Introduction to Parallel Computation

Wolfgang Schreiner
Research Institute for Symbolic Computation (RISC-Linz)
Johannes Kepler University, A-4040 Linz, Austria

Wolfgang.Schreiner@risc.uni-linz.ac.at
http://www.risc.uni-linz.ac.at/people/schreine

A Graph-Theoretical Problem

"All Pairs Shortest Paths"

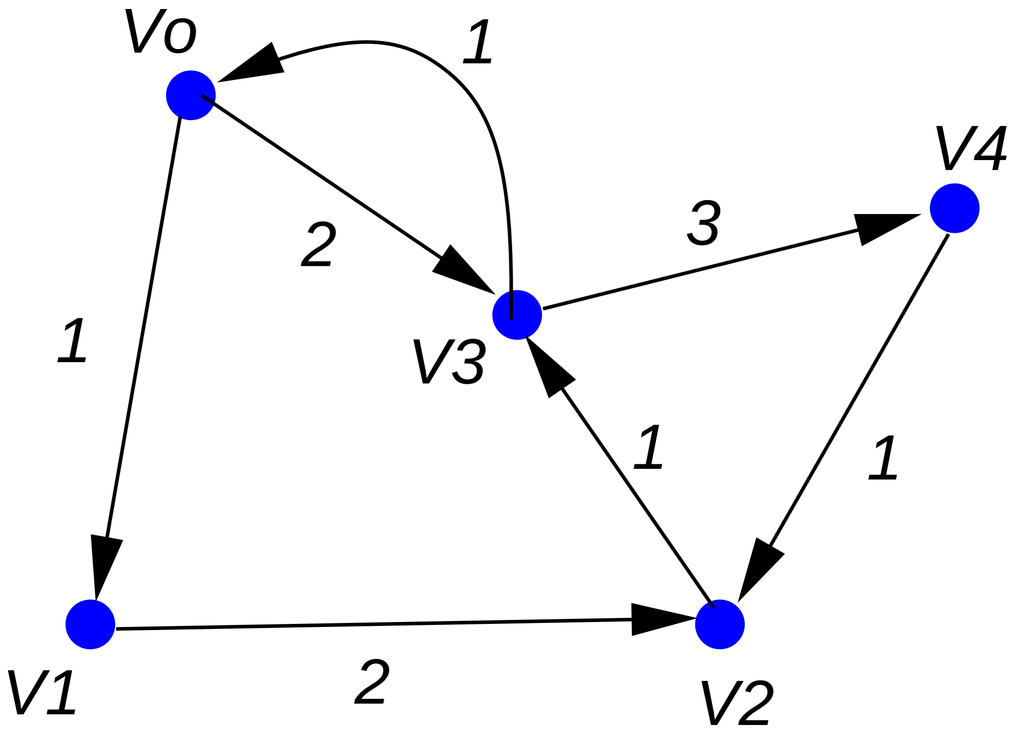
- Given: graph $G = (V, A)$
  - Directed acyclic graph, $|V| = n$
- Representation: weight matrix W

$$W(i,j) = \begin{cases} 0 & \text{if } i = j \\ w > 0 & \text{if edge of length } w \\ & \text{between nodes } i \text{ and } j \\ \infty & \text{if no edge between } i \text{ and } j \end{cases}$$

- Wanted: distance matrix D

$$D(i,j) = \begin{cases} 0 & \text{if } i = j \\ d > 0 & \text{if } i \neq j \text{ where } d \text{ length of} \\ & \text{shortest path between } i \text{ and } j \\ \infty & \text{if no edge between } i \text{ and } j \end{cases}$$

Example



W	v_0	v_1	v_2	v_3	v_4
v_0	0	1	∞	2	∞
v_1	∞	0	2	∞	∞
v_2	∞	∞	0	1	∞
v_3	1	∞	∞	0	3
v_4	∞	∞	1	∞	0

$\Rightarrow$

D	v_0	v_1	v_2	v_3	v_4
v_0	0	1	3	2	5
v_1	4	0	2	3	6
v_2	2	3	0	1	4
v_3	1	2	4	0	3
v_4	3	4	1	2	0

Solution Idea

- Construct sequence of matrices
  - $D_0, D_1, \ldots, D_{n-1}$
  - D_i describes all shortest paths with not more than i edges.
- Consequence: $D_{n-1} = D$
- Proof
  - Assume shortest path p with more than $n-1$ edges. Then there is some node v twice in this path i.e. $p = \langle i, \ldots, \underline{v, \ldots, v}, \ldots, j \rangle$. But then path $p' = \langle i, \ldots, v, \ldots, j \rangle$ is shorter!

Construction

$$D_0(i,j) = \begin{cases} 0 & \text{if } i = j \\ \infty & \text{if } i \neq j \end{cases}$$
$$D_1(i,j) = W(i,j)$$

$$D_{r+1}(i,j) = ?$$

Two Cases

Let $l \leq r$ be the number of edges of the shortest path between i and j captured by $D_r(i,j)$.

1. $|p| = l$
   - $p = \langle \underbrace{i, \ldots, j}_{l \text{ edges}} \rangle$

     $D_{r+1}(i,j) = D_r(i,j)$
2. $|p| = r + 1$
   - $p = \langle \underbrace{i, \ldots, k}_{r \text{ edges}}, j \rangle$

     $D_{r+1}(i,j) = D_r(i,k) + W(k,j)$

$$D_{r+1}(i,j) = \min\{D_r(i,j), \min_k\{D_r(i,k) + W(k,j)\}\}$$

Sequential Algorithm

```
AllPairsShortestPaths(W):
  D = W
  for r=1 to n-1 do
    D = MatMin(D, W)
  return D

MatMin(D, W):
  for i=1 to n do
    for j=1 to n do
      E[i,j] = D[i,j]
      for k=1 to n do
        E[i,j] = min(E[i,j],D[i,k]+W[k,j])
  return E
```

Observation

`MatMin` has same structure as matrix multiplication ($+ \longrightarrow$ min$, * \rightarrow +$).

- Define $D \times W = \texttt{MatMin}(D, W)$
- Begin: $D_1 = W$
- General: $D_i = D^{i-1} \times W = W^i$
- End: $D = D_{n-1} = W^{n-1}$

Problem solution is essentially repeated matrix multiplication!

Optimization

- Instead of computing
  - $W^1, W^2, W^3, \ldots, W^{n-1}$
- we can compute
  - $W^1, W^2, W^4, \ldots, W^{2^s}$
    (where $2^s \geq n-1$).

True for matrix multiplication as well as for `MatMin` (since both are associative).

```
AllPairsShortestPaths(W):
  D = W
  for r=1 to s do
    D = MatMin(D, D)
  return D
```

We can reduce n matrix multiplications to log n square computations!

Time Analysis

- n nodes.
- log n square computations.
- n^3 (min,$+$) operations for each square computation.

Sequential time complexity $O(\log\ n * n^3)$

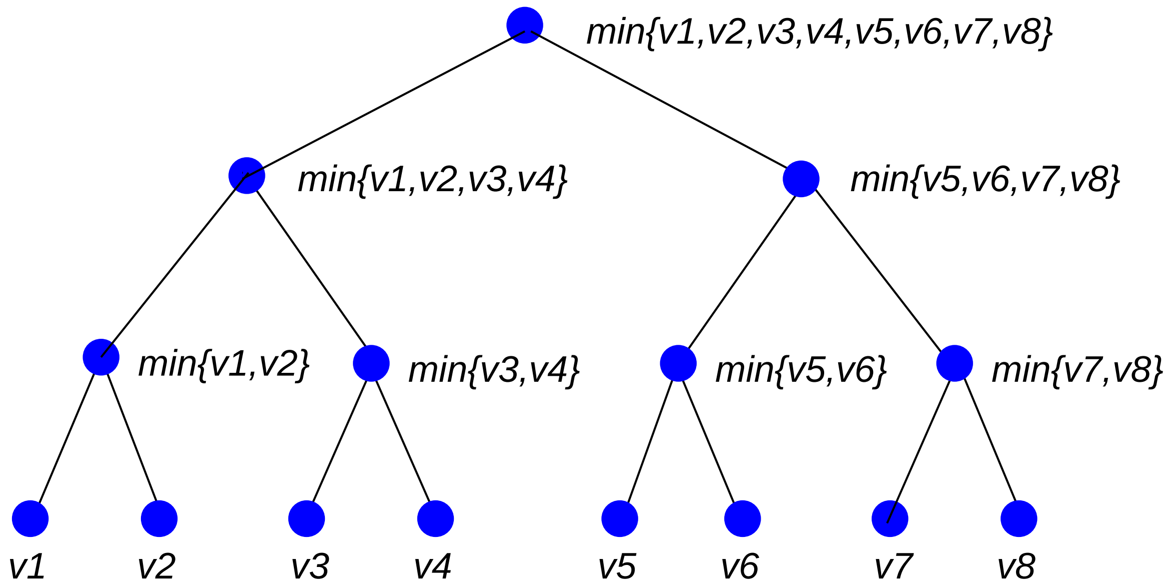
Parallel Algorithm

- Sequence of square operations.
- Each square op. contains n^3 independent (min,$+$) operations that can be performed in parallel yielding n^3 results.
- Each of the n^2 entries $D(i, j)$ is a minimum of n values.
- Time complexity:
  - log n square computations.
  - 1 time step for all (min,$+$) operations.
  - log n time to compute each of the n^2 minimums.

Parallel time complexity $O(\log^2 n)$

Minimum of n Values

Tree-like minimum construction



Depth of tree = *computation time* = $O(\log n)$

Comparison

- General
  - Sequential: $O(\log n * n^3)$
  - Parallel: $O(\log^2 n)$
  - Processors: $O(n^3)$
- Time/processor product
  - Sequential product: $O(\text{log } n * n^3)$
  - Parallel product: $O(\text{log}^2 n * n^3)$

Parallel algorithm is in some sense less efficient than sequential one!

Parallel Machine Models

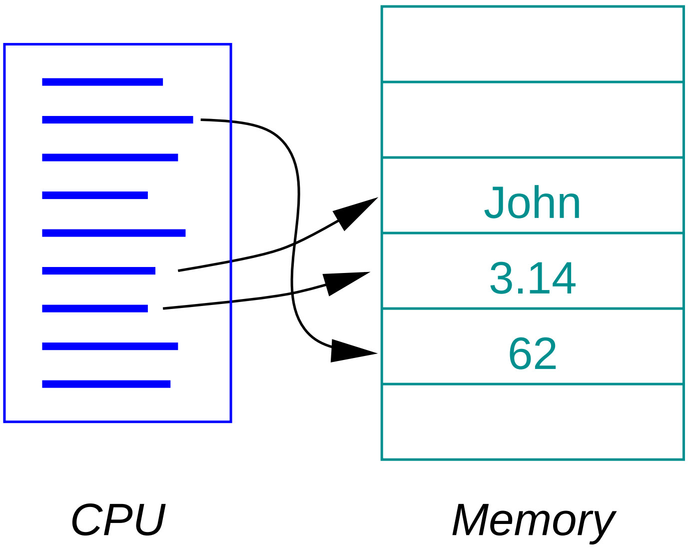
- How to define algorithm?
- How to implement algorithm?
- How to analyze algorithm?

We need a parallel machine and programming model!

Sequential Machine Model

Von Neumann Computer, Random Access Machine (RAM).

- Central Processing Unit (CPU)
  - Executes a stored program.
- Storage Unit (Memory)
  - Random access to any memory cell.

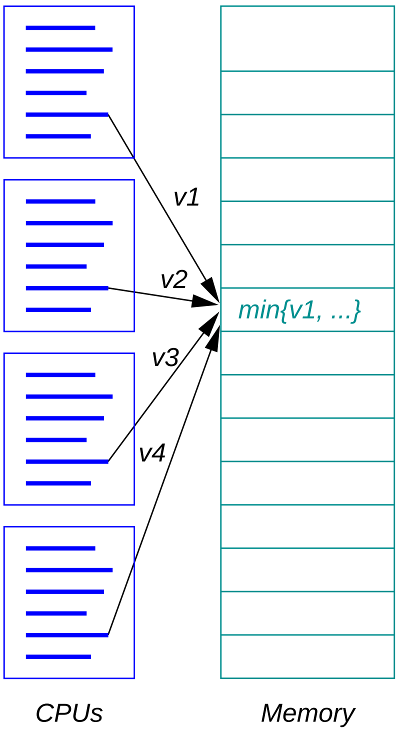


Program execution is sequence of read/write operations on memory.

PRAM Model

Parallel Random Access Machine.

- Set of CPUs.
- CPUs operate on same memory.
- CPUs execute same program lock-step.



Theoretical model for designing and analyzing parallel algorithms.

PRAM Variants

Restrictions of access to memory cells.

- EREW (exclusive read, exclusive write)
  - Only one access to individual memory cell at a time.
- CREW (concurrent read, exclusive write).
  - Multiple concurrent reads to a memory cell, but writes are exclusive.
- CRCW (concurrent read, concurrent write).
  - Multiple concurrent writes allowed, random value (maximum value, sum, ...) is written.

Different variants may yield different complexities of parallel algorithms.

PRAM Program

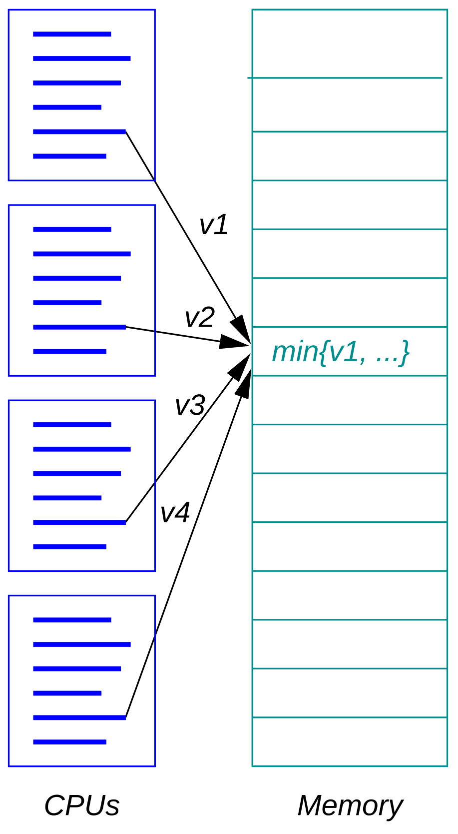
```
AllPairsShortestPaths(W):
  D = W
  for r=1 to s do
    D = MatMin(D, D)
  return D

MatMin(D, W):
  for i=1 to n do in parallel
    for j=1 to n do in parallel
      for k=1 to n do in parallel
        M[i,j,k] =
          min(D[i,j], D[i,k]+W[k,j])
      E[i,j] = MIN(M[i,j])
  return E
```

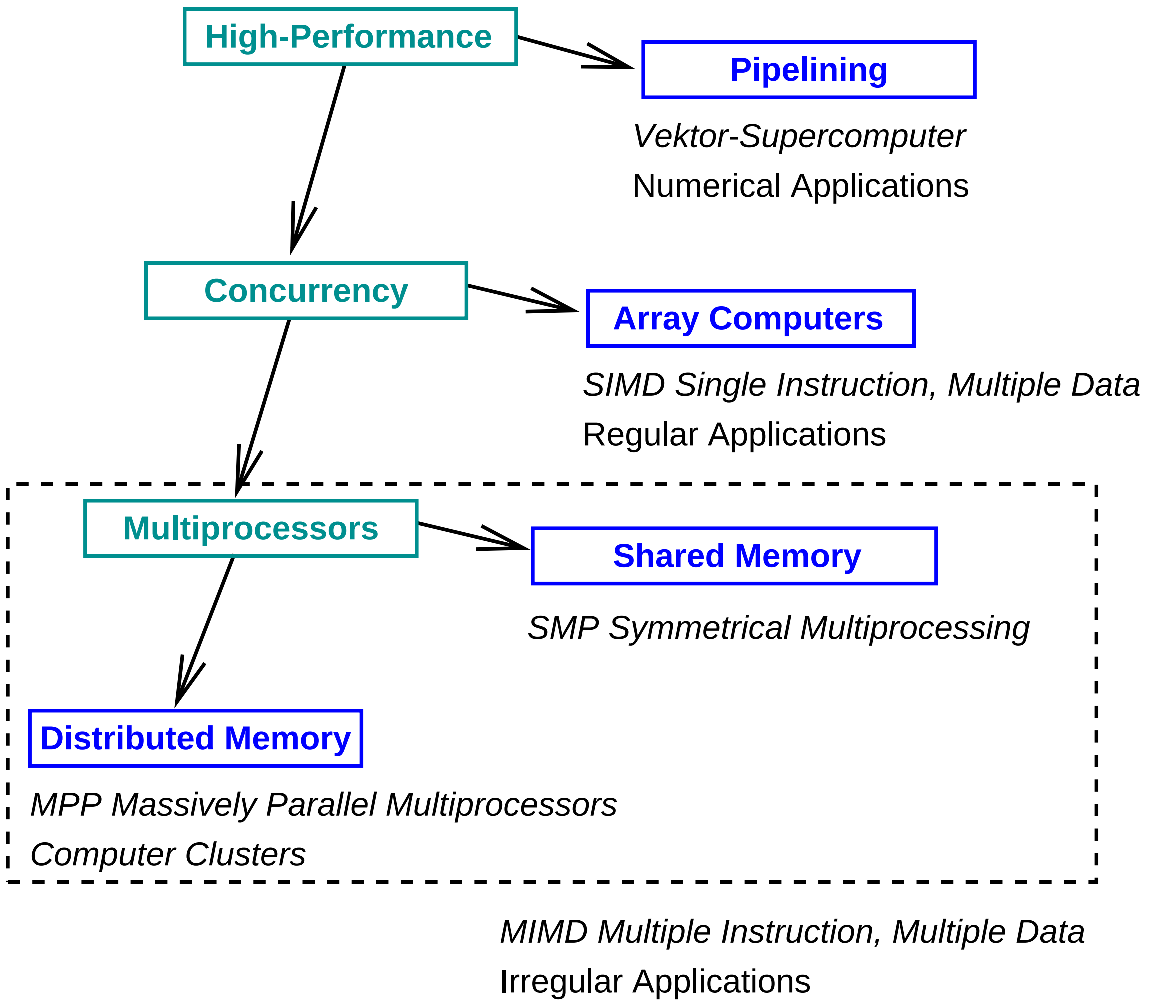
MIN computes minimum of n values.

Complexity of MIN

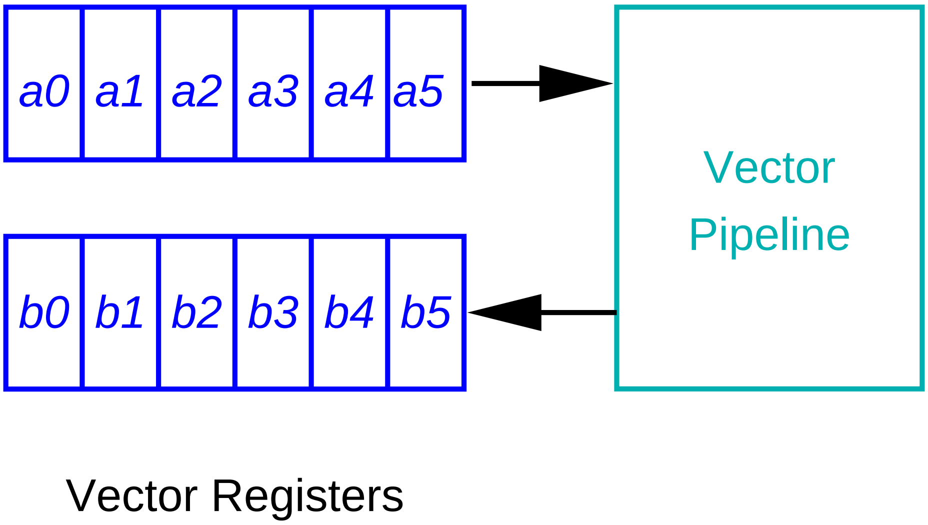
- EREW, CREW: $O(\log n)$
  - Tree-like combination of values.
- CRCW: $O(1)$
  - All values written simultaneously into same cell, minimum value remains.



High-Performance Architectures

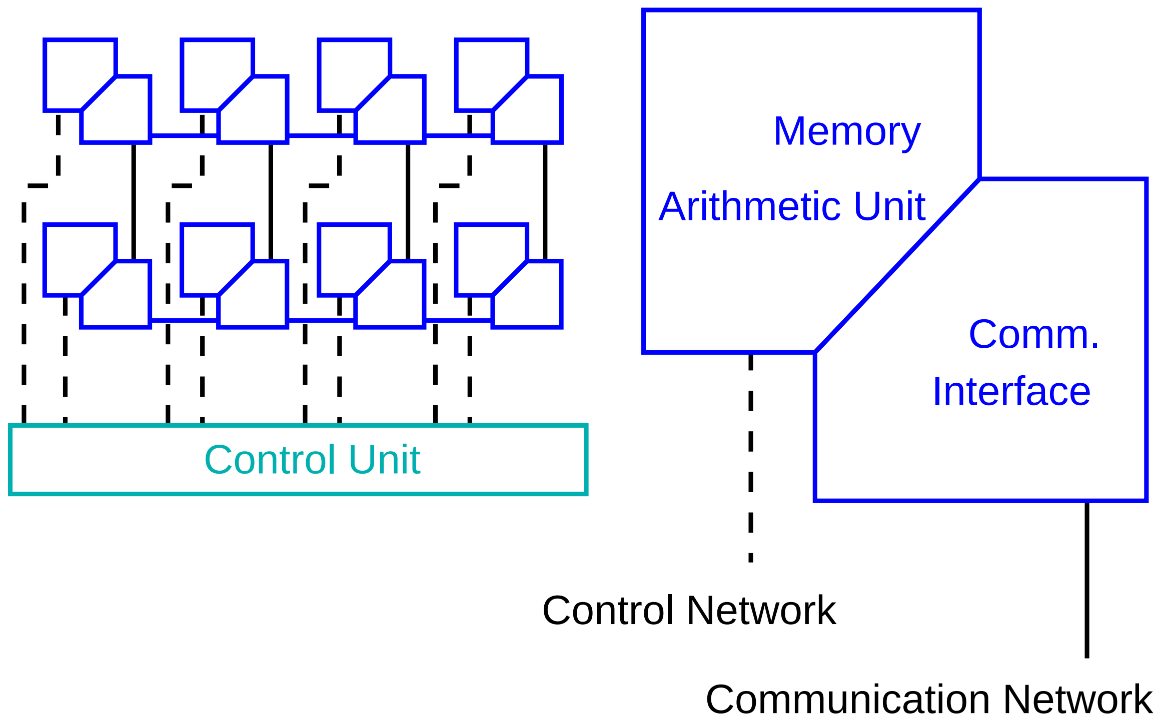


Vector Supercomputers



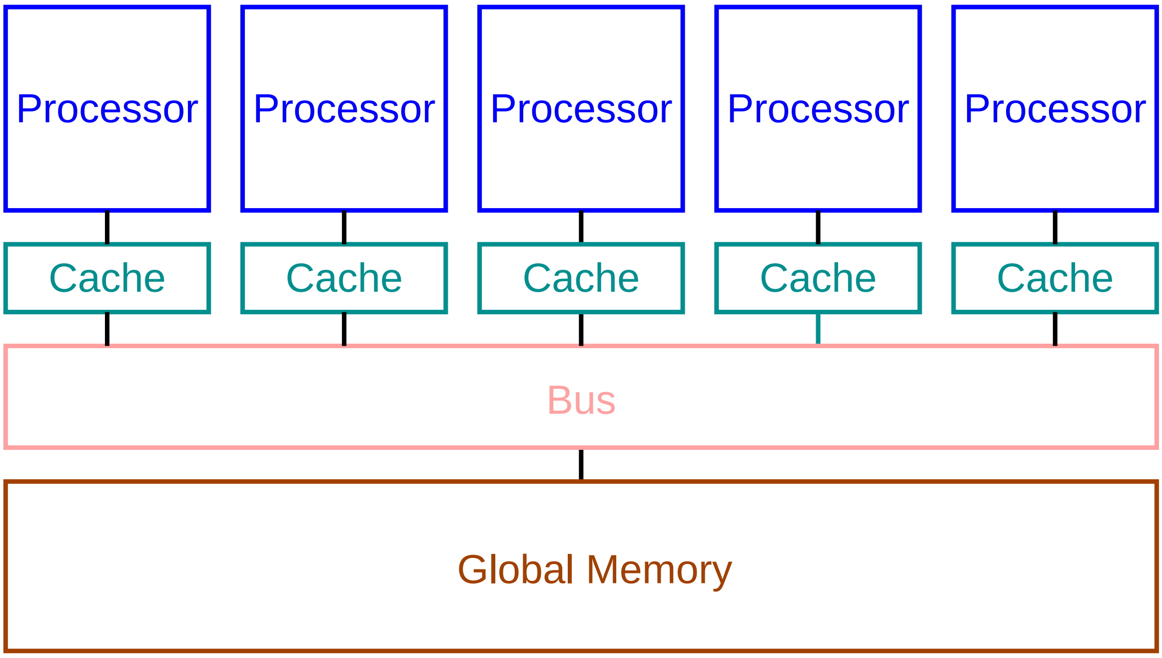
Vectors of numbers are passed through arithmetic pipeline.

SIMD Array Computers



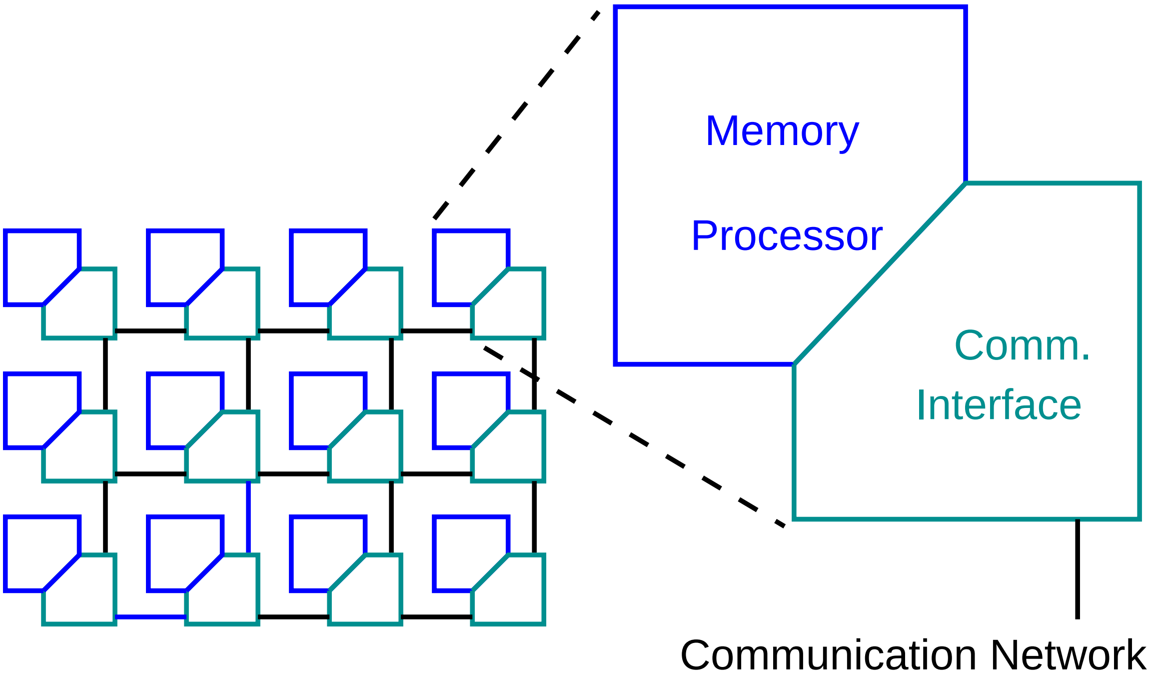
Array of arithmetic units operates and communicates in lock-step.

Shared Memory Multiprocessors



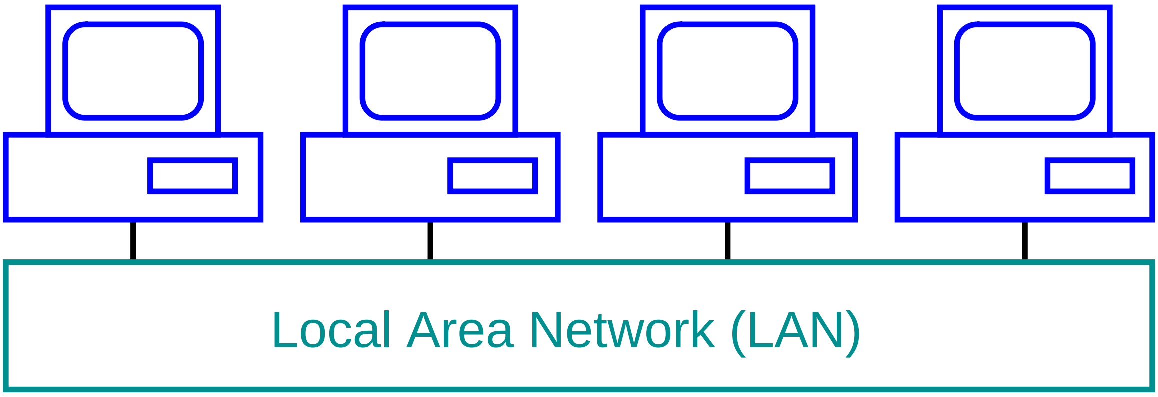
Processors operate asynchronously on shared memory.

Distributed Memory Multiprocessors



Processors operate asynchronously on local memory and communicate by point-to-point network.

Computer Clusters



Cluster of independent computers cooperate via local network.

A Parallel Program

```
ccccccccccccccccccccccccccccccccccccccccccccccccc
c      Matrix Multiplication MPI Program      c
c      For this simple version, # of procssors c
c      equals # of columns in matrix          c
c                                             c
c      To run, mpirun -np 4 a.out             c
ccccccccccccccccccccccccccccccccccccccccccccccccc

      include 'mpif.h'

      parameter (ncols=4, nrows=4)
      integer a(ncols,nrows), b(ncols,nrows), c(ncols,nrows)
      integer buf(ncols),ans(nrows)
      integer myid, root, numprocs, ierr, status(MPI_STATUS_SIZE)
      integer sender, count

      call MPI_INIT(ierr)
      call MPI_COMM_RANK( MPI_COMM_WORLD, myid, ierr )
      call MPI_COMM_SIZE( MPI_COMM_WORLD, numprocs, ierr )

      if(numprocs.ne.4) then
        print *, "Please run this exercise on 4 processors"
        call MPI_FINALIZE(ierr)
        stop
      endif

      root = 0
      tag = 100
      count = nrows*ncols

c     Master initializes and then dispatches to others
      IF ( myid .eq. root ) then

         do j=1,ncols
            do i=1,nrows
               a(i,j) = 1
               b(i,j) = j
            enddo
         enddo
```

A Parallel Program (Contd)

```
c        send a to all other process
         call MPI_BCAST(a,count,MPI_INTEGER,root,MPI_COMM_WORLD,ierr)

c        send one column of b to each other process
         do j=1,numprocs-1
            do i = 1,nrows
               buf(i) = b(i,j+1)
            enddo
         call MPI_SEND(buf,nrows,MPI_INTEGER,j,tag,MPI_COMM_WORLD,ierr)
         enddo

c        Master does his own part here
         do i=1,nrows
            ans(i) = 0
            do j=1,ncols
               ans(i) = ans(i) + a(i,j) * b(i,1)
            enddo
         c(i,1) = ans(i)
         enddo

c        then receives answers from others

         do j=1,numprocs-1
            call MPI_RECV(ans, nrows, MPI_INTEGER, MPI_ANY_SOURCE,
     $           MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)

            sender = status(MPI_SOURCE)
            do i=1,nrows
               c(i,sender+1) = ans(i)
            enddo

         enddo

         do i=1,nrows
            write(6,*)(c(i,j),j=1,ncols)
         enddo
```

A Parallel Program (Contd)

```
      ELSE

c        slaves receive a, and one column of b, then compute dot product
         call MPI_BCAST(a,count,MPI_INTEGER,root,MPI_COMM_WORLD,ierr)

         call MPI_RECV(buf, nrows, MPI_INTEGER, root,
     $        MPI_ANY_TAG, MPI_COMM_WORLD, status, ierr)

         do i=1,nrows
            ans(i) = 0
            do j=1,ncols
               ans(i) = ans(i) + a(i,j) * buf(j)
            enddo
         enddo

         call MPI_SEND(ans,nrows,MPI_INTEGER,root,0,MPI_COMM_WORLD,ierr)

      ENDIF

      call MPI_FINALIZE(ierr)

      stop
      end
```

Literature

- Ian T. Foster, *Designing and Building Parallel Programs — Concepts and Tools for Parallel Software Engineering*, Addison Wesley, Reading, Massachusetts, 1995. Online version: `http://www.mcs.anl.gov/dbpp`
- Michael J. Quinn, *Parallel Computing — Theory and Practice*, 2nd edition, McGraw-Hill, New York, 1994.
- Kai Hwang, *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, New York, 1993.