# Current Parallel Machines

CS 258 Parallel Computer Architecture
Lecture 2

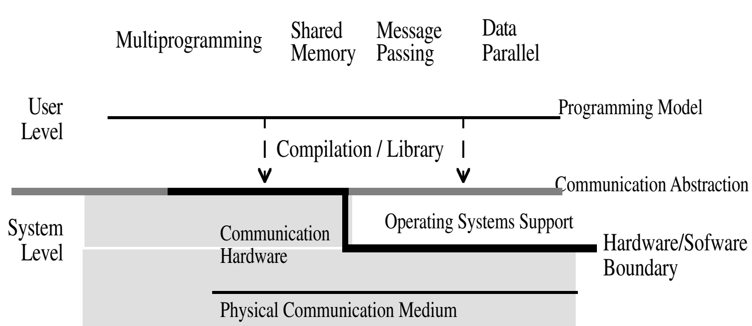




## Communication Architecture

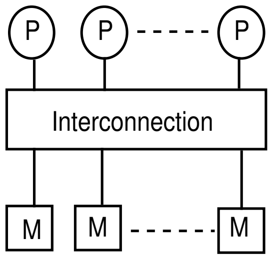
- Extension of "computer architecture" to support communication and cooperation
  - **Instruction Set Architecture and organization**

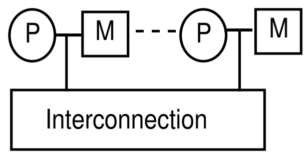
  **plus**
  - **Communications Architecture**



## Shared Memory Architectures

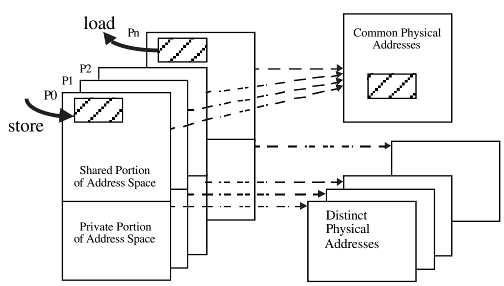
- Key Feature: All processors in the system can directly access all memory locations in the system, thus providing a convenient and fast (?) mechanism for processors to communicate.
  - **convenient: (i) location transparency; (ii) abstraction supported is same as that on current day uniprocessors.**




- Memory can be centrally placed or distributed in such machines.

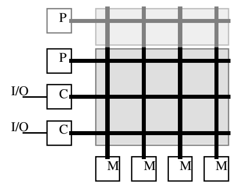
## Single Shared Address Space



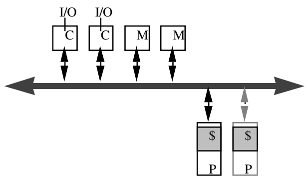

**Communication, sharing and synchronization through store / load on shared variables (test&set, exch, load-locked, store-conditional)**

## Hardware Organizations

- "Mainframe" approach
  - **motivated by multiprogramming**
  - **extends xbar**
  - **latency**
  - **bandwidth**
  - **cost**



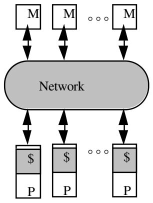

- "Minicomputer" approach
  - **motivated by multiprogramming and transaction processing**
  - **used heavily for parallel comp.**
  - **latency**
  - **bandwidth**
    - » **caching is key**
  - **cost**

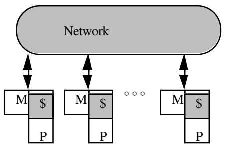






## Scalability

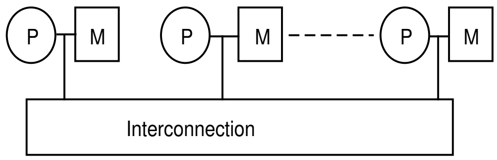



- Latency, Bandwidth, and Cost
- Construct shared memory out of simple message transactions across a general-purpose network
  - **read-request, read-response**
- Caching?

## Message Passing Architectures

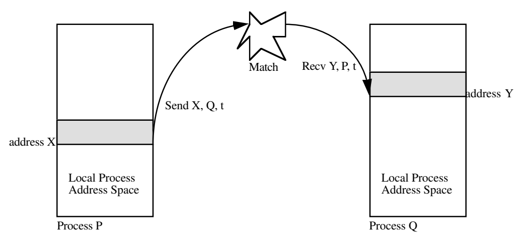
- Processors can directly access only local memory, and all communication and synchronization happens via messages.








## Message Passing Abstraction




- Send + Matching Recv provides a memory-memory copy plus a pairwise synchronization event
- Substantial layer of software between user primitives and HW organization
- Highly structured programs (SPMD) are common
- Sending a message often incurs many overheads:
  - **building a header; copying data into network buffers; sending data; receiving data into buffers; copying data from kernel to user space.**

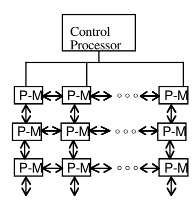
## Evolution and Convergence

- Early hypercubes provide FIFO on each link
  - **HW close to Prog. Model**
  - **Soon replaced by DMA transfers and non-blocking operation**
- Diminishing role of topology
  - **store&forward in the comm. software, in the routers**
  - **general purpose (cut-through) rougint**
- Contraint that only local addresses are used in artifact of P.M.
- User-level networks (CM-5)
- Complete message processor (Paragon, Meiko CS-2)
- User-to-user DMA
- Most shared-memory efforts now optimized toward msg passing (Alewife, Flash)
  - » **A single-address space machine is specialized to handle memory read/write messages in hardware and is thus faster at such operations.**




## Data Parallel Architectures

- 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, general communication, and cheap global synchronization.
  - **driven originally by simple O.D.E.**



## Application

- Example:

  **Each PE contains an employee record with his/her salary**

  **Code:**
```
If salary > 100K then
        salary = salary *1.05
else
        salary = salary *1.10
```
  **Logically, the whole operation is a single step**

  **Notion of enabled processors.**
- Other examples:
  - **Finite differences, linear algebra, ...**
  - **Document searching, graphics, image processing, ...**
- Recent Machines:
  - **Thinking Machines CM-1, CM-2 and CM-5**
  - **Maspar MP-1 and MP-2,**
  - **Wavetracer**



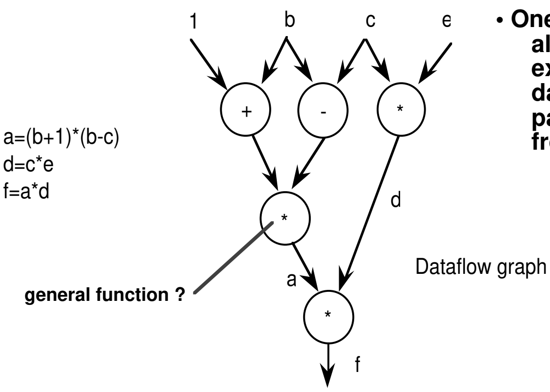

## Evolution and Convergence

- Rigid control structure (SIMD in Flynn's Taxonomy)
  - **SISD = uniprocessor, MIMD = multiprocessor**
- Cost savings in centralized instruction sequencer
  - **60's when CPU was a cabinet of eqipment**
  - **mid 80's when 32-bit slices of datapath would just fit on a chip**
- Simple, regular calculations usually have good locality
  - **realize on SM or MP machine with decent compiler**
  - **may still require fast global synchronization**
- Programming model converges with SPMD
  - **single program multiple data**

## Dataflow Architectures

- In the dataflow model, instructions are activated by the availability of data operands for instructions.
- In contrast, in control flow models, computation is a series of instructions with implicit or explicit sequencing between them.

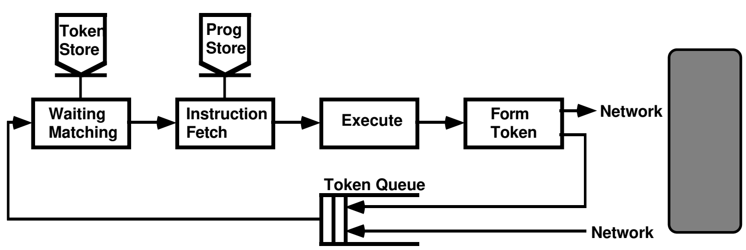
a=(b+1)*(b-c)
d=c*e
f=a*d




- **One of the advantages is that all dependencies are explicitly present in the dataflow graph, so parallelism is not hidden from hardware.**



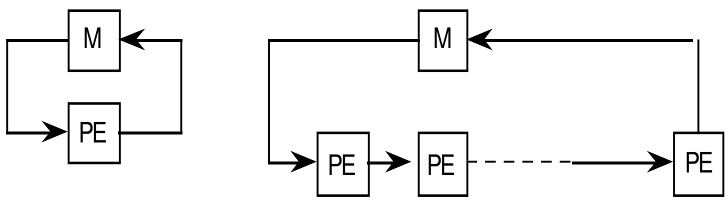

## Dataflow Machine Structure




- Static vs Dynamic Dataflow Machines
  - **function (task) invocation**
- Role of local state
- Granularity of operations
- Efficient handling of complex data structures (arrays)
- Hardware complexity
- Problems due to Excess parallelism
- Convergence toward fast dispatch on user level message

## Systolic Architectures

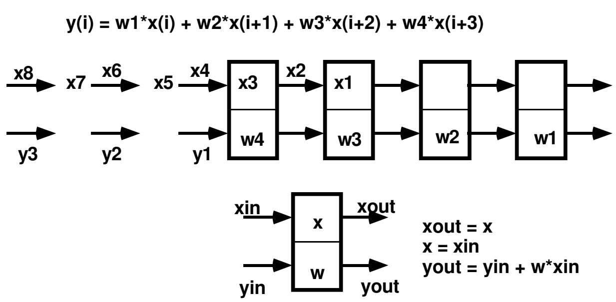
- **Basic principle: By replacing a single PE by a regular array of PEs, and by carefully orchestrating the flow of data between the PEs, high throughput can be achieved without increasing memory bandwdith requirements.**
- Distinguishing features from regular pipelined computers:
  - **• Array structure can be non linear (e.g, hexagonal)**
  - **• Pathways between PEs may be multidirectional**
  - **• PEs may have local instruction and data memory, and are more complex than the stage of a pipelined computer.**





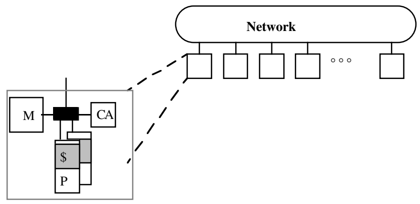



## Example: 1D convolution




- Issues:
  - **System integration: Shipping data from host array and back**
  - **Cell architecture and communication architecture (Warp, iWarp)**
  - **Software for automatically mapping computations to systolic arrays**
  - **General purpose systolic arrays**

## Convergence toward a Generic Parallel Machine




- Node: one or more processors, memory system, plus *communication assist*
  - **network interface**
  - **integration with processor, memory system, OS determines fundamental capabilities**
- Scalable network





## Fundamental Architectural Issues

- Naming: How is communicated data and/or partner referenced?
- Synchronization: How can producers and consumers of data coordinate their activities?
- Latency: How long does it take to communicatedata in a protected fashion?
- Bandwidth: How much data can be communicated per second? How many operations per second?

## Naming

- Single Global Linear-Address-Space (shared memory)
  vs.
- Single Global Segmented-Name-Space (global objects)
  vs.
- Multiple Local Address/Name Spaces (message passing)

- Naming strategy has implications on:
      » **- Programmer / Software**
      » **- Performance**
      » **- Design Complexity**

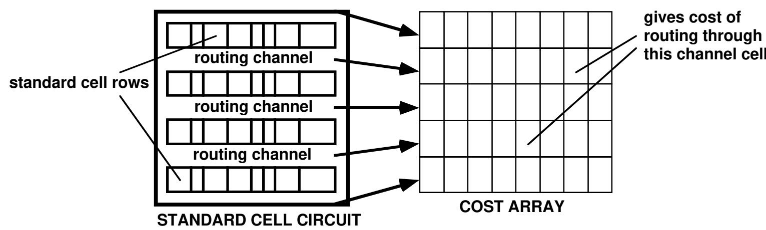



## Synchronization

- Coordination mainly takes three forms:
  - **multual exclusion (e.g., spin-locks)**
  - **event notification**
    - » **point-to-point (e.g., producer-consumer interactions)**
    - » **global / point-to-multipoint (e.g., end of phase indication)**
  - **global operations (e.g., sum)**
- Issues:
  - **synchronization name space (all address space or small portion)**
  - **granularity (per byte, per word, ... ==> overhead)**
  - **low latency, low serialization (hot spot handling)**
  - **many proposals**
    - » **test-&-set, compare-&-swap, ldLocked-stCond,**
    - » **F/E bits and traps,**
    - » **queue-based locks, fetch&op with combining, ...**
    - » **scans, fuzzy barriers, . . .**

## Programmer / Software

- **Example: LocusRoute (standard cell router)**




```
while (route_density_improvement > threshold)
  {
    for (i = 1 to num_wires) do
      {
        - rip old wire route out
        - explore new routes
        - place wire using best new route
      }
  }
```

## Shared-Memory Implementation

- Shared memory algorithm:
  - **Divide cost-array into regions**
    - **logically assign regions to PEs**
  - **Assign wires to PEs based on the region in which center lies**
  - **Do load balancing using stealing when local queue empty**
- Good points:
  - **Good load balancing**
  - **Mostly local accesses**
  - **High cache-hit ratio**
- Not so good
  - **non-deterministic**
  - **potential hot spots**
  - **amount of parallelism**

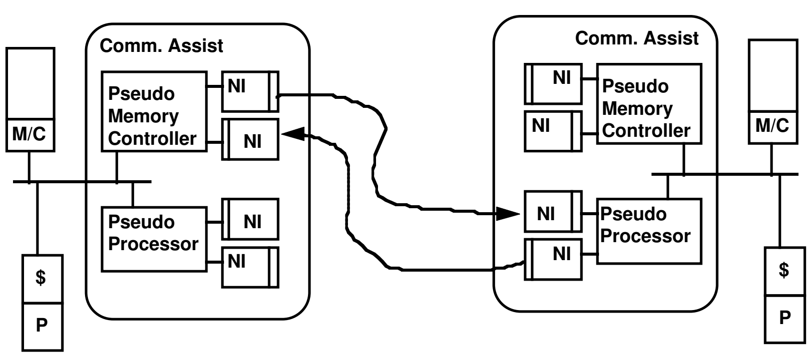
 

## Message-Passing Implementations

- Solution-1:
  - Distribute wires and cost-array regions as in sh-mem implementation
  - Handle when wire-path crosses to remote region
    - » • send computation to remote PE, or
    - » • send messages to access remote data
    - »
- Solution-2:
  - Wires distributed as in sh-mem implementation
  - Each PE has copy of full cost array
    - » one owned region, plus potentially stale copy of others
    - » send frequent updates so that copies not too stale
  - Consequences:
    - » waste of memory in replication
    - » stale data => poorer quality results or more iterations
- In either case, lots of thinking needed on programmer's part




## HW Complexity: Shared physical address space



## Performance

- 1. Efficient naming
  - **virtual to physical using TLBs**
  - **ability to name only the relevant portion of objects (almost)**
- 2. Ease and efficiency of caching
  - **caching is natural and well understood**
  - **can be automated using hardware (fixed sized blocks)**
  -
- 3. Communication overhead
  - **low since protection is built into the memory system**
  - **easy for hardware to packetize requests and replies**
- 4. Integration of latency toleration
  - **demand-driven: consistency models, prefetching, multiple-ctxt PEs**
  - **can be extended to push data to other PEs plus bulk transfer**




## Latency

- Ability to deal with latency is critical with fast processors
-
- Options for dealing with latency:
  - **1. Lower frequency of long latency events**
    - » **• algorithmic changes, computation and data distribution, ...**
    - »
  - **2. Reduce latency**
    - » **caching of shared data, network-interface design, network, ...**
    - »
  - **3. Tolerate latency**
    - » **message-passing overlaps communication and computation (under programmer control)**
    - » **shared-memory overlaps access completion and computation using consistency model or prefetching**

## Bandwidth

- MP programs have private and global bandwidth requirements
- 
- Private bandwith requirements can be supported using:
  - **distributing main memory among PEs**
  - **application changes, local caches, mem system design, ...**
  - 
- • Global bandwidth requirements can be supported using:
  - **scalable interconnect technology**
  - **distributed main memory and caches**
  - **efficient network interfaces**
  - **application changes (e.g., avoiding hot-spots)**

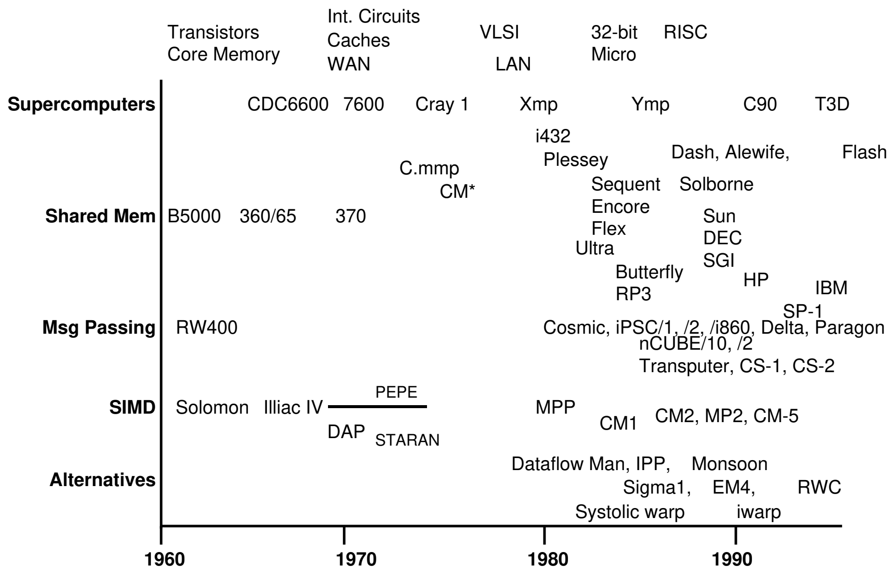



## Cost of Communication

$$\textbf{Cost} = \textbf{Frequency} \times \left( \textbf{Overhead} + \textbf{Latency} + \frac{\textbf{Transfer Size}}{\textbf{Bandwidth}} - \textbf{Overlap} \right)$$

- Frequency - number of communications per unit work
  - **algorithm, placement, replication, bulk**
- Overhead - processor cycles spent initiating or handling
  - **protection check, status, buffer mgmt, copies, events**
- Latency - time to move bits from source to destination
  - **comm. assist, topology, routing, congestion**
- Transfer Time - time through bottleneck
  - **comm. assist, links, congestion**
- Overlap - portion overlapped with useful work
  - **comm. assist, comm. operations, proc. design**

## Historical Perspective




cs258 2.29