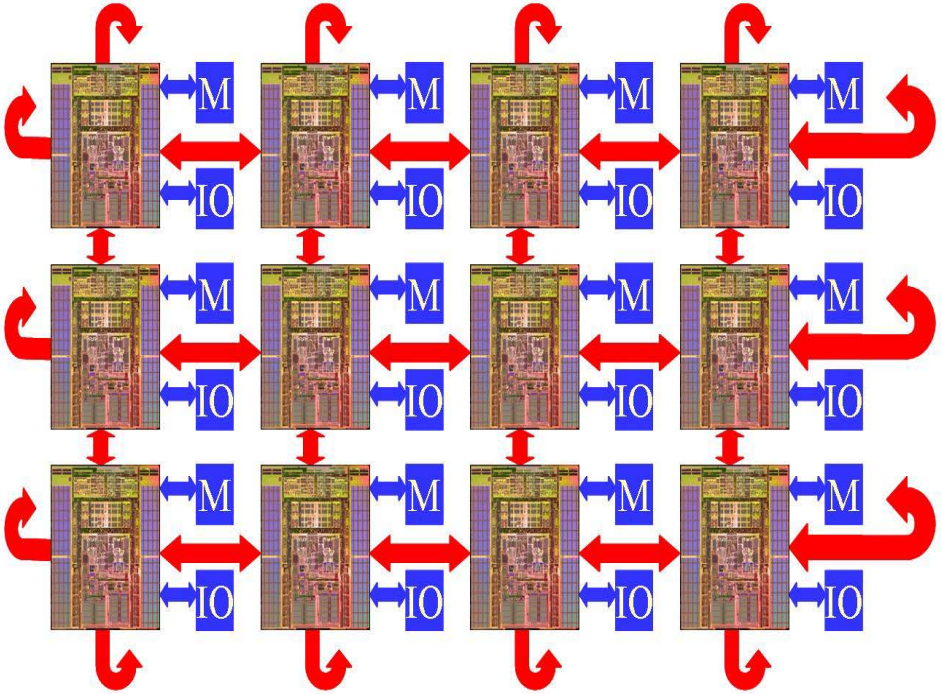
18-742

Lecture 1

Multiprocessor Computer Architecture



Alpha 21364 Shared-Memory Multiprocessor

Spring 2005

Prof. Babak Falsafi

http://www.ece.cmu.edu/~ece742

Slides developed in part by Profs. Adve, Falsafi, Hill, Lebeck, Reinhardt, Smith, and Singh of University of Illinois, Carnegie Mellon University, University of Wisconsin, Duke University, University of Michigan, and Princeton University.

Homework 0

- **Due by this Friday**
- **My way to learn about you**
- **Learn your name and computer architecture background**

No homeworks graded until homework 0 handed in!!

Readings

- **Reader 1**
  - **The book will arrive later. There will be a copies of chapter 1 outside of A302 after class**
  - **Chapter 9 of H,J&S will be on the web**

18-742 Class Info

- **Instructor: Professor Babak Falsafi**
  - **URL: http://www.ece.cmu.edu/~babak**
- **Research interests:**
  - **Bridging the processor/memory performance gap**
  - **Nanoscale CMOS computer systems**
  - **Analytic & simulation tools for design evaluation**
- **TA:**
  - **Brian Gold (bgold@cmu.edu)**
- **Administrative Assistant: Matt Koeske (koeske@ece)**
- **Class info:**
  - **URL: http://www.ece.cmu.edu/~ece742**
    - » **Username & password provided in class**
  - **Newsgroup: cmu.ece.class.ece742 (use nntp.ece.cmu.edu)**

18-742 Class Meeting Time

- **Notice!**
  - **each lecture is 80 minutes long**
  - **class meets between 3:00pm - 4:20pm on MW**
  - **office hours: Mon. 2:00-3:00 & Fri. 3:00-4:00 pm**
  - **Brian's office hours: Wed. 11:00-12:00 & Thur. 3:30-4:30**

Who Should Take 18-742?

- **Graduate students (MS/IMB/PhD)**
  1. Computer architects to be
  2. Computer system designers

- **Required Background**
  - 18-741 (Advanced Computer Architecture)

- **Course expectations:**
  - Heavily discussion oriented
  - Will loosely follow text *(Culler & Singh)*
  - With emphasis on cutting-edge issues/research
  - Students will review list of research papers *(almost weekly)*
  - Independent research project *(a major component)*
  - Individual feedback upon request *(come to office hours!)*

18-742: Components

- **Text**
  - **Parallel Computer Architecture: A Hardware/Software Approach**
  - **recommended: *Readings in Computer Architecture***
- **Readers + Review**
  - **list of papers: classic + state of the art**
  - **short written review per paper**
- **Homework (individual write-up, group discussion okay)**
- **Project**
  - **mostly original research**
  - **groups of two**
- **Heavy use of web page for interaction!**
  - **www.ece.cmu.edu/~ece742**

Grading

- **Grade breakdown**

  Project: 30%
  Homework: 20%
  Exam 1: 25%
  Exam 2: 25%

- **Participation + Discussion count toward the Project grade**

Grading (Cont.)

- **No late homeworks, no kidding**
- **Group studies are encouraged**
- **Group discussions are encouraged**
- **All homeworks must be results of individual work** *(identify the persons you discussed the problem with)*

- *There is no tolerance for academic dishonesty. Please refer to the University Policy on cheating and plagiarism. Discussion and group studies are encouraged, but all submitted material must be the student's individual work (or in case of the project, individual group work).*

Computer Architecture Curriculum

18-447 — Introduction to computer architecture
18-741 — Advanced computer architecture
18-742 — Multiprocessor architecture
18-743 — Power-aware architecture
18-744 — Microarchitecture design and implementation
18-747 — Advanced topics

- **Related areas:**
  - » circuits: VLSI, digital circuit design, CAD
  - » systems: compilers, OS, database systems, networks, embedded computing, fault-tolerant computing
  - » evaluation: queuing theory, analysis of variance, confidence intervals, etc.

Computer Architecture Activity

Computer Architecture Lab at Carnegie Mellon (CALCM) @ www.ece.cmu.edu/CALCM

Send mail to calcm-list-request@ece

body: subscribe calcm-list

Seminars

- **CALCM weekly seminar**
- **LCS/SDI/Intel weekly seminar**
- **CSSI weekly seminar**
- **Computer systems lecture series**

Outline

- **Motivation & Applications**
- **Theory, History, & A Generic Parallel Machine**
- **UMA, NUMA, Examples, & More History**
- **Programming Models**
  - **Shared Memory**
  - **Message Passing**
  - **Data Parallel**
- **Issues in Programming Models**
  - **Function: naming, operations, & ordering**
  - **Performance: latency, bandwidth, etc.**

Motivation

- 18-741 is about computers with one processor
- This course uses N processors in a computer to get
  - Higher Throughput via many jobs in parallel
  - Improved Cost-Effectiveness (e.g., adding 3 processors may yield 4X throughput for 2X system cost)
  - To get Lower Latency from shrink-wrapped software (e.g., databases and web servers today, but more tomorrow)
  - Lower latency through Parallelizing your application (but this is hard)
- Need faster than today's microprocessor?
  - Wait for tomorrow's microprocessor
  - Use many microprocessors in parallel

Applications: Science and Engineering

- **Examples**
  - **Weather prediction**
  - **Evolution of galaxies**
  - **Oil reservoir simulation**
  - **Automobile crash tests**
  - **Drug development**
  - **VLSI CAD**
  - **Nuclear BOMBS!**
- **Typically model physical systems or phenomena**
- **Problems are 2D or 3D**
- **Usually requires “number crunching”**
- **Involves “true” parallelism**

Applications: Commercial

- **Examples**
  - **On-line transaction processing (OLTP)**
  - **Decision support systems (DSS)**
  - **“app servers”**
- **Involves data movement, not much number crunching**
  - **OTLP has many small queries**
  - **DSS has fewer large queries**
- **Involves throughput parallelism**
  - **inter-query parallelism for OLTP**
  - **intra-query parallelism for DSS**

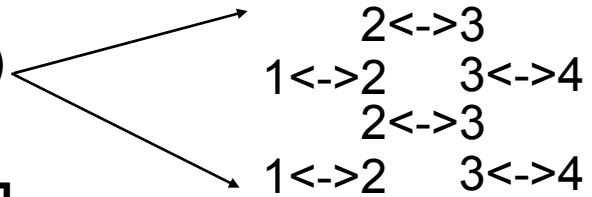
Applications: Multi-media/home

- **Examples**
  - **speech recognition**
  - **data compression/decompression**
  - **3D graphics**
- **Will become ubiquitous**
- **Involves everything (crunching, data movement, true parallelism, and throughput parallelism)**

In Theory

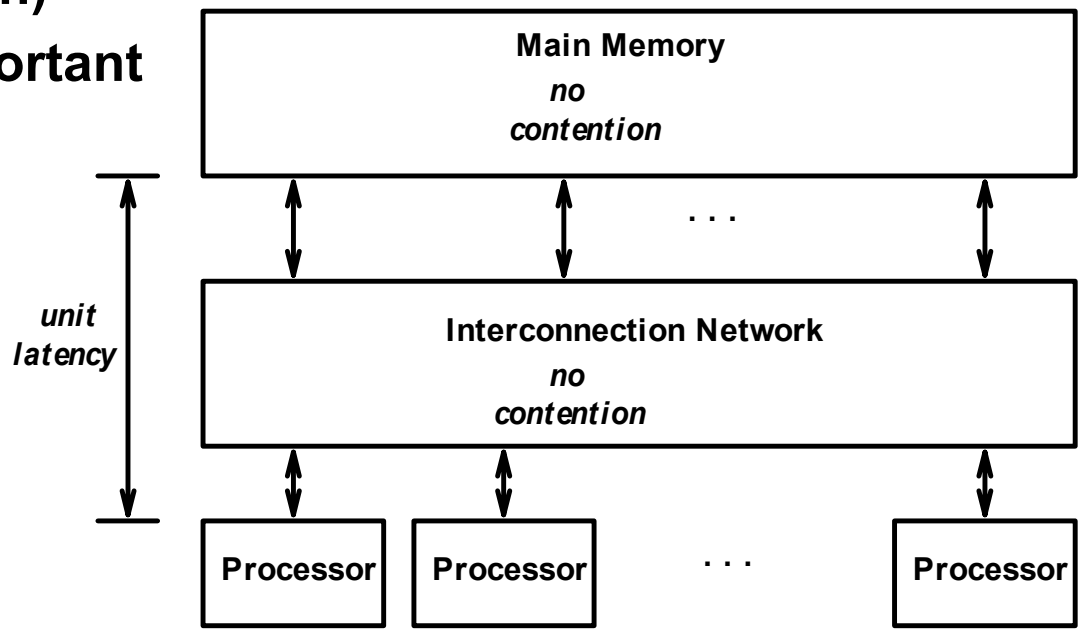
- **Sequential**
  - **Time to sum n numbers? O(n)**
  - **Time to sort n numbers? O(n log n)**
  - **What model? RAM**

- **Parallel**
  - **Time to sum? Tree for O(log n)**
  - **Time to sort? Non-trivially O(log n)**
  - **What model?**
    - » **PRAM [Fortune Willie STOC78]**
    - » **P processors in lock-step**
    - » **One memory (e.g., CREW for concurrent read exclusive write)**



Perfection: the PRAM Model

- Parallel RAM
- Fully shared memory
- Unit latency
- No memory contention (unrestricted bandwidth)
- Data placement unimportant



Perfection is NOT Achievable

- **Latencies grow as the system size grows**
- **Bandwidths are restricted by memory organizations and interconnection networks**
- **Dealing with reality leads to division between**

  UMA: Uniform Memory Access

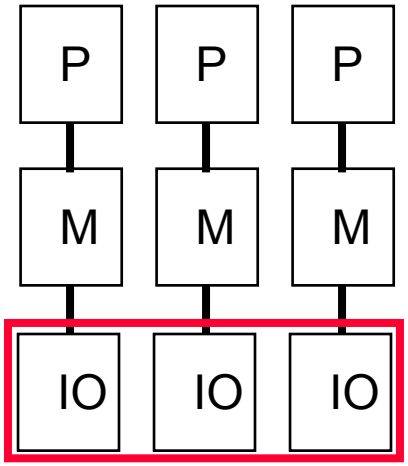
  and

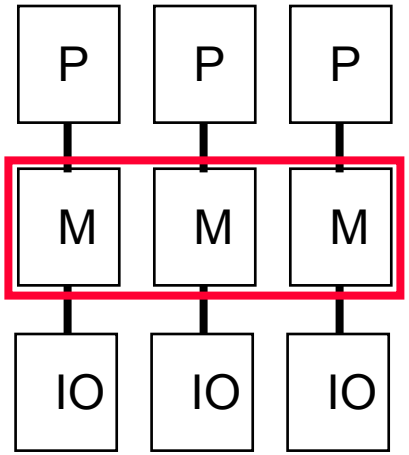
  NUMA: Non-Uniform Memory Access

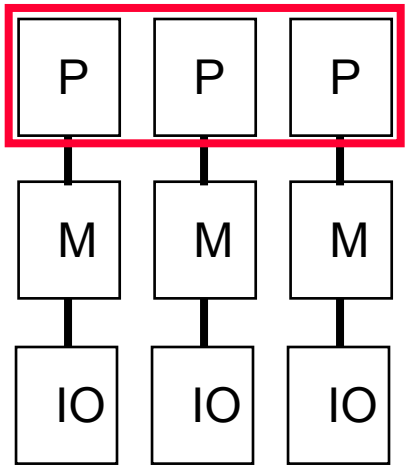
But in Practice, How Do You

- **Name a datum (e.g. all say a[I])?**
- **Communicate values?**
- **Coordinate and synchronize?**
- **Select processing node size (few-bit ALU to a PC)?**
- **Select number of nodes in system?**

Historical View





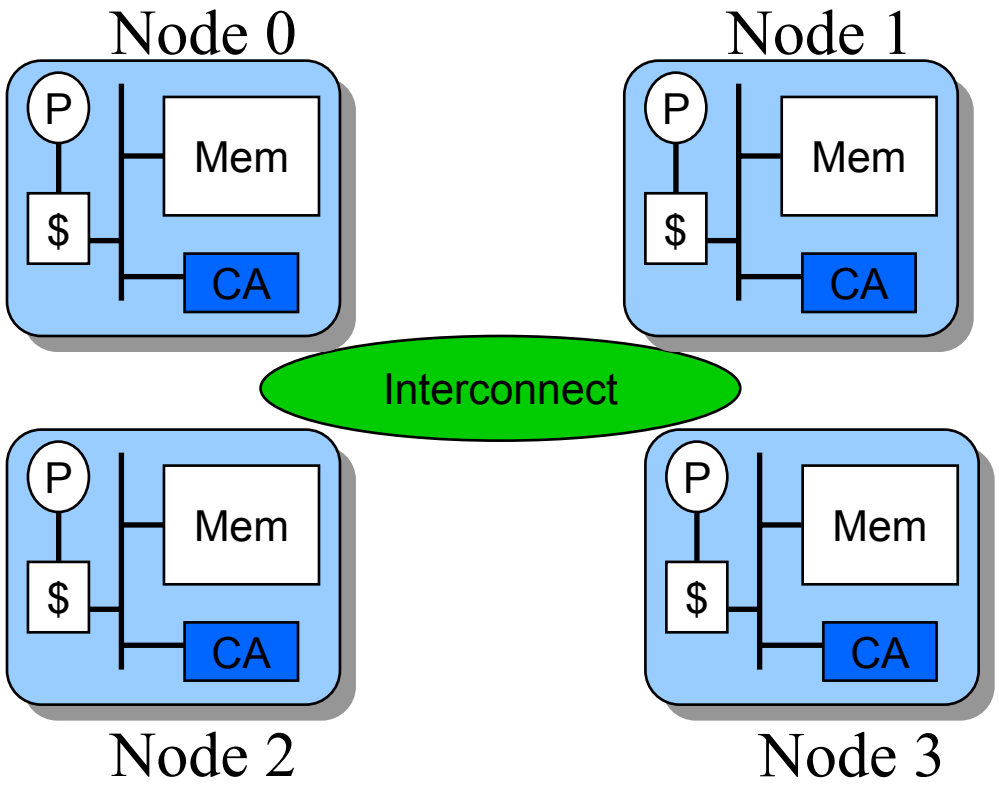


Join At:	I/O (Network)	Memory	Processor
Program With:	Message Passing	Shared Memory	(Dataflow/Systolic), Single-Instruction Single-Data (SIMD) ==> Data Parallel

Historical View, cont.

- **Machine --> Programming Model**
  - **Join at network so program with message passing model**
  - **Join at memory so program with shared memory model**
  - **Join at processor so program with SIMD or data parallel**
- **Programming Model --> Machine**
  - **Message-passing programs on message-passing machine**
  - **Shared-memory programs on shared-memory machine**
  - **SIMD/data-parallel programs on SIMD/data-parallel machine**
- **But**
  - **Isn't hardware basically the same? Processors, memory, & I/O?**
  - **Why not have generic parallel machine & program with model that fits the problem?**

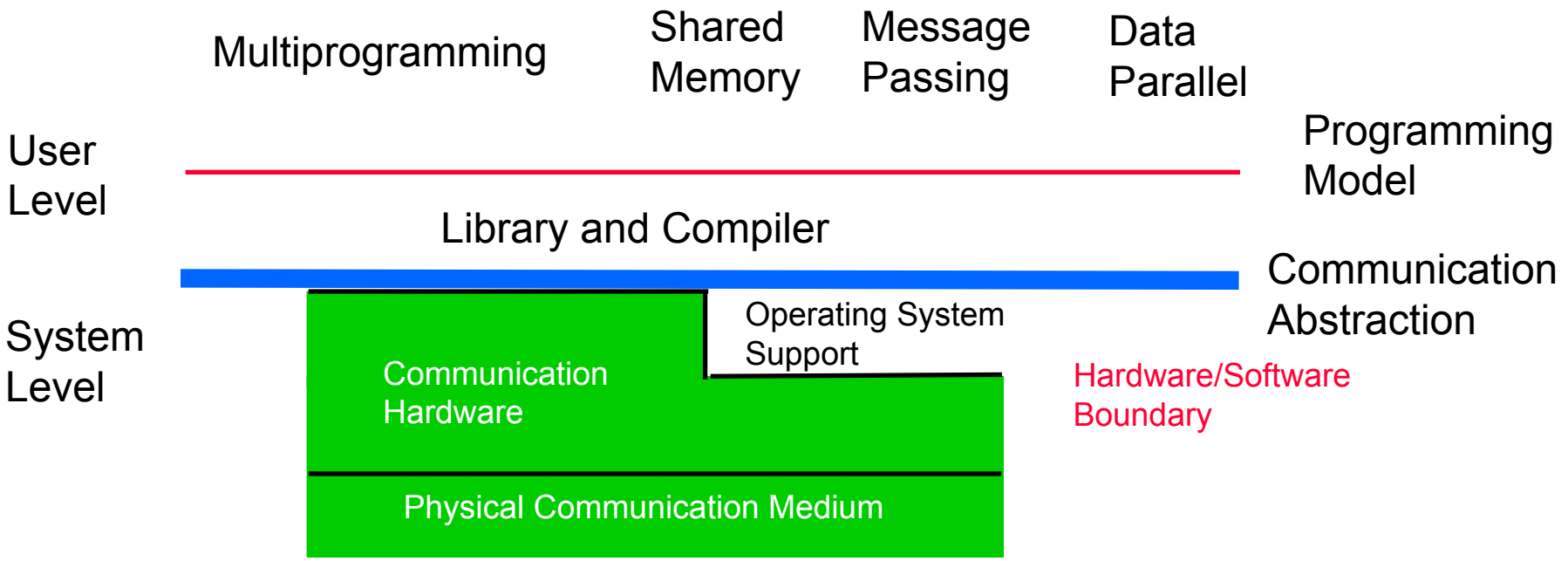
A Generic Parallel Machine



- **Separation of programming models from architectures**
- **All models require communication**
- **Node with processor(s), memory, communication assist**

Today's Parallel Computer Architecture

- **Extension of traditional computer architecture to support communication and cooperation**
  - **Communications architecture**



Simple Problem

```
for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
        sum = sum + A[i]
```

- How do I make this parallel?

Simple Problem

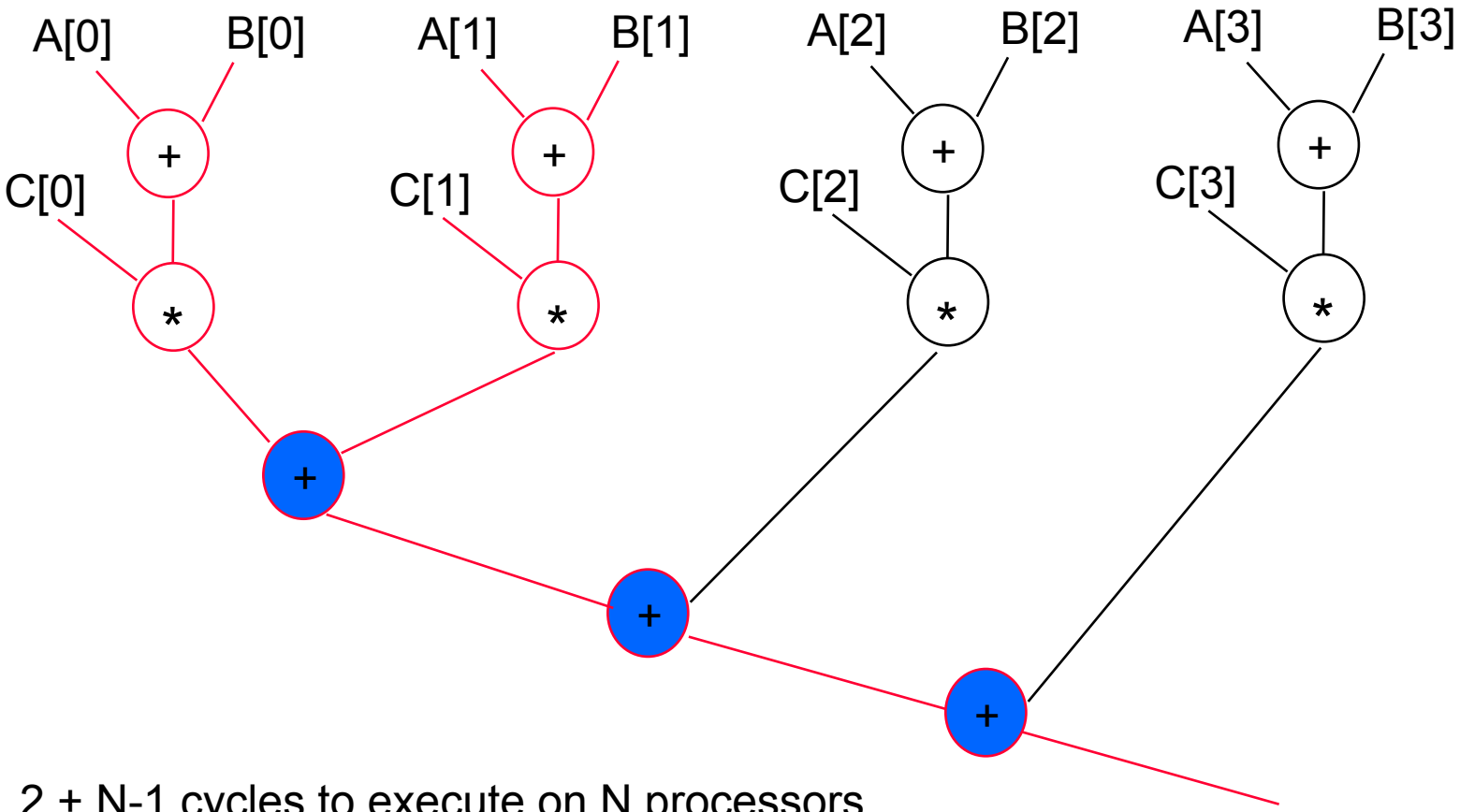
```
for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
        sum = sum + A[i]
```

- Split the loops
  - » Independent iterations

```
for i = 1 to N
        A[i] = (A[i] + B[i]) * C[i]
for i = 1 to N
        sum = sum + A[i]
```

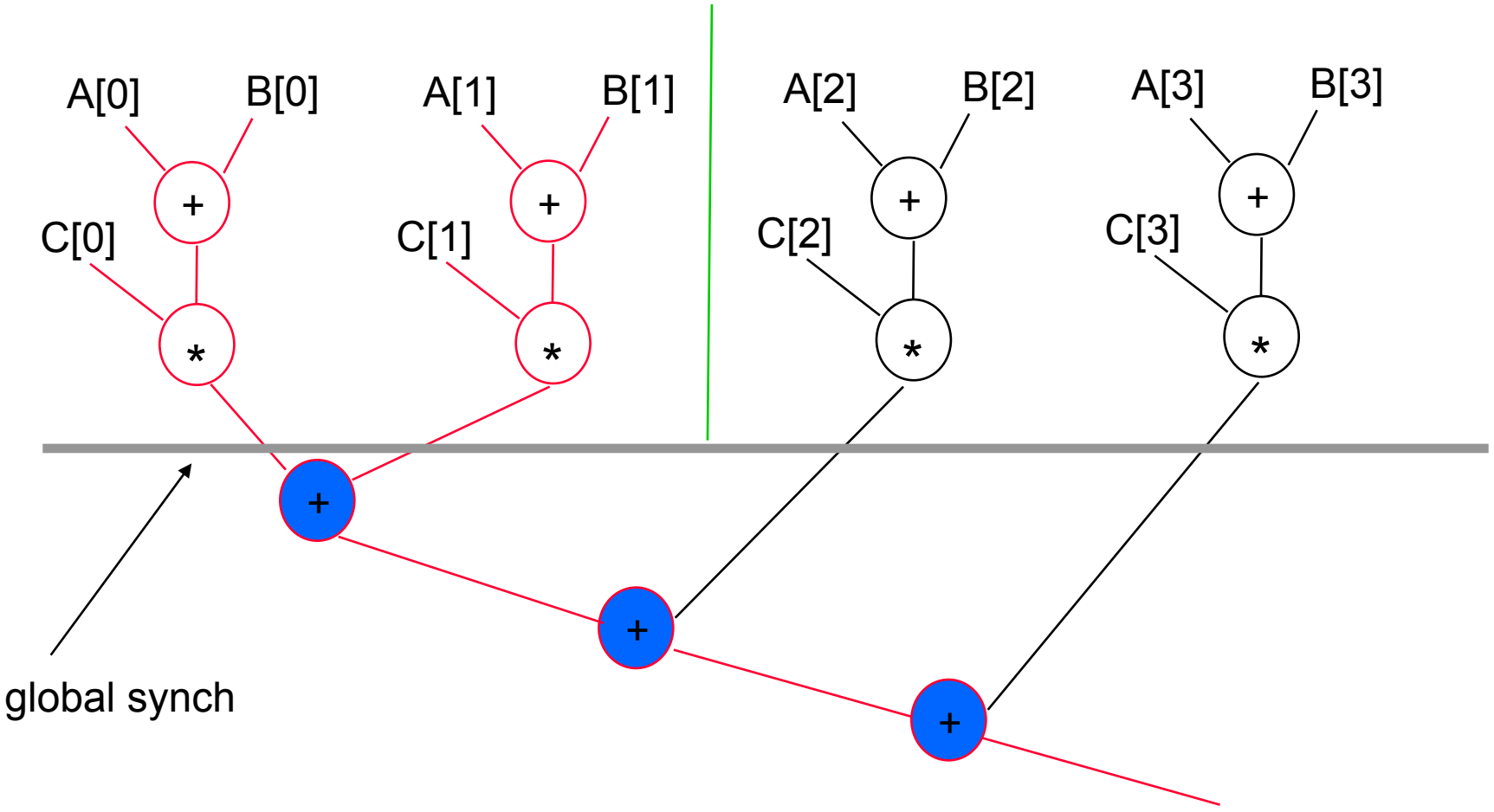
- Data flow graph?

Data Flow Graph



2 + N-1 cycles to execute on N processors
what assumptions?

Partitioning of Data Flow Graph



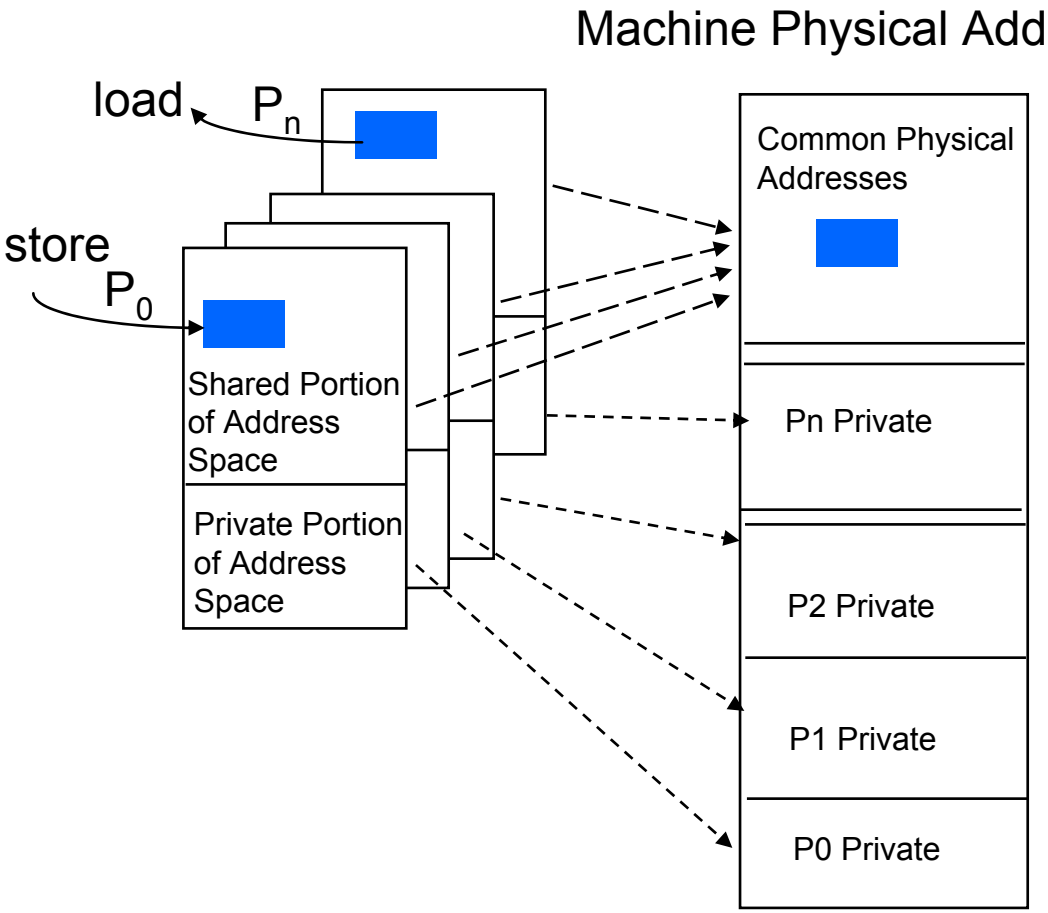
Programming Model

- **Historically, Programming Model == Architecture**
- **Thread(s) of control that operate on data**
- **Provides a communication abstraction that is a contract between hardware and software (ala ISA)**

<u>Current Models</u>

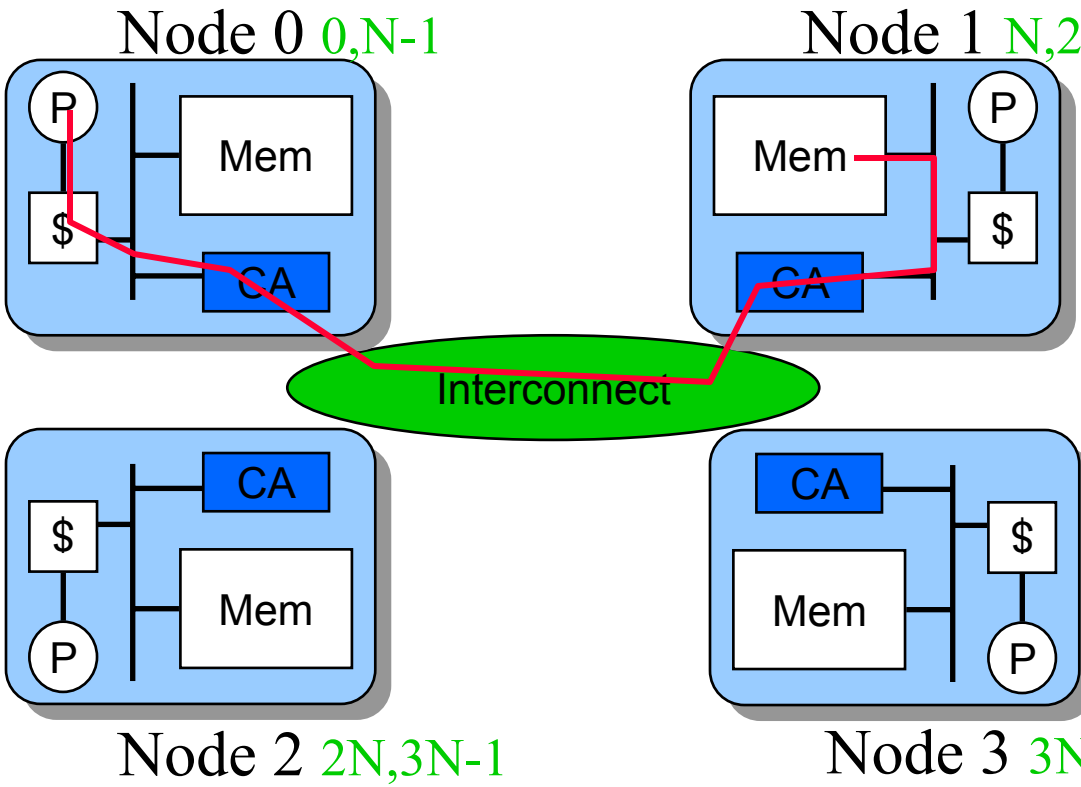
- **Shared Memory**
- **Message Passing**
- **Data Parallel (Shared Address Space)**
- **(Data Flow)**

Shared (Physical) Memory



- **Communication, sharing, and synchronization with store / load on shared variables**
- **Must map virtual pages to physical page frames**
- **Consider OS support for good mapping**

Shared (Physical) Memory on Generic MP



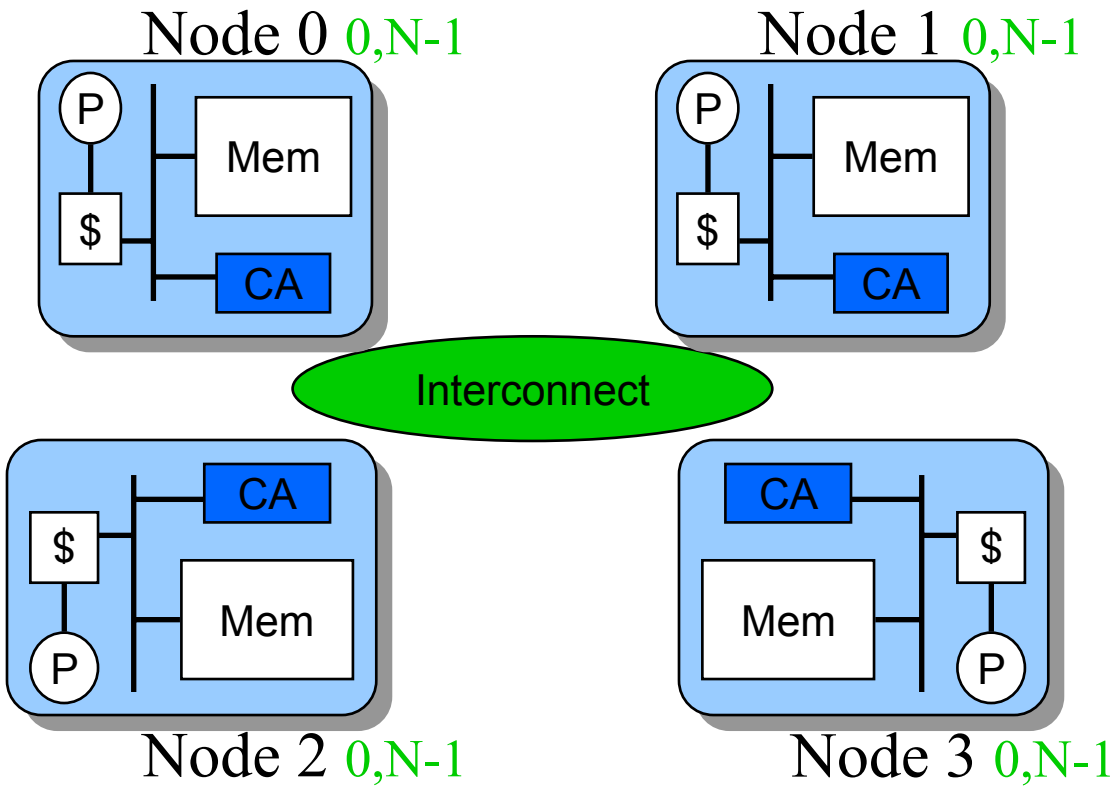
Keep private data and frequently used shared data on same node as computation

Return of The Simple Problem

```
private int i, my_start, my_end, mynode;
shared float A[N], B[N], C[N], sum;
for i = my_start to my_end
        A[i] = (A[i] + B[i]) * C[i]
GLOBAL_SYNCH;
if (mynode == 0)
        for i = 1 to N
                sum = sum + A[i]
```

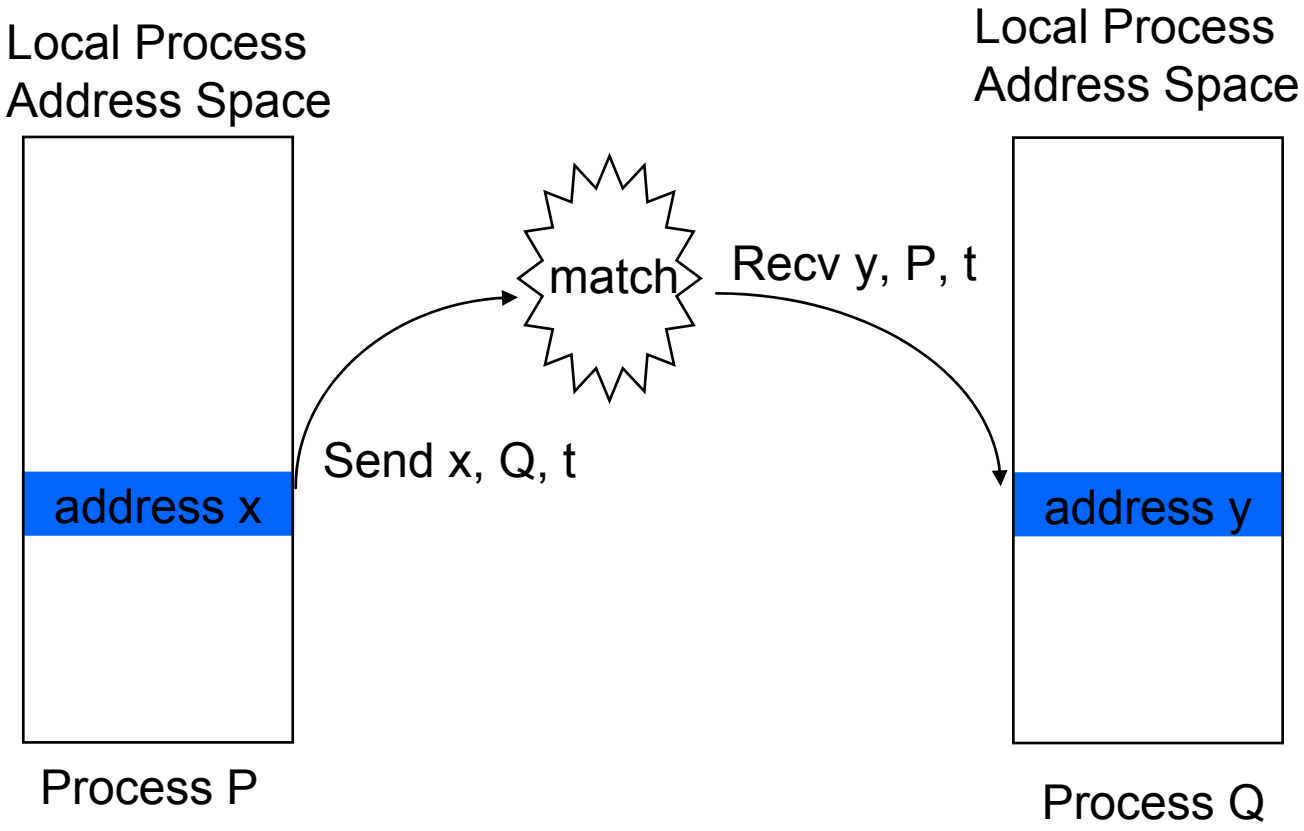
- Can run this on any shared memory machine

Message Passing Architectures



- **Cannot directly access memory on another node**
- **IBM SP-2, Intel Paragon**
- **Cluster of workstations**

Message Passing Programming Model



- **User level send/receive abstraction**
  - **local buffer (x,y), process (Q,P) and tag (t)**
  - **naming and synchronization**

The Simple Problem Again

```
int i, my_start, my_end, mynode;
float A[N/P], B[N/P], C[N/P], sum;
for i = 1 to N/P
        A[i] = (A[i] + B[i]) * C[i]
        sum = sum + A[i]
if (mynode != 0)
        send (sum,0);
if (mynode == 0)
        for i = 1 to P-1
                recv(tmp,i)
                sum = sum + tmp
```

- Send/Recv communicates and synchronizes
- P processors

Separation of Architecture from Model

- **At the lowest level SM sends messages**
  - **HW is specialized to expedite read/write messages**
- **What programming model / abstraction is supported at user level?**
- **Can I have shared-memory abstraction on message passing HW?**
- **Can I have message passing abstraction on shared memory HW?**

Data Parallel

- **Programming Model**
  - **operations are performed on each element of a large (regular) data structure in a single step**
  - **arithmetic, global data transfer**
- **Processor is logically associated with each data element**
- **Early architectures directly mirrored programming model**
  - **Many, bit serial processors**
- **Today we have FP units and caches on microprocessors**
- **Can support data parallel model on SM or MP architecture**

The Simple Problem Strikes Back

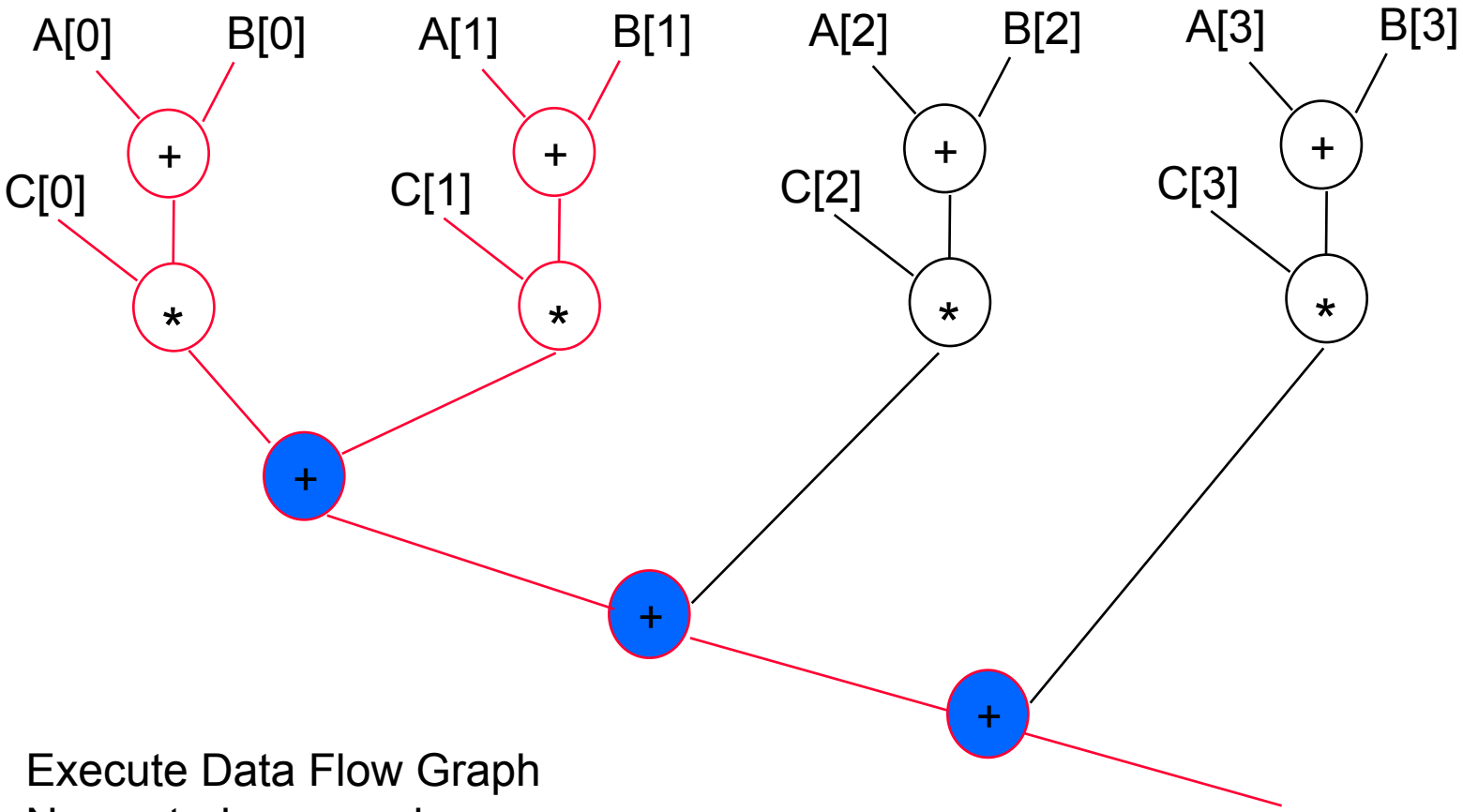
Assuming we have N processors

A = (A + B) * C

sum = global_sum (A)

- **Language supports array assignment**
- **Special HW support for global operations**
- **CM-2 bit-serial**
- **CM-5 32-bit SPARC processors**
  - **Message Passing and Data Parallel models**
  - **Special control network**
- **Chapter 11…..**

Data Flow Architectures



Execute Data Flow Graph
No control sequencing

Data Flow Architectures

- Explicitly represent data dependencies (Data Flow Graph)
- No artificial constraints, like sequencing instructions!
  - Early machines had no registers or cache
- Instructions can "fire" when operands are ready
  - Remember tomasulo's algorithm
- How do we know when operands are ready?
- Matching store
  - large associative search!
- Later machines moved to coarser grain (threads)
  - allowed registers and cache for local computation
  - introduced messages (with operations and operands)