Advanced Computer Architectures

(High Performance Processors and Systems)

Multiprocessors

Politecnico di Milano

v1

Alessandro Verosimile <alessandro.verosimile@polimi.it>

Marco D. Santambrogio <marco.santambrogio@polimi.it>

Outline

- Multiprocessors
  - Flynn taxonomy
  - SIMD architectures
  - Vector architectures
  - MIMD architectures

Flynn Taxonomy (1966)

- SISD - Single Instruction Single Data
  - Uniprocessor systems
- MISD - Multiple Instruction Single Data
  - No practical configuration and no commercial systems
- SIMD - Single Instruction Multiple Data
  - Simple programming model, low overhead, flexibility, custom integrated circuits
- MIMD - Multiple Instruction Multiple Data
  - Scalable, fault tolerant, *off-the-shelf* micros

Which kind of multiprocessors?

- Many of the early multiprocessors were SIMD – SIMD model received great attention in the ’80’s (vector processors, multimedia instructions) and today they are getting back to the spotlight!

- MIMD has emerged as architecture of choice for general-purpose multiprocessors

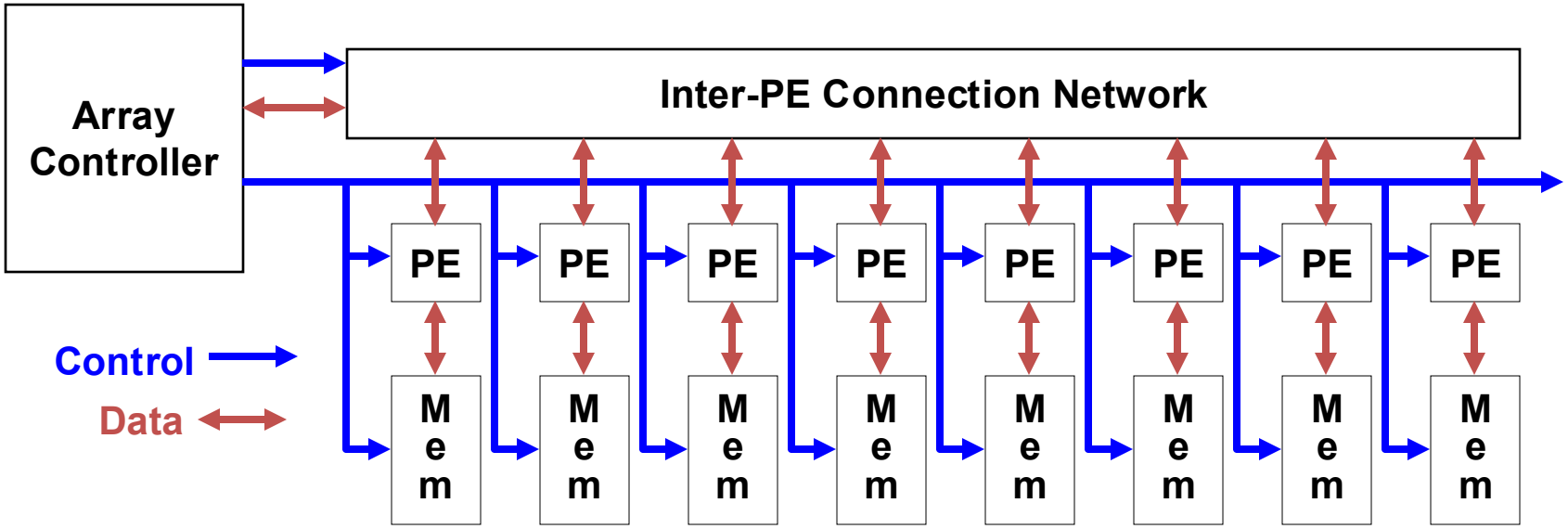
- Lets see these architectures more in details..

SIMD - Single Instruction Multiple Data

- Same instruction executed by multiple processors using different data streams.
- Each processor has its own data memory.
- Single instruction memory and control processor to fetch and dispatch instructions
- Processors are typically special-purpose.
- Simple programming model.

SIMD Architecture

Central controller broadcasts instructions to multiple processing elements (PEs)



- ✓ Only requires one controller for whole array
- ✓ Only requires storage for one copy of program
- ✓ All computations fully synchronized

SIMD model

- Synchronized units: single Program Counter
- Each unit has its own addressing registers
  - Can use different data addresses
- Motivations for SIMD:
  - Cost of control unit shared by all execution units
  - Only one copy of the code in execution is necessary
- Real life:
  - SIMD have a mix of SISD instructions and SIMD
  - A host computer executes sequential operations
  - SIMD instructions sent to all the execution units, which has its own memory and registers and exploit an interconnection network to exchange data

Reality: Sony Playstation 2000

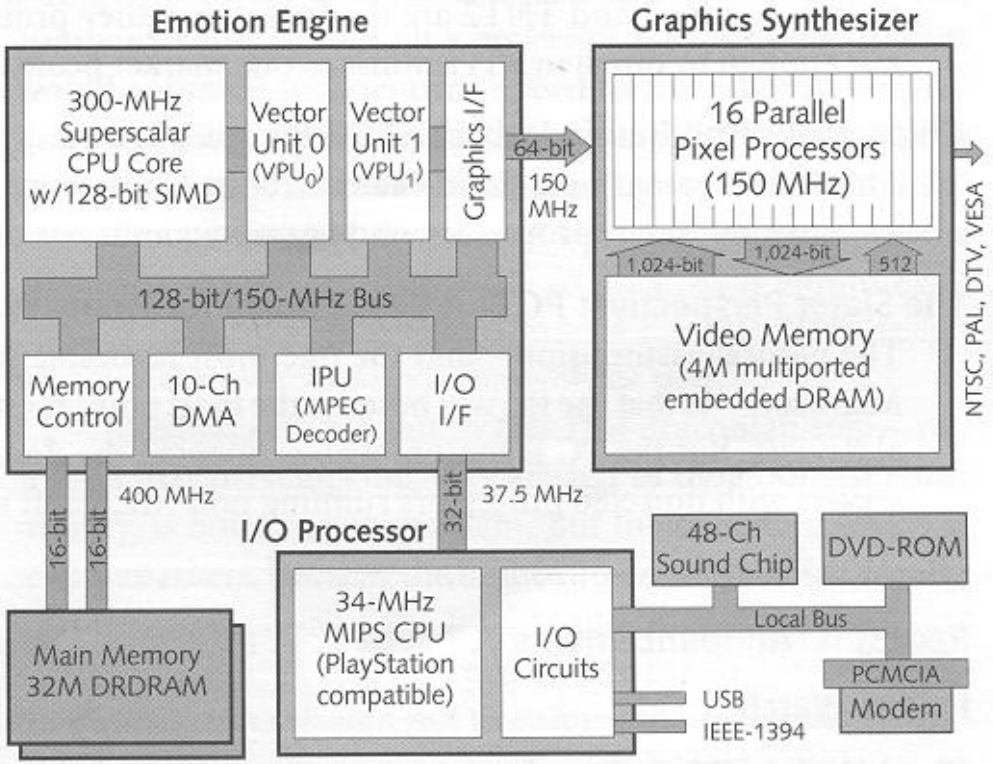
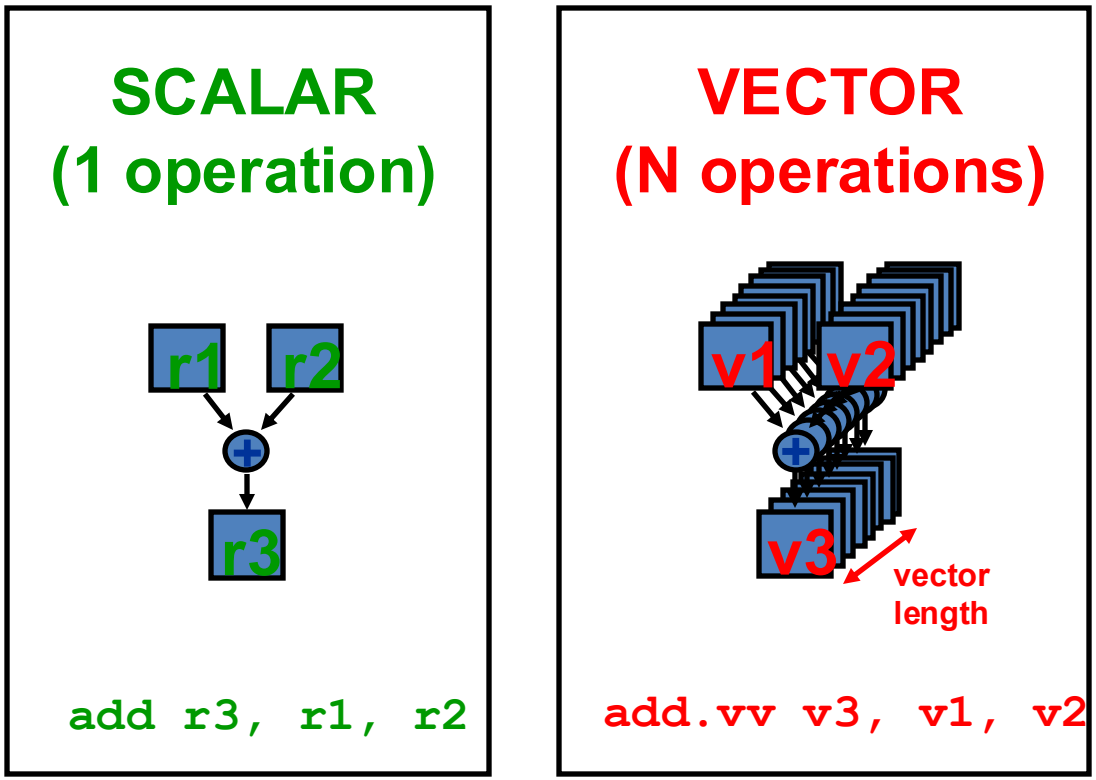


Figure 1. PlayStation 2000 employs an unprecedented level of parallelism to achieve workstation-class 3D performance.

Figure 2. PlayStation 2000 screenshot. (Source: Namco)

Alternative Model: Vector Processing

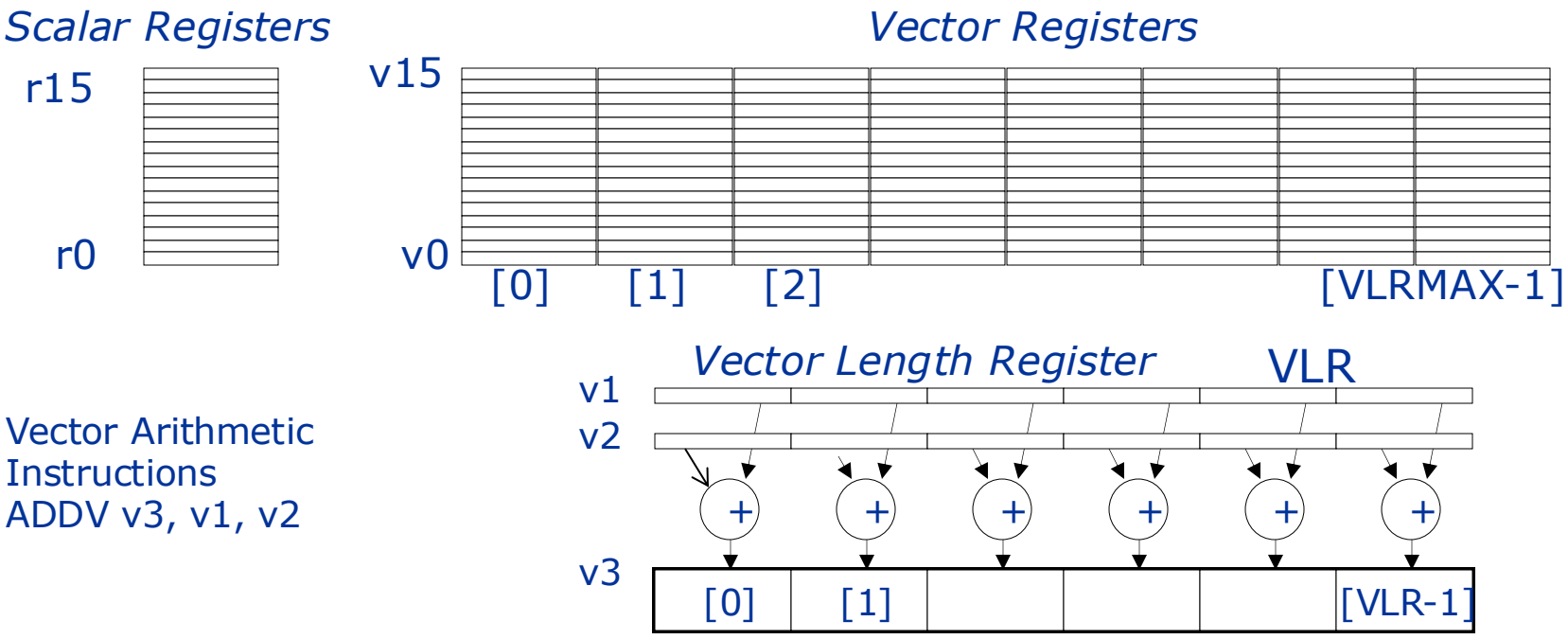
- Vector processors have high-level operations that work on linear arrays of numbers: "vectors"



Styles of Vector Architectures

- A vector processor consists of a pipelined scalar unit (may be out-of order or VLIW) + vector unit
- *memory-memory vector processors*: all vector operations are memory to memory
- *vector-register processors*: all vector operations between vector registers (except load and store)
  - Vector equivalent of load-store architectures
  - Includes all vector machines since late 1980s: Cray, Convex, Fujitsu, Hitachi, NEC

Vector programming model



Vector Arithmetic Instructions
ADDV v3, v1, v2

Vector Code Example

```
# C code
for (i=0;i<64; i++)
  C[i] = A[i]+B[i];
```

Vector Code Example

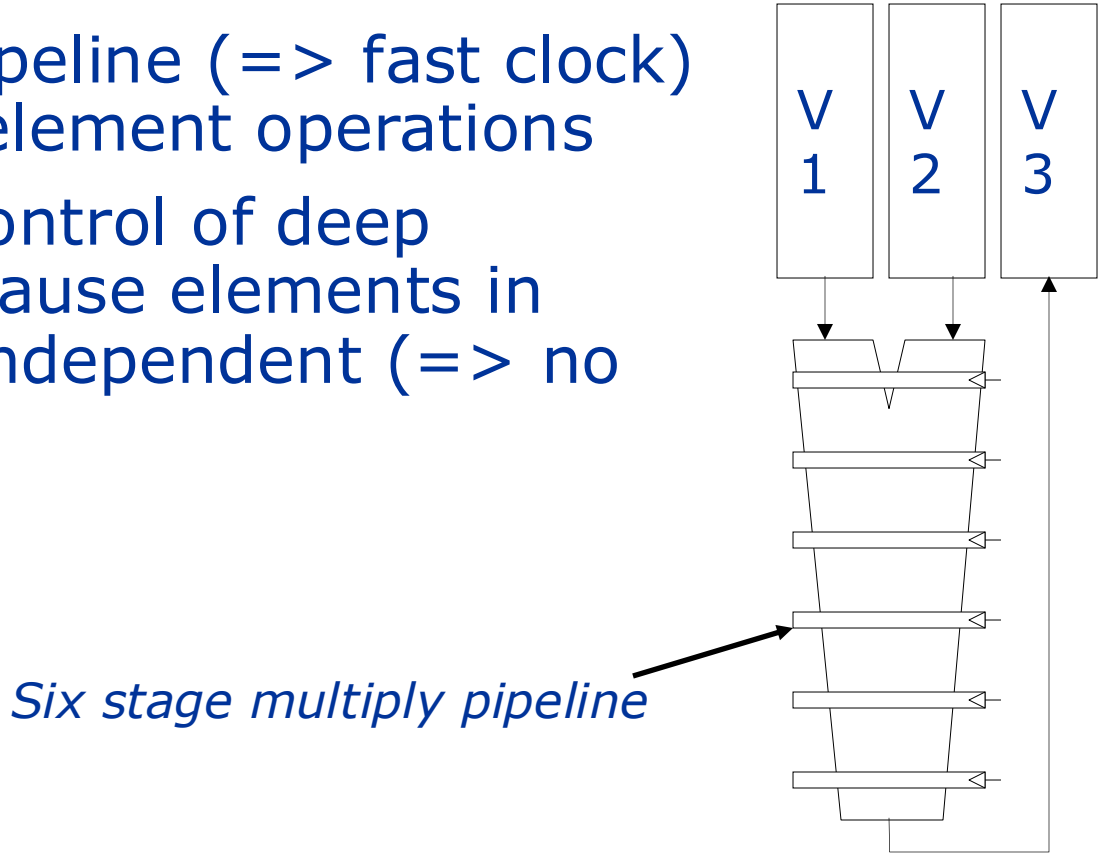
# C code	# Scalar Code	
for (i=0;i<64; i++) C[i] = A[i]+B[i];	LI R4, #64 loop: L.D F0, 0(R1) L.D F2, 0(R2) ADD.D F4, F2, F0 S.D F4, 0(R3) DADDIU R1, 8 DADDIU R2, 8 DADDIU R3, 8 DSUBIU R4, 1 BNEZ R4, loop	

Vector Code Example

# C code	# Scalar Code	# Vector Code
for (i=0;i<64; i++) C[i] = A[i]+B[i];	LI R4, #64 loop: L.D F0, 0(R1) L.D F2, 0(R2) ADD.D F4, F2, F0 S.D F4, 0(R3) DADDIU R1, 8 DADDIU R2, 8 DADDIU R3, 8 DSUBIU R4, 1 BNEZ R4, loop	LI VLR, #64 LV V1, R1 LV V2, R2 ADDV.D V3,V1,V2 SV V3, R3

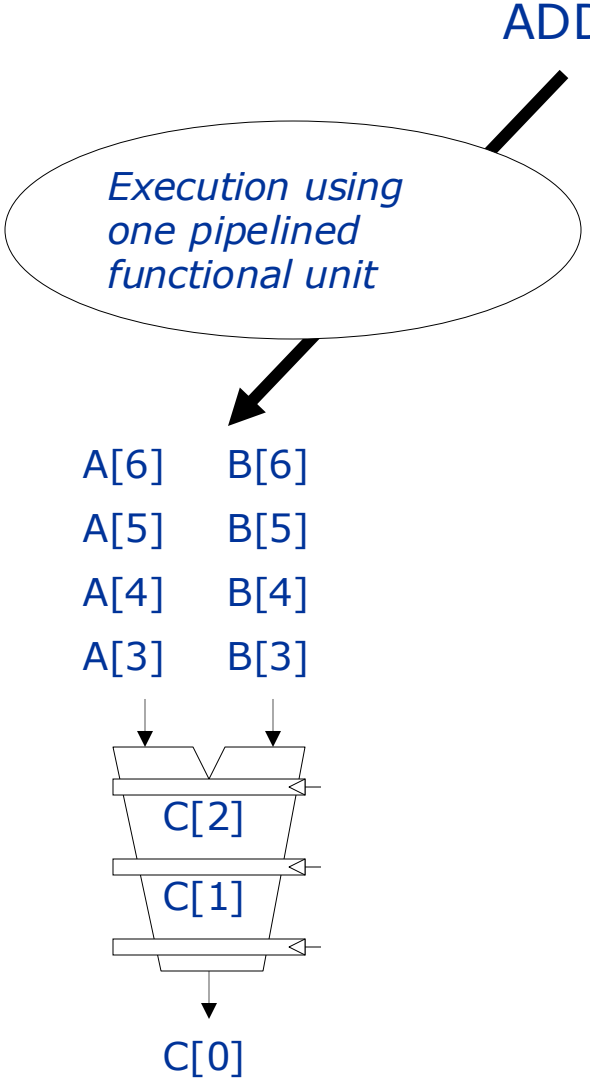
Vector Arithmetic Execution

- Use deep pipeline (=> fast clock) to execute element operations
- Simplifies control of deep pipeline because elements in vector are independent (=> no hazards!)

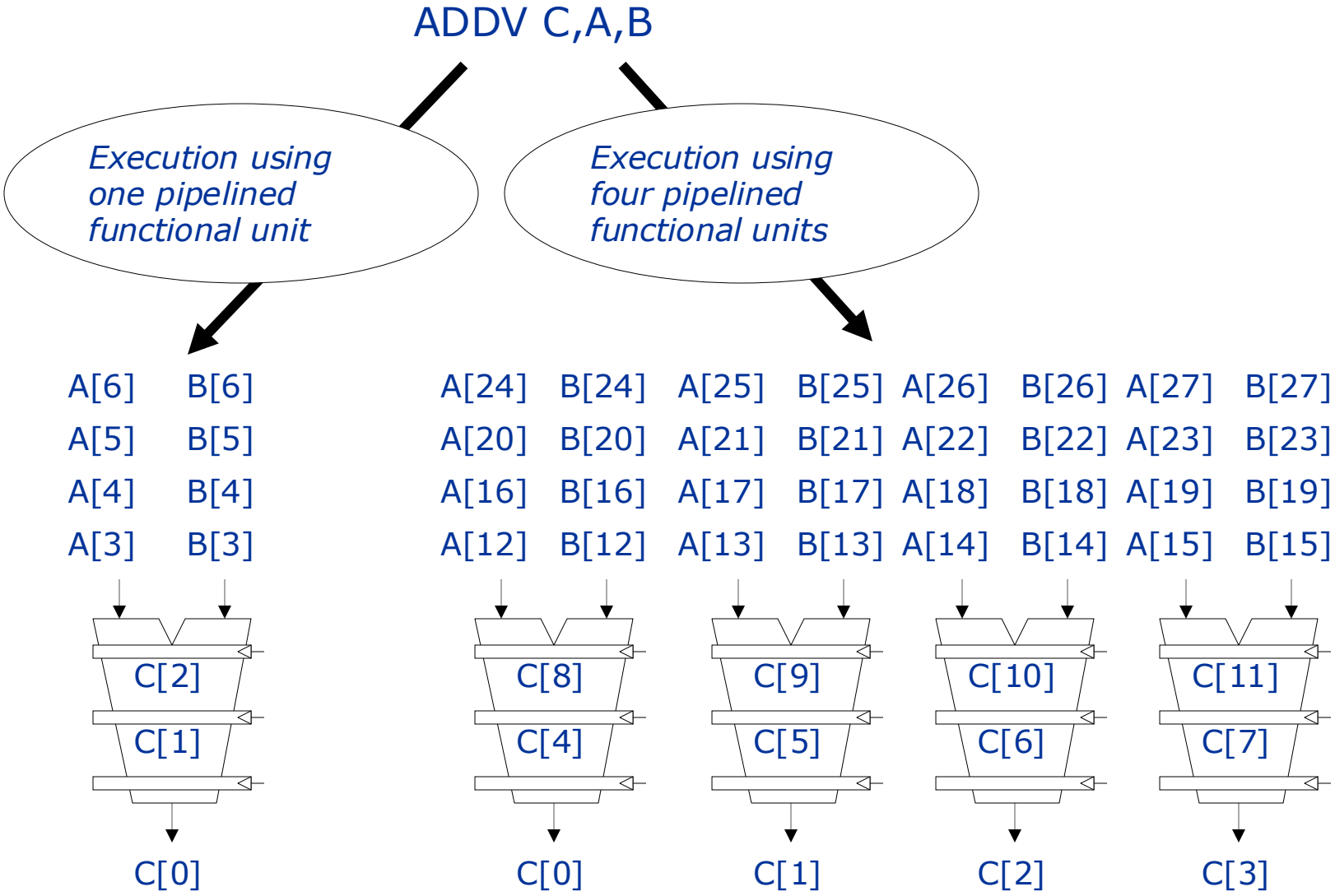


V3 ← v1 * v2

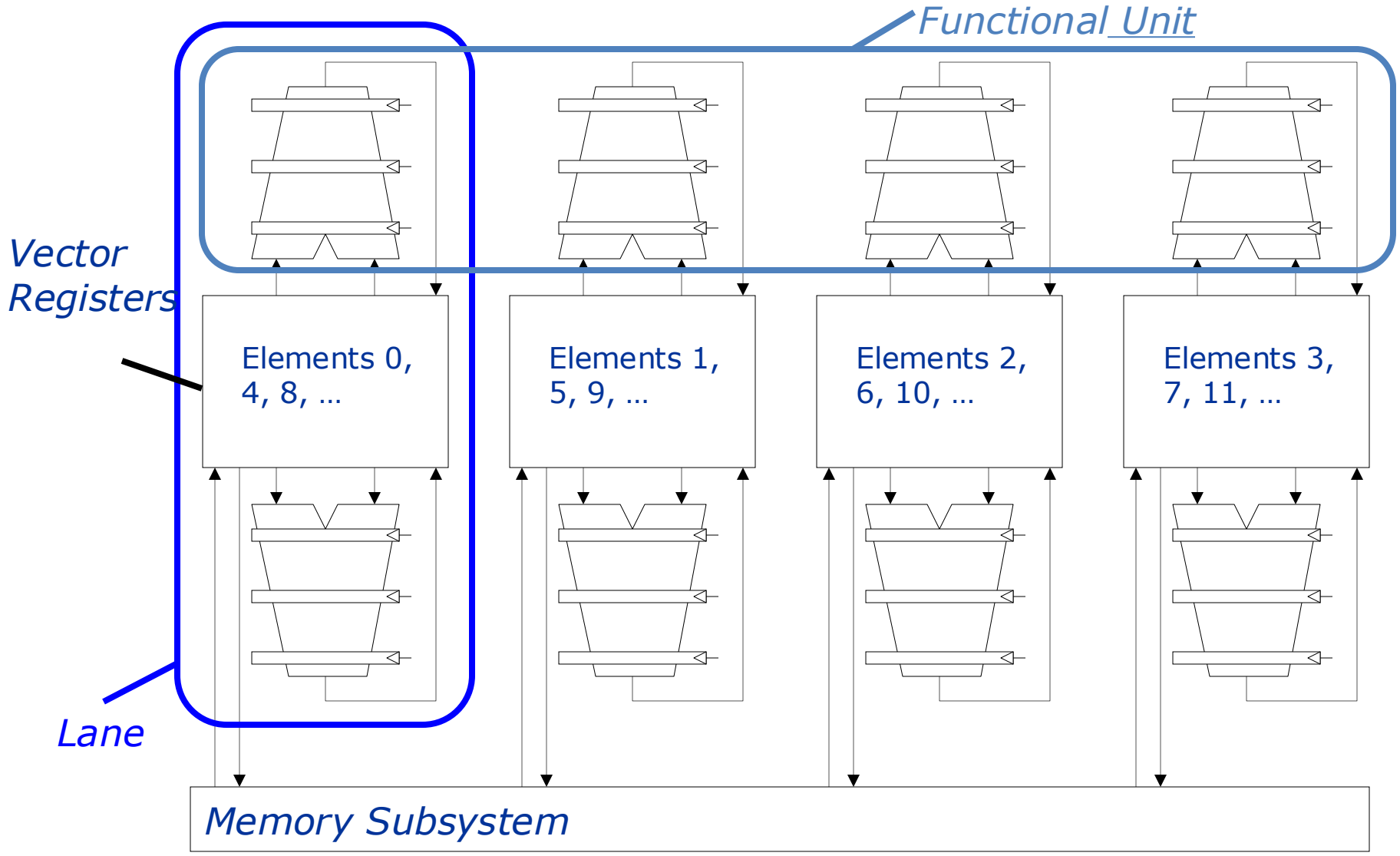
Vector Instruction Execution



Vector Instruction Execution



Vector Unit Structure



Vector Applications

Limited to scientific computing?

- Multimedia Processing (compress., graphics, audio synth, image proc.)
- Standard benchmark kernels (Matrix Multiply, FFT, Convolution, Sort)
- Lossy Compression (JPEG, MPEG video and audio)
- Lossless Compression (Zero removal, RLE, Differencing, LZW)
- Cryptography (RSA, DES/IDEA, SHA/MD5)
- Speech and handwriting recognition
- Operating systems/Networking (`memcpy`, `memset`, parity, checksum)
- Databases (hash/join, data mining, image/video serving)
- Language run-time support (stdlib, garbage collection)
- even SPECint95

Why MIMD?

Why MIMD?

- MIMDs are flexible – they can function as single-user machines for high performances on one application, as multiprogrammed multiprocessors running many tasks simultaneously, or as some combination of such functions;

Why MIMD?

- MIMDs are flexible – they can function as single-user machines for high performances on one application, as multiprogrammed multiprocessors running many tasks simultaneously, or as some combination of such functions;

- Can be built starting from standard CPUs (such is the present case nearly for all multiprocessors!).

MIMD - Multiple Instruction Multiple Data

- Each processor fetches its own instructions and operates on its own data.
- Processors are often off-the-shelf microprocessors.
- Scalable to a variable number of processor nodes.
- Flexible:
    - single-user machines focusing on high-performance for one specific application,
    - multi-programmed machines running many tasks simultaneously,
    - some combination of these functions.
- Cost/performance advantages due to the use of off-the-shelf microprocessors.
- Fault tolerance issues.

MIMD

- To exploit a MIMD with ***n*** processors
  - at least ***n*** threads or processes to execute
  - independent threads typically identified by the programmer or created by the compiler.
  - Parallelism is contained in the threads
    - *thread-level parallelism*.

MIMD

- To exploit a MIMD with ***n*** processors
  - at least ***n*** threads or processes to execute
  - independent threads typically identified by the programmer or created by the compiler.
  - Parallelism is contained in the threads
    - *thread-level parallelism*.

- Thread: from a large, independent process to parallel iterations of a loop. Important: *parallelism is identified by the software* (not by hardware as in superscalar CPUs!)... keep this in mind, we'll use it!

MIMD Machines

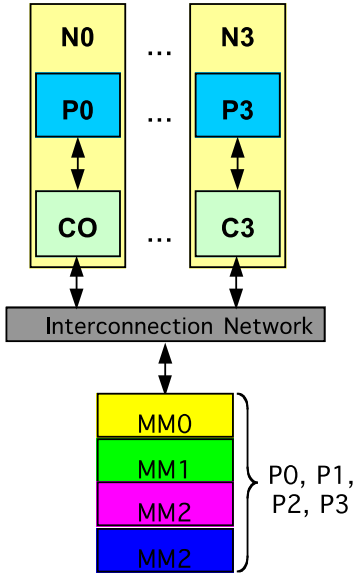
Existing MIMD machines fall into 2 classes, depending on the number of processors involved, which in turn dictates a memory organization and interconnection strategy.

MIMD Machines

Existing MIMD machines fall into 2 classes, depending on the number of processors involved, which in turn dictates a memory organization and interconnection strategy.

Centralized shared-memory architectures

- at most few dozen processor chips (< 100 cores)
- Large caches, single memory multiple banks
- Often called symmetric multiprocessors (SMP) and the style of architecture called Uniform Memory Access (UMA)

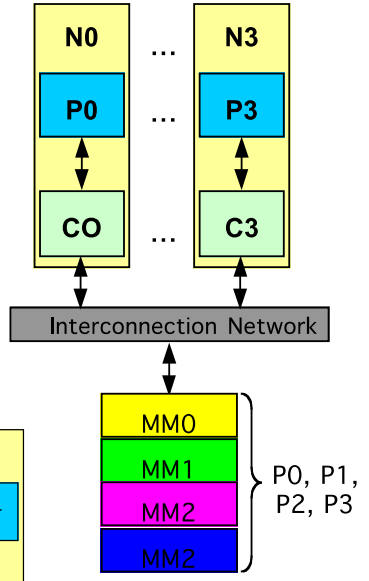


MIMD Machines

Existing MIMD machines fall into 2 classes, depending on the number of processors involved, which in turn dictates a memory organization and interconnection strategy.

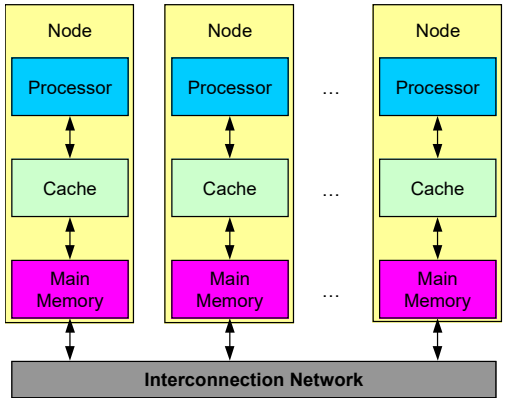
Centralized shared-memory architectures

- at most few dozen processor chips (< 100 cores)
- Large caches, single memory multiple banks
- Often called symmetric multiprocessors (SMP) and the style of architecture called Uniform Memory Access (UMA)



Distributed memory architectures

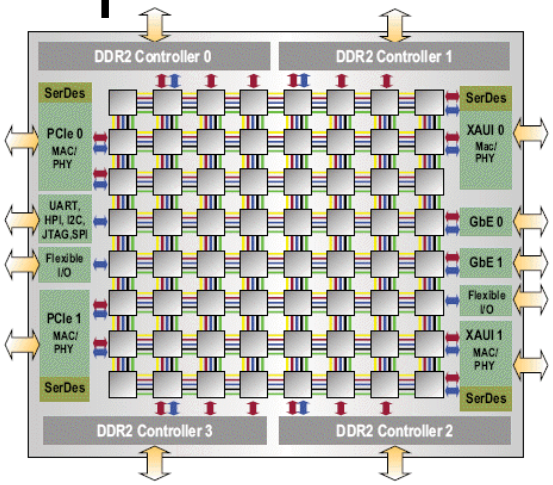
- To support large processor counts
- Requires high-bandwidth interconnect
- Disadvantage: data communication among processors

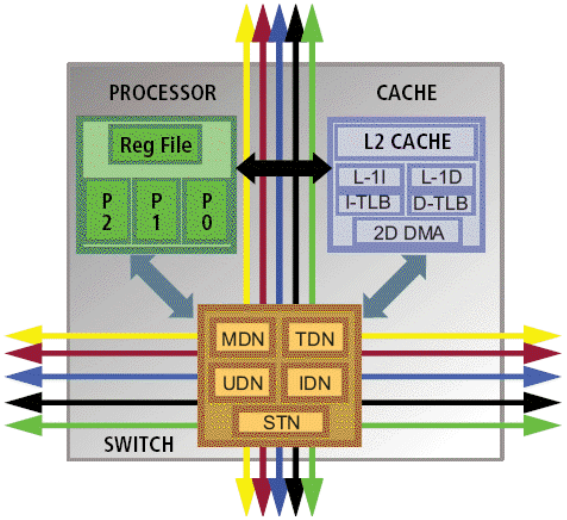


The Tilera Example

The Tilera Example

From Computer Desktop Encyclopedia
Reproduced with permission.
© 2007 Tilera Corporation





TILERA®

Key issues to design multiprocessors

- How many processors?
- How powerful are processors?
- How do parallel processors share data?
- Where to place the physical memory?
- How do parallel processors cooperate and coordinate?
- What type of interconnection topology?
- How to program processors?
- How to maintain cache coherency?
- How to maintain memory consistency?
- How to evaluate system performance?


CREATE THE MOST
AMAZING GAME CONSOLE

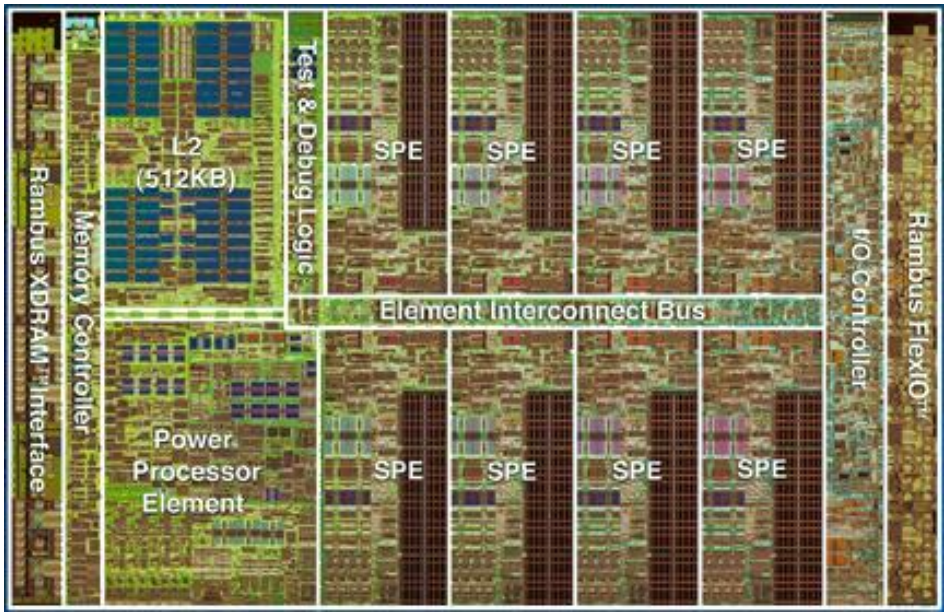
One core

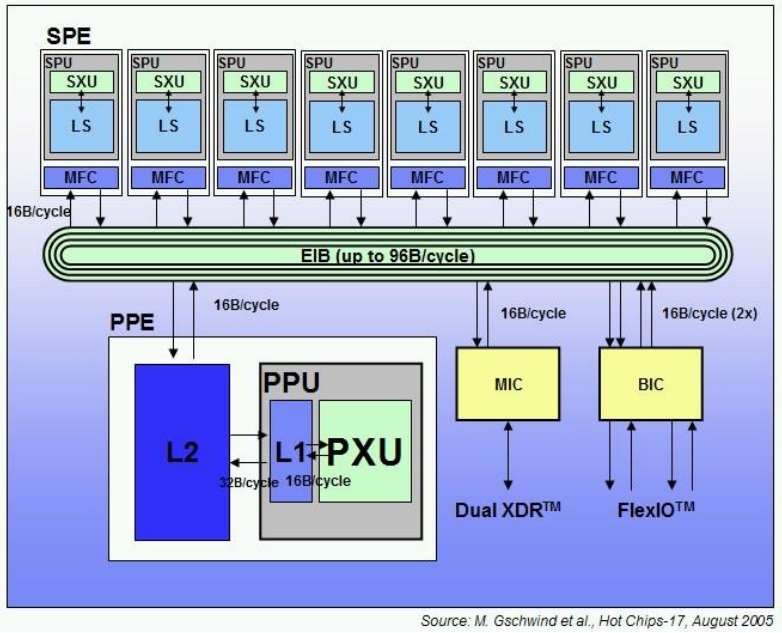
- A 64-bit Power Architecture core
- Two-issue superscalar execution
- Two-way multithreaded core
- In-order execution
- Cache
  - 32 KB instruction and a 32 KB data Level 1 cache
  - 512 KB Level 2 cache
  - The size of a cache line is 128 bytes

- One core to rule them all

Cell: PS3

- Cell is a heterogeneous chip multiprocessor
  - One 64-bit Power core
  - 8 specialized co-processors
    - based on a novel single-instruction multiple-data (SIMD) architecture called SPU (Synergistic Processor Unit)



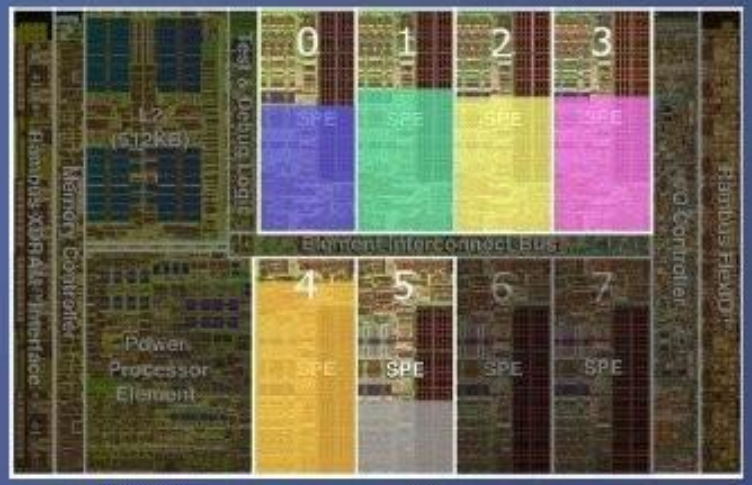


Source: M. Gschwind et al., Hot Chips-17, August 2005

Duck Demo

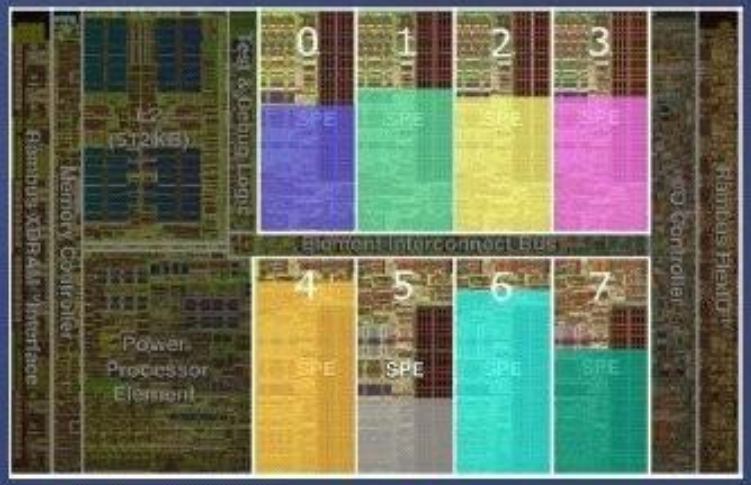
https://www.dropbox.com/s/f66abjcwp0kwf6z/DuckDemo.wmv?dl=0

Duck Demo SPE Usage



Old Code – no machine vision – 6 SPEs

- SPE0 – Surface water physics
- SPE1 – Splash physics
- SPE2 – Boat 1 physics
- SPE3 – Boat 2 physics
- SPE4 – Collision physics
- SPE5 – Graphics



Old Code - machine vision – 8 SPEs

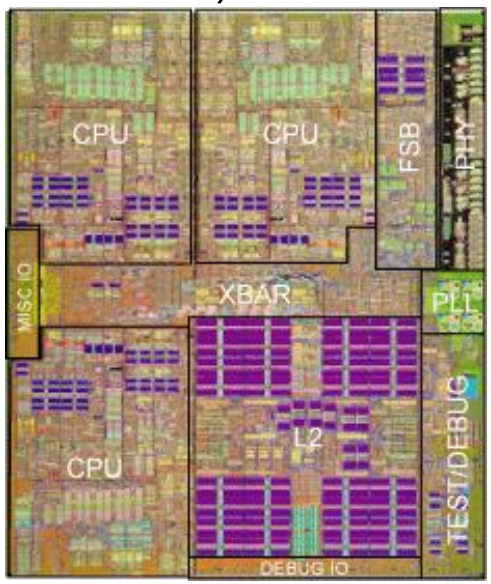
- SPE0-SPE5 UNCHANGED

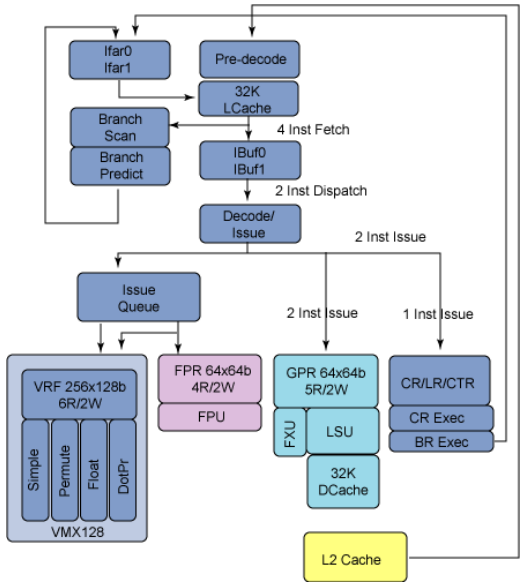
Added machine vision, particle water

- SPE6 – Particle water physics
- SPE7 – Machine vision

Xenon: XBOX360

- Three symmetrical cores
  - each two way SMT-capable and clocked at 3.2 GHz
- SIMD: VMX128 extension for each core
- 1 MB L2 cache (lockable by the GPU) running at half-speed (1.6 GHz) with a 256-bit bus



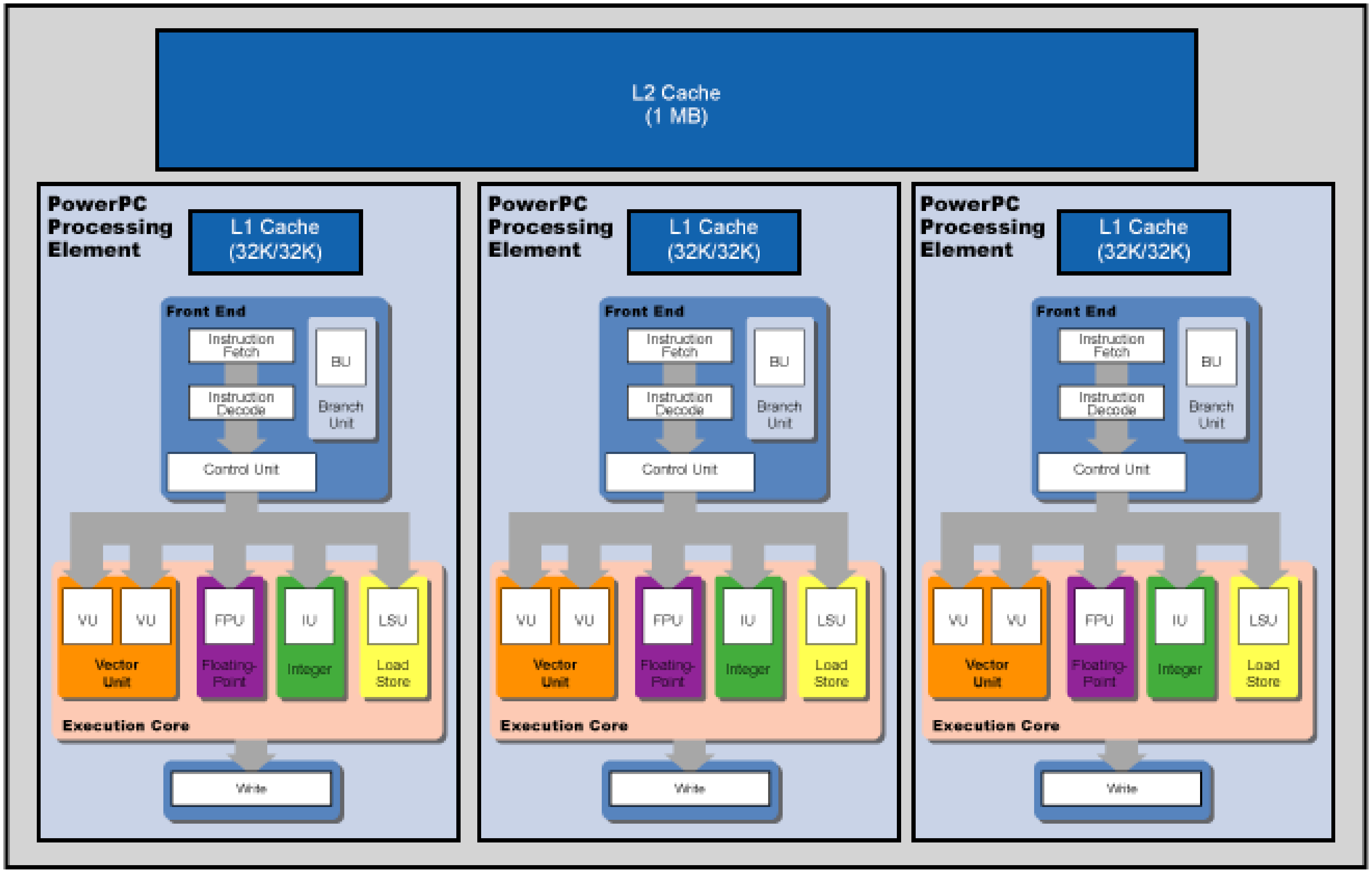


Microsoft vision

- Microsoft envisions a procedurally rendered game as having at least two primary components:
  - Host thread: a game's host thread will contain the main thread of execution for the game
  - Data generation thread: where the actual procedural synthesis of object geometry takes place

- These two threads could run on the same PPE, or they could run on two separate PPEs.
- In addition to the these two threads, the game could make use of separate threads for handling physics, artificial intelligence, player input, etc.

The Xenon architecture

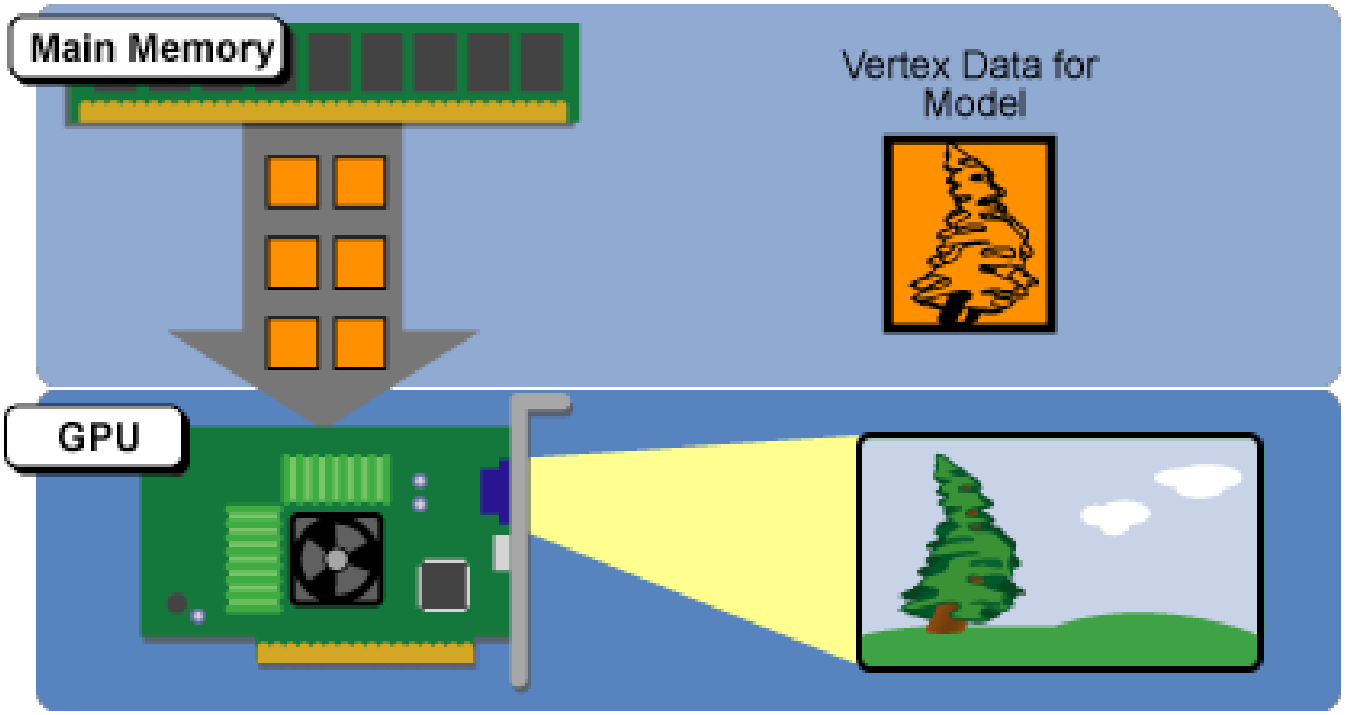


From ILP to TLP: from the processor to the programmer

- Keep it simple
  - Stripping out hardware that's intended to optimize instruction scheduling at runtime.
  - Neither the Xenon nor the Cell have an instruction window
    - Instructions pass through the processor in the order in which they're fetched
    - Two adjacent, non-dependent instructions are executed in parallel where possible
- Static execution
  - Is simple to implement
  - Takes up much less die space than dynamic execution
    - since the processor doesn't need to spend a lot of transistors on the instruction window and related hardware. Those transistors that the lack of an instruction window frees up can be used to put more actual execution units on the die.
- Rethink how you organize the processor
  - You can't just eliminate the instruction window and replace it with more execution

Procedural Synthesis in a nutshell

- Procedural synthesis is about making optimal use of system bandwidth and main memory by dynamically generating lower-level geometry data from statically stored higher-level scene data



Procedural Synthesis in a nutshell

- Procedural synthesis is about making optimal use of system bandwidth and main memory by dynamically generating lower-level geometry data from statically stored higher-level scene data

- For 3D games
  - Artists use a 3D rendering program to produce content for the game
  - Each model is translated into a collection of polygons
  - Each polygons is represented in the computer's memory as collections of vertices

Procedural Synthesis in a nutshell

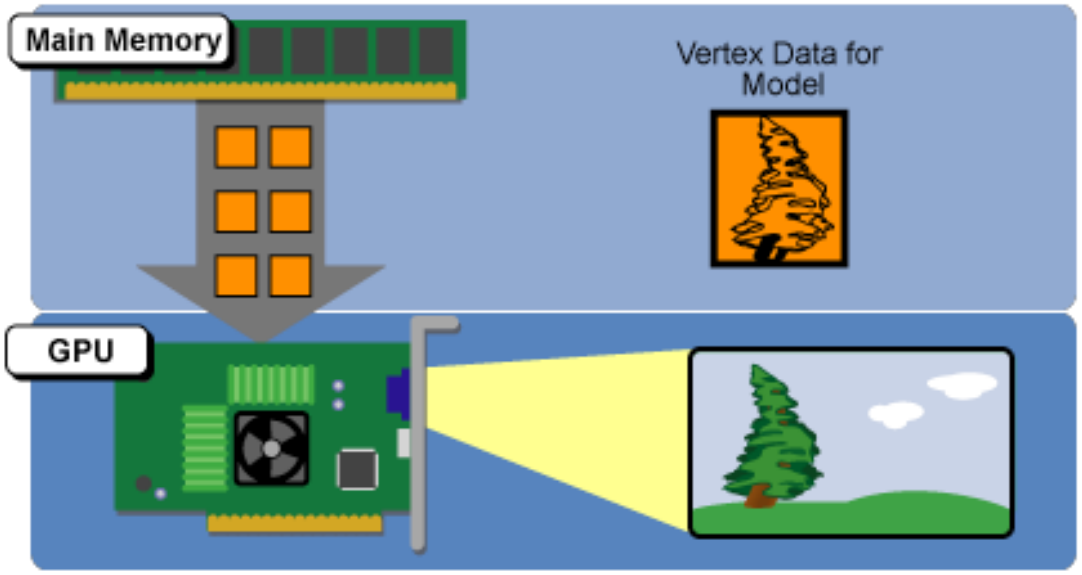
- Procedural synthesis is about making optimal use of system bandwidth and main memory by dynamically generating lower-level geometry data from statically stored higher-level scene data

- For 3D games
  - Artists use a 3D rendering program to produce content for the game
  - Each model is translated into a collection of polygons
  - Each polygons is represented in the computer's memory as collections of vertices

- When the computer is rendering a scene in a game in real-time
  - Models that are being displayed on the screen start out in main memory as stored vertex data
  - That vertex data is fed from main memory into the GPU
    - where it is then rendered into a 3D image and output to the monitor as a sequence of frames.

Limitations

- There are two problems
  - The costs of creating art assets for a 3D game are going through the roof along with the size and complexity of the games themselves
  - Console hardware's limited main memory sizes and limited bus bandwidth



Advanced Computer Architectures

(High Performance Processors and Systems)

Multiprocessors

Politecnico di Milano