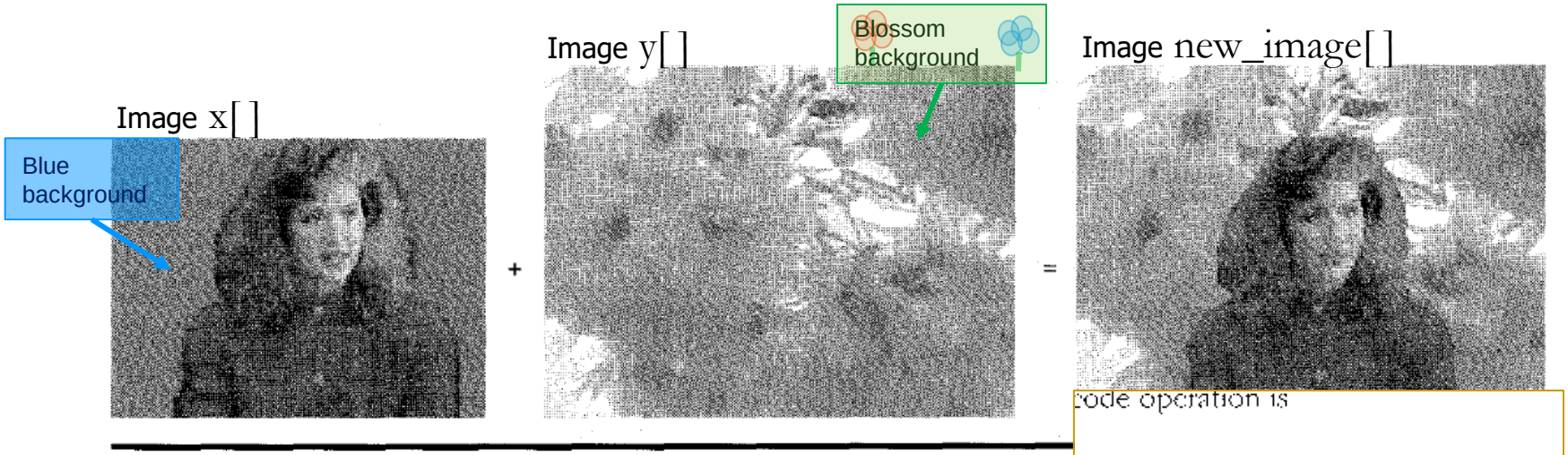


Figure 8. Chroma keying: image overlay using a background color.

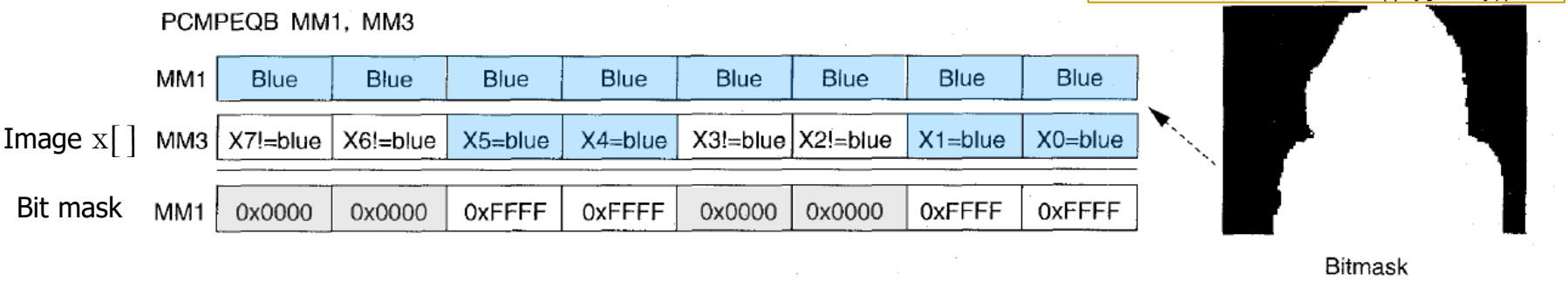
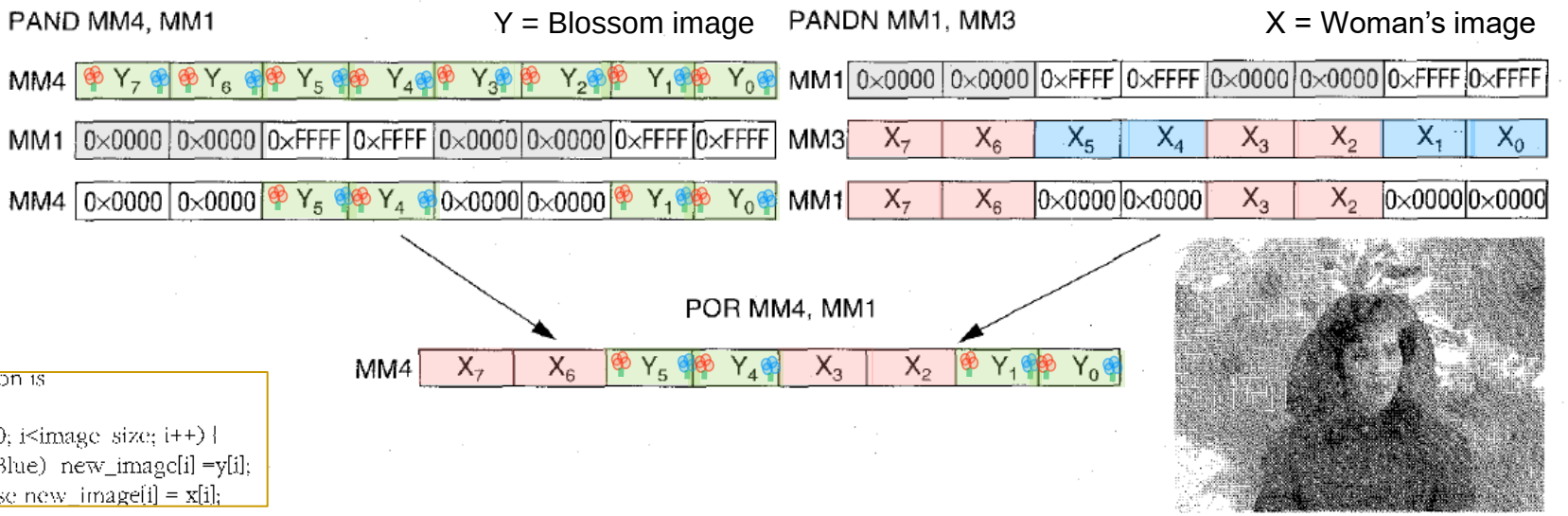


Figure 9. Generating the selection bit mask.

MMX Example: Image Overlaying (II)



code operation is

```
for (i=0; i<image_size; i++) {
if (x[i] == Blue)  new_image[i] =y[i];
        else new_image[i] = x[i];
```

Figure 10. Using the mask with logical MMX instructions to perform a conditional select.

```
Movq    mm3, mem1    /* Load eight pixels from
                        woman's image
Movq    mm4, mem2    /* Load eight pixels from the
                        blossom image
Pcmpeqb mm1, mm3
Pand    mm4, mm1
Pandn   mm1, mm3
Por     mm4, mm1
```

Figure 11. MMX code sequence for performing a conditional select.

Peleg and Weiser, "MMX Technology Extension to the Intel Architecture," IEEE Micro, 1996.

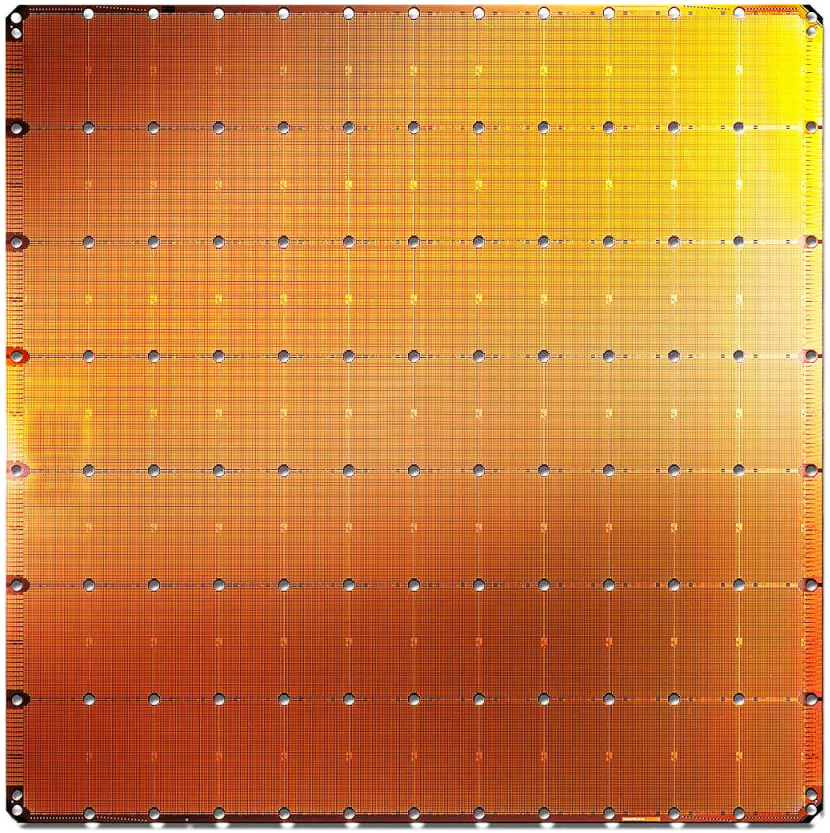
From MMX to AMX in x86 ISA

- MMX
  - 64-bit MMX registers for integers
- SSE (Streaming SIMD Extensions)
  - SSE-1: 128-bit XMM registers for integers and single-precision floating point
  - SSE-2: Double-precision floating point
  - SSE-3, SSSE-3 (supplemental): New instructions
  - SSE-4: New instructions (not multimedia specific), shuffle operations
- AVX (Advanced Vector Extensions)
  - AVX: 256-bit floating point
  - AVX2: 256-bit floating point with FMA (Fused Multiply Add)
  - AVX-512: 512-bit
- AMX (Advanced Matrix Extensions)
  - Designed for AI/ML workloads
  - 2-dimensional registers
  - Tiled matrix multiply unit (TMUL)

https://www.intel.sg/content/dam/www/public/apac/xa/en/pdfs/ssg/Intel_Processor_Architecture_SIMD_Instructions.pdf
https://gcc.gnu.org/wiki/cauldron2014?action=AttachFile&do=get&target=Cauldron14_AVX-512_Vector_ISA_Kirill_Yukhin_20140711.pdf

SIMD Operations in Modern (Machine Learning) Accelerators

Cerebras's Wafer Scale Engine (2019)



- The largest ML accelerator chip (2019)
- 400,000 cores

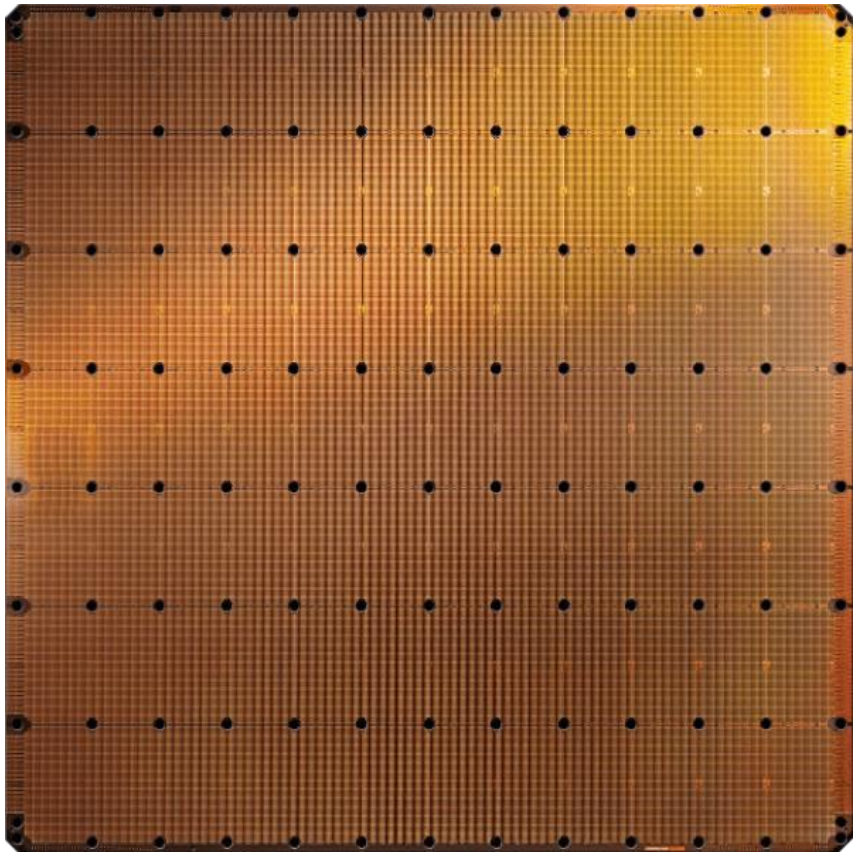
Cerebras WSE
1.2 Trillion transistors
46,225 mm^2

Largest GPU
21.1 Billion transistors
815 mm^2
NVIDIA TITAN V

https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning

https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

Cerebras's Wafer Scale Engine-2 (2021)



- The largest ML accelerator chip (2021)
- 850,000 cores

Cerebras WSE-2
2.6 Trillion transistors
46,225 mm^2

Largest GPU
54.2 Billion transistors
826 mm^2
NVIDIA Ampere GA100

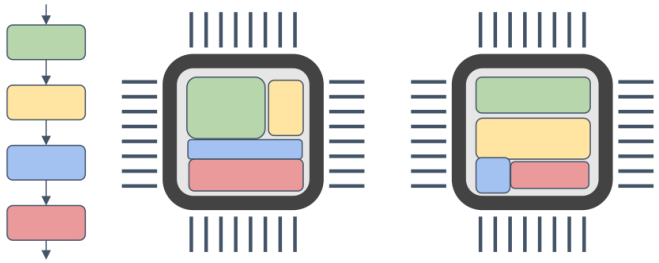
https://www.anandtech.com/show/14758/hot-chips-31-live-blogs-cerebras-wafer-scale-deep-learning

https://www.cerebras.net/cerebras-wafer-scale-engine-why-we-need-big-chips-for-deep-learning/

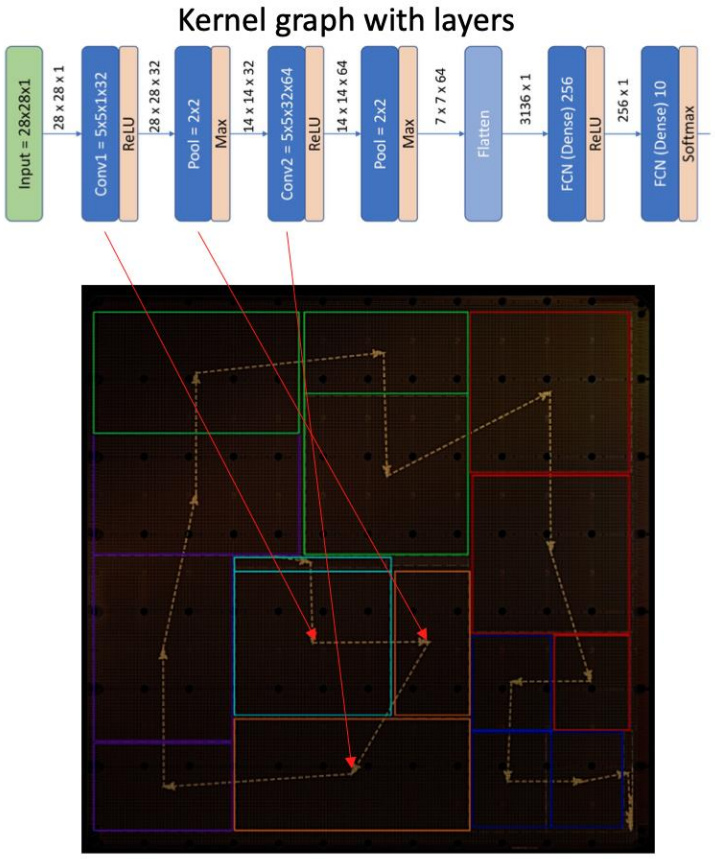
Size, Place, and Route in Cerebras's WSE

- Neural network mapping onto the whole wafer is a challenge

Multiple possible mappings



An example mapping



Layers mapped on Wafer Scale Engine

Different dies of the wafer work on different layers of the neural network: MIMD machine

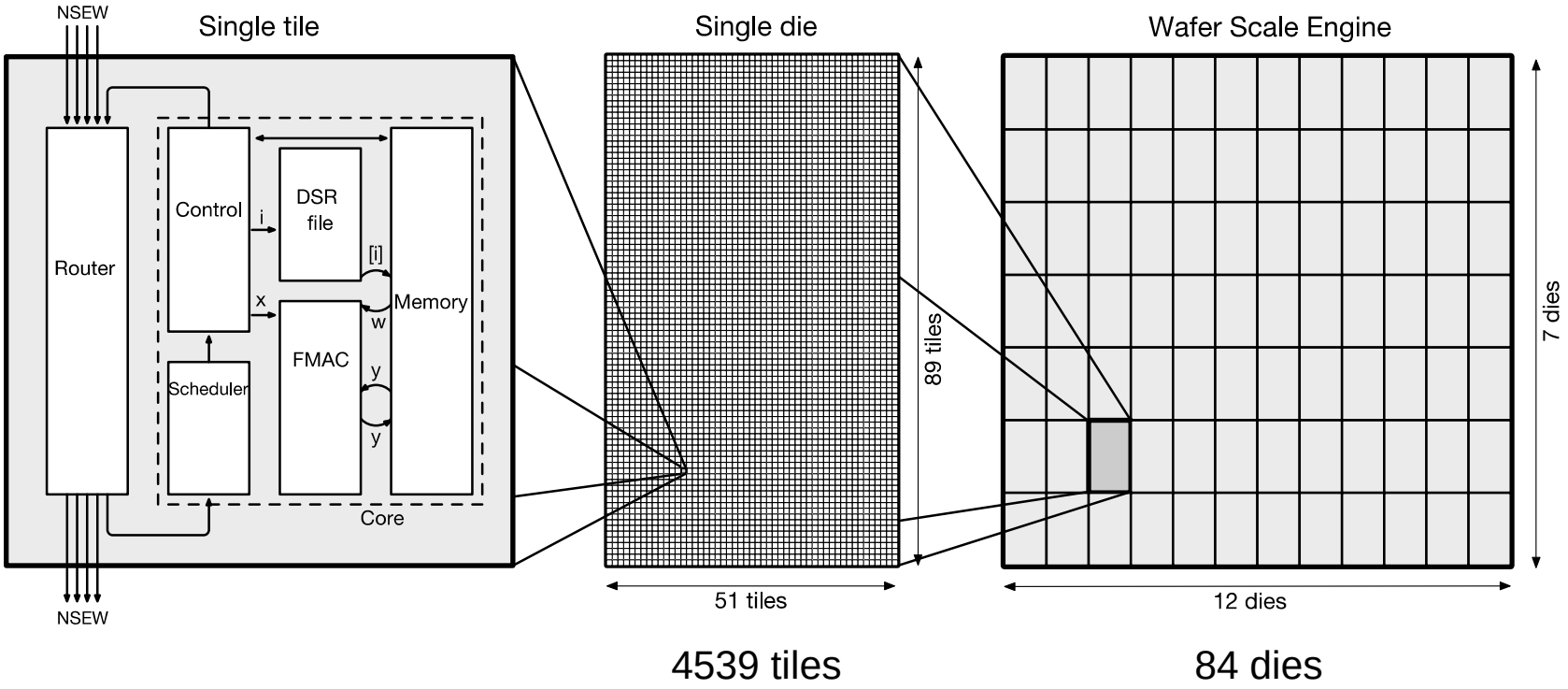
Recall: Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- SISD: Single instruction operates on single data element
- SIMD: Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

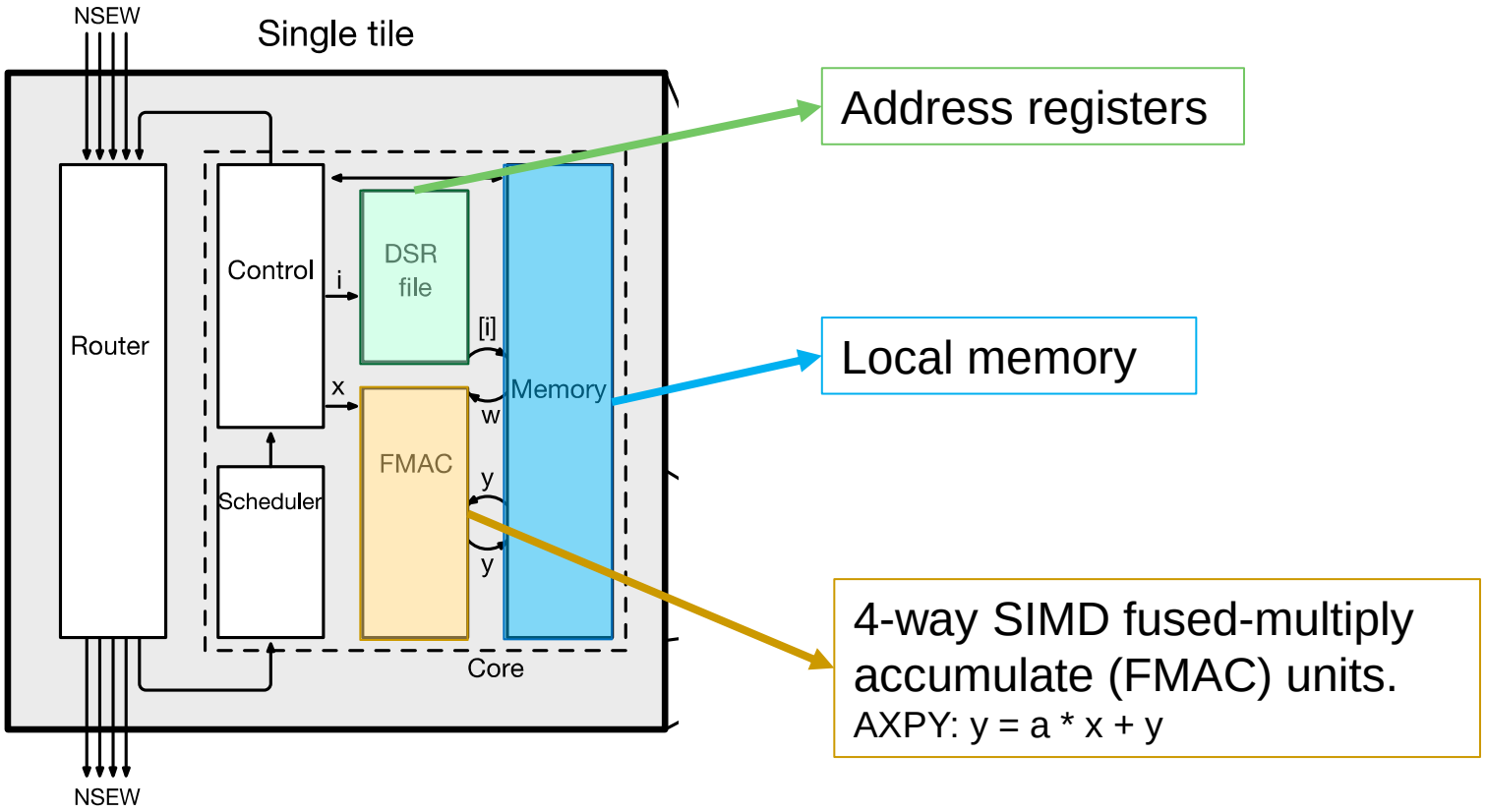
A MIMD Machine with SIMD Processors (I)

- MIMD machine
  - Distributed memory (no shared memory)
  - 2D-mesh interconnection fabric



A MIMD Machine with SIMD Processors (II)

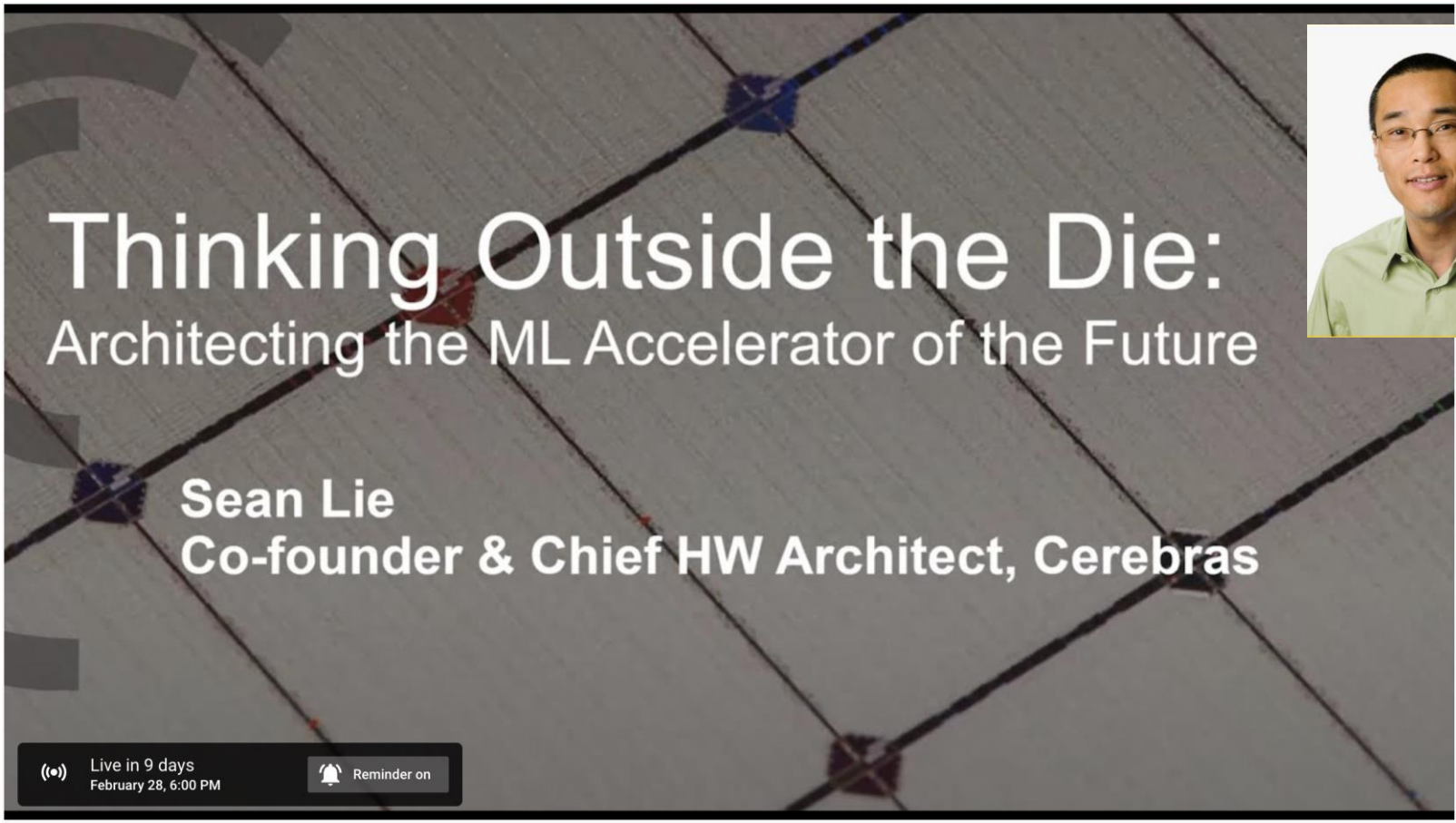
- SIMD processors
  - 4-way SIMD for 16-bit floating point operands
  - 48 KB of local SRAM



Rocki et al., "Fast stencil-code computation on a wafer-scale processor." SC 2020.

More on the Cerebras WSE

https://www.youtube.com/watch?v=x2-qB0J7KHw



Digital Design & Computer Arch.

Lecture 20: SIMD Processors

Dr. Juan Gómez Luna
Prof. Onur Mutlu
ETH Zürich

Spring 2022

12 May 2022