Parallel Programming Platforms

Alexandre David
B2-206

http://www.cs.aau.dk/~adavid/teaching/MTP-06/

Today

- Implicit Parallelism (2.1)
- Limitations of Memory System Performance (2.2)
- Dichotomy of Parallel Computing Platforms (2.3)

Motivations

- Bottlenecks in computers:
  - Processor
  - Memory
  - Datapath
- Addressed with multiplicity.
- Parallelization not solution to everything
  - Sub-optimal serial code bad
  - Optimize serial first (similar characteristics)

Trends in Microprocessors

- Processor speed increase exponentially
- More and more transistors: How to use them wisely?
- Multiple functional units run multiple instructions in the same clock cycle: **superscalar** processors.
- How to select and execute instructions?

Pipelining and Superscalar Execution

- Pipeline idea: overlap stages in instruction execution.
- Example of car factory.
- The good: higher throughput.
- The bad: penalty of branch miss prediction.
- Multiple pipelines: several functional units.

Pipelining and Superscalar Execution

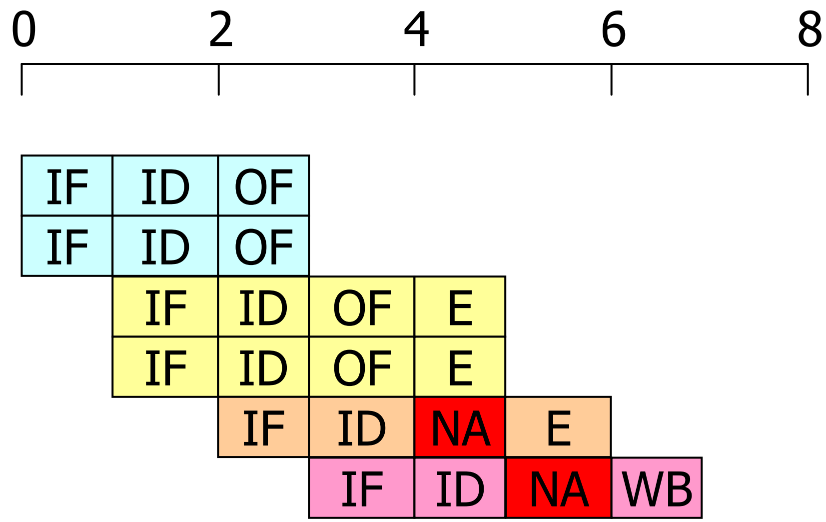
Compiler

c=a+b+c+d
as
c=(a+b)+(c+d)

CPU

1. load R1,@1000
2. load R2,@1008
3. addR1,@1004
4. addR2,@100C
5. add R1,R2
6. store R1,@2000

Instruction cycles



2x IF, ID, OF, … in the same cycle: **superscalar**.

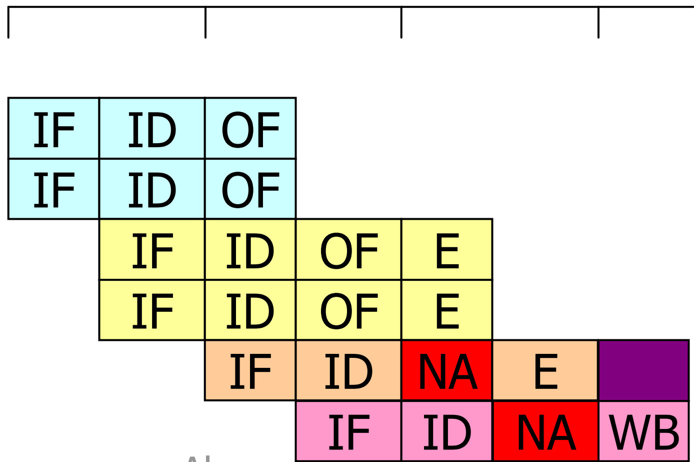
Pipelining and Superscalar Execution

- Imagine another ordering (or factorization by the compiler): different performance.
- Resolve data dependency.
- Reordering by CPU possible (out-of-order execution).
- Resource dependency.

Limitations

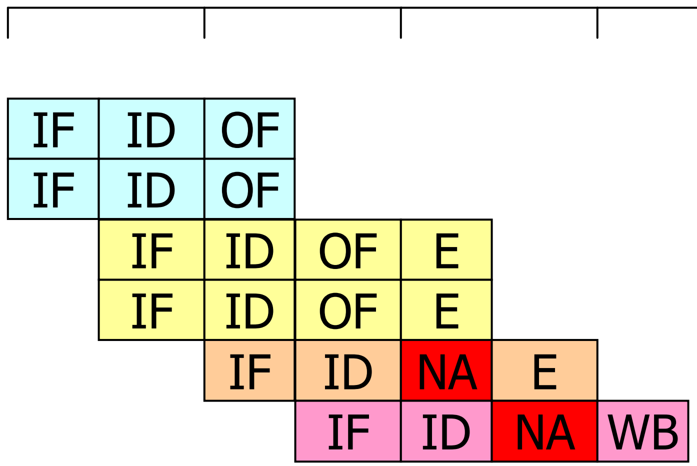
- Bottleneck: slowest stage -> small stages to go fast -> long pipelines
  - BUT miss prediction gives big penalties
- How to keep busy the functional units?

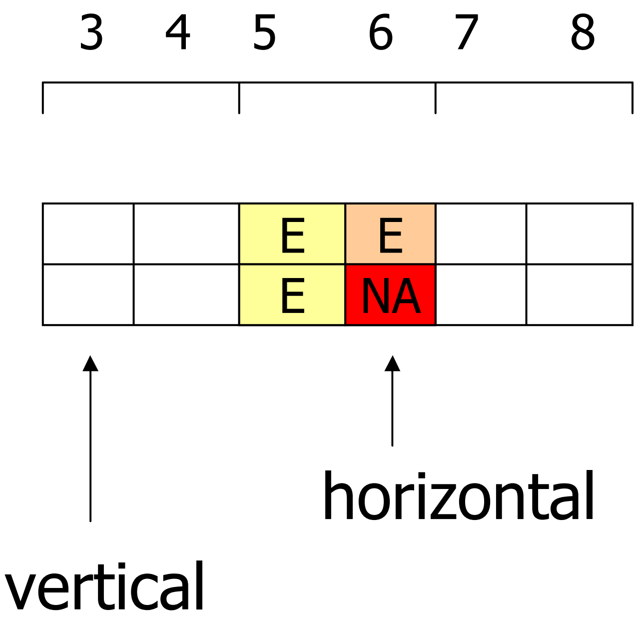
Vertical waste: no instruction on execution unit. Here no instruction on the adder unit.



Horizontal waste: parts of execution units used.

Adder Utilization (fig 2.1)





Adder functional unit: execute = E
2 units.

VLIWP

- Bundle instructions together to simplify the superscalar scheduler.
- IA64 (Itanium) is an example.
- Problems:
  - *Rely a lot* on the compiler.
  - Limited parallelism (not dynamic).

Limitations of Memory System Performance

- The memory system is most often the bottleneck.
- Performance captured by
  - latency and
  - bandwidth.
- Remark: In practice latency is complicated to define: CL2, CL3, 2-2-2-5,…

Effect on Performance: An Example

- Processor @1GHz (1ns cycle), DRAM with 100ns latency, capable of executing 4 IPC.
- 4 IPC @1GHz -> 4GFLOPS *peak rating*.
- Processor must wait 100 cycles for every request.
  - Vector operations (dot product) @10MFLOPs.

Improving with Cache

- Note: Often “$$” on pictures (cash).
- Hierarchical memory architecture with several levels of cache (2 common).
- Instruction and data separate for L1.
- Low latency, high bandwidth, but small.
- Why does it improve???

Why is $$ good?

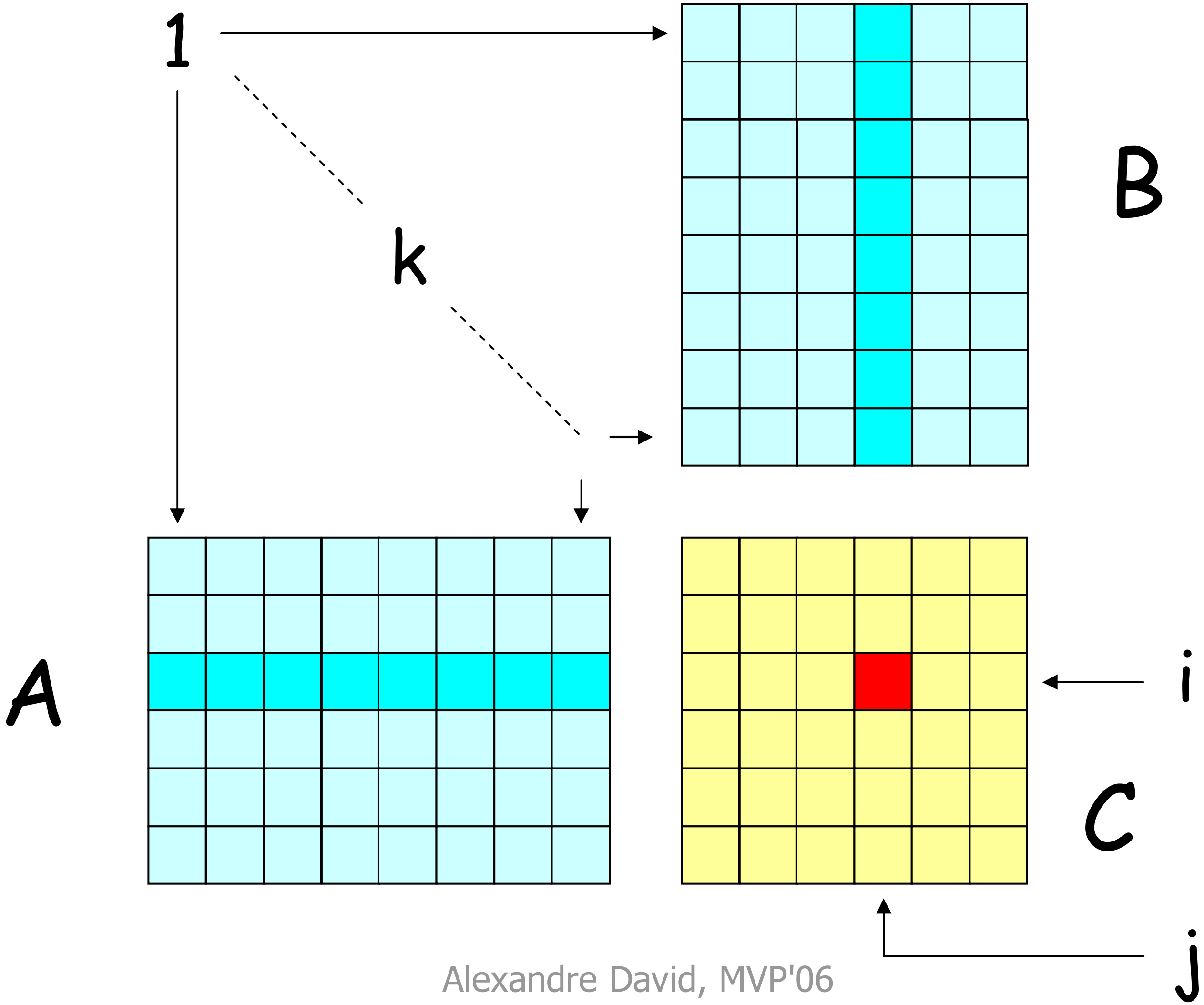
- Temporal locality
  - Repeated access to the **same** data in a small window of time.
- Spatial locality
  - **Consecutive** data accessed by successive instructions.
- Vital assumptions, almost always hold.
- Very important for parallel computing.

Matrix Multiplication Example

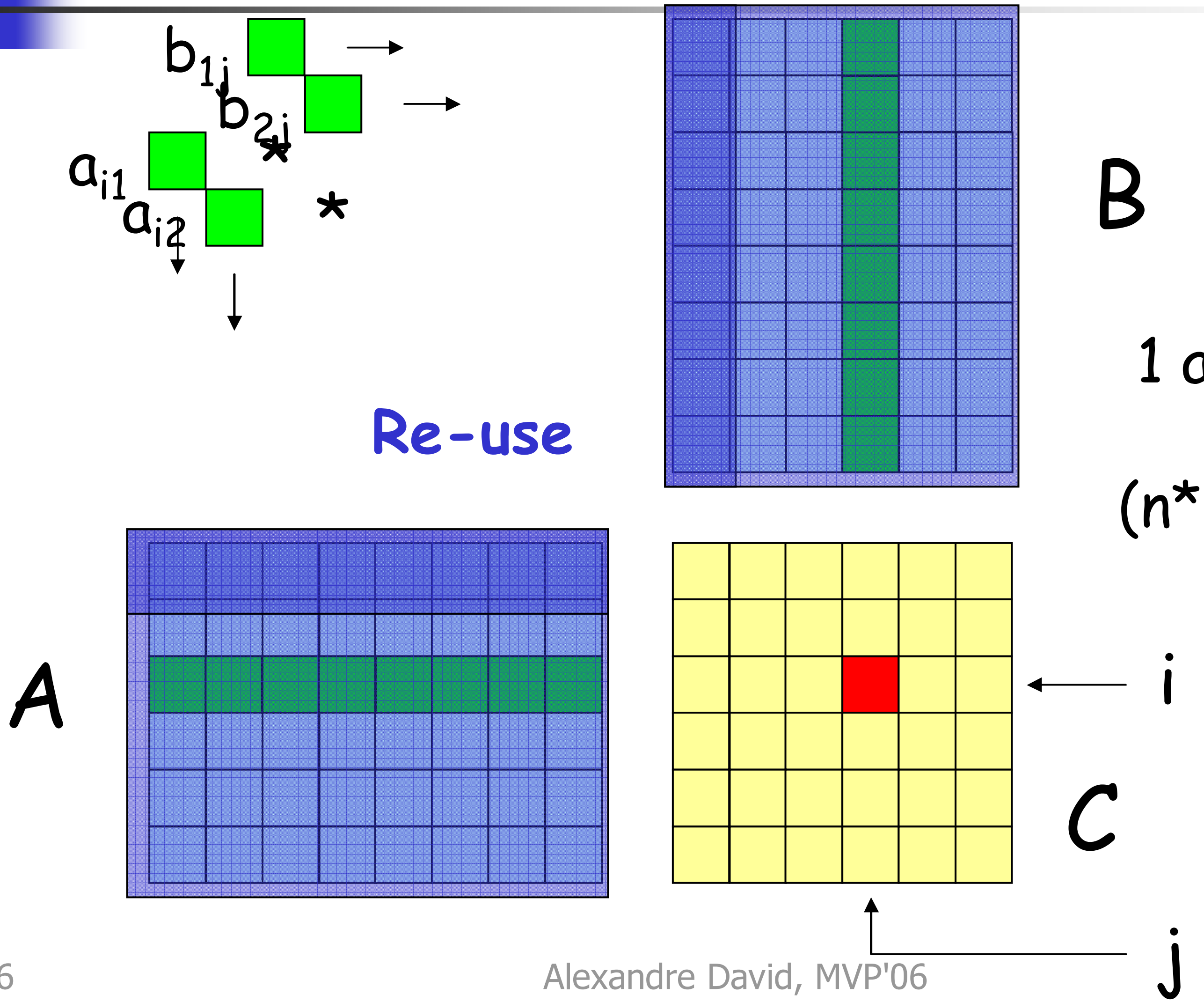
- Common example, will be used many times in the course.
- C=A*B, where A (n*m), B (p*n), and C (p*m) are matrices.

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Matrix Multiplication Example



Matrix Multiplication Example



1 add & mul/k
n^3 total
(n*n matrices).

Cache Characteristics

- Hit ratio (behavior): fraction of references satisfied by the cache.
- Cache line (= bus width): granularity.
- Associativity (architecture): “collision list” to reduce cache eviction.
- For the matrix: $2n^2$ fetches from memory to *populate the cache*, and then n^3 direct accesses at full speed.

Impact on Memory Bandwidth (and Latency)

- Access to successive words much better than random access.
  - Higher bandwidth (whole cache line at once)
  - Better latency (successive words already in cache)

Example: Strided Access

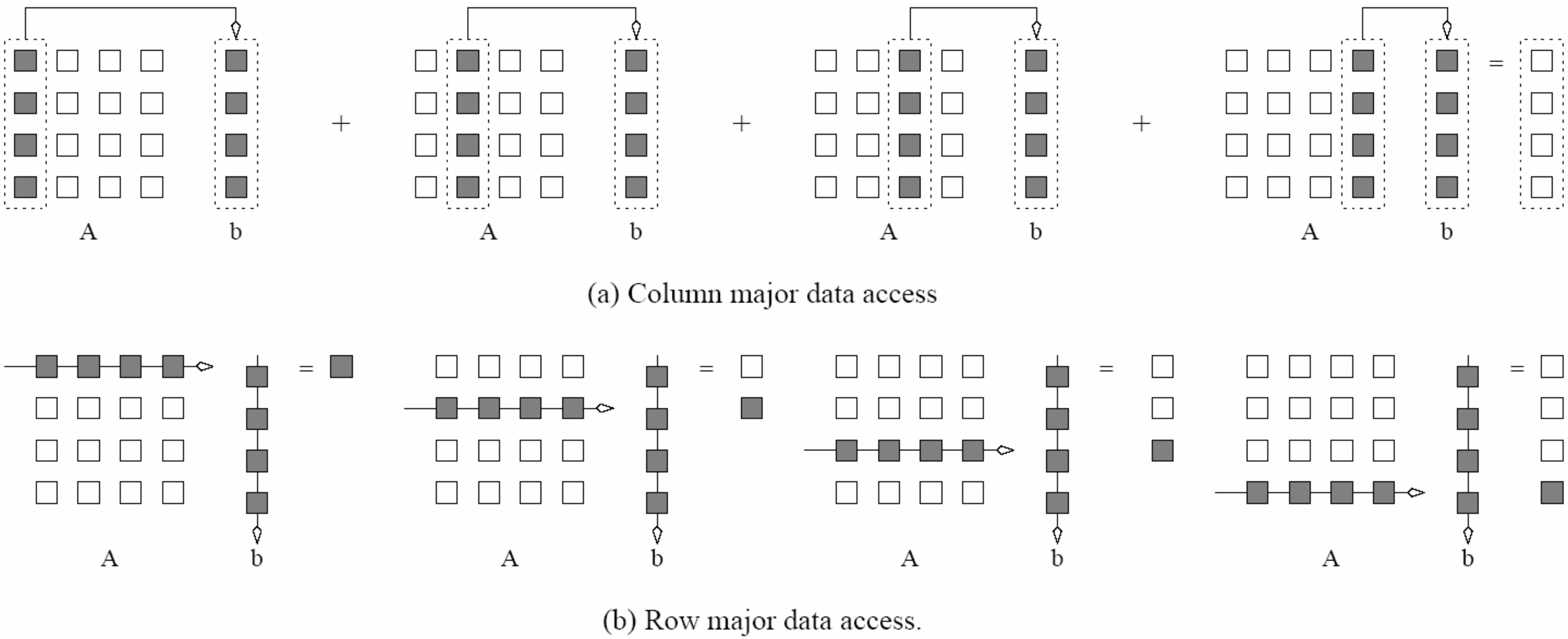


Figure 2.2 Multiplying a matrix with a vector: (a) multiplying column-by-column, keeping a running sum; (b) computing each element of the result as a dot product of a row of the matrix with the vector.

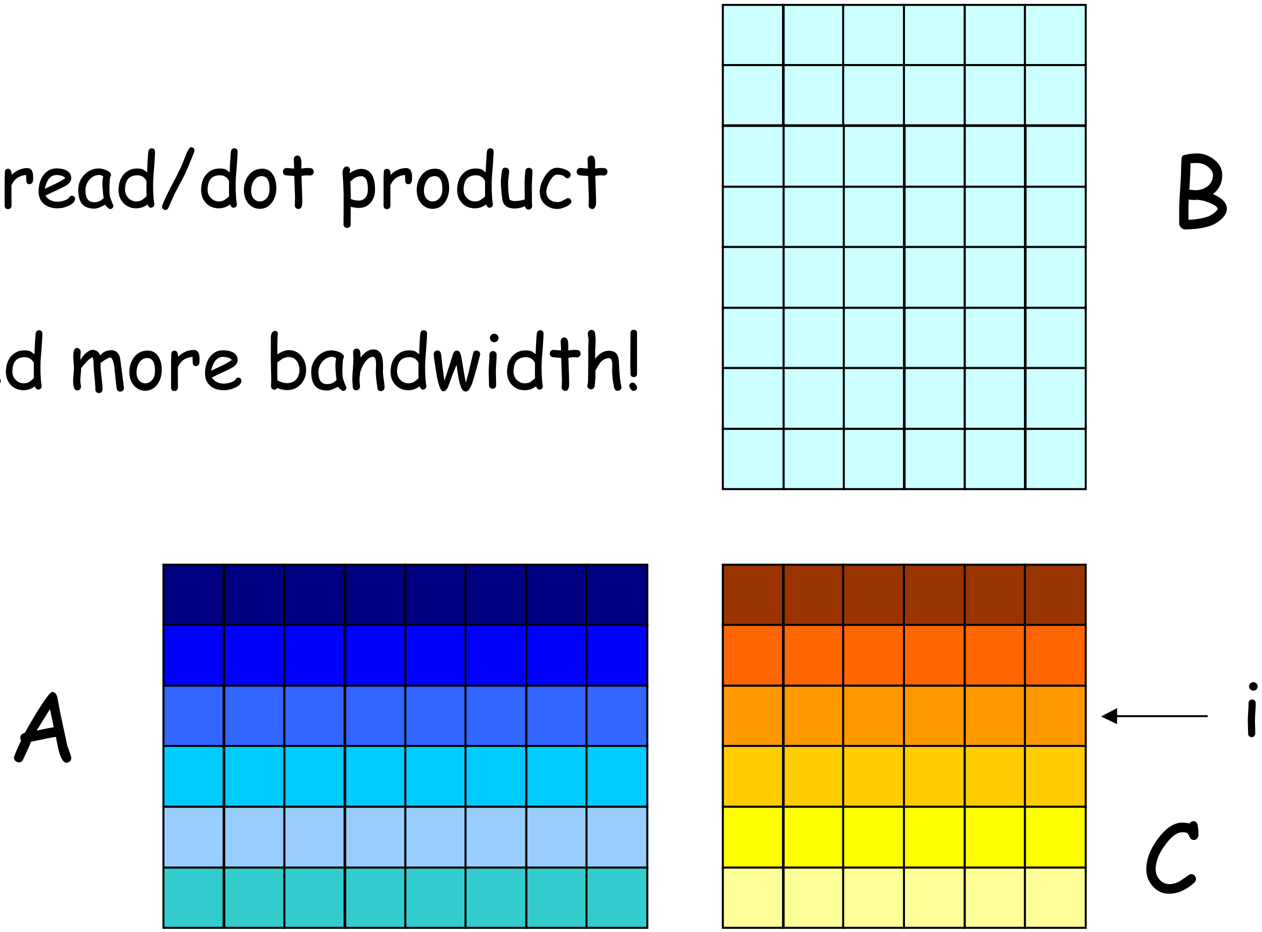
Other Approaches to Hide Latency

- Prefetching
  - but may evict useful data because cache is small.
- Multi-threading
  - but needs higher bandwidth because all the threads share the same bus.

Multi-threading

1 thread/dot product

BUT: need more bandwidth!



Summary on Memory

- Exploit spatial and temporal locality in programs. For sequential and parallel programs!
- Operations/memory accesses is a good indicator of tolerance to memory bandwidth.
- Processing is cheap, memory is expensive.

Dichotomy of Parallel Computing Platforms

- Logical organization: programmer’s view.
- Physical organization: actual hardware.
- Two critical components:
  - expressing parallel tasks (control structure)
  - specifying interaction between them (communication model).

Control Structure

- Parallelism can be expressed at different levels of granularity
  - from instruction level parallelism
  - to processes.
- SIMD: single instruction stream, multiple data stream.
- MIMD: multiple instruction stream …

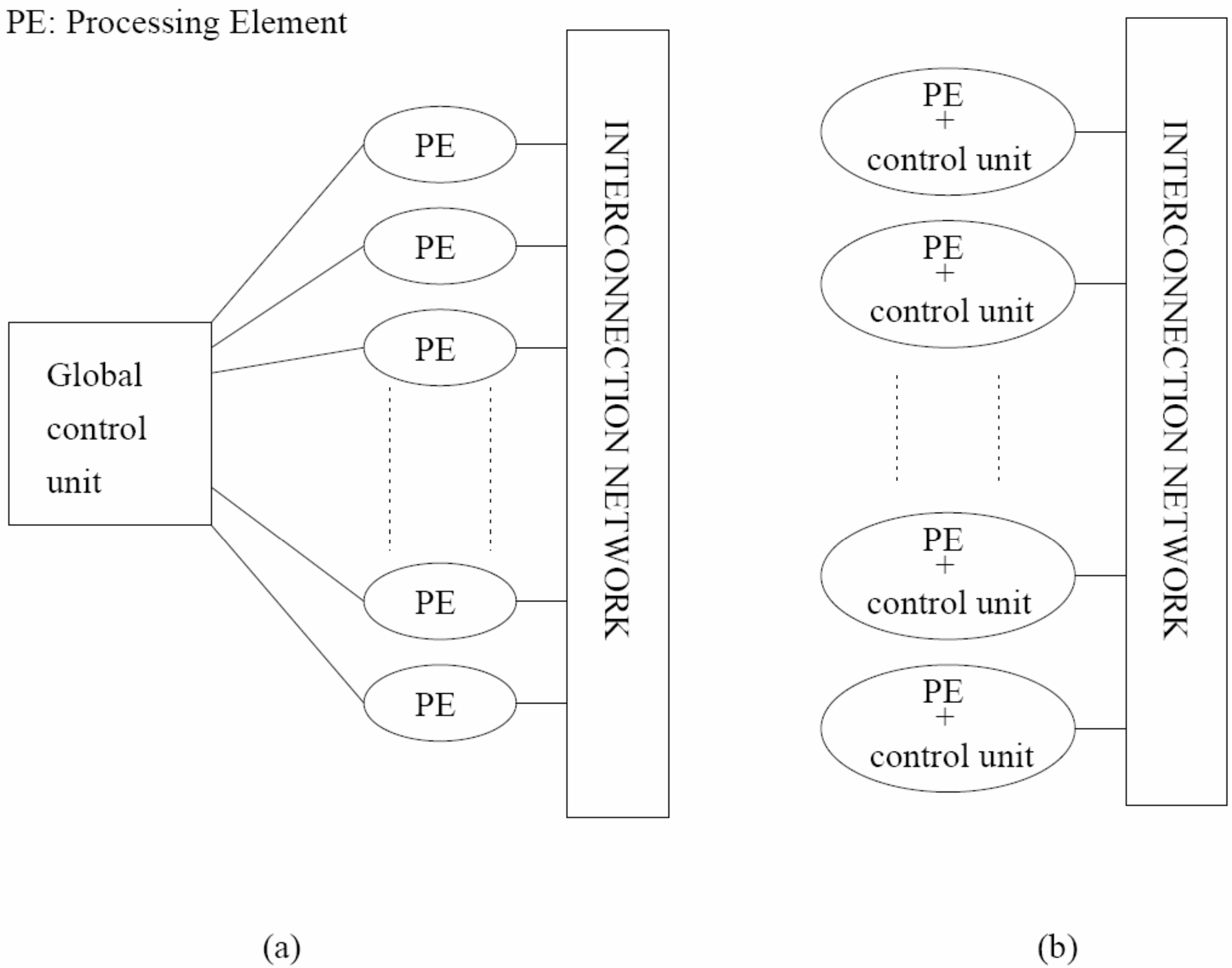
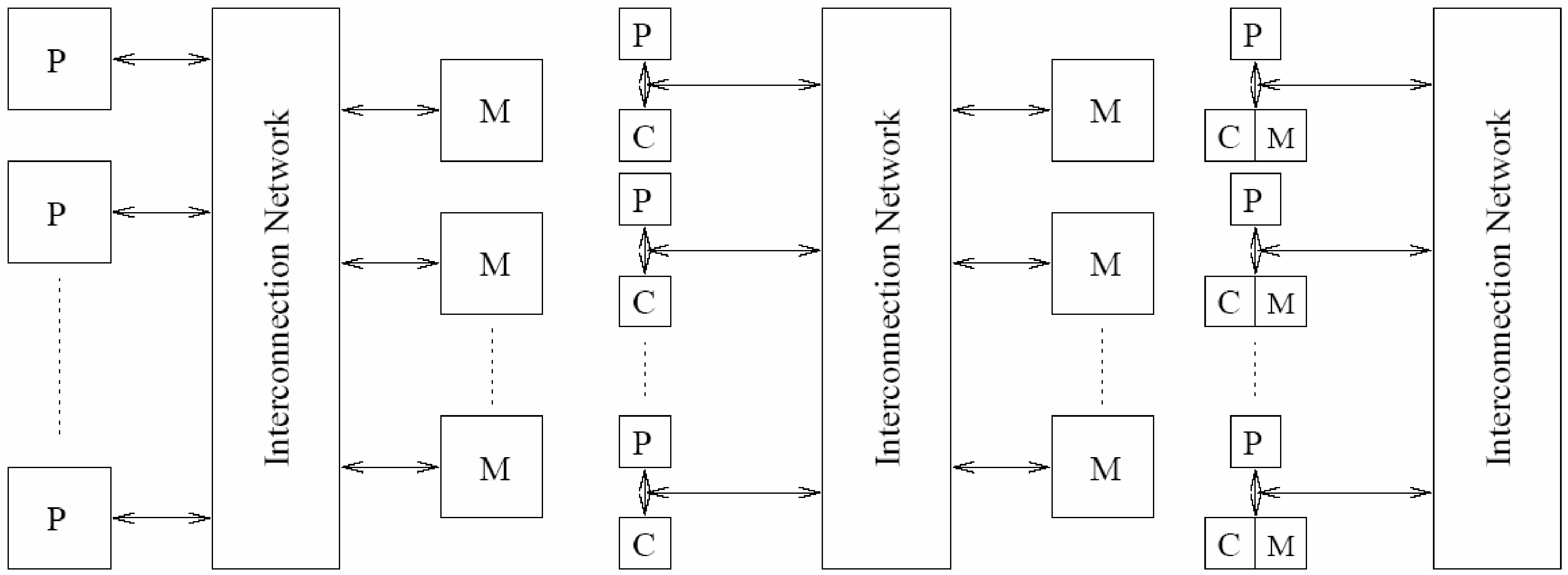


Figure 2.3 A typical SIMD architecture (a) and a typical MIMD architecture (b).

Communication Model: Shared Address Space

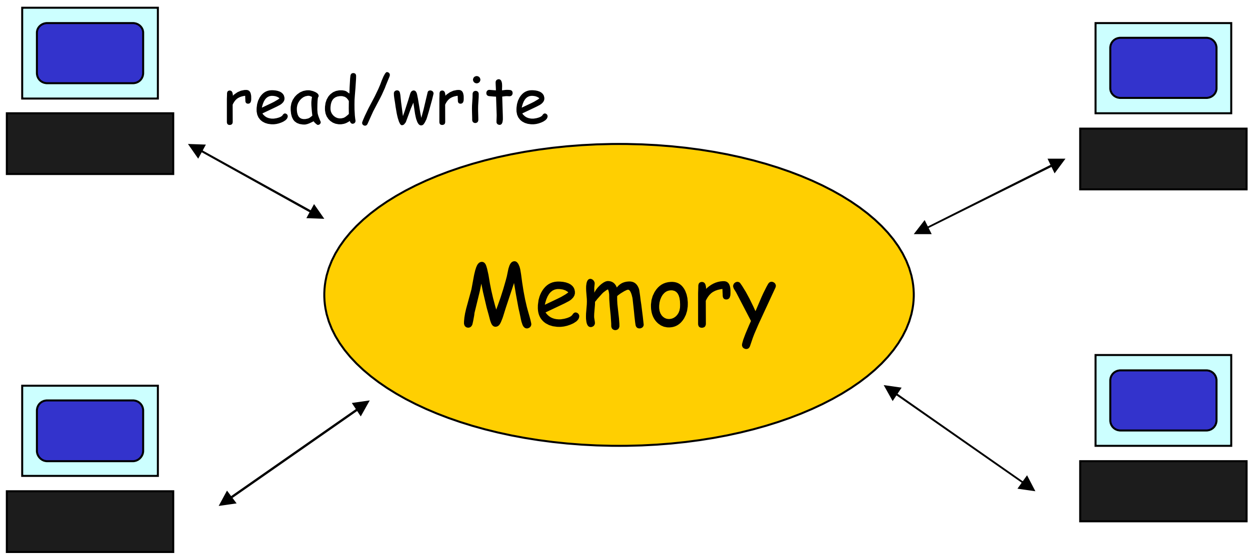
- Memory shared between several processors.
  - NUMA different access time
  - UMA same access time.
  - Cases with local cache considered UMA.
- Easier programming, one address space
  - but cache coherence mechanisms needed,
  - But need to solve contention (writes).

UMA vs. NUMA



(a) (b) (c)

Communication Model: Shared Address Space



Implemented as shared memory computers or distributed memory computers.

Message-Passing Platforms

- Memory private to processors.
- Interaction via messages
  - Send/receive primitives.
  - MPI libraries.
- Hardware needed: good network interconnect.