Introduction to Parallel Programming

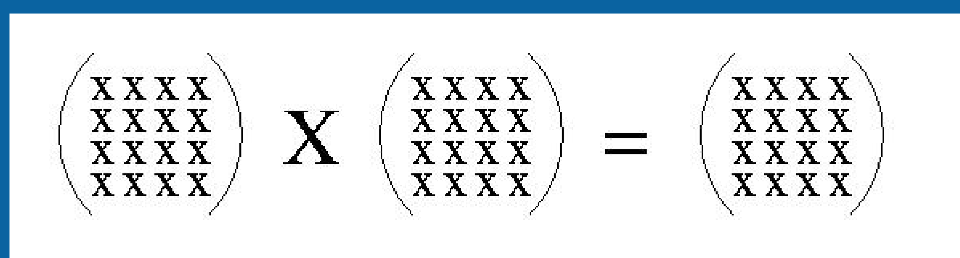
- Language notation: message passing
- 5 parallel algorithms of increasing complexity:
  - ⋆ Matrix multiplication
  - ⋆ Successive overrelaxation
  - ⋆ All-pairs shortest paths
  - ⋆ Linear equations
  - ⋆ Search problem

Message Passing

- SEND(destination, message)
  - ⋆ blocking: wait until message has arrived
  - ⋆ nonblocking: continue immediately
- RECEIVE(source, message)
- RECEIVE-FROM-ANY(message)
  - ⋆ blocking: wait until message is available
  - ⋆ nonblocking: test if message is available

Parallel Matrix Multiplication

- Given two $N \times N$ matrices A and B
- Compute $C = A \times B$
- $C_{ij} = A_{i1}B_{1j} + A_{i2}B_{2j} + .. + A_{iN}B_{Nj}$



Sequential Matrix Multiplication

```
for (i = 1; i <= N; i++)
   for (j = 1; j <= N; j++)
      C[i,j] = 0;
      for (k = 1; k <= N; k++)
         C[i,j] += A[i,k] * B[k,j];
```

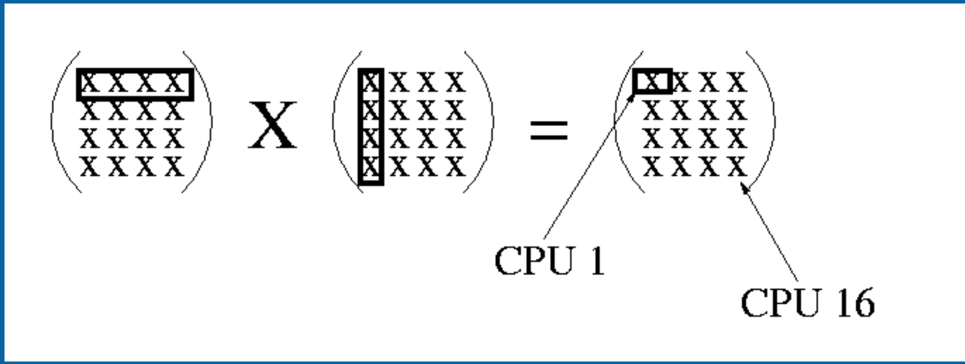
- The order of the operations is overspecified
- Everything can be computed in parallel

Parallel Algorithm 1

Each processor computes 1 element of C

Requires N^2 processors

Need 1 row of A and 1 column of B as input



Parallel Algorithm 1

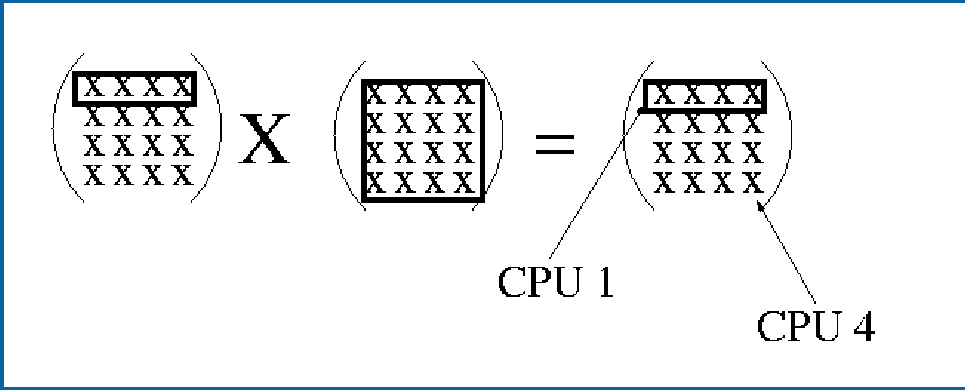
```
Master (processor 0):
   for (i = 1; i <= N; i++)
      for (j = 1; j <= N; j++)
         SEND(p++, A[i,*], B[*,j], i, j);
   for (x = 1; x <= N*N; x++)
       RECEIVE_FROM_ANY(&result, &i, &j);
       C[i,j] = result;
Slaves:
   int Aix[N], Bxj[N], Cij;
   RECEIVE(0, &Aix, &Bxj, &i, &j);
   Cij = 0;
   for (k = 1; k <= N; k++) Cij += Aix[k] * Bxj[k];
   SEND(0, Cij , i, j);
```

Parallel Algorithm 2

Each processor computes 1 row (N elements) of C

Requires N processors

Need entire B matrix and 1 row of A as input



Parallel Algorithm 2

```
Master (processor 0):
   for (i = 1; i <= N; i++)
      SEND(i, A[i,*], B[*,*], i);
   for (x = 1; x <= N; x++)
       RECEIVE_FROM_ANY(&result, &i);
       C[i,*] = result[*];
Slaves:
   int Aix[N], B[N,N], C[N];
   RECEIVE(0, &Aix, &B, &i);
   for (j = 1; j <= N; j++)
      C[j] = 0;
      for (k = 1; k <= N; k++) C[j] += Aix[k] * B[j,k];
   SEND(0, C[*] , i);
```

Problem: need larger granularity

So far, each parallel task needs as much communication as computation

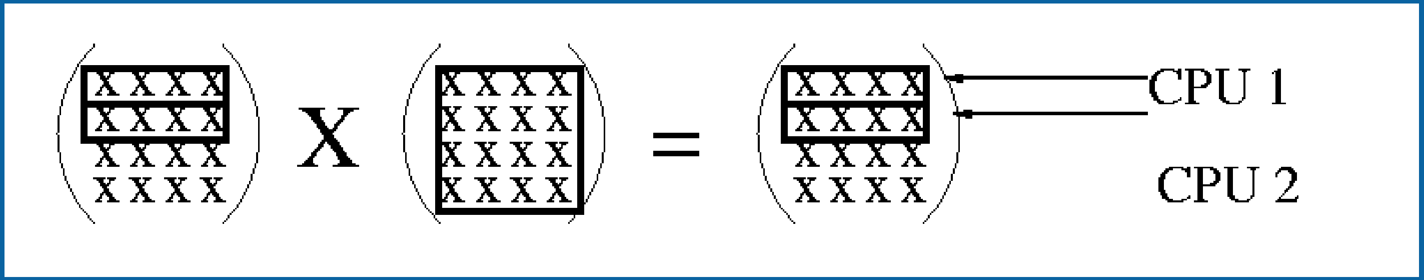
Assumption: $N \gg P$ (i.e. we solve a *l*arge problem)

Assign many rows to each processor

Parallel Algorithm 3

Each processor computes $\frac{N}{P}$ rows of C

Need entire B matrix and $\frac{N}{P}$ rows of A as input



Parallel Algorithm 3

```
Master (processor 0):
   int result[N, N/nprocs];
   int inc = N / nprocs;   /* number of rows per cpu */
   int lb = 1;
   for (i = 1; i <= nprocs; i++)
      SEND(i, A[lb .. lb+inc-1, *], B[*,*], lb, lb+inc-1);
      lb += inc;
   for (x = 1; x <= nprocs; x++)
       RECEIVE_FROM_ANY(&result, &lb);
       for (i = 1; i <= N/nprocs; i++)
          C[lb+i-1, *] = result[i, *];
```

Parallel Algorithm 3 (Cnt'd)

```
Slaves:
   int A[N/nprocs, N], B[N,N], C[N/nprocs, N];
   RECEIVE(0, &A, &B, &lb, &ub);
   for (i = lb; i <= ub; i++)
      for (j = 1; j <= N; j++)
         C[i,j] = 0;
         for (k = 1; k <= N; k++)
            C[i,j] += A[i,k] * B[k,j];
   SEND(0, C[*,*] , lb);
```

Comparison

alg.	parallelism (#jobs)	communication per job	comput. per job	ratio comp/comm
1.	N^2	$N + N + 1$	N	$O(1)$
2.	N	$N + N^2 + N$	N^2	$O(1)$
3.	P	$\frac{N^2}{P} + N^2 + \frac{N^2}{P}$	$\frac{N^3}{P}$	$O(\frac{N}{P})$

If $N \gg P$, algorithm 3 will have low communication overhead

Its *grain size* is high

Discussion

- Matrix multiplication is trivial to parallelize
- Getting good performance is a problem
- Need right grain size
- Need large input problem

Successive Overrelaxation (SOR)

Iterative method for solving Laplace equations

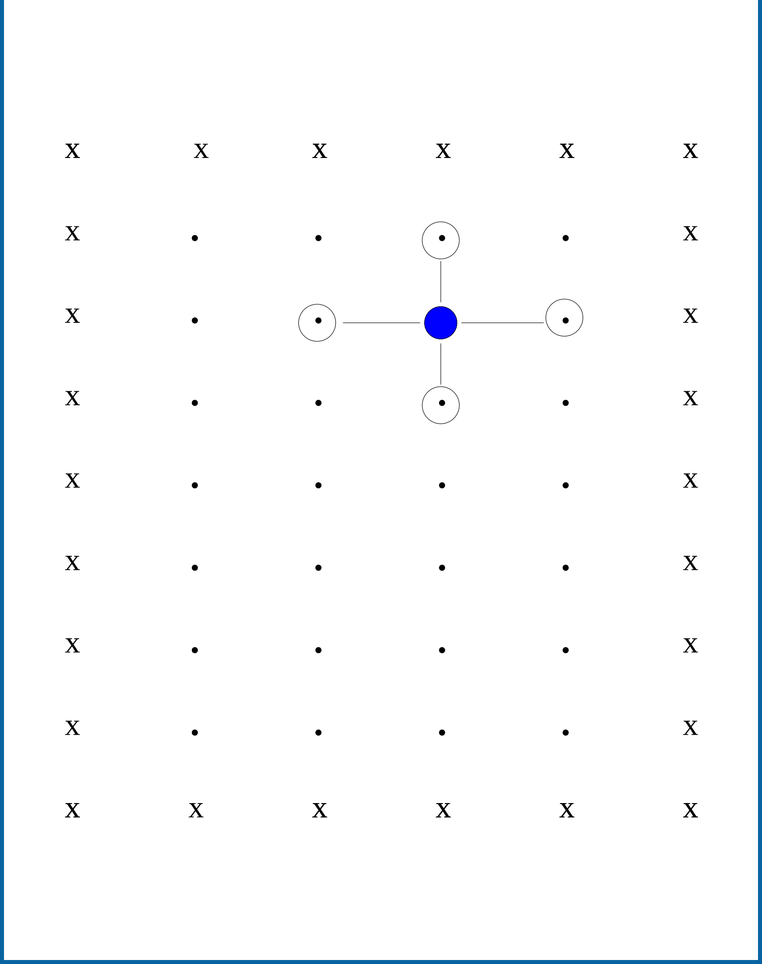
Repeatedly updates elements of a grid

```
float G[1:N, 1:M], Gnew[1:N, 1:M];
for (step = 0; step < NSTEPS; step++)
   for (i = 2; i < N; i++)  /* update grid */
      for (j = 2; j < M; j++)
         Gnew[i,j] = f(G[i,j], G[i-1,j], G[i+1,j],
                       G[i,j-1], G[i,j+1]);
   G = Gnew;
```

SOR example

x	x	x	x	x	x
x	.	.	.	.	x
x	.	.	.	.	x
x	.	.	.	.	x
x	.	.	.	.	x
x	.	.	.	.	x
x	.	.	.	.	x
x	.	.	.	.	x
x	x	x	x	x	x

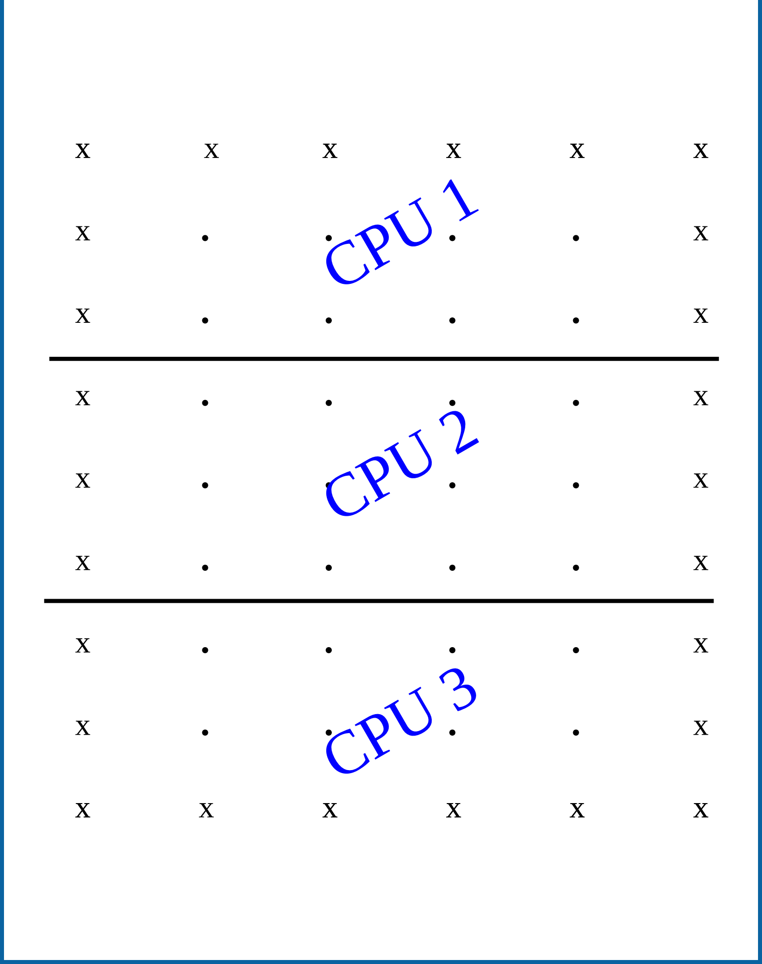
SOR example



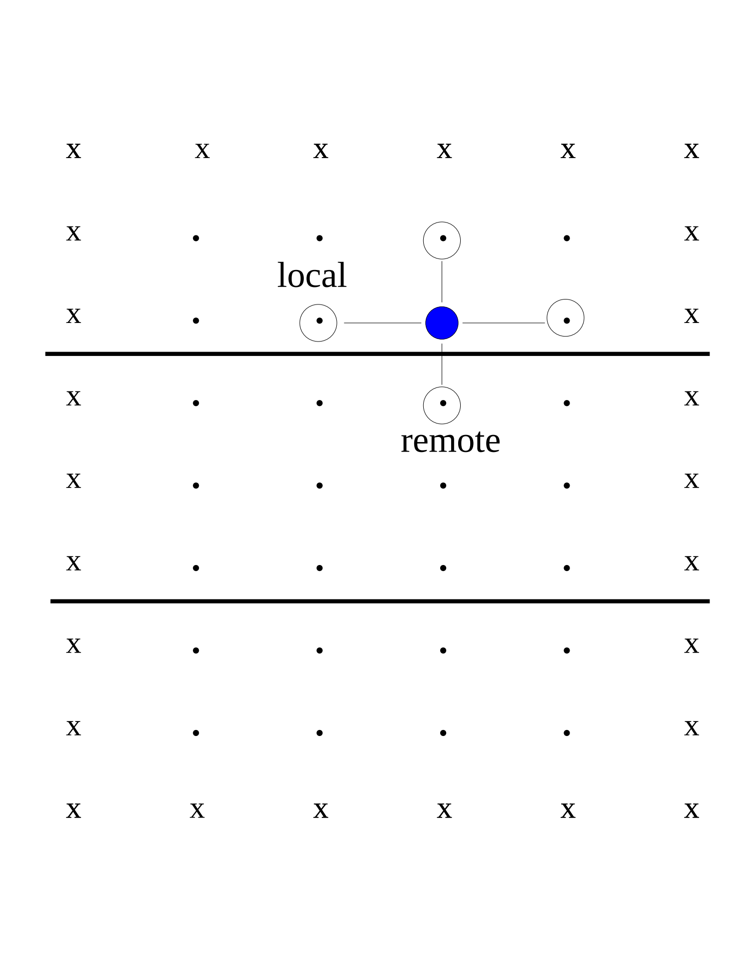
Parallelizing SOR

- Domain decomposition on the grid
- Each processor owns N/P rows
- Need communication between neighbors to exchange elements at processor boundaries

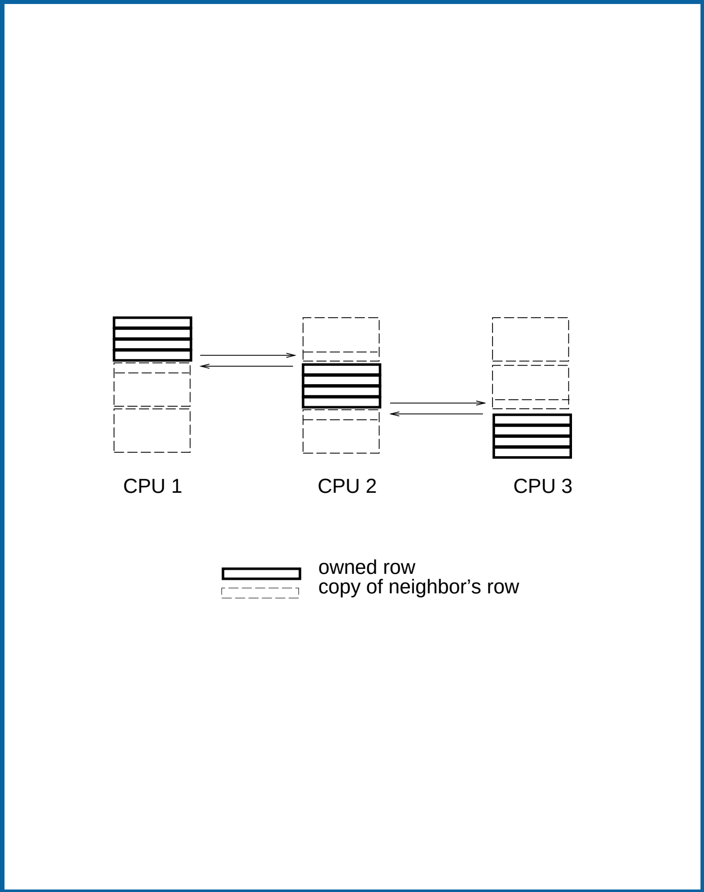
SOR example partitioning



SOR example partitioning



Communication scheme



Each CPU communicates with left & right neighbor (if existing)

Parallel SOR

```
float G[lb-1:ub+1, 1:M], Gnew[lb-1:ub+1, 1:M];
for (step = 0; step < NSTEPS; step++)
   SEND(cpuid-1, G[lb]);         /* send 1st row left */
   SEND(cpuid+1, G[ub]);        /* send last row right */
   RECEIVE(cpuid-1, G[lb-1]);   /* receive from left */
   RECEIVE(cpuid+1, G[ub+1]); /* receive from right */
   for (i = lb; i <= ub; i++)      /* update my rows */
      for (j = 2; j < M; j++)
         Gnew[i,j] = f(G[i,j], G[i-1,j], G[i+1,j],
                              G[i,j-1], G[i,j+1]);
   G = Gnew;
```

Performance of SOR

Communication and computation during each iteration:

- Each processor sends/receives 2 messages with M reals
- Each processor computes $N/P * M$ updates

The algorithm will have good performance if

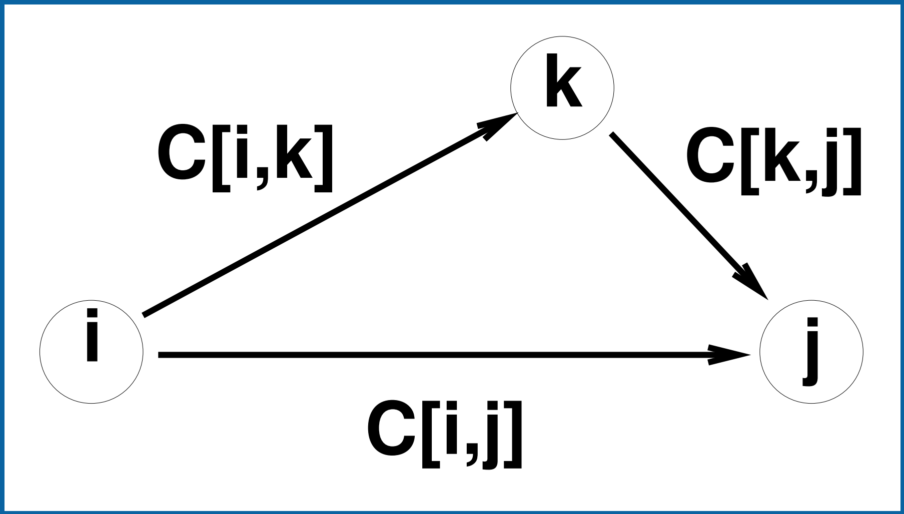
- Problem size is large: $N \gg P$
- Message exchanges can be done in parallel

All-pairs Shorts Paths (ASP)

- Given a graph G with a distance table C:
  - ⋆ C[i,j] = length of direct path from node i to node j
- Compute length of shortest path between any two nodes in G

Floyd's Sequential Algorithm

- Basic step:




```
for (k = 1; k <= N; k++)
    for (i = 1; i <= N; i++)
        for (j = 1; j <= N; j++)
            C[i,j] = MIN(C[i,j], C[i,k] + C[k,j]
```

Parallelizing ASP

- Distribute rows of C among the P processors
- During iteration k, each processor executes

  C[i,j] $=$ MIN(C[i,j], C[i,k] $+$ C[k,j]);
  on its own rows i, so it needs these rows and row k
- Before iteration k, the processor owning row k sends it to all the others

Parallel ASP Algorithm

```
int lb, ub;          /* lower/upper bound for this CPU */
int rowK[N], C[lb:ub, N];        /* pivot row ; matrix */

for (k = 1; k <= N; k++)
   if (k >= lb && k <= ub)                /* do I have it? */
      rowK = C[k,*];
      for (p = 1; p <= nproc; p++) /* broadcast row */
         if (p != myprocid) SEND(p, rowK);
   else
      RECEIVE_FROM_ANY(&rowK);              /* receive row */
   for (i = lb; i <= ub; i++)          /* update my rows */
      for (j = 1; j <= N; j++)
         C[i,j] = MIN(C[i,j], C[i,k] + rowK[j]);
```

Parallel ASP Algorithm

```
int lb, ub;          /* lower/upper bound for this CPU */
int rowK[N], C[lb:ub, N];       /* pivot row ; matrix */

for (k = 1; k <= N; k++)









   for (i = lb; i <= ub; i++)      /* update my rows */
      for (j = 1; j <= N; j++)
         C[i,j] = MIN(C[i,j], C[i,k] + rowK[j]);
```

Parallel ASP Algorithm

```
int lb, ub;          /* lower/upper bound for this CPU */
int rowK[N], C[lb:ub, N];        /* pivot row ; matrix */

for (k = 1; k <= N; k++)
   if (k >= lb && k <= ub)                /* do I have it? */
      rowK = C[k,*];
      for (p = 1; p <= nproc; p++) /* broadcast row */
         if (p != myprocid) SEND(p, rowK);
   else
      RECEIVE_FROM_ANY(&rowK);              /* receive row */
   for (i = lb; i <= ub; i++)          /* update my rows */
      for (j = 1; j <= N; j++)
         C[i,j] = MIN(C[i,j], C[i,k] + rowK[j]);
```

Performance Analysis ASP

Per iteration:

- 1 CPU sends $P - 1$ messages with N integers
- Each CPU does $\frac{N}{P} \times N$ comparisons

Communication/computation ratio is small if $N \gg P$

… but, is the Algorithm Correct?

?

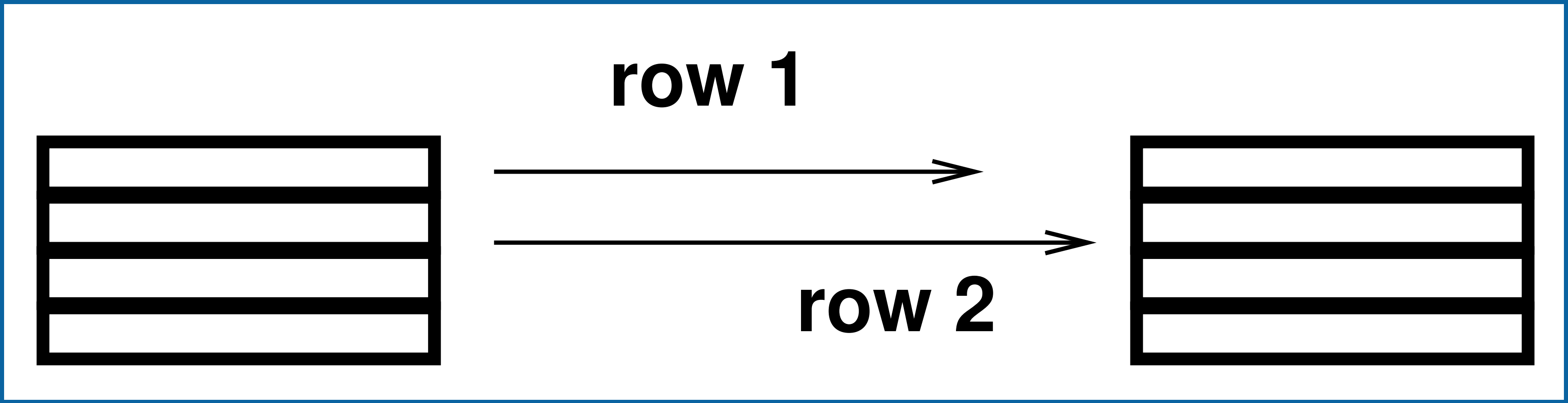
Parallel ASP Algorithm

```
int lb, ub;
int rowK[N], C[lb:ub, N];

for (k = 1; k <= N; k++)
   if (k >= lb && k <= ub)
      rowK = C[k,*];
      for (p = 1; p <= nproc; p++)
         if (p != myprocid) SEND(p, rowK);
   else
      RECEIVE_FROM_ANY(&rowK);
   for (i = lb; i <= ub; i++)
      for (j = 1; j <= N; j++)
         C[i,j] = MIN(C[i,j], C[i,k] + rowK[j]);
```

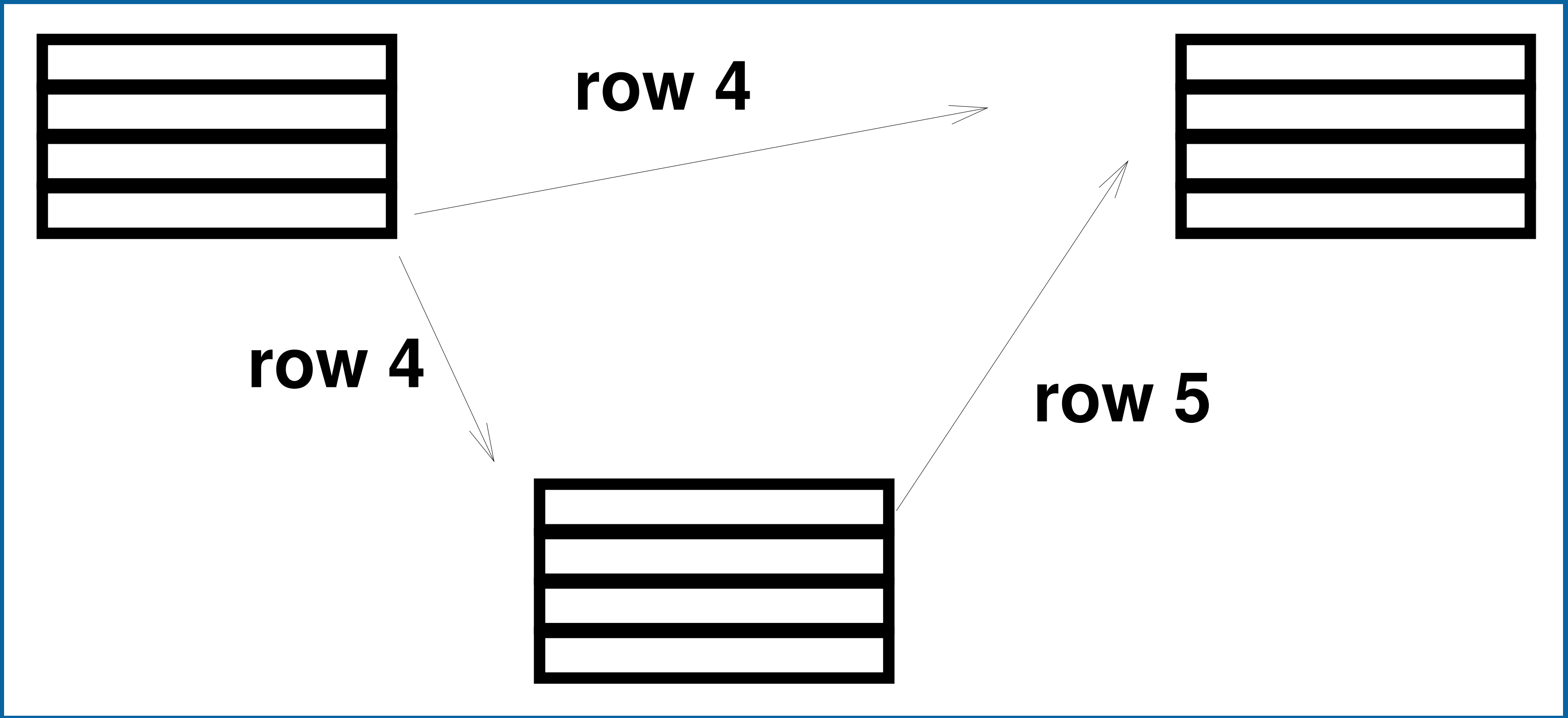
Non-FIFO Message Ordering

Row 2 may be received before row 1



FIFO Ordering

Row 5 may be received before row 4



Correctness

Problems:

- Asynchronous non-FIFO SEND
- Messages from different senders may overtake each other

Correctness

Problems:

- Asynchronous non-FIFO SEND
- Messages from different senders may overtake each other

Solutions:

Correctness

Problems:

- Asynchronous non-FIFO SEND
- Messages from different senders may overtake each other

Solutions:

- Synchronous SEND (less efficient)

Correctness

Problems:

- Asynchronous non-FIFO SEND
- Messages from different senders may overtake each other

Solutions:

- Synchronous SEND (less efficient)
- Barrier at the end of outer loop (extra communication)

Correctness

Problems:

- Asynchronous non-FIFO SEND
- Messages from different senders may overtake each other

Solutions:

- Synchronous SEND (less efficient)
- Barrier at the end of outer loop (extra communication)
- Order incoming messages (requires buffering)

Correctness

Problems:

- Asynchronous non-FIFO SEND
- Messages from different senders may overtake each other

Solutions:

- Synchronous SEND (less efficient)
- Barrier at the end of outer loop (extra communication)
- Order incoming messages (requires buffering)
- RECEIVE(cpu, msg) (more complicated)

Linear equations

- Linear equations:

$$a_{1,1}x_1 + a_{1,2}x_2 + ...a_{1,n}x_n = b_1$$

$$...$$

$$a_{n,1}x_1 + a_{n,2}x_2 + ...a_{n,n}x_n = b_n$$

- Matrix notation: $Ax = b$
- Problem: compute x, given A and b
- Linear equations have many important applications

  Practical applications need huge sets of equations

Solving a linear equation

- Two phases:

  Upper-triangularization $\longrightarrow Ux = y$
  Back-substitution $\longrightarrow x$

- Most computation time is in upper-triangularization

- Upper-triangular matrix:

  U[i,i] $= 1$
  U[i,j] $= 0$ if $i > j$

Sequential Gaussian elimination

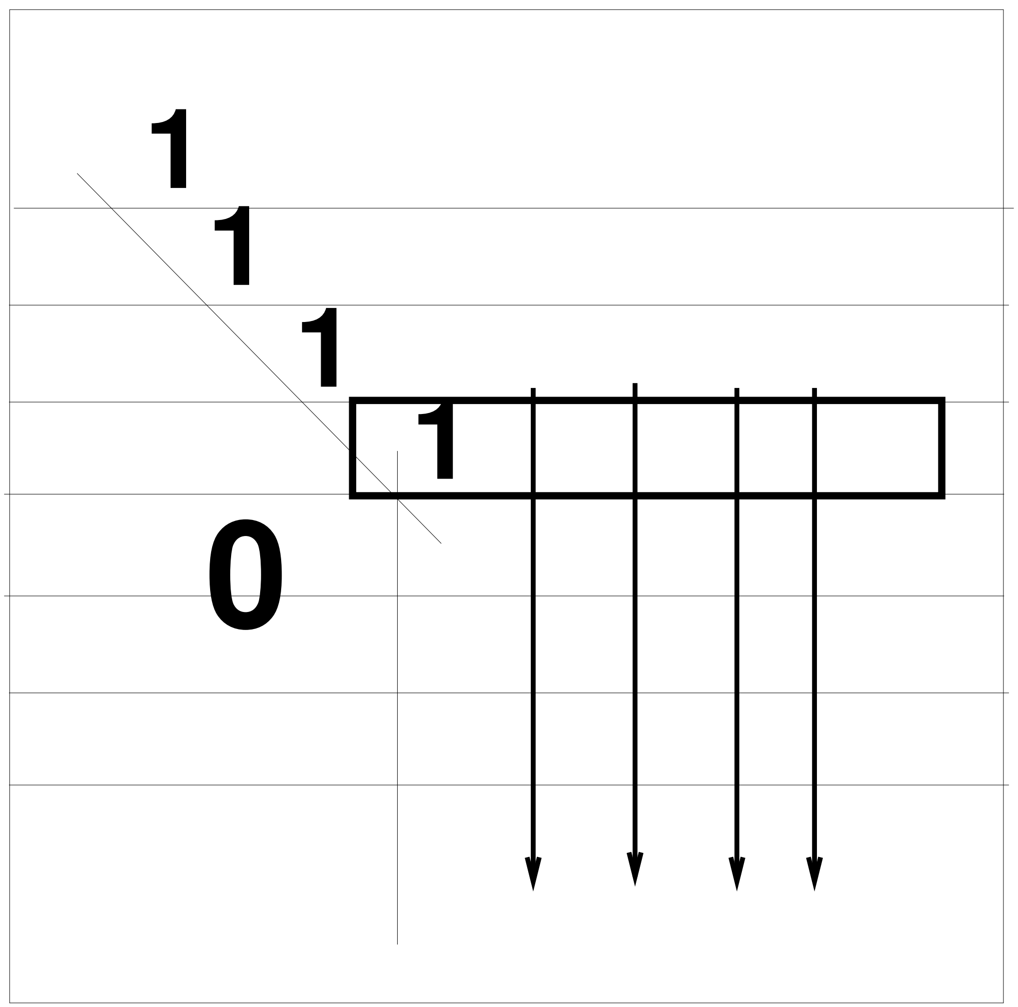
```
for (k = 1; k <= N; k++)
    for (j = k+1; j <= N; j++)
        A[k,j] = A[k,j] / A[k,k]
    y[k] = b[k] / A[k,k]
    A[k,k] = 1
    for (i = k+1; i <= N; i++)
        for (j = k+1; j <= N; j++)
            A[i,j] = A[i,j] - A[i,k] * A[k,j]
        b[i] = b[i] - A[i,k] * y[k]
        A[i,k] = 0
```

- Converts $Ax = b$ into $Ux = y$
- Sequential algorithm uses $\frac{2}{3}N^3$ operations

Parallelizing Gaussian elimination

- Row-wise partitioning scheme
  - ⋆ Each CPU gets one row (*striping*)
  - ⋆ Execute one (outer-loop) iteration at a time
- Communication requirement:
  - ⋆ During iteration k, CPUs $P_{k+1} \ldots P_{n-1}$ need part of row k
  - ⋆ This row is stored on CPU P_k
  - ⋆ $\longrightarrow$ need partial broadcast (multicast)

Communication



multicast

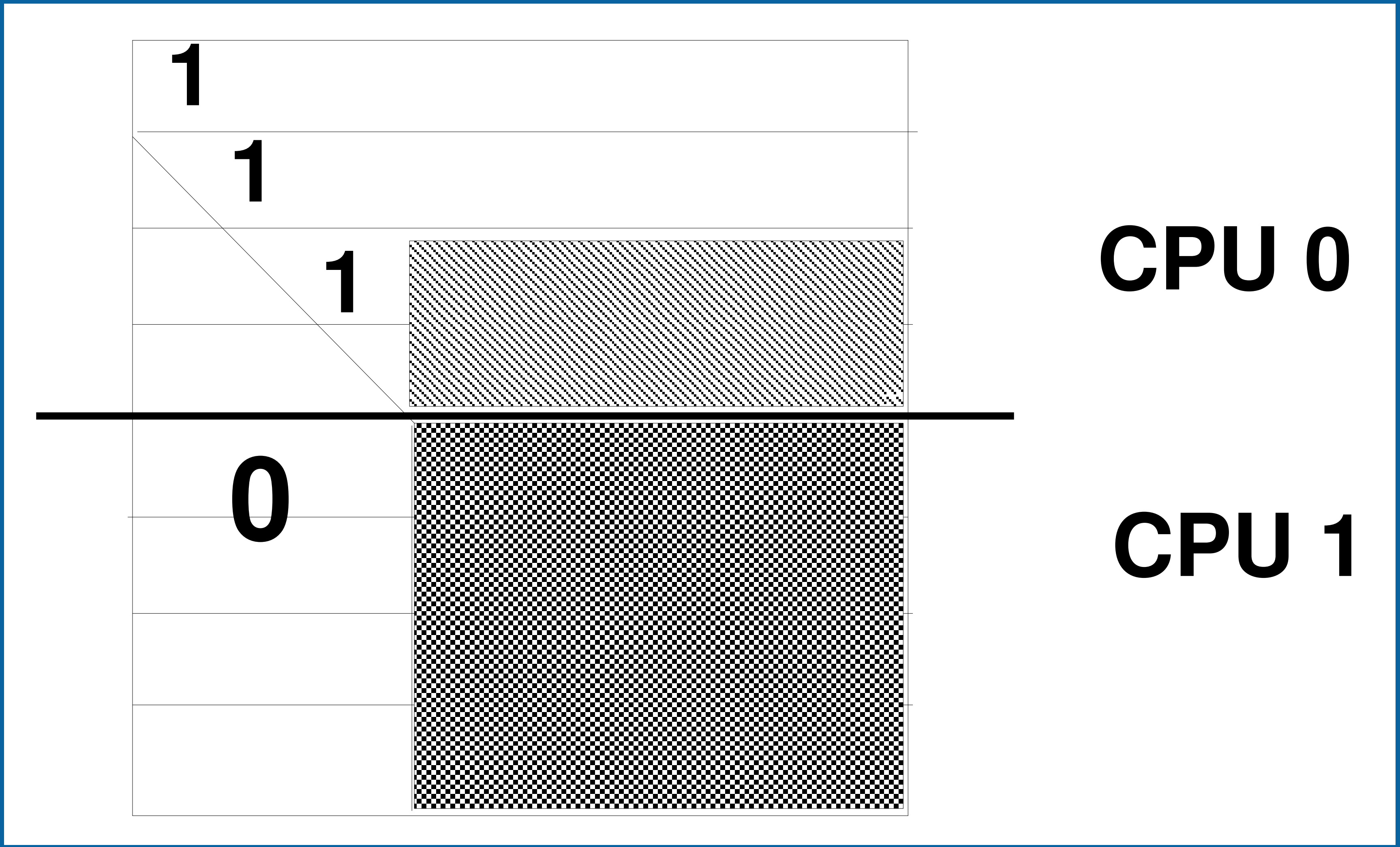
Performance problems

- Communication overhead (multicast)
- Load imbalance

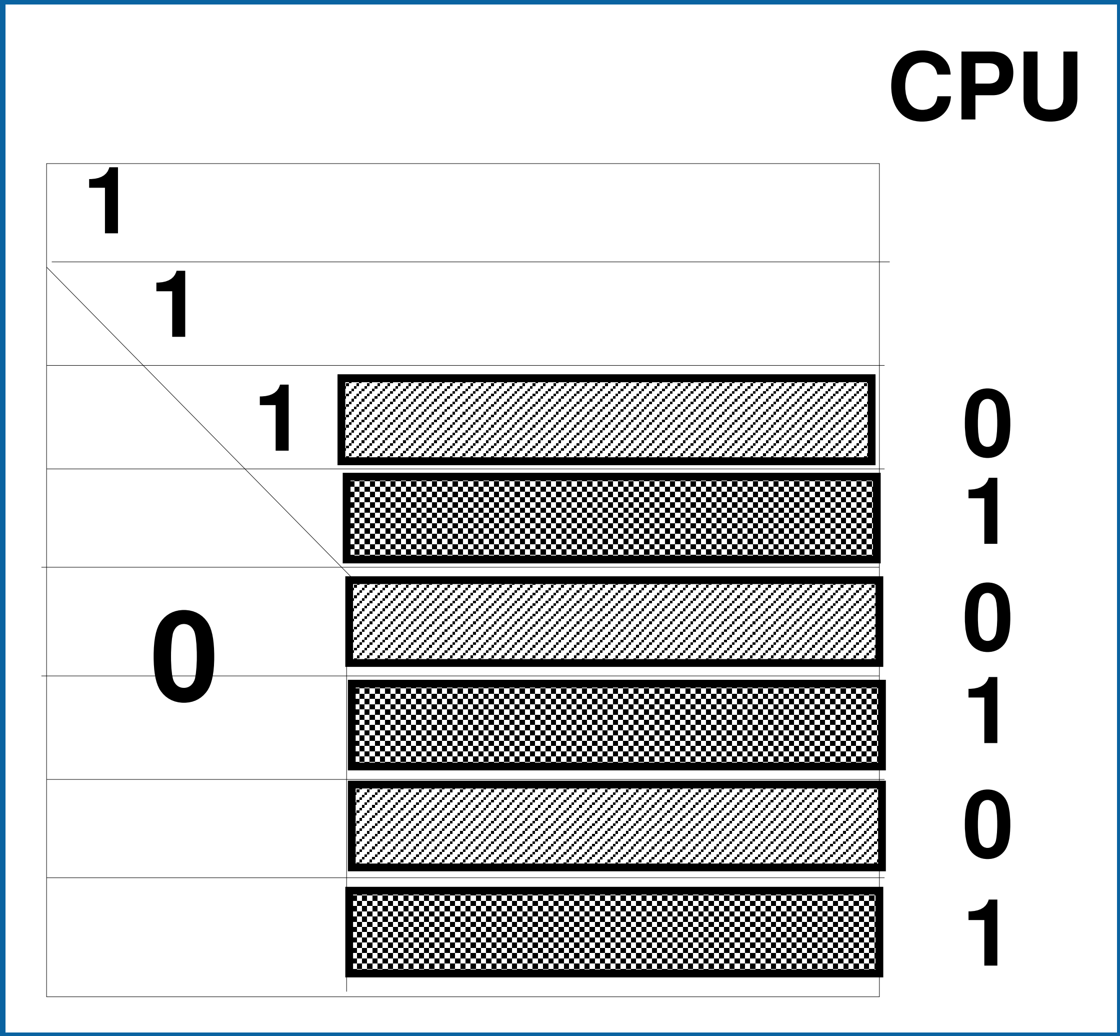
  CPUs $P_0 \ldots P_k$ are idle during iteration k
- In general, number of CPUS is less than n

  Choice between block-striped and cyclic-striped distribution
- Block-striped distribution has high load-imbalance
- Cyclic-striped distribution has less load-imbalance

Block-striped distribution



Cyclic-striped distribution



A Search Problem

Given an array A[1..N] and an item x, check if x is present in A

```
int present = false;
for (i = 1; !present && i <= N; i++)
    if (A[i] == x) present = true;
```

Parallel Search on 2 CPUs

```
int lb, ub;
int A[lb:ub];

for (i = lb; i <= ub; i++)
  if (A[i] == x)
     print("Found item");
     SEND(1-cpuid);  /* send other CPU empty message*/
     exit();
  /* check message from other CPU: */
  if (NONBLOCKING_RECEIVE(1-cpuid)) exit()
```

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

1. if x not present $\Rightarrow$ factor 2

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

1. if x not present $\Rightarrow$ factor 2
2. if x present in A[1 .. 50] $\Rightarrow$ factor 1

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

1. if x not present $\Rightarrow$ factor 2
2. if x present in A[1 .. 50] $\Rightarrow$ factor 1
3. if A[51] $=$ x $\Rightarrow$ factor 51

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

1. if x not present $\Rightarrow$ factor 2
2. if x present in A[1 .. 50] $\Rightarrow$ factor 1
3. if $A[51] = x \Rightarrow$ factor 51
4. if $A[75] = x \Rightarrow$ factor 3

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

1. if x not present $\Rightarrow$ factor 2
2. if x present in A[1 .. 50] $\Rightarrow$ factor 1
3. if A[51] = x $\Rightarrow$ factor 51
4. if A[75] = x $\Rightarrow$ factor 3

In case 2 the parallel program does more work than the sequential program $\Rightarrow$ *search overhead*

Performance Analysis

How much faster is the parallel program than the sequential program for N=100 ?

1. if x not present $\Rightarrow$ factor 2
2. if x present in A[1 .. 50] $\Rightarrow$ factor 1
3. if A[51] = x $\Rightarrow$ factor 51
4. if A[75] = x $\Rightarrow$ factor 3

In case 2 the parallel program does more work than the sequential program $\Rightarrow$ *search overhead*

In cases 3 and 4 the parallel program does less work $\Rightarrow$ *negative search overhead*

Discussion

Several kinds of performance overhead

Discussion

Several kinds of performance overhead

- Communication overhead

Discussion

Several kinds of performance overhead

- Communication overhead
- Load imbalance

Discussion

Several kinds of performance overhead

- Communication overhead
- Load imbalance
- Search overhead

Discussion

Several kinds of performance overhead

- Communication overhead
- Load imbalance
- Search overhead

Making algorithms correct is nontrivial

Discussion

Several kinds of performance overhead

- Communication overhead
- Load imbalance
- Search overhead

Making algorithms correct is nontrivial

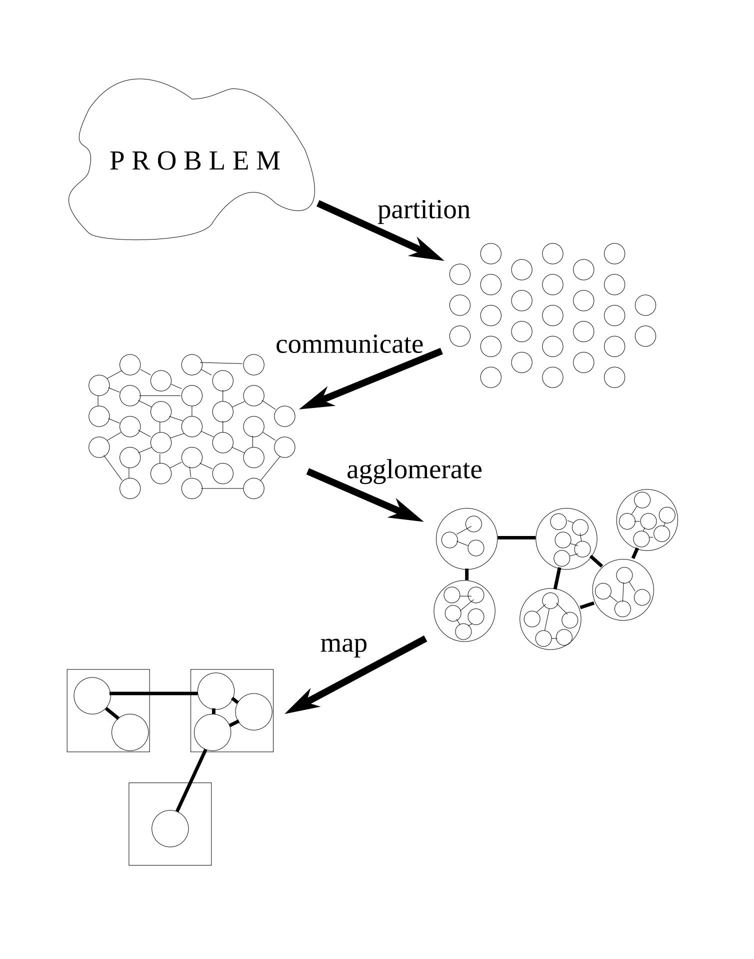
- Message ordering

Designing Parallel Algorithms

Source: Designing and building parallel programs (Ian Foster, 1995)

- Partitioning
- Communication
- Agglomeration
- Mapping

Figure 2.1 from Foster's book



Partitioning

- Domain decomposition
  - ⋆ Partition the data
  - ⋆ Partition computations on data (owner-computes rule)
- Functional decomposition
  - ⋆ Divide computations into subtasks
  - ⋆ E.g. search algorithms

Communication

- Analyze data-dependencies between partitions
- Use communication to transfer data
- Many forms of communication, e.g.
  - ⋆ Local communication with neighbors (SOR)
  - ⋆ Global communication with all processors (ASP)
  - ⋆ Synchronous (blocking) communication
  - ⋆ Asynchronous (nonblocking) communication

Agglomeration

Reduce communication overhead by

- increasing granularity
- improving locality

Mapping

- On which processor to execute each subtask?
- Put concurrent tasks on different CPUs
- Put frequently communicating tasks on same CPU?
- Avoid load imbalances

Summary

Hardware and software models

Example applications

- Matrix multiplication
- Successive overrelaxation
- All-pairs shortest paths
- Linear equations
- Search problem

Designing parallel algorithms