Computer Architecture and Performance

David Newman

(from Tom Logan Slides from Ed Kornkven)

Outline

- **Performance Architectures**
  - Superscalar
  - Pipelined
  - Parallel
  - Vector
- **Memory Hierarchy**

Scalar Operations

For the statement: Z = a*b + c*d

1. Load a into register R0
2. Load b into R1
3. Multiply R2 = R0 * R1
4. Load c into R3
5. Load d into R4
6. Multiply R5 = R3 * R4
7. Add R6 = R2 + R5
8. Store R6 into z

- **Does Order Matter?**

**slide from SIPE* http://www.oscer.ou.edu/education.php

Basic Computer Control: The Fetch-Execute Cycle

Do Forever:

- Fetch instruction from memory
- Decode instruction; Fetch registers
- Compute addresses; Execute instruction
- Read/Write memory; Update program counter
- Update registers

- **This is the behavior of a basic CPU**
- **How can we speed this up?**

Computer Speedup Idea #1

Do Forever:

- Fetch instruction from memory
- Decode instruction; Fetch registers
- Compute addresses; Execute instruction
- Read/Write memory; Update program counter
- Update registers

Pipelining:

Do all these steps in parallel

Computer Speedup Idea #2

Do Forever:

{ – Fetch instruction from memory }

– Decode instruction; Fetch registers

– Compute addresses; Execute instruction

– Read/Write memory; Update program counter

– Update registers

Superscalar:

Do > 1 instruction in parallel

Computer Speedup Idea #3

Do Forever:

– Fetch instruction from memory	**Vector instruction**
– Decode instruction; Fetch registers	**Vector registers**
– Compute addresses; Execute instruction	
– Read/Write memory; Update program counter	**Vector registers**
– Update registers	**Vector registers**

Vector:

Operate on many data per instruction

Computer Speedup Idea #4

Do Forever:

- Fetch instruction from memory
- Decode instruction; Fetch registers
- Compute addresses; Execute instruction
- Read/Write memory; Update program counter
- Update registers

$\Big\}$ **One PE**

Parallel:

Use many processors

Superscalar Architectures

- **Superscalar**
  - Issues more than one instruction at a time
  - Must have multiple functional units to do the work
  - E.g., an integer unit and a floating point unit which can execute in parallel
- **Common in modern microprocessors**

Superscalar Operations

For the statement: Z = a*b + c*d

1. Load a into R0 AND

load b into R1

2. Multiply R2 = R0 * R1 AND

load c into R3 AND

load d into R4

3. Multiply R5 = R3 * R4

4. Add R6 = R2 + R5

5. Store R6 into z

**slide from SIPE* http://www.oscer.ou.edu/education.php

Pipelined Architectures

- **Pipelining**
  - Operations are divided into stages
  - When an operand finishes a stage, that stage is available for another operand
  - An N-stage pipeline has speedup of N minus pipeline overhead
    - Depth of pipeline has practical limits
- **Common in modern microprocessors**

Fast and Slow Operations

  - Fast: sum, add, subtract, multiply
  - Medium: divide, mod (i.e., remainder)
  - Slow: transcendental functions (sqrt, sin, exp)
  - Incredibly slow: power x^y for real x and y

- **On most platforms, divide, mod and transcendental functions are not pipelined, so a code will run faster if most of it is just adds, subtracts and multiplies.**
- **For example, solving an N x N system of linear equations by LU decomposition uses on the order of N^3 additions and multiplications, but only on the order of N divisions.**

**slide from SIPE* http://www.oscer.ou.edu/education.php

What Can Prevent Pipelining?

- **Certain events make it very hard (maybe even impossible) for compilers to pipeline a loop, such as:**
  - array elements accessed in random order
  - loop body too complicated
  - if statements inside the loop (on some platforms)
  - premature loop exits
  - function/subroutine calls
  - I/O

**slide from SIPE* http://www.oscer.ou.edu/education.php

Vector Architectures

- **Vector Processing**
  - A datatype, instructions and hardware for operating on *vectors* -- 1-D arrays of data
  - Deeply pipelined vector processing units
  - Vector registers
  - Vector versions of LOAD, STORE and arithmetic operations

Vector Architectures (cont.)

- **Furthermore, vector supercomputers also**
  - Have a specialized memory system for very high bandwidth
  - A large number of registers
  - Many opportunities for parallelism
    - Pipelined scalar and vector units
    - Multiple pipelines and functional units
    - Overlapping vector and scalar operations
    - Multiple CPUs

Parallel Architectures

- **Multiple processing elements (PEs) with varying degrees of memory sharing between PEs (Beware of terminology!)**
  - Shared memory - all processors can access all memory locations in approximately the same time (aka *multiprocessors* or *symmetric multiprocessors*)
  - Nonshared (distributed) memory - each PE has its "own" memory which it can access quickly; memory of other PEs is accessed over interconnect (aka *multicomputers* or *MPPs*)
  - Combinations of these -- Nearby local (fast but small) memory plus larger remote (e.g., on other PEs) memory (longer latency but larger) (aka *NUMA machines*)
    - See http://lse.sourceforge.net/numa/faq/

Memory Hierarchy

- **The most significant challenge in programming parallel machines is effectively managing the memory hierarchy**
  - The components of the storage system
  - Closer to CPU ⇒ smaller, faster
  - Farther from CPU ⇒ larger, slower

Memory Hierarchy (cont.)

Memory Hierarchy Stages

- Registers
  - On-CPU; directly referenced by instruction set; virtually no access delay (latency)
- Cache
  - Small, fast memory close to CPU; holds most recently accessed code or data
- Main memory
  - Local memory - fast access by owning CPU
  - Node memory - reasonably fast access by CPUs of owning node (grouping of CPUs)
  - Global memory - accessible by any CPU
- Disk - persistent storage

Memory Hierarchy (cont.)

Level	Latency in CPU cycles	Capacity in K Bytes
Registers	1	10^1
L1 Cache	A few	10^2
L2 Cache	More than L1	10^3
L3 Cache	More than L2	10^4
Main memory	10^2	10^7
Disk	10^5	10^9 +

ARSC

Example: Some Microprocessors

	Pentium 4 3.2	Pentium 4 3.2 EE (2MB L3)	Athlon XP 3200+	Athlon 64 FX-51
L1 latency, cycles	2	2	3	3
L2 latency, cycles	19	19	20	13
L3 latency, cycles		43		
Memory latency, cycles	204	206	180	125
L1 latency, ns	0.63	0.63	1.36	1.36
L2 latency, ns	5.94	5.94	9.09	5.90
L3 latency, ns		13.44		
Memory latency, ns	63.75	64.38	81.82	56.81

Impact of Memory Hierarchy

- **Suppose we can use the L1 cache 90% of the time on the Pentium 4**
- **How much of a speedup do we get compared to just using main memory?**

  (Using Amdahl's Law)

  speedup = 1 / (0.10 + (0.90/L1 cache speedup))

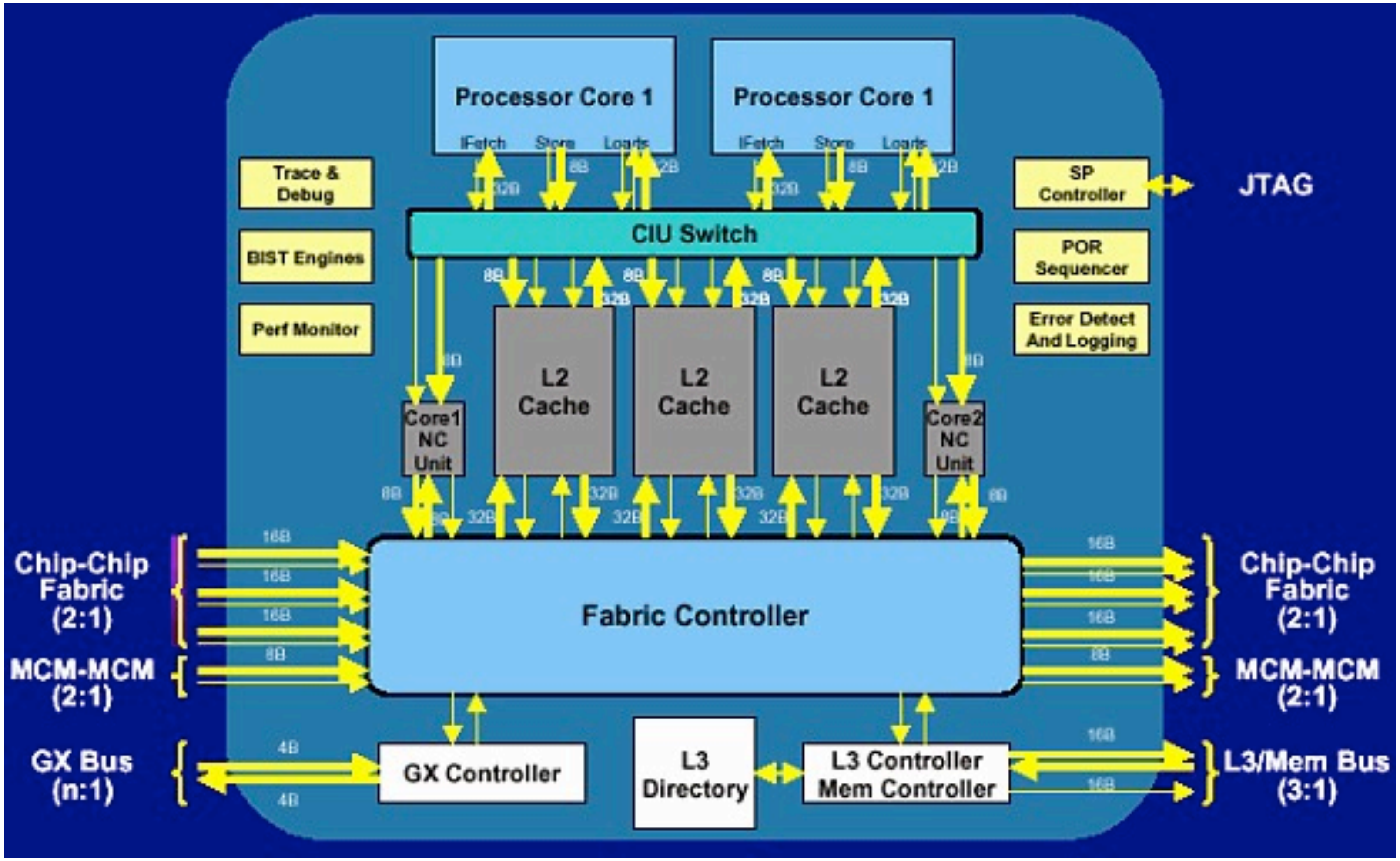
  From the previous slide, L1 cache speedup = 204/2 = 102

  So speedup = 1 / (0.10 + (0.90/102)) = 9.2

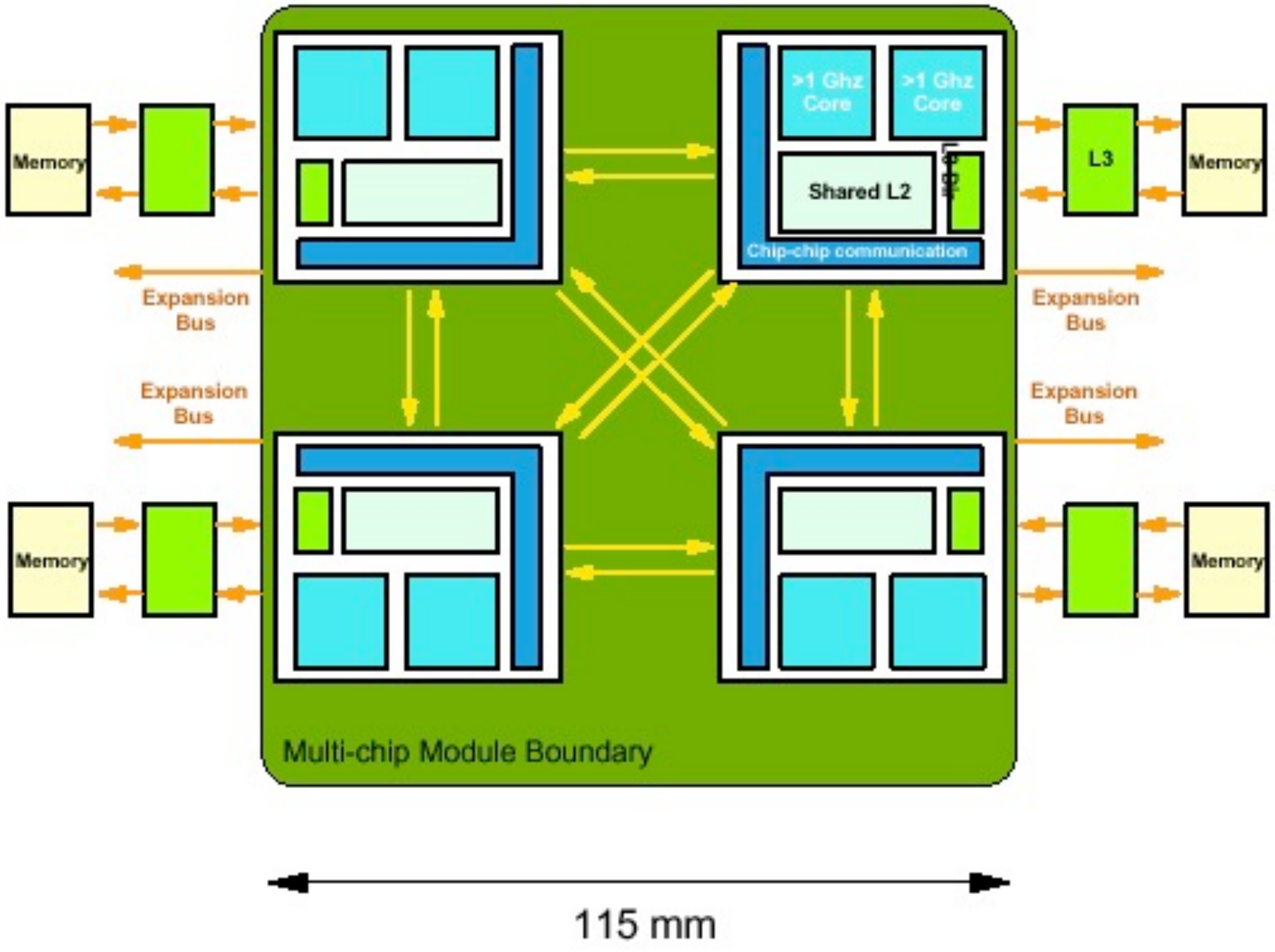
Example: Power4

Level	Latency in CPU cycles	Capacity in Bytes
Registers	1	≈1 KB
L1 Cache	4	128 KB
L2 Cache	12-20	1.45 MB
L3 Cache	More than L2	32 MB
Main memory	10^2	16 GB
Disk	10^5	1+ TB

Memory Hierarchy (cont.)



Memory Hierarchy (cont.)



Memory Management

- **Three main focal points for programmer**
  1. Cache
  2. Main memory
  3. Interconnection network

Cache Management

- **Be aware of locality**
  - Spatial
  - Temporal
- **Most recently used data or instructions will be in the cache -- reuse them**

Main Memory Management

- **Understand memory layout for your language**
  - C: row major
  - Fortran: column major
- **Special case to watch for:**
  - Accessing memory by 2k strides
  - E.g. loop nests that access rows/columns

Network Management

- **Minimize remote data references**
  - Arrange data on processors carefully
  - Perhaps keep a local copy of some remote data -- e.g. *halo*
  - Use non-blocking messaging
  - Many other techniques -- wait for MPI classes