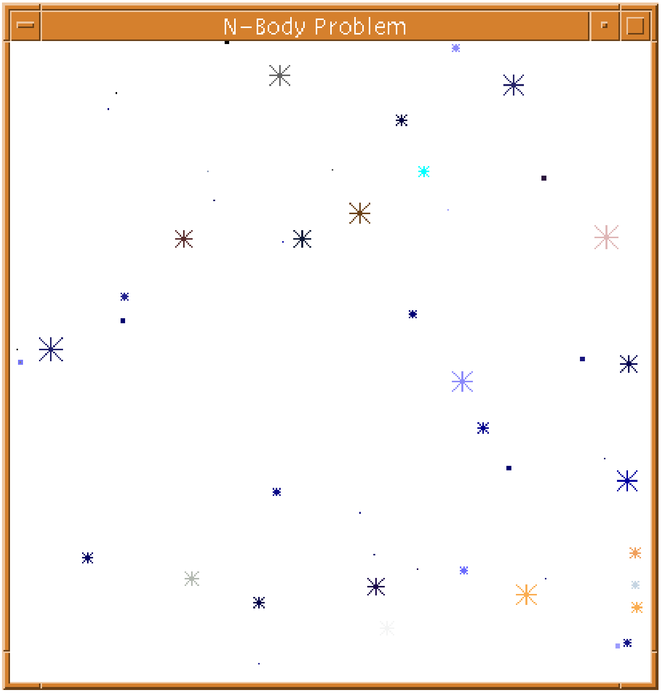


Figure 1.1 Astrophysical *N*-body simulation by Scott Linssen (undergraduate University of North Carolina at Charlotte [UNCC] student).

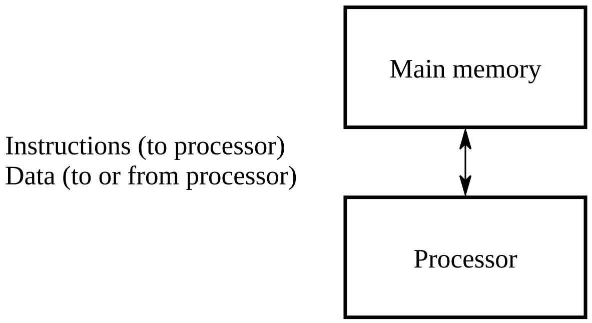


Figure 1.2 Conventional computer having a single processor and memory.

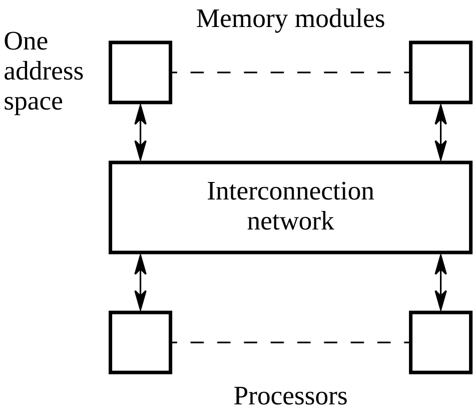


Figure 1.3 Traditional shared memory multiprocessor model.

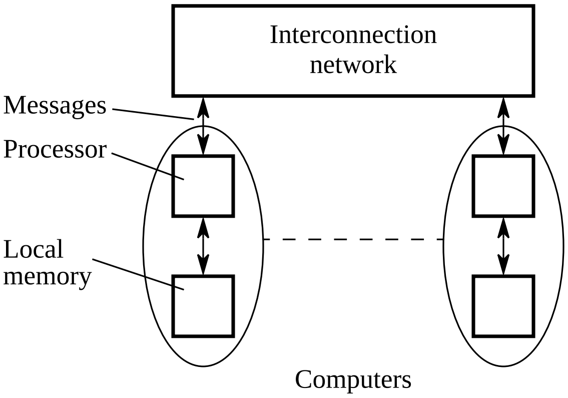


Figure 1.4 Message-passing multiprocessor model (multicomputer).

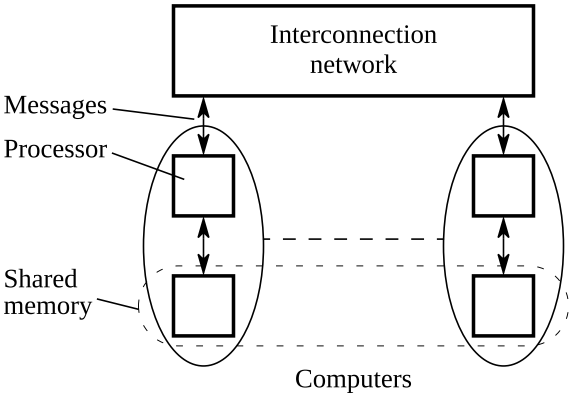


Figure 1.5 Shared memory multiprocessor implementation.

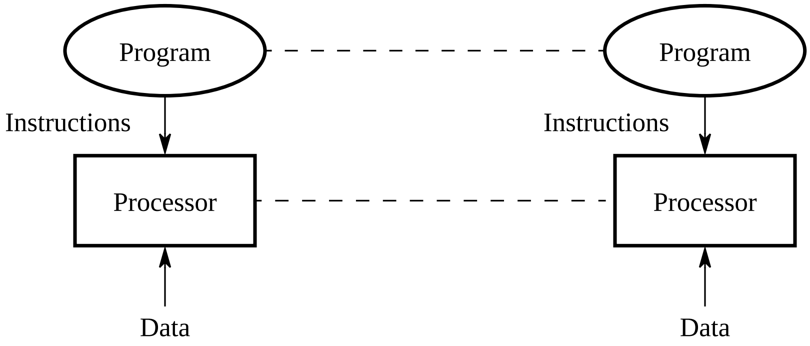


Figure 1.6 MPMD structure.

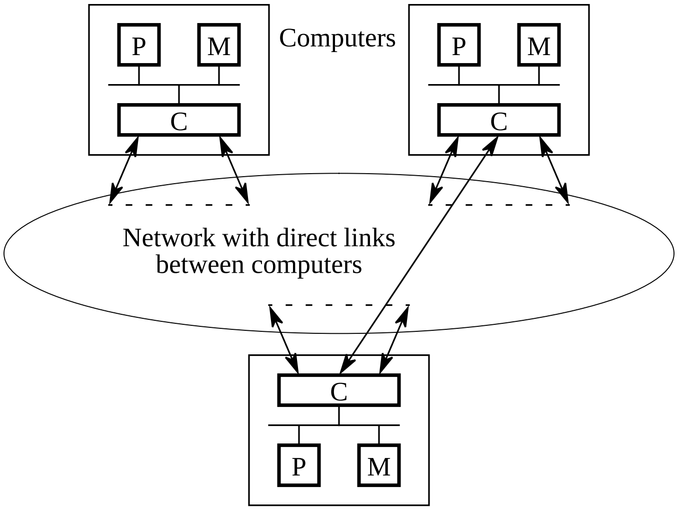


Figure 1.7 Static link multicomputer.

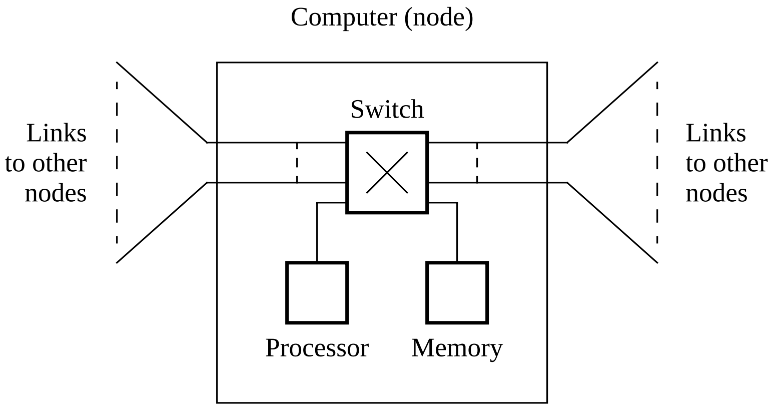


Figure 1.8 Node with a switch for internode message transfers.

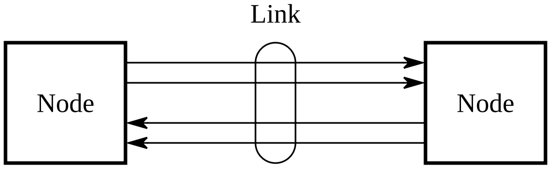


Figure 1.9 A link between two nodes with separate wires in each direction.

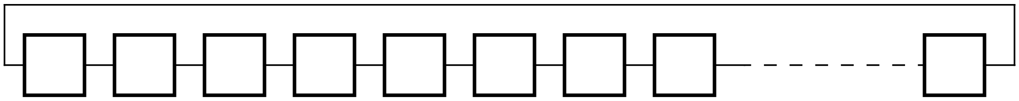


Figure 1.10 Ring.

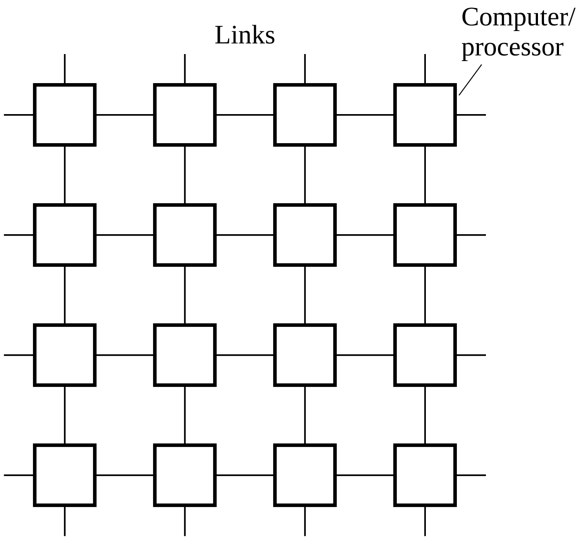


Figure 1.11 Two-dimensional array (mesh).

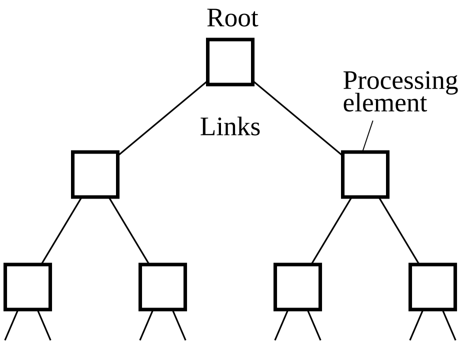


Figure 1.12 Tree structure.

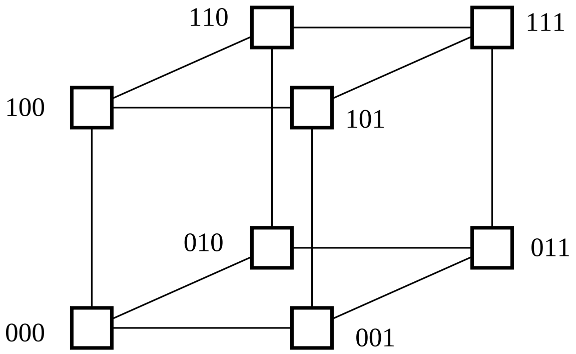


Figure 1.13 Three-dimensional hypercube.

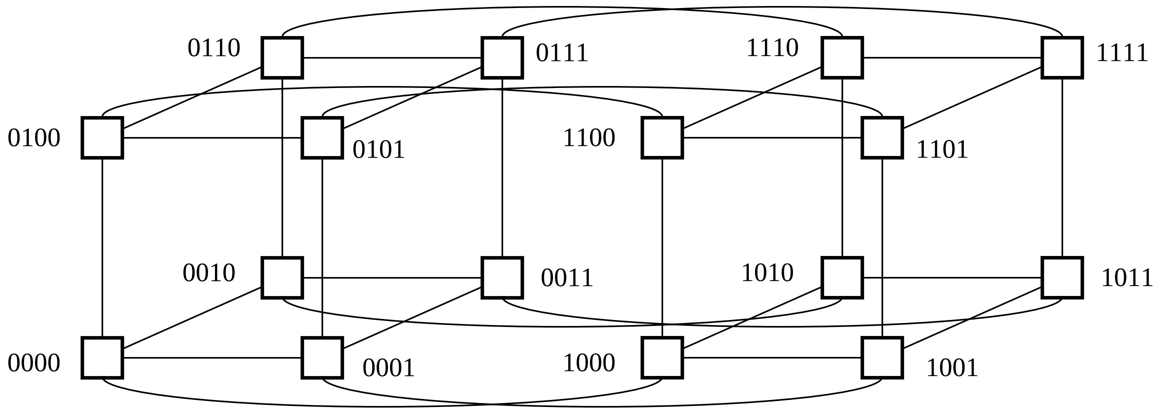


Figure 1.14 Four-dimensional hypercube.

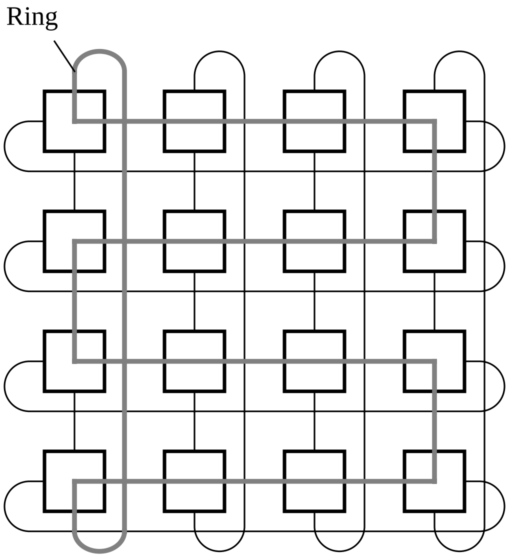


Figure 1.15 Embedding a ring onto a torus.

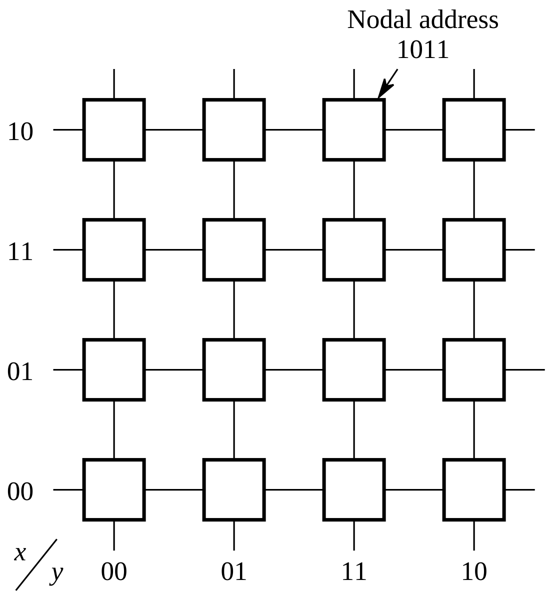


Figure 1.16 Embedding a mesh into a hypercube.

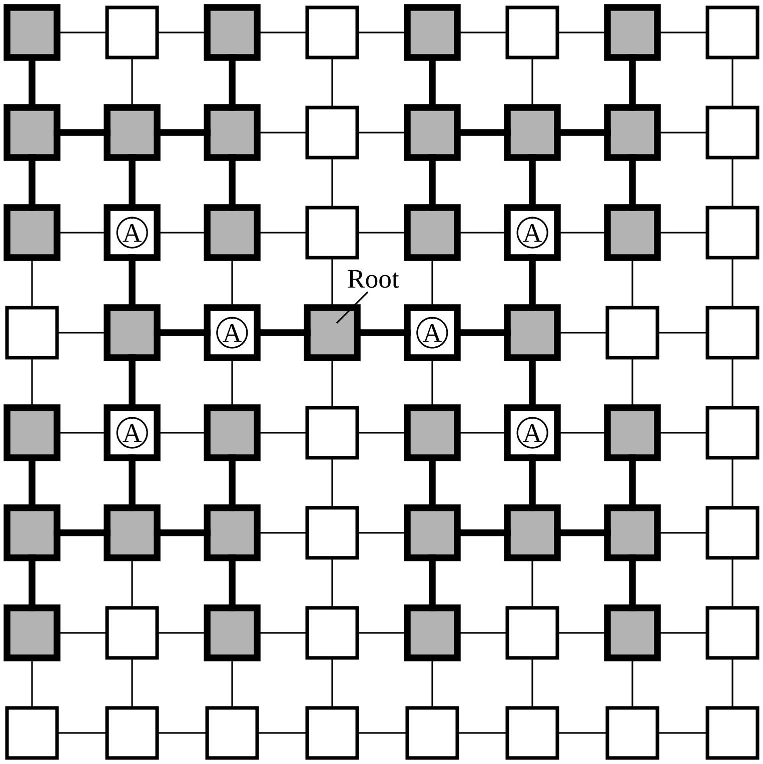


Figure 1.17 Embedding a tree into a mesh.

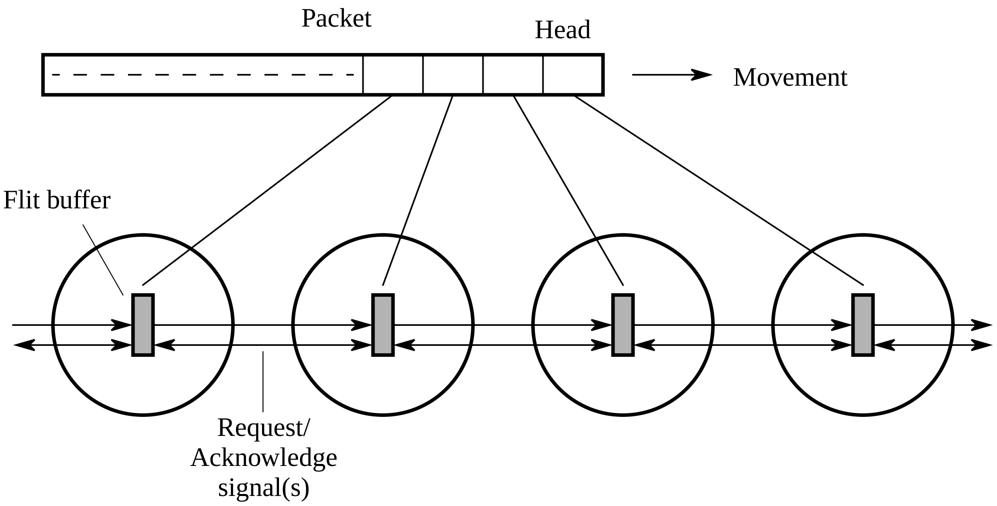


Figure 1.18 Distribution of flits.

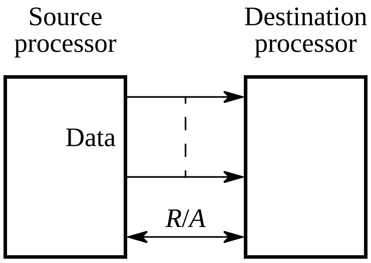


Figure 1.19 A signaling method between processors for wormhole routing (Ni and McKinley, 1993).

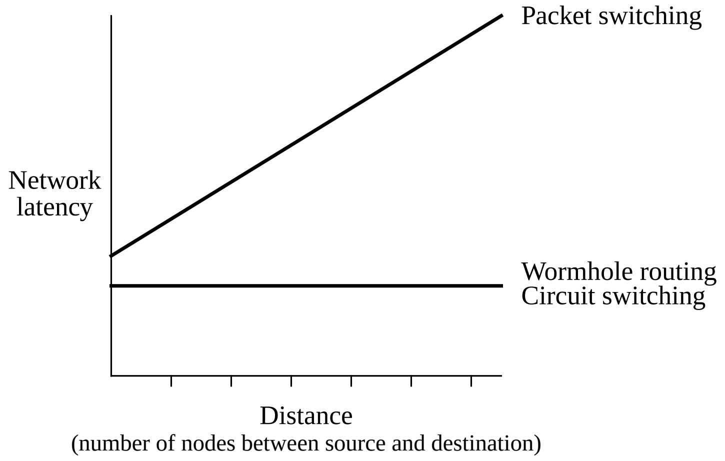


Figure 1.20 Network delay characteristics.

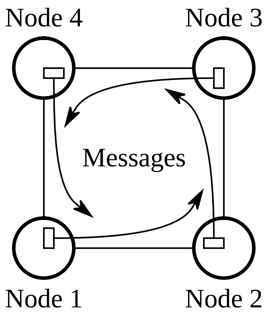


Figure 1.21 Deadlock in store-and-forward networks.

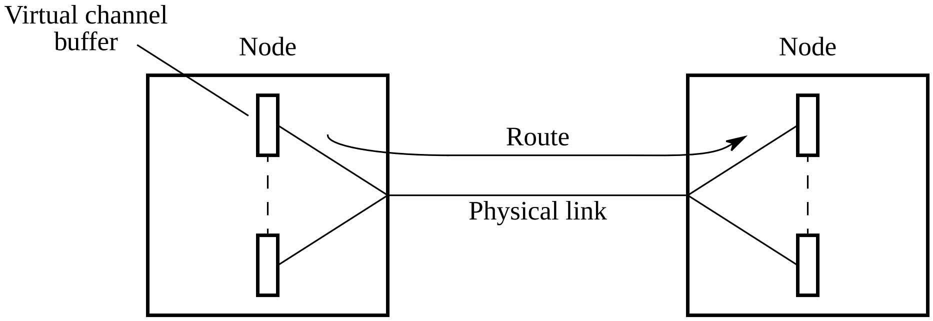


Figure 1.22 Multiple virtual channels mapped onto a single physical channel.

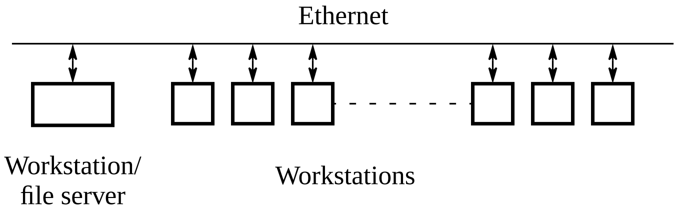


Figure 1.23 Ethernet-type single wire network.

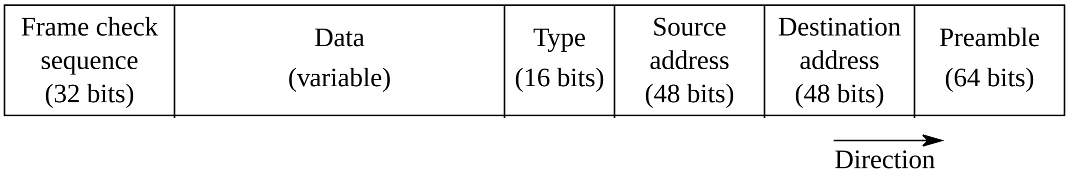


Figure 1.24 Ethernet frame format.

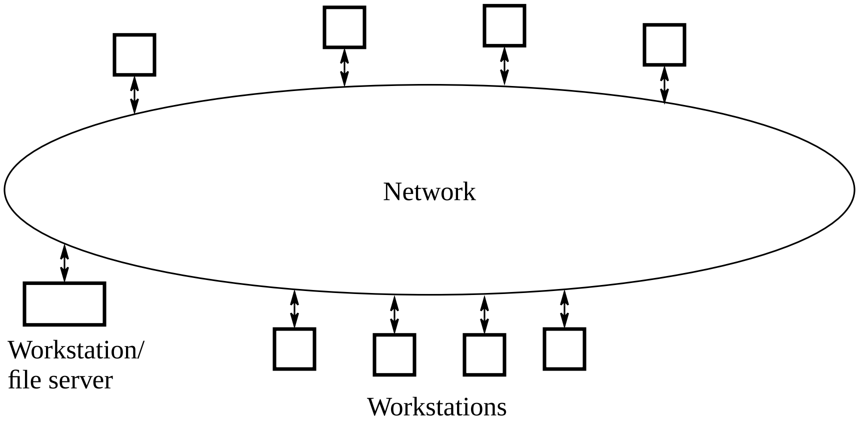


Figure 1.25 Network of workstations connected via a ring.

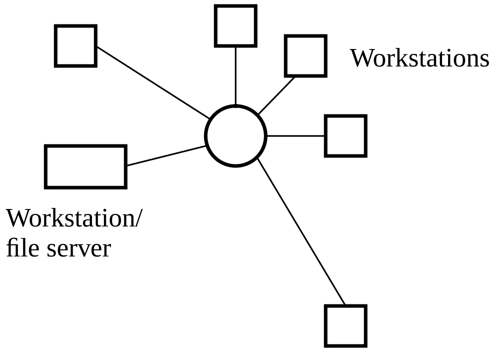
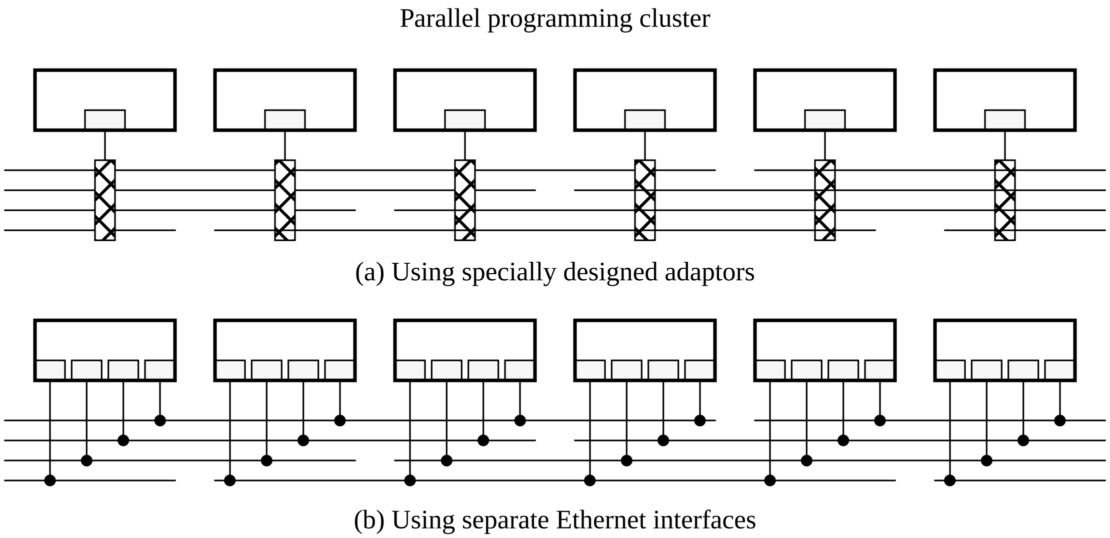


Figure 1.26 Star connected network.



Parallel programming cluster

(a) Using specially designed adaptors

(b) Using separate Ethernet interfaces

Figure 1.27 Overlapping connectivity Ethernets.

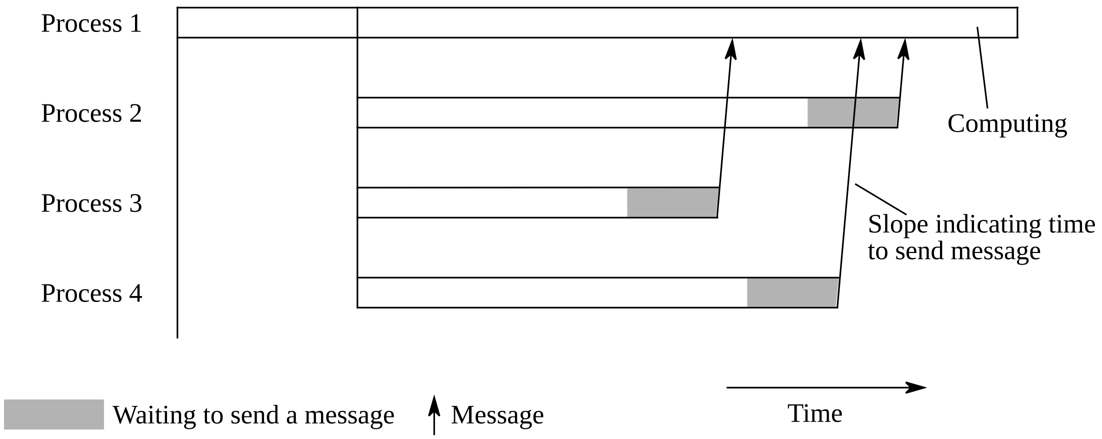

Process 1
Process 2
Process 3
Process 4
Computing
Slope indicating time to send message
Waiting to send a message
Message
Time

Figure 1.28 Space-time diagram of a message-passing program.

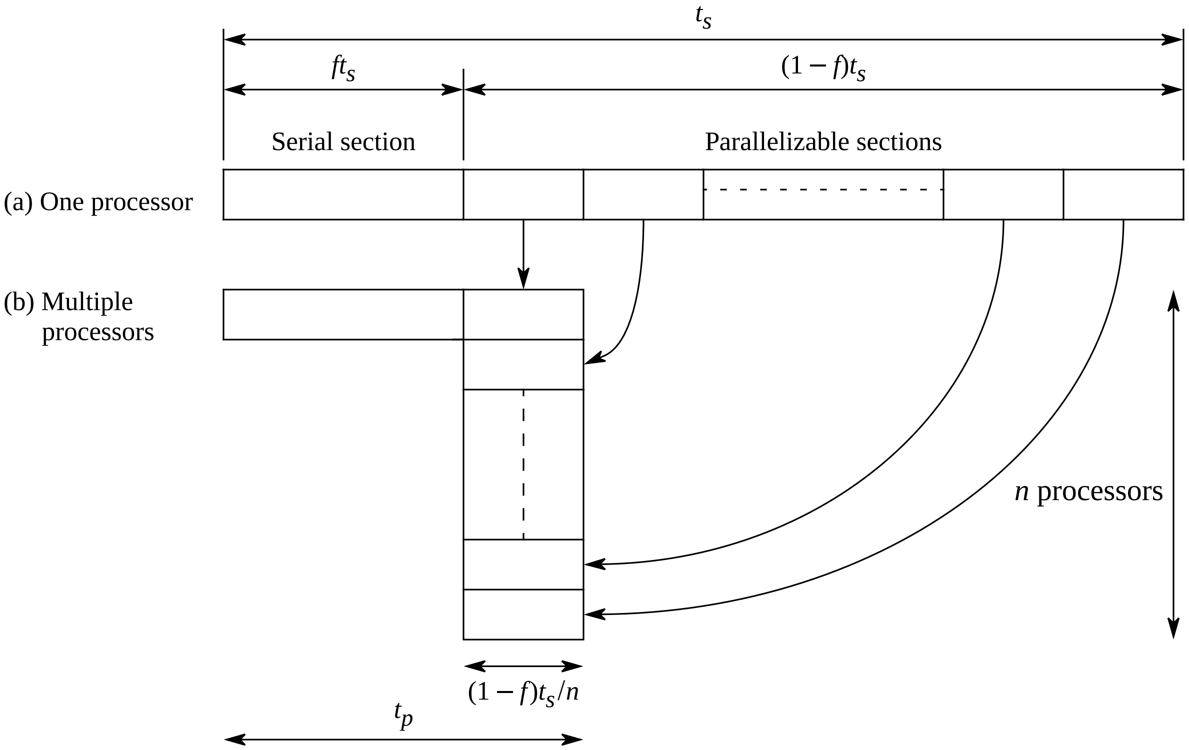


Figure 1.29 Parallelizing sequential problem — Amdahl's law.

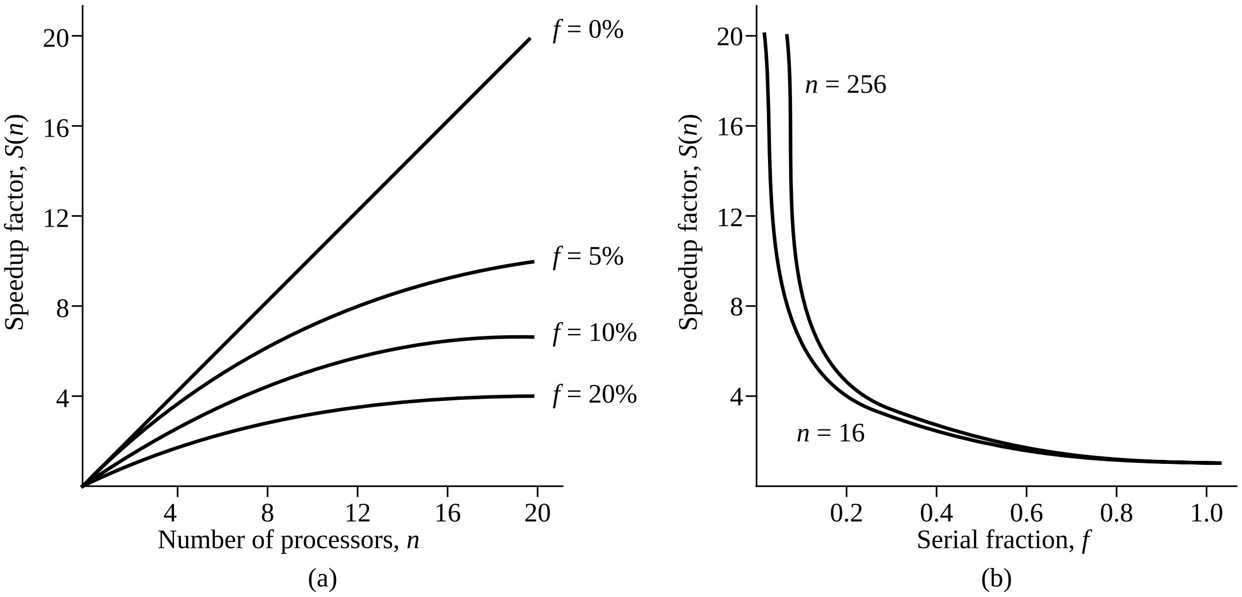


Figure 1.30 (a) Speedup against number of processors. (b) Speedup against serial fraction, *f*.

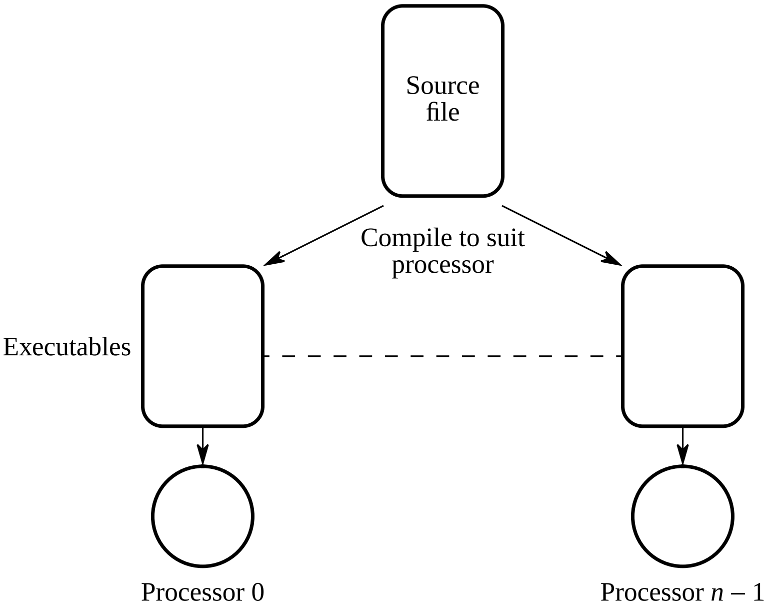


Figure 2.1 Single program, multiple data operation.

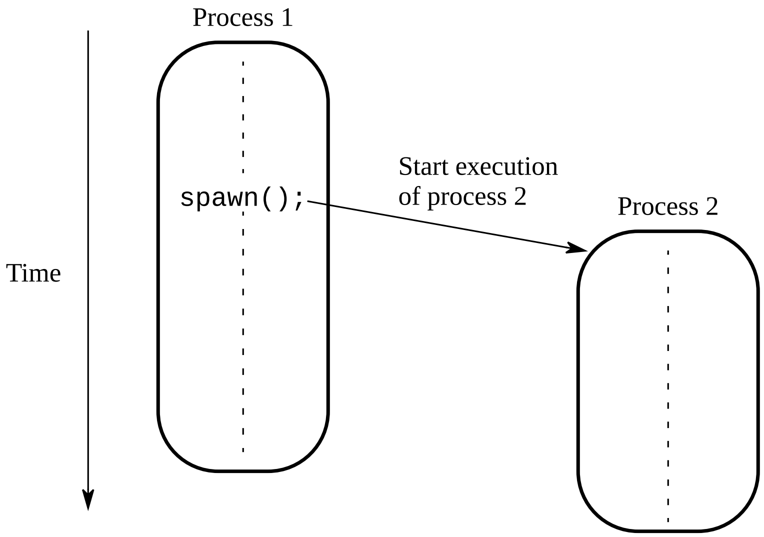


Figure 2.2 Spawning a process.

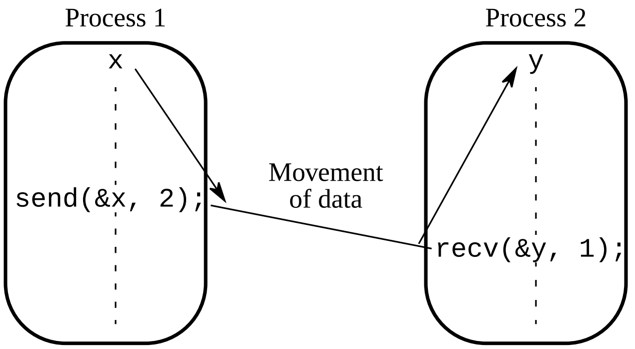
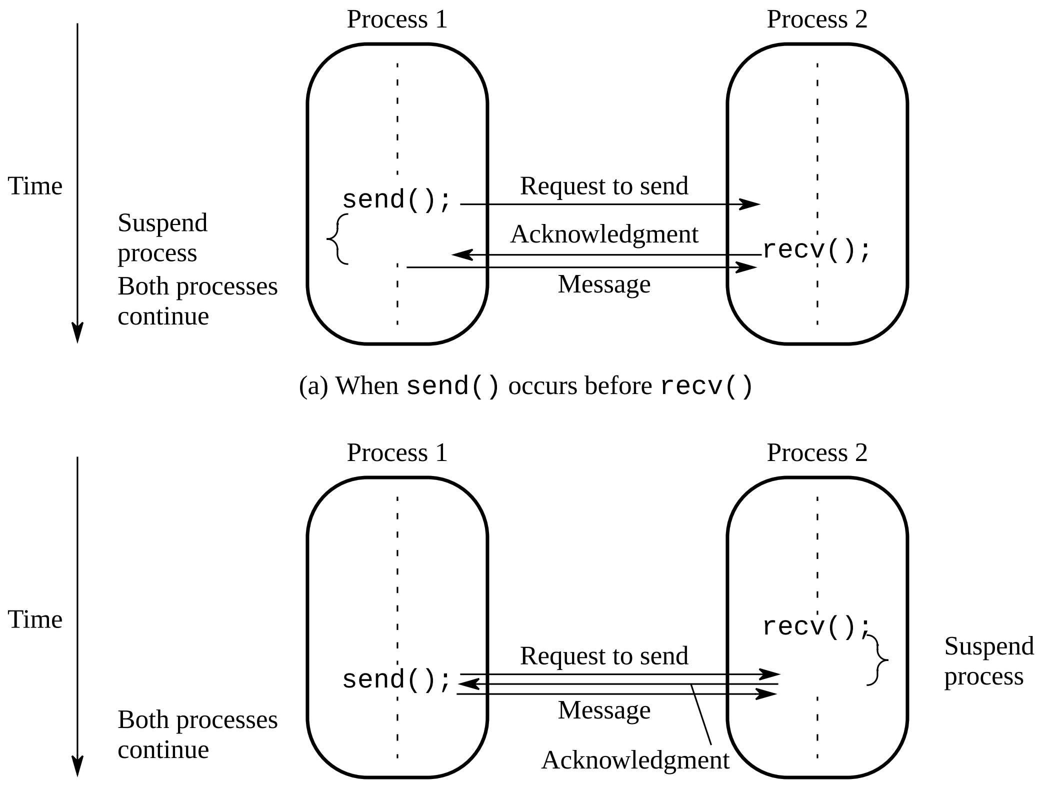


Figure 2.3 Passing a message between processes using `send()` and `recv()` library calls.



(a) When `send()` occurs before `recv()`

(b) When `recv()` occurs before `send()`

Figure 2.4 Synchronous `send()` and `recv()` library calls using a three-way protocol.

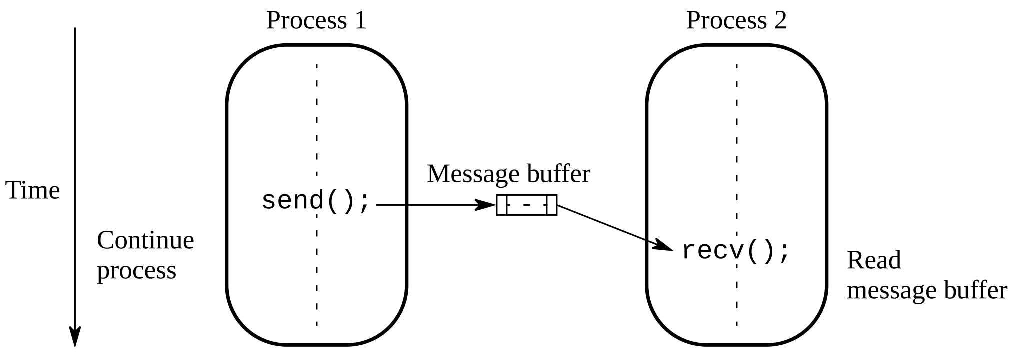


Figure 2.5 Using a message buffer.

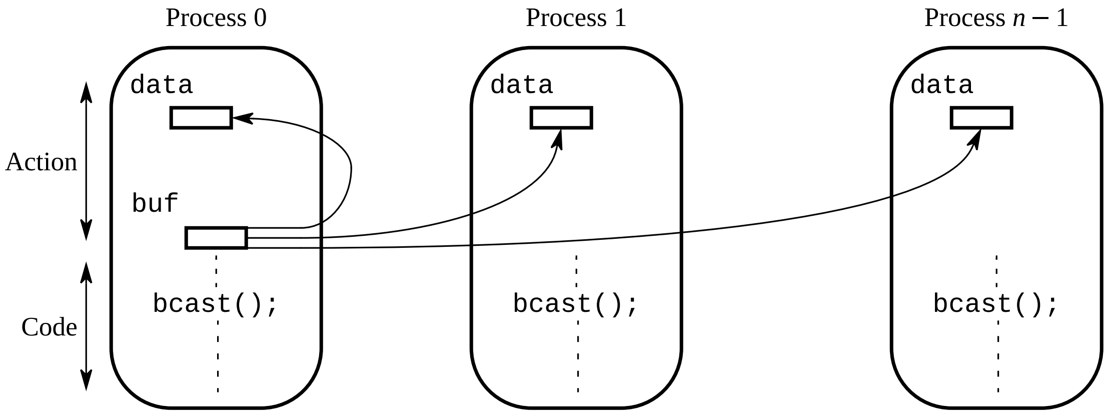


Figure 2.6 Broadcast operation.

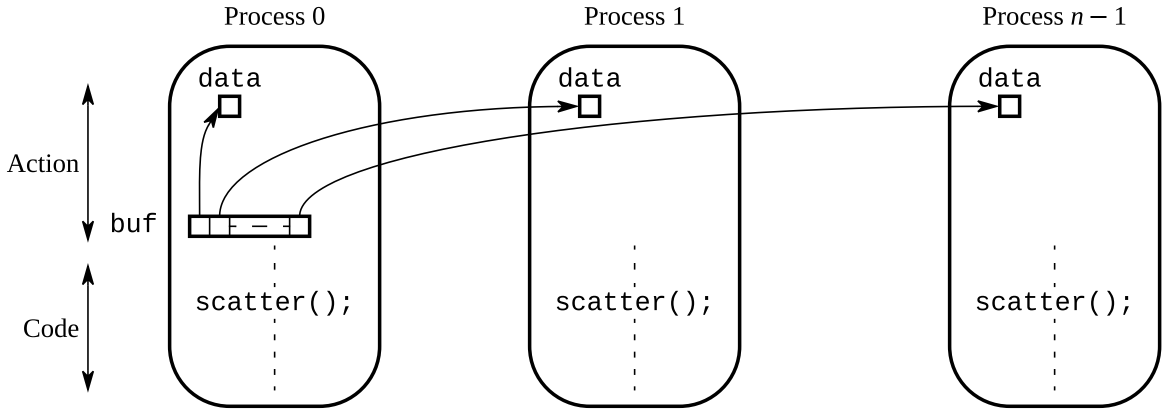


Figure 2.7 Scatter operation.

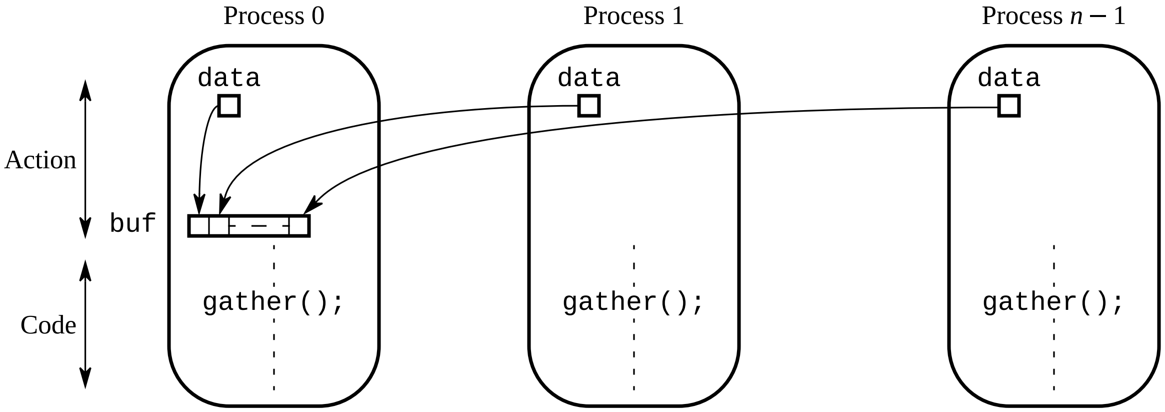


Figure 2.8 Gather operation.

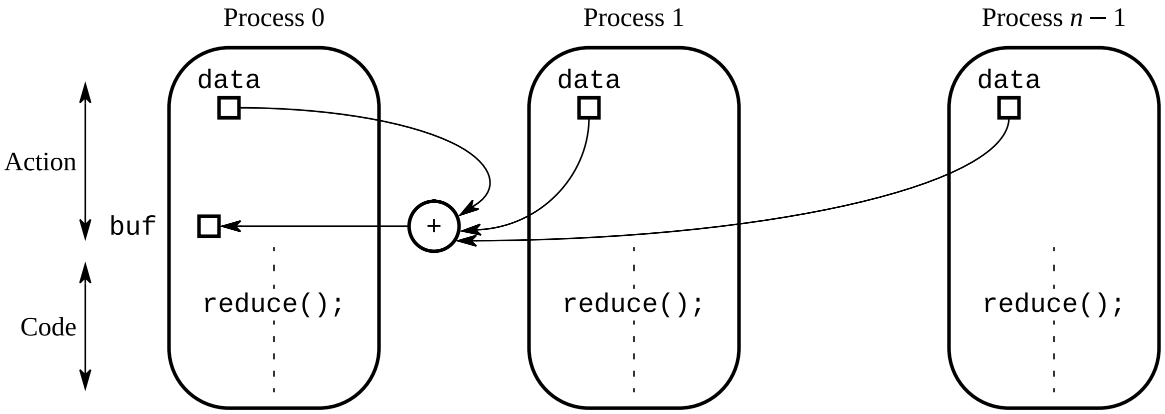


Figure 2.9 Reduce operation (addition).

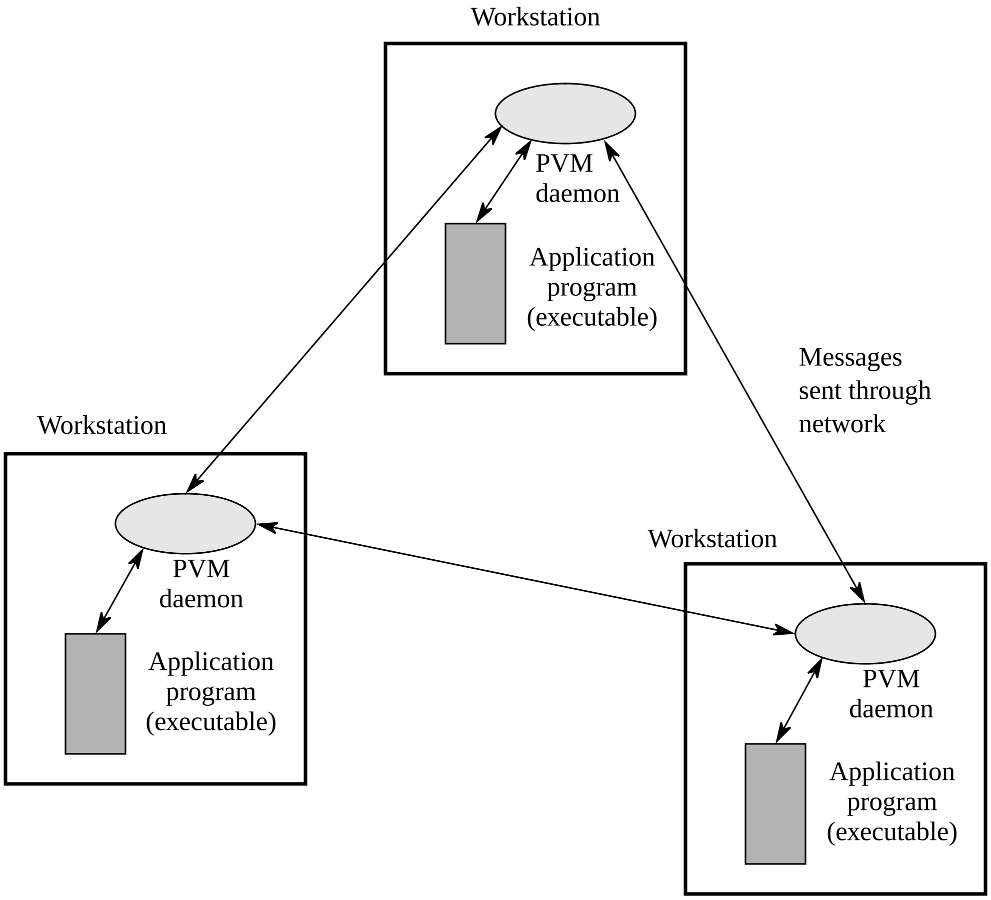

Workstation
PVM daemon
Application program (executable)
Messages sent through network
Workstation
PVM daemon
Application program (executable)
Workstation
PVM daemon
Application program (executable)

Figure 2.10 Message passing between workstations using PVM.

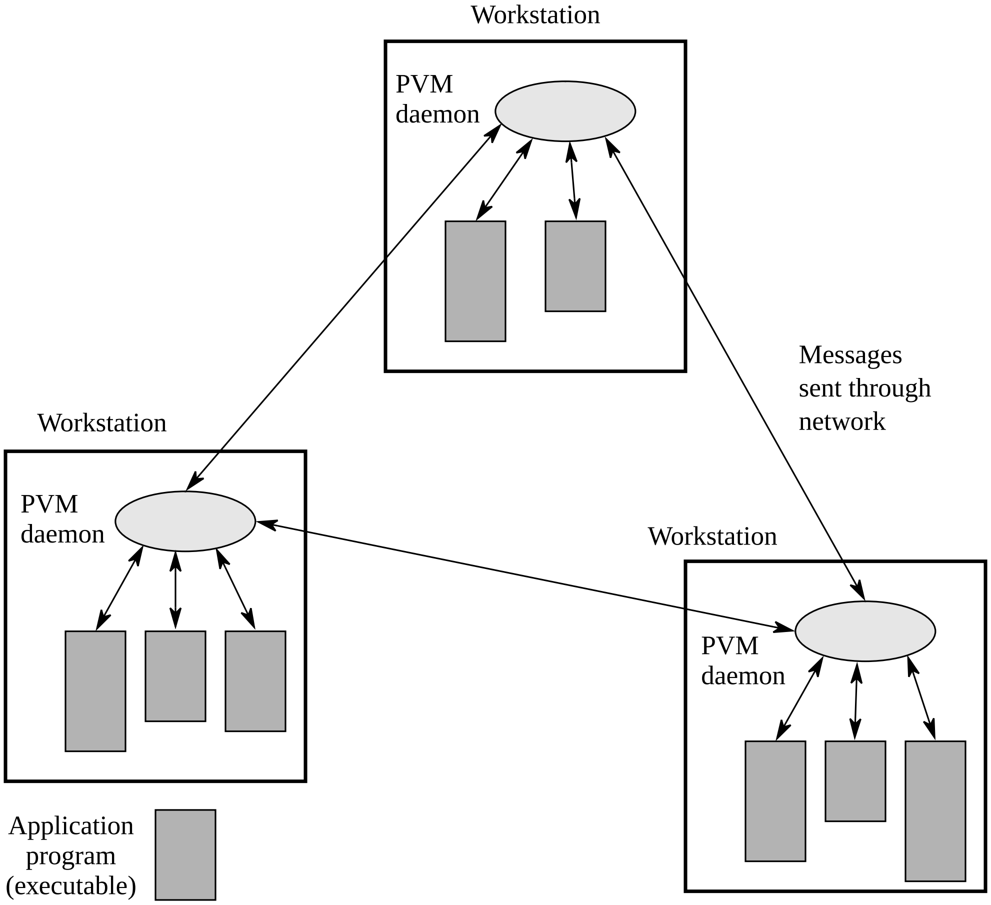


Figure 2.11 Multiple processes allocated to each processor (workstation).

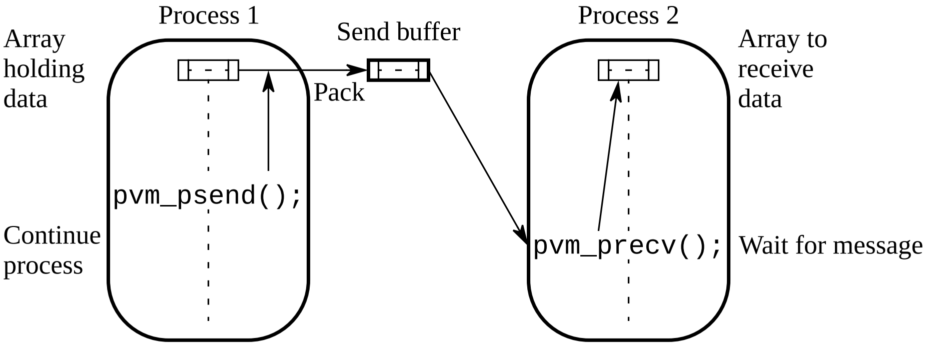


Figure 2.12 `pvm_psend()` and `pvm_precv()` system calls.

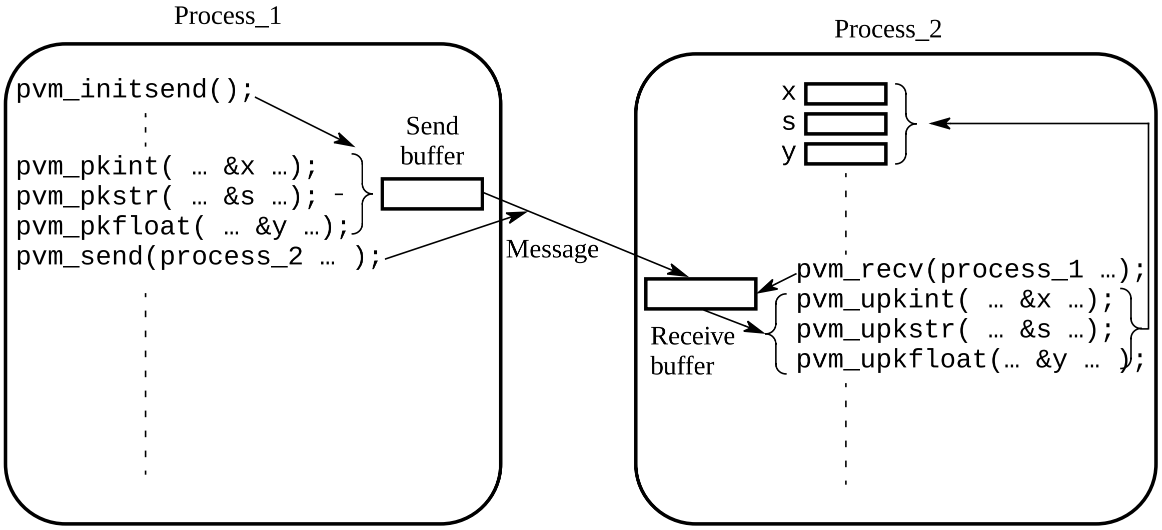
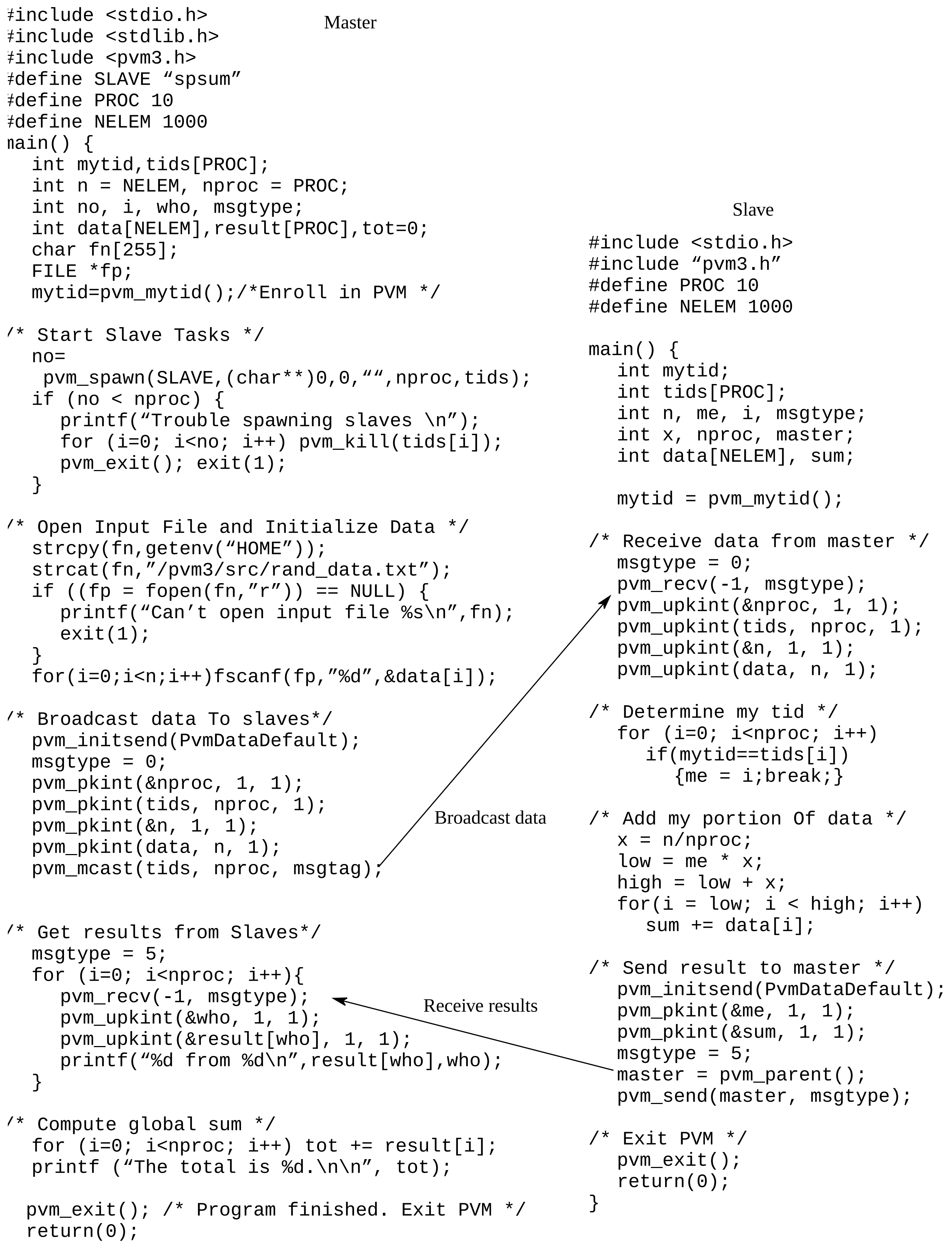


Figure 2.13 PVM packing messages, sending, and unpacking.



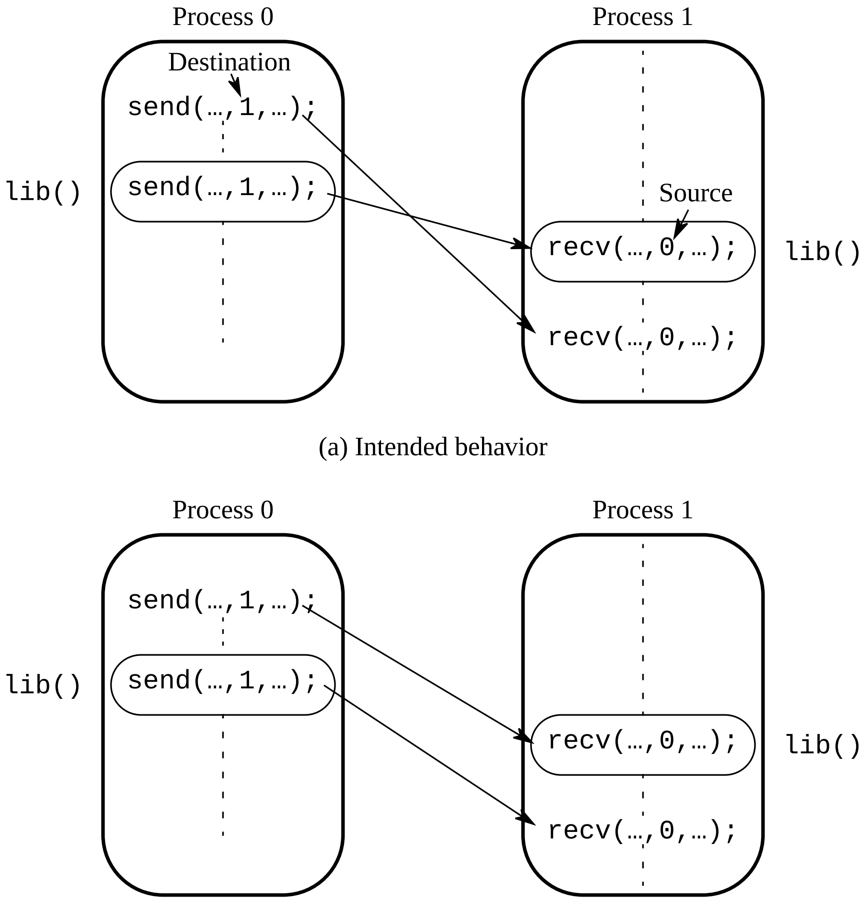
Master

```
#include <stdio.h>
#include <stdlib.h>
#include <pvm3.h>
#define SLAVE “spsum”
#define PROC 10
#define NELEM 1000
main() {
  int mytid,tids[PROC];
  int n = NELEM, nproc = PROC;
  int no, i, who, msgtype;
  int data[NELEM],result[PROC],tot=0;
  char fn[255];
  FILE *fp;
  mytid=pvm_mytid();/*Enroll in PVM */

/* Start Slave Tasks */
  no=
   pvm_spawn(SLAVE,(char**)0,0,“”,nproc,tids);
  if (no < nproc) {
    printf(“Trouble spawning slaves \n”);
    for (i=0; i<no; i++) pvm_kill(tids[i]);
    pvm_exit(); exit(1);
  }

/* Open Input File and Initialize Data */
  strcpy(fn,getenv(“HOME”));
  strcat(fn,”/pvm3/src/rand_data.txt”);
  if ((fp = fopen(fn,”r”)) == NULL) {
    printf(“Can’t open input file %s\n”,fn);
    exit(1);
  }
  for(i=0;i<n;i++)fscanf(fp,”%d”,&data[i]);

/* Broadcast data To slaves*/
  pvm_initsend(PvmDataDefault);
  msgtype = 0;
  pvm_pkint(&nproc, 1, 1);
  pvm_pkint(tids, nproc, 1);
  pvm_pkint(&n, 1, 1);
  pvm_pkint(data, n, 1);
  pvm_mcast(tids, nproc, msgtag);


/* Get results from Slaves*/
  msgtype = 5;
  for (i=0; i<nproc; i++){
    pvm_recv(-1, msgtype);
    pvm_upkint(&who, 1, 1);
    pvm_upkint(&result[who], 1, 1);
    printf(“%d from %d\n”,result[who],who);
  }

/* Compute global sum */
  for (i=0; i<nproc; i++) tot += result[i];
  printf (“The total is %d.\n\n”, tot);

  pvm_exit(); /* Program finished. Exit PVM */
  return(0);
```

Slave

```
#include <stdio.h>
#include “pvm3.h”
#define PROC 10
#define NELEM 1000

main() {
  int mytid;
  int tids[PROC];
  int n, me, i, msgtype;
  int x, nproc, master;
  int data[NELEM], sum;

  mytid = pvm_mytid();

/* Receive data from master */
  msgtype = 0;
  pvm_recv(-1, msgtype);
  pvm_upkint(&nproc, 1, 1);
  pvm_upkint(tids, nproc, 1);
  pvm_upkint(&n, 1, 1);
  pvm_upkint(data, n, 1);

/* Determine my tid */
  for (i=0; i<nproc; i++)
    if(mytid==tids[i])
      {me = i;break;}

/* Add my portion Of data */
  x = n/nproc;
  low = me * x;
  high = low + x;
  for(i = low; i < high; i++)
    sum += data[i];

/* Send result to master */
  pvm_initsend(PvmDataDefault);
  pvm_pkint(&me, 1, 1);
  pvm_pkint(&sum, 1, 1);
  msgtype = 5;
  master = pvm_parent();
  pvm_send(master, msgtype);

/* Exit PVM */
  pvm_exit();
  return(0);
}
```

Figure 2.14 Sample PVM program.



(a) Intended behavior

(b) Possible behavior

Figure 2.15 Unsafe message passing with libraries.

```
#include “mpi.h”
#include <stdio.h>
#include <math.h>
#define MAXSIZE 1000

void main(int argc, char *argv)
{
   int myid, numprocs;
   int data[MAXSIZE], i, x, low, high, myresult, result;
   char fn[255];
   char *fp;

   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);

   if (myid == 0) {            /* Open input file and initialize data */
      strcpy(fn,getenv(“HOME”));
      strcat(fn,”/MPI/rand_data.txt”);
      if ((fp = fopen(fn,”r”)) == NULL) {
         printf(“Can’t open the input file: %s\n\n”, fn);
         exit(1);
      }
      for(i = 0; i < MAXSIZE; i++) fscanf(fp,”%d”, &data[i]);
   }

   /* broadcast data */
   MPI_Bcast(data, MAXSIZE, MPI_INT, 0, MPI_COMM_WORLD);

/* Add my portion Of data */
   x = n/nproc;
   low = myid * x;
   high = low + x;
   for(i = low; i < high; i++)
      myresult += data[i];
   printf(“I got %d from %d\n”, myresult, myid);

/* Compute global sum */
   MPI_Reduce(&myresult, &result, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);
   if (myid == 0) printf(“The sum is %d.\n”, result);

   MPI_Finalize();
}
```

Figure 2.16 Sample MPI program.

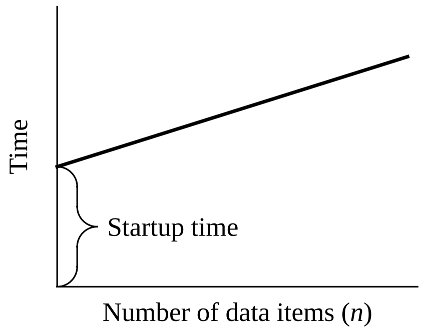


Figure 2.17 Theoretical communication time.

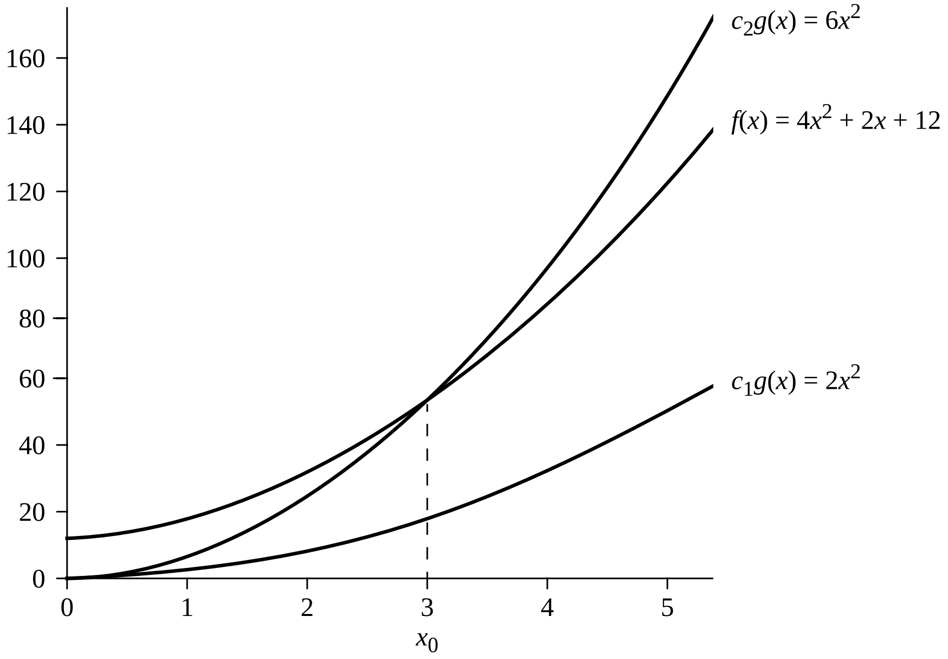


Figure 2.18 Growth of function $f(x) = 4x^2 + 2x + 12$.

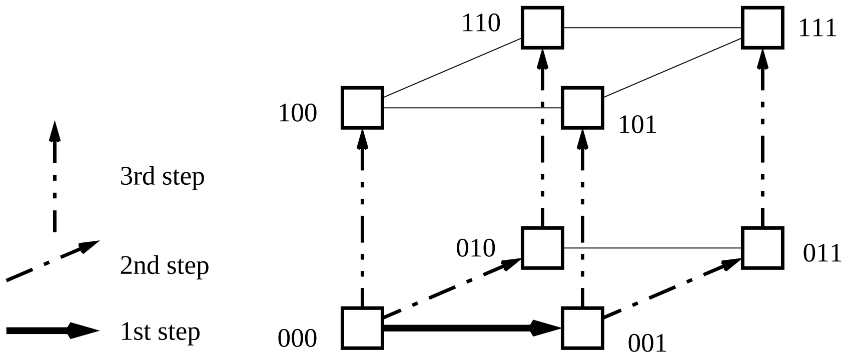


Figure 2.19 Broadcast in a three-dimensional hypercube.

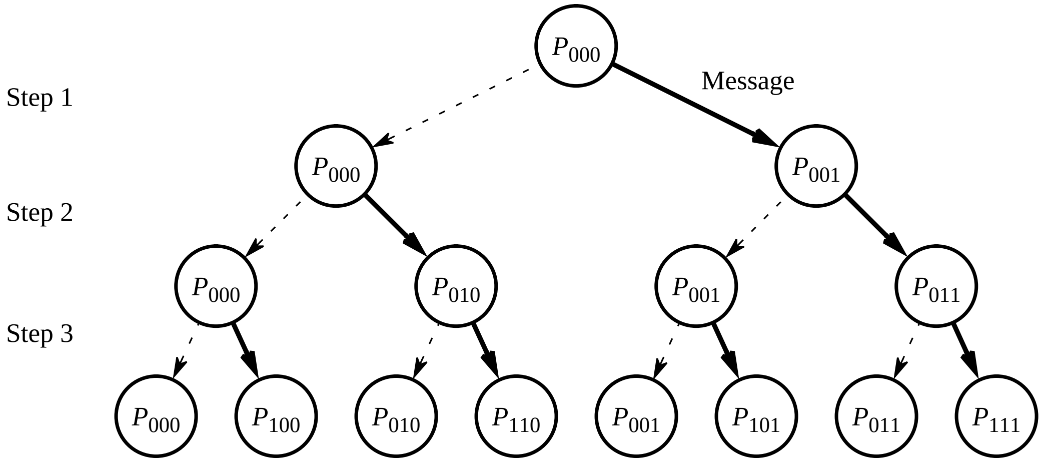


Figure 2.20 Broadcast as a tree construction.

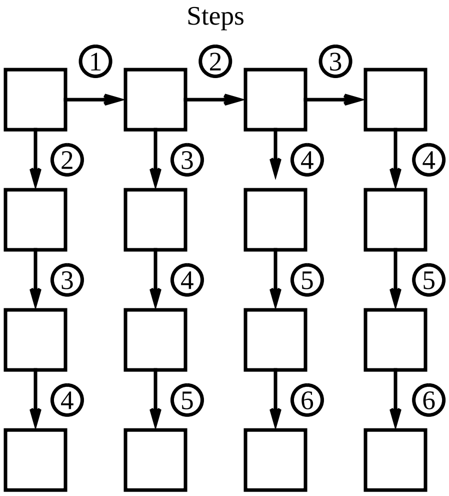


Figure 2.21 Broadcast in a mesh.

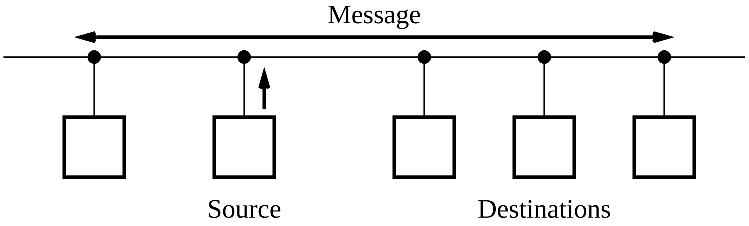


Figure 2.22 Broadcast on an Ethernet network.

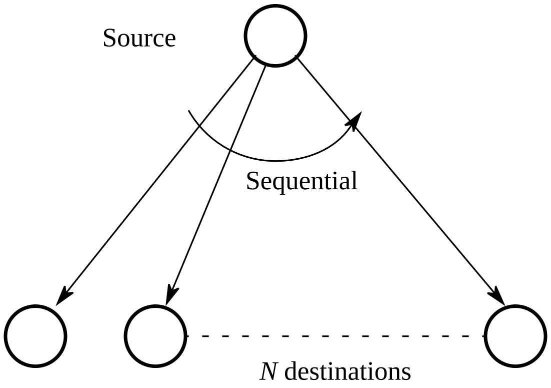


Figure 2.23 1-to-*N* fan-out broadcast.

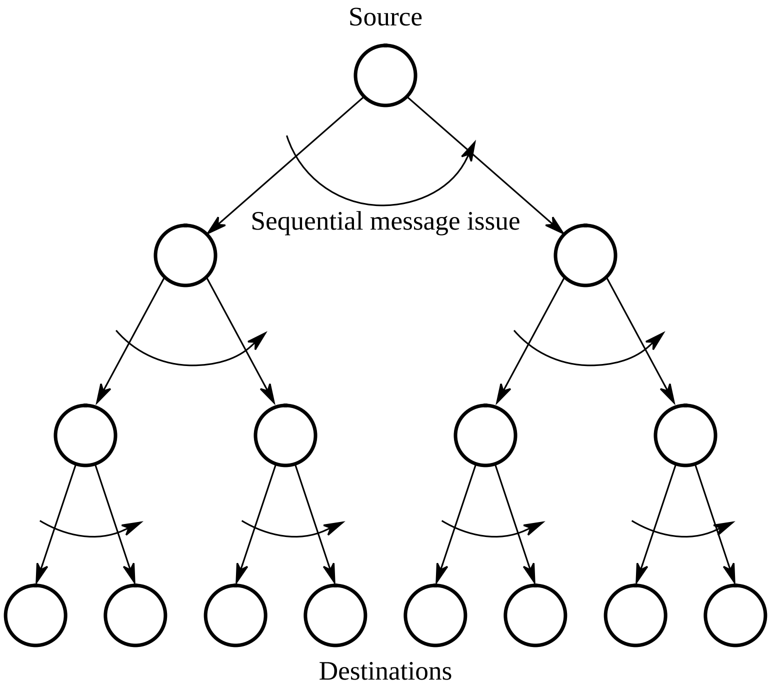


Figure 2.24 1-to-*N* fan-out broadcast on a tree structure.

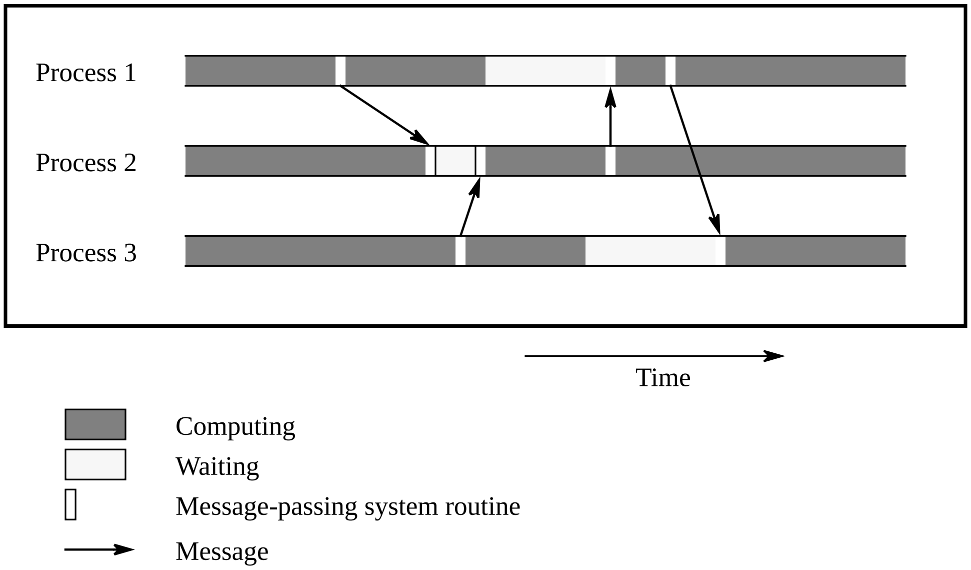


Figure 2.25 Space-time diagram of a parallel program.

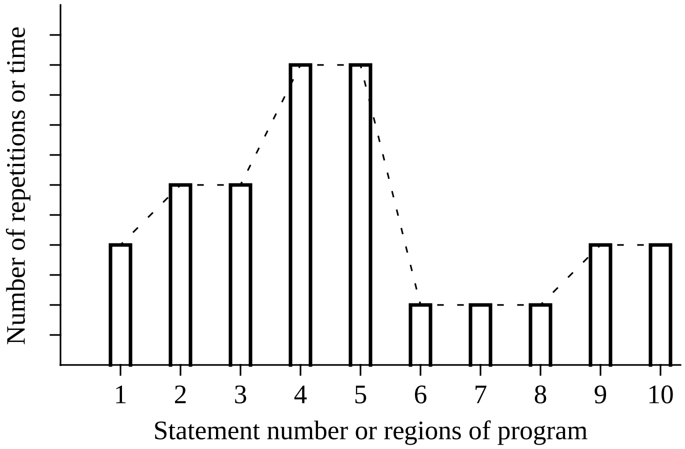


Figure 2.26 Program profile.

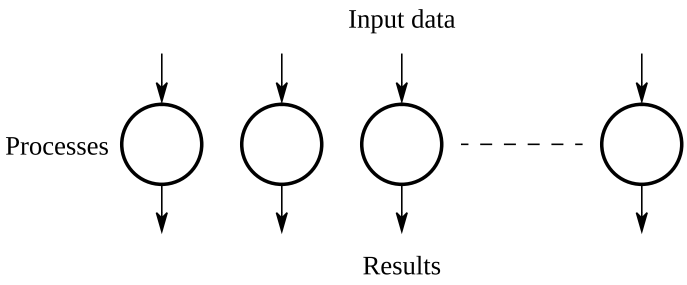


Figure 3.1 Disconnected computational graph (embarrassingly parallel problem).

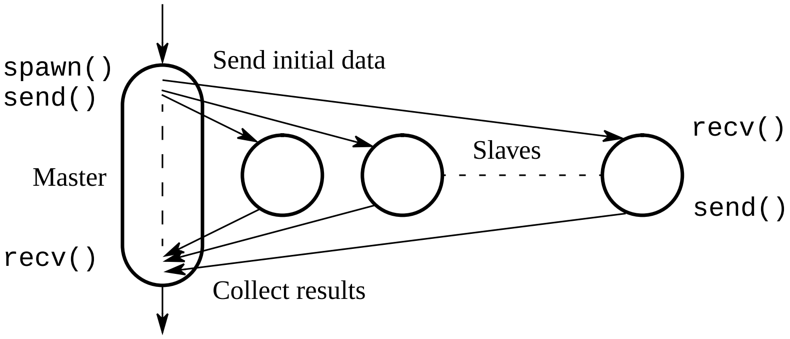
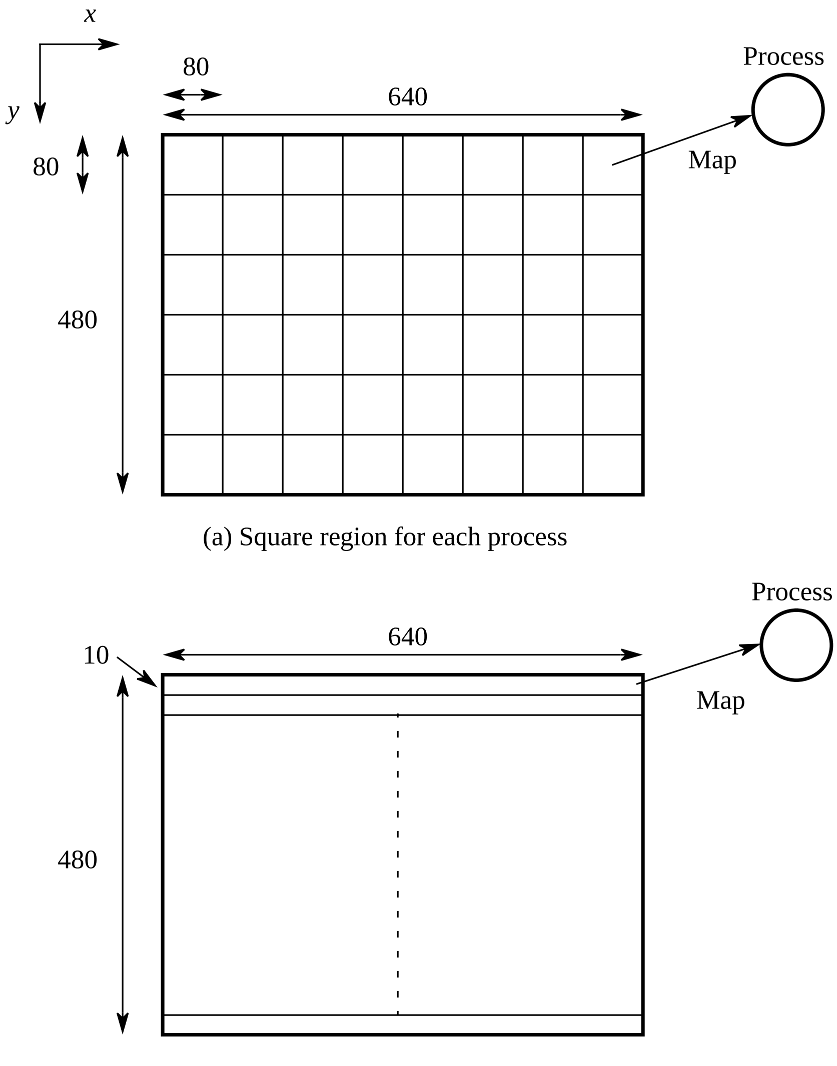


Figure 3.2 Practical embarrassingly parallel computational graph with dynamic process creation and the master-slave approach.



(a) Square region for each process

(b) Row region for each process

Figure 3.3 Partitioning into regions for individual processes.

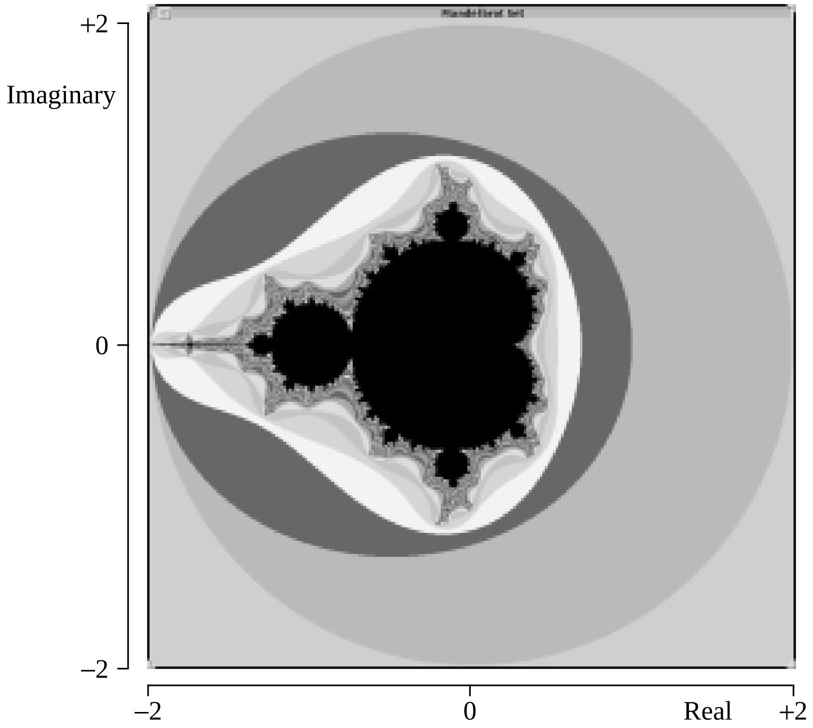


Figure 3.4 Mandelbrot set.

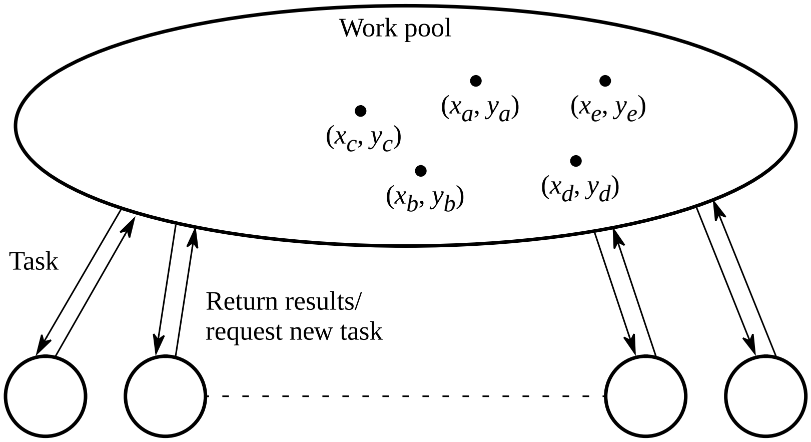


Figure 3.5 Work pool approach.

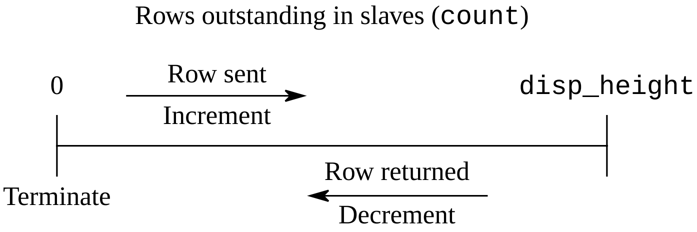


Figure 3.6 Counter termination.

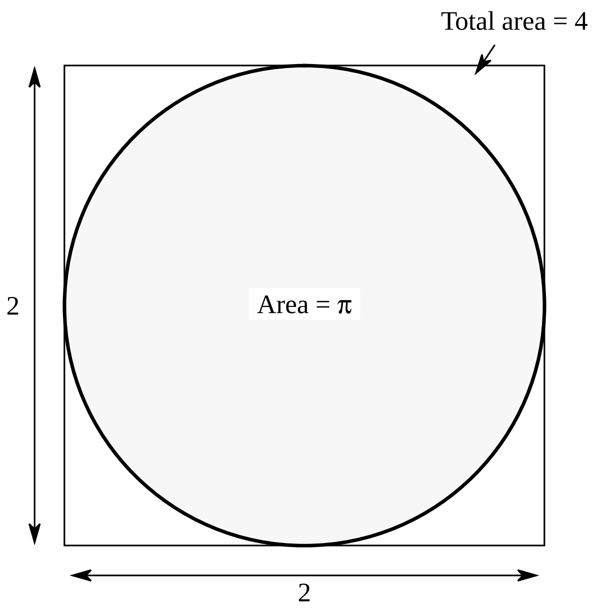


Figure 3.7 Computing π by a Monte Carlo method.

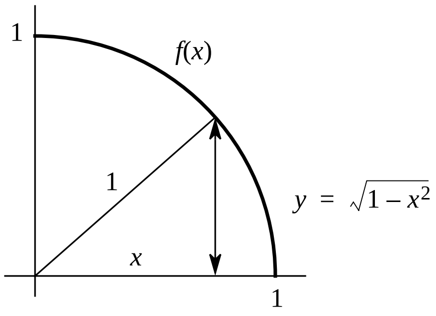


Figure 3.8 Function being integrated in computing π by a Monte Carlo method.

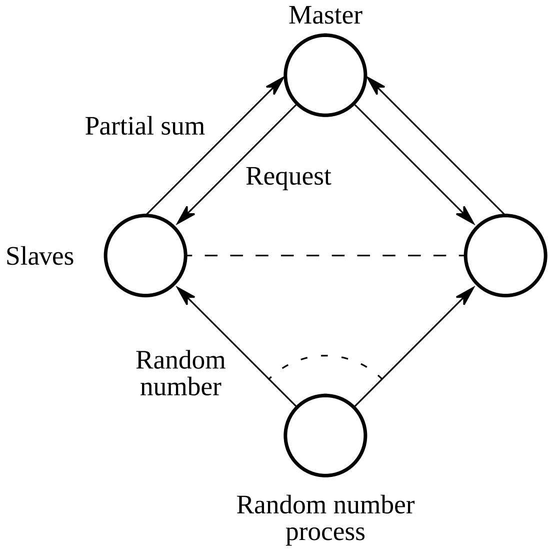


Figure 3.9 Parallel Monte Carlo integration.

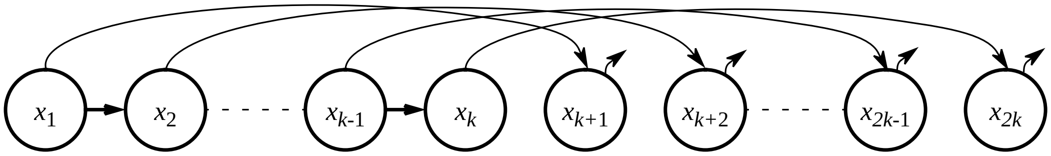


Figure 3.10 Parallel computation of a sequence.

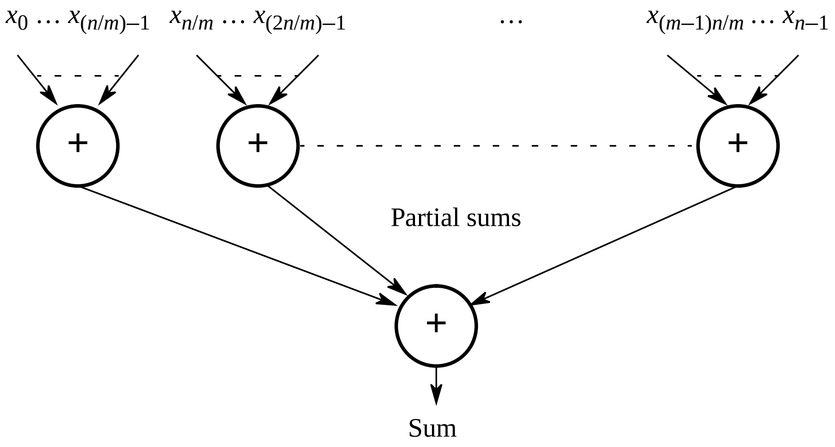


Figure 4.1 Partitioning a sequence of numbers into parts and adding the parts.

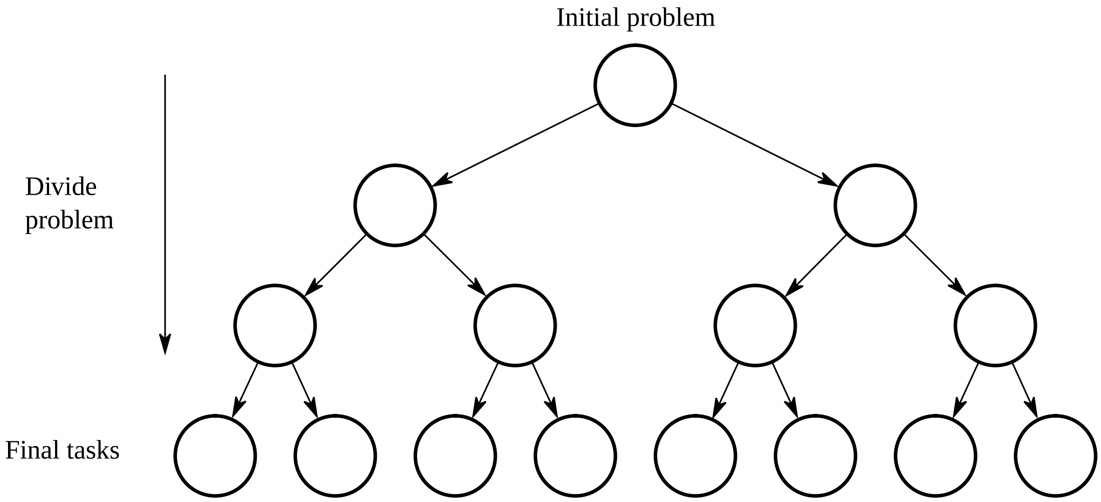


Figure 4.2 Tree construction.

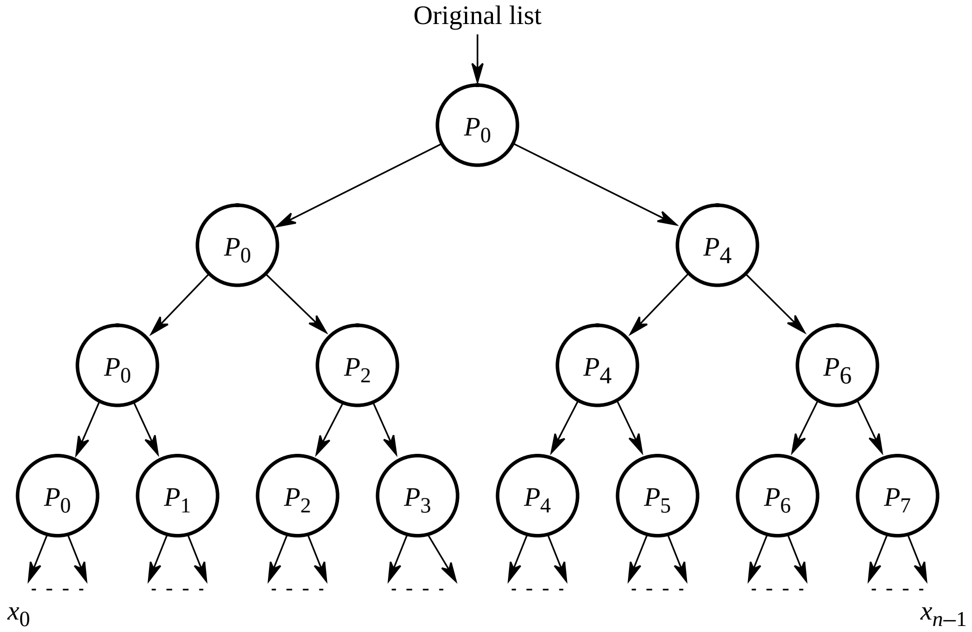


Figure 4.3 Dividing a list into parts.

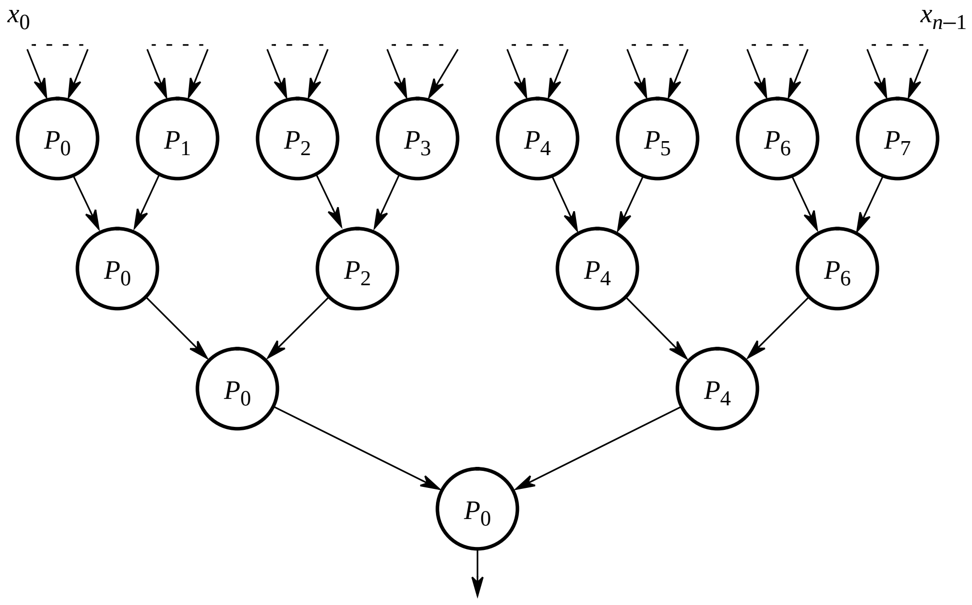


Figure 4.4 Partial summation.

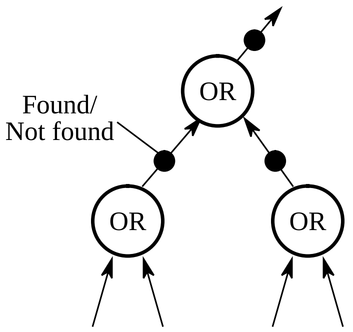


Figure 4.5 Part of a search tree.

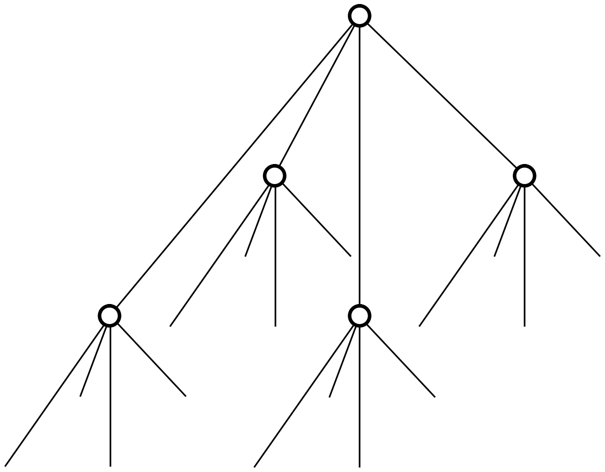


Figure 4.6 Quadtree.

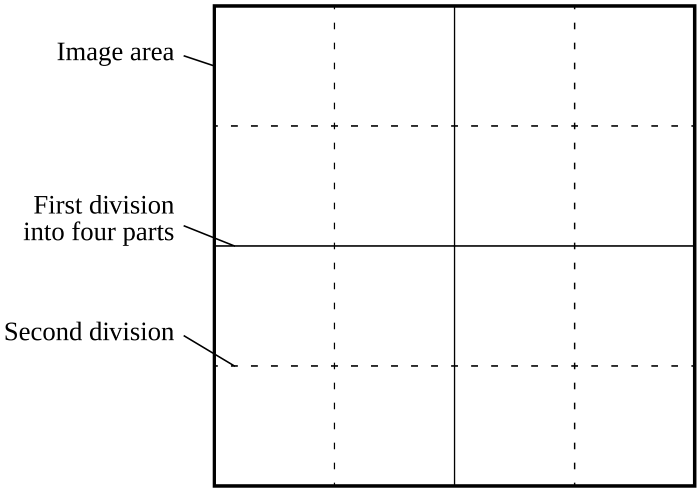


Figure 4.7 Dividing an image.

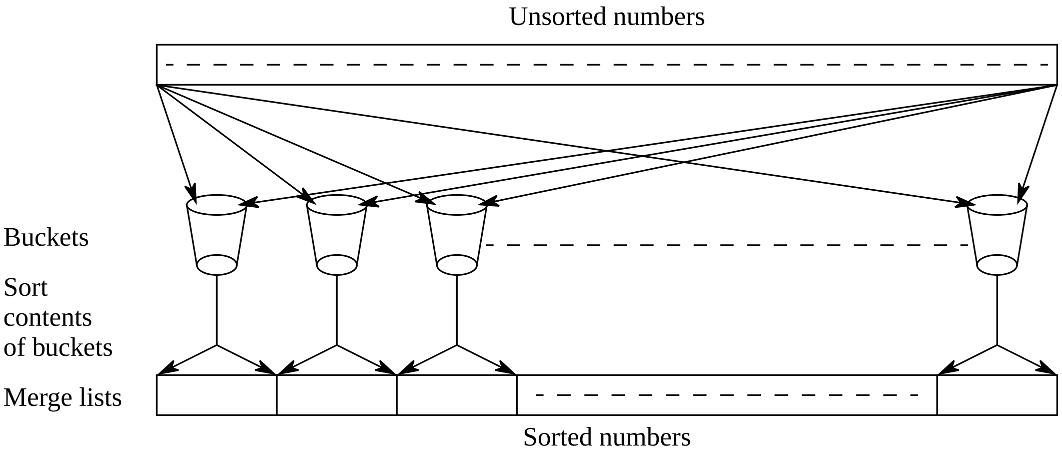


Figure 4.8 Bucket sort.

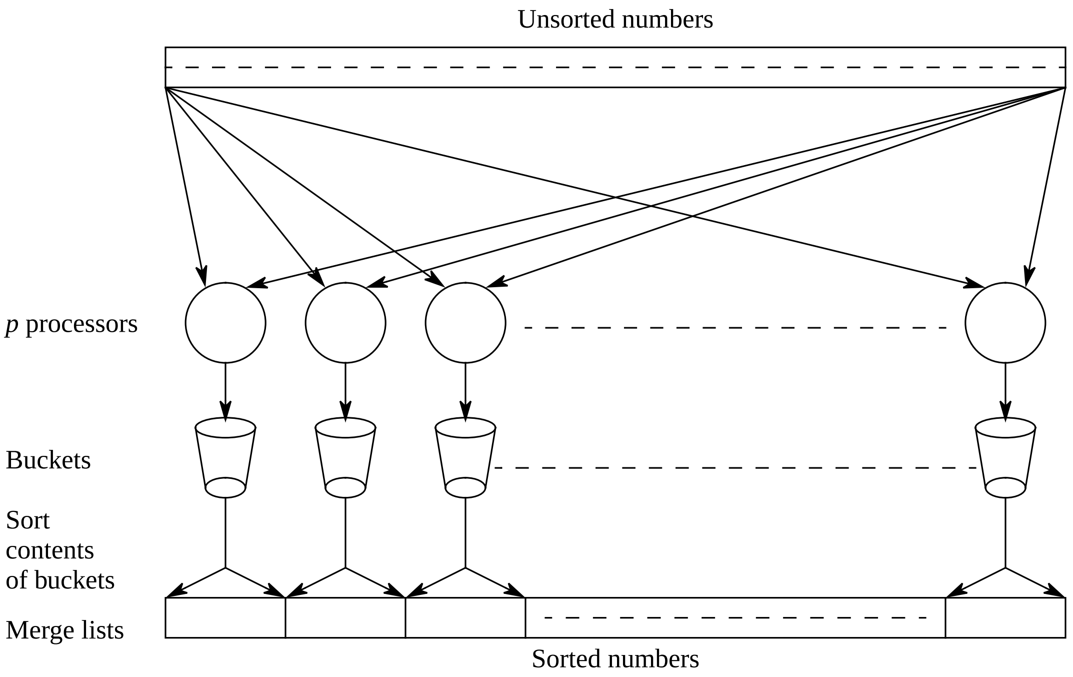


Figure 4.9 One parallel version of bucket sort.

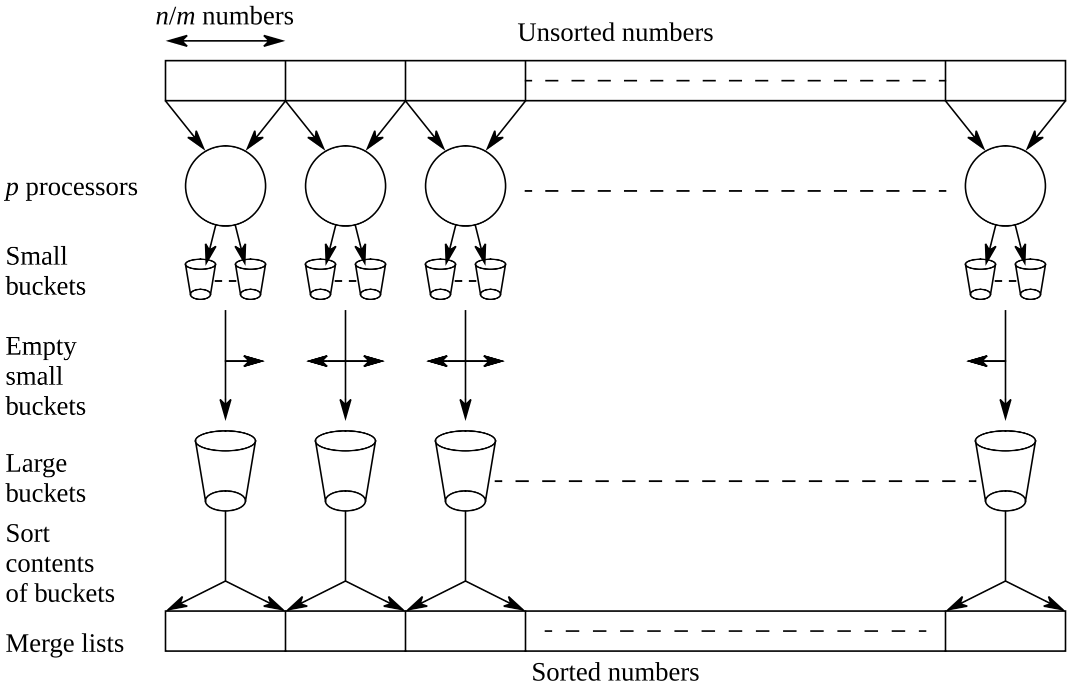


Figure 4.10 Parallel version of bucket sort.

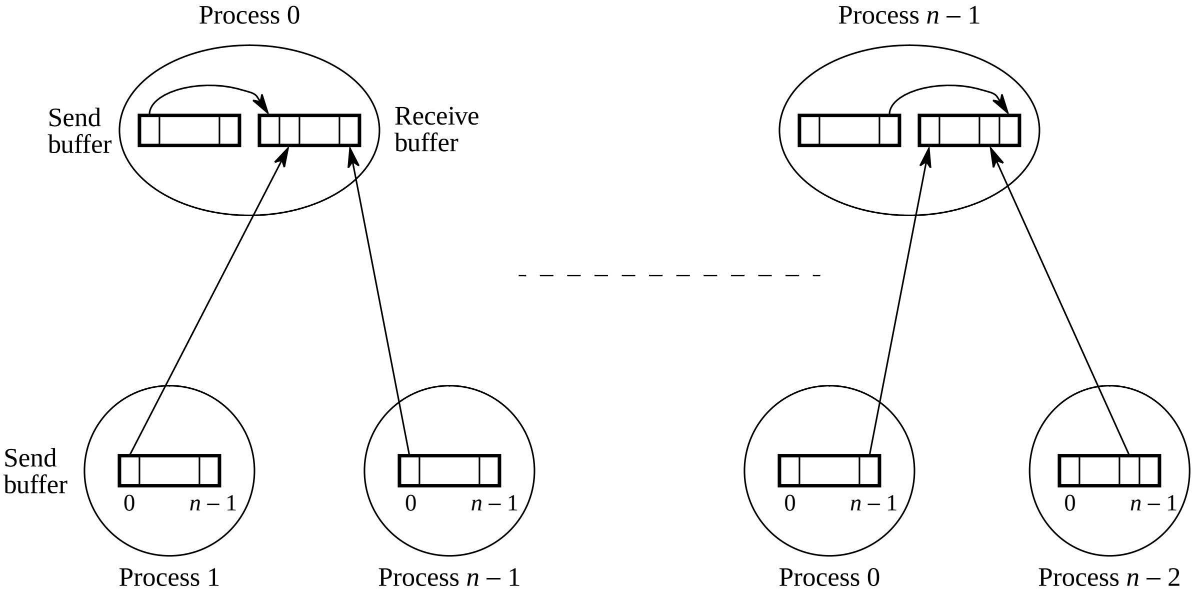


Figure 4.11 “All-to-all” broadcast.

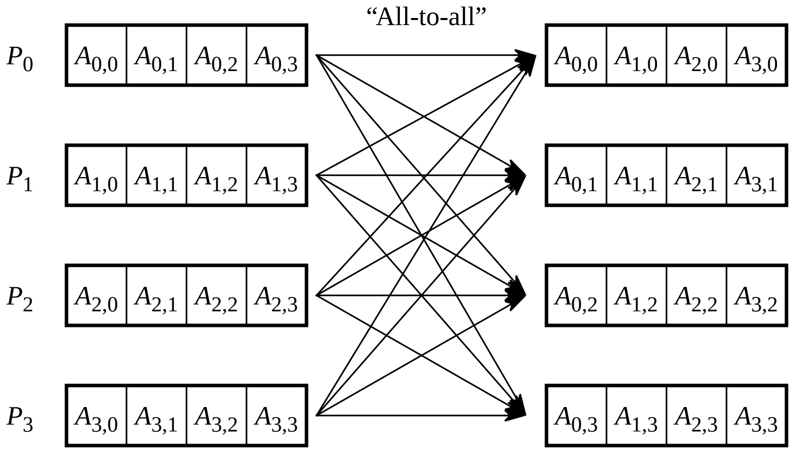

"All-to-all"
P_0
P_1
P_2
P_3
$A_{0,0}$ $A_{0,1}$ $A_{0,2}$ $A_{0,3}$
$A_{1,0}$ $A_{1,1}$ $A_{1,2}$ $A_{1,3}$
$A_{2,0}$ $A_{2,1}$ $A_{2,2}$ $A_{2,3}$
$A_{3,0}$ $A_{3,1}$ $A_{3,2}$ $A_{3,3}$
$A_{0,0}$ $A_{1,0}$ $A_{2,0}$ $A_{3,0}$
$A_{0,1}$ $A_{1,1}$ $A_{2,1}$ $A_{3,1}$
$A_{0,2}$ $A_{1,2}$ $A_{2,2}$ $A_{3,2}$
$A_{0,3}$ $A_{1,3}$ $A_{2,3}$ $A_{3,3}$

Figure 4.12 Effect of "all-to-all" on an array.

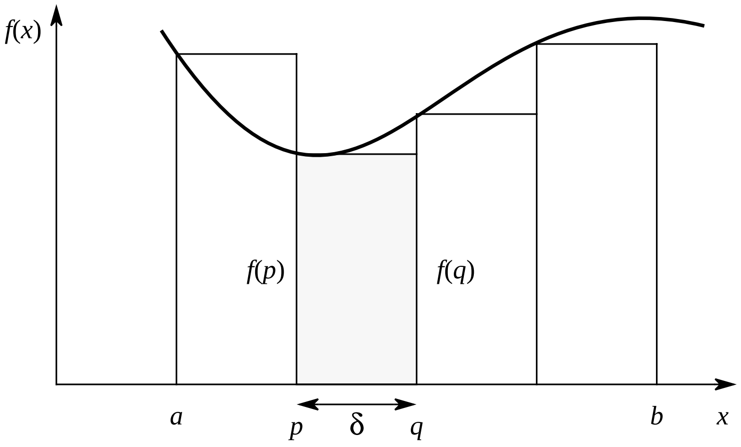


Figure 4.13 Numerical integration using rectangles.

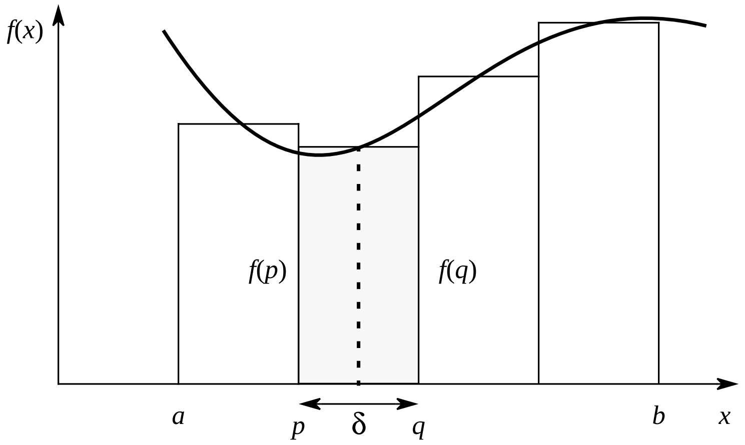


Figure 4.14 More accurate numerical integration using rectangles.

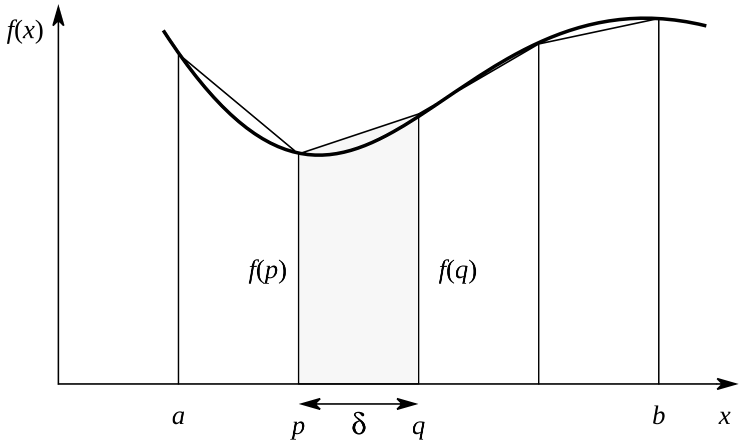


Figure 4.15 Numerical integration using the trapezoidal method.

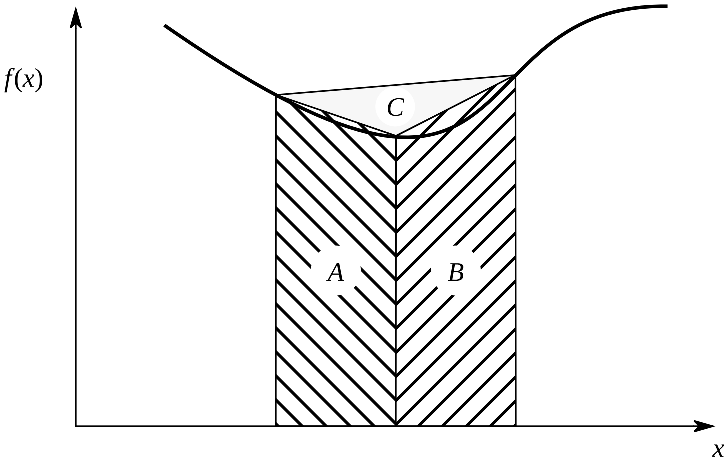


Figure 4.16 Adaptive quadrature construction.

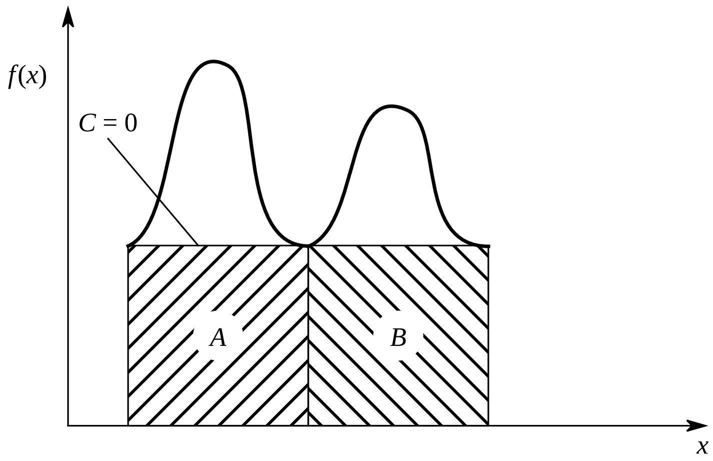


Figure 4.17 Adaptive quadrature with false termination.

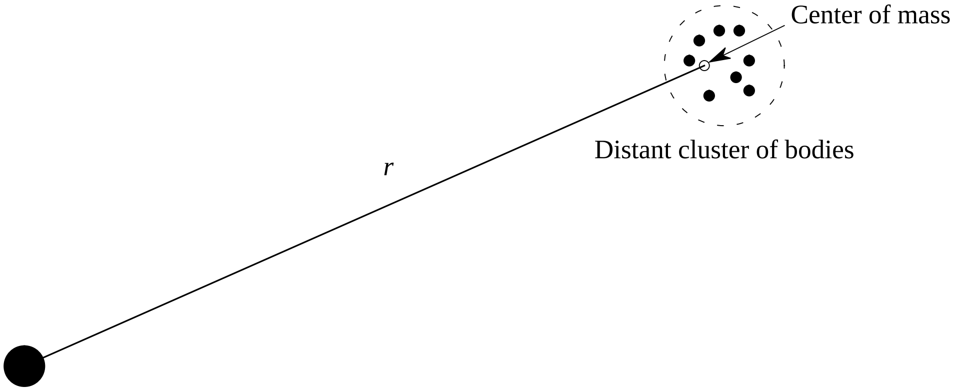
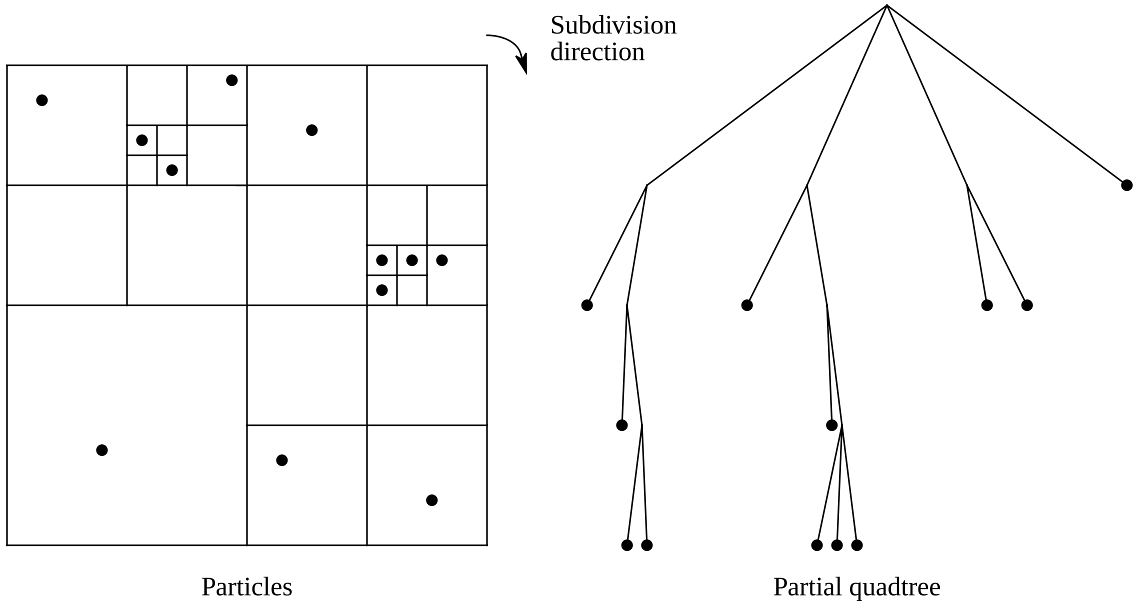


Figure 4.18 Clustering distant bodies.



Particles

Partial quadtree

Figure 4.19 Recursive division of two-dimensional space.

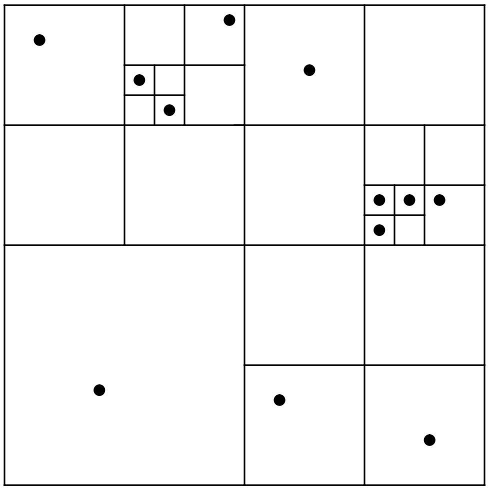


Figure 4.20 Orthogonal recursive bisection method.

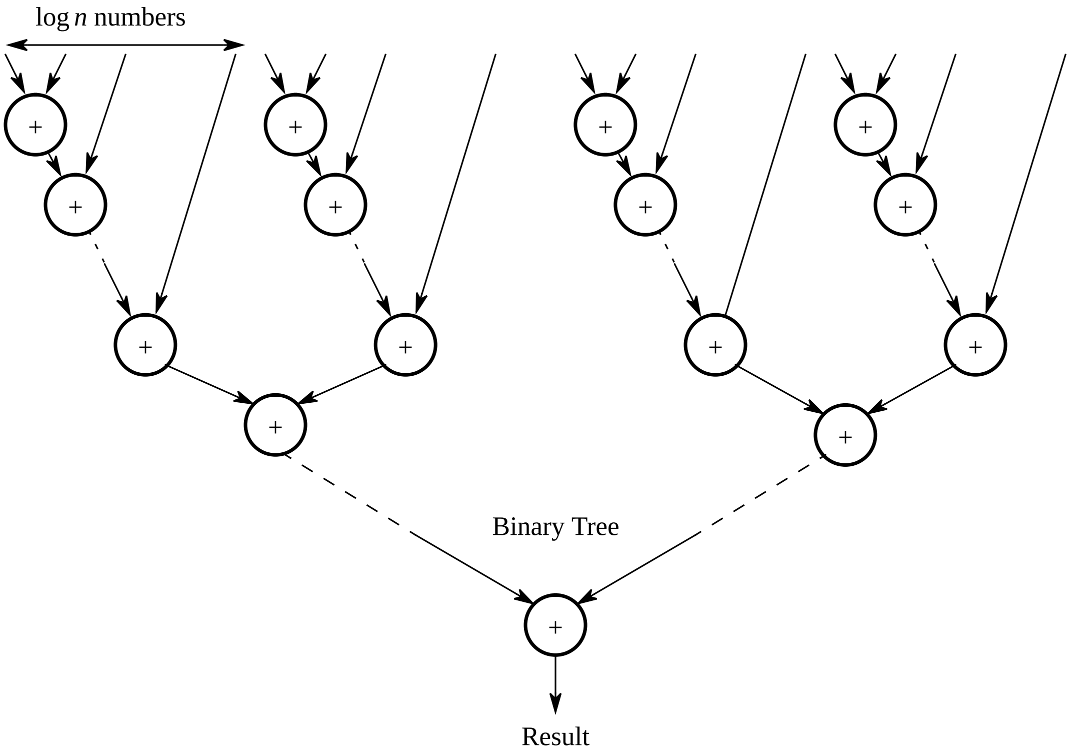


Figure 4.21 Process diagram for Problem 4-12(b).

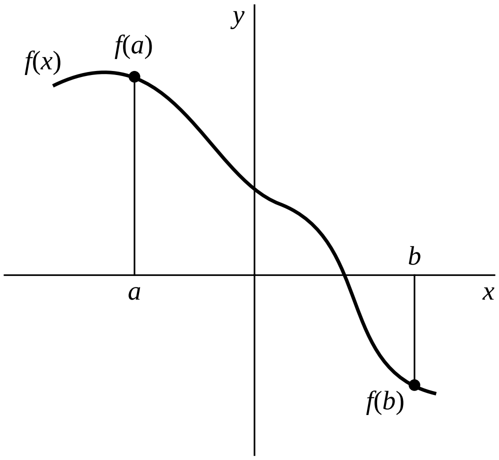


Figure 4.22 Bisection method for finding the zero crossing location of a function.

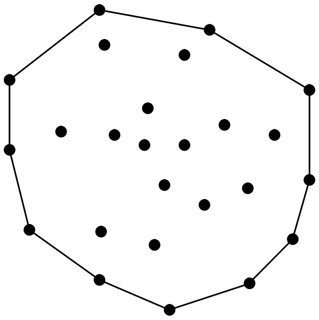


Figure 4.23 Convex hull (Problem 4-22).

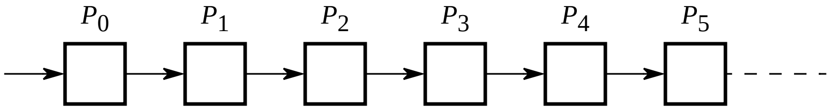


Figure 5.1 Pipelined processes.

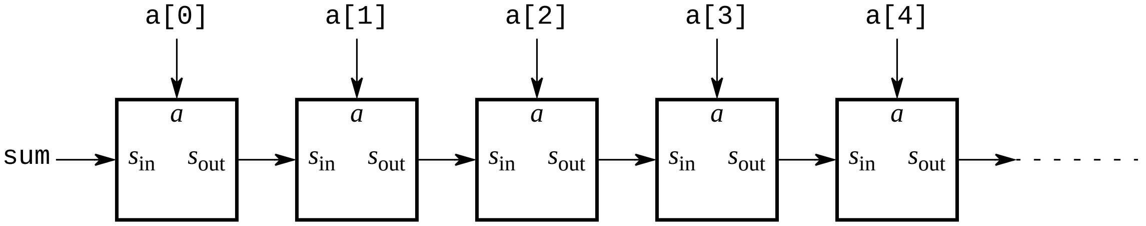


Figure 5.2 Pipeline for an unfolded loop.

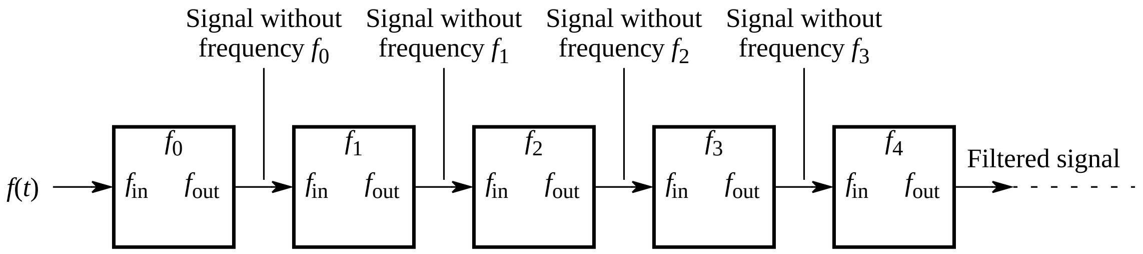
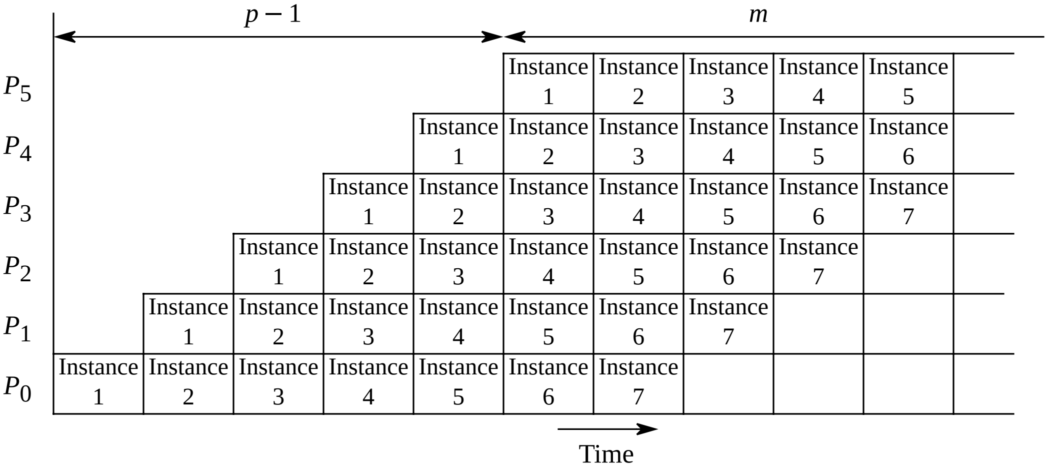


Figure 5.3 Pipeline for a frequency filter.



P_5						Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	
P_4					Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	
P_3				Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7	
P_2			Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7		
P_1		Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7			
P_0	Instance 1	Instance 2	Instance 3	Instance 4	Instance 5	Instance 6	Instance 7				

Figure 5.4 Space-time diagram of a pipeline.

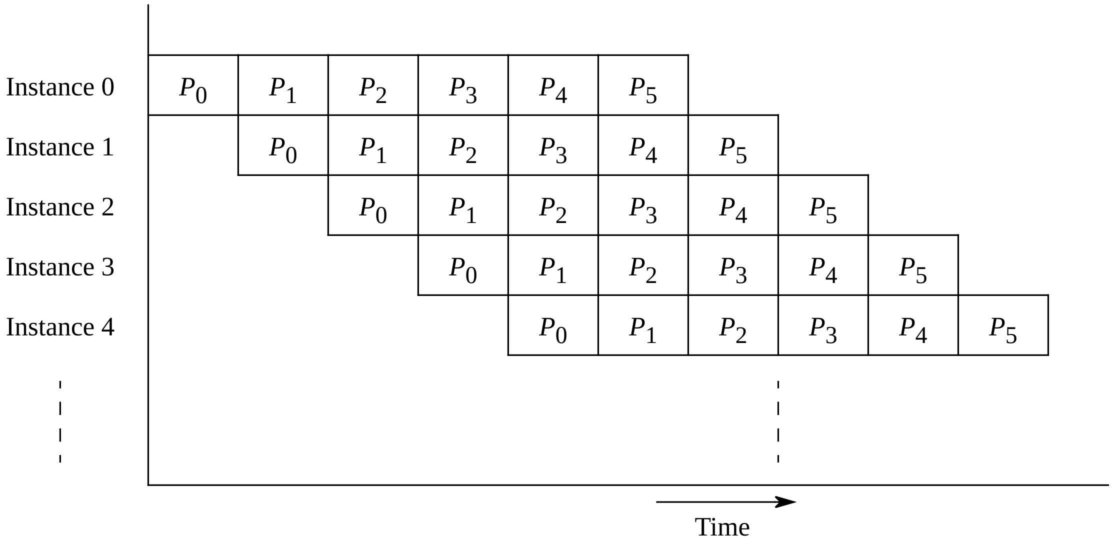
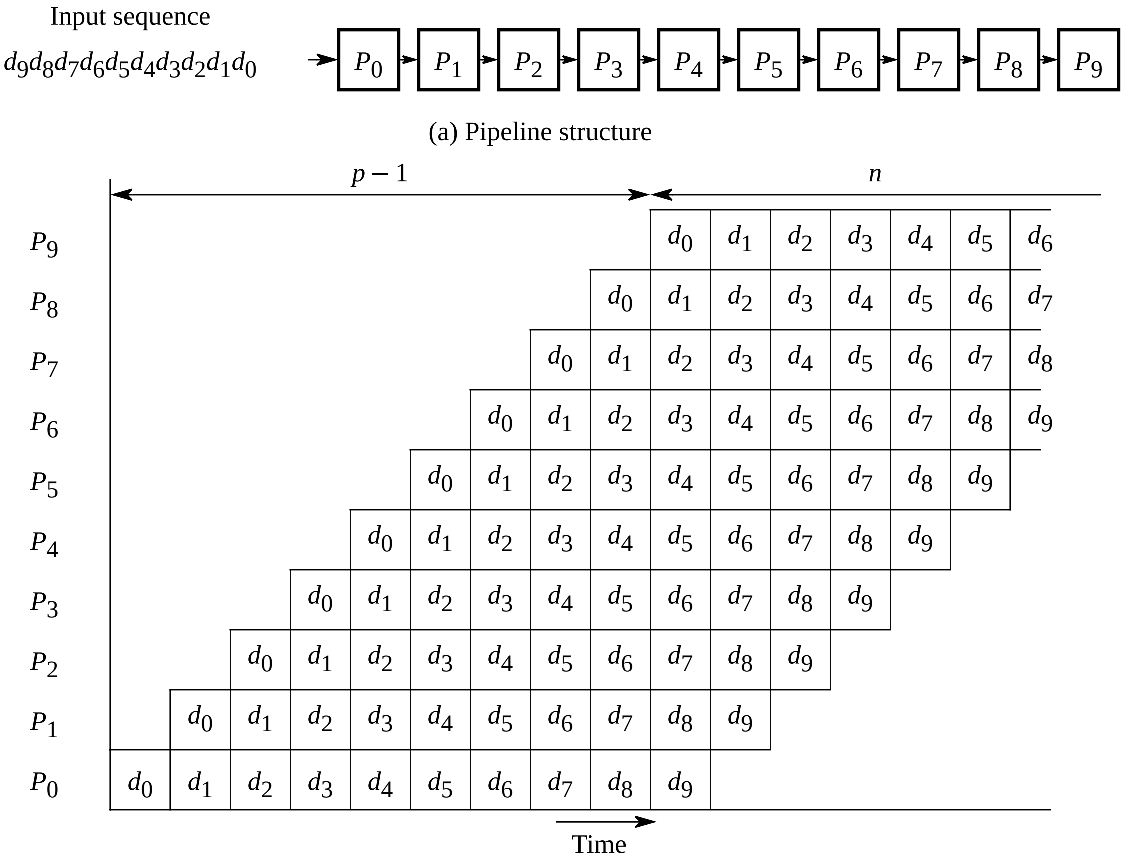

Instance 0 P_0 P_1 P_2 P_3 P_4 P_5
Instance 1 P_0 P_1 P_2 P_3 P_4 P_5
Instance 2 P_0 P_1 P_2 P_3 P_4 P_5
Instance 3 P_0 P_1 P_2 P_3 P_4 P_5
Instance 4 P_0 P_1 P_2 P_3 P_4 P_5
Time

Figure 5.5 Alternative space-time diagram.

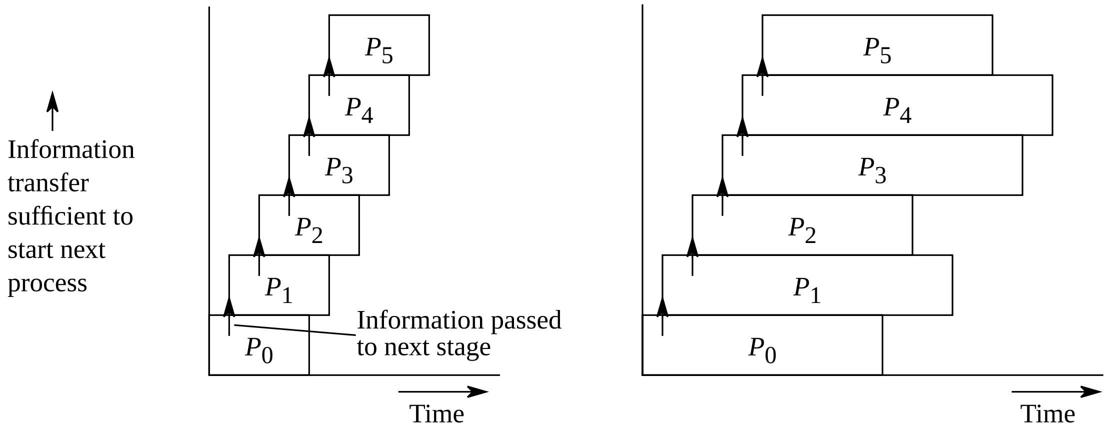


(a) Pipeline structure

P_9										d_0	d_1	d_2	d_3	d_4	d_5	d_6
P_8									d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7
P_7								d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8
P_6							d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9
P_5						d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9	
P_4					d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9		
P_3				d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9			
P_2			d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9				
P_1		d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9					
P_0	d_0	d_1	d_2	d_3	d_4	d_5	d_6	d_7	d_8	d_9						

(b) Timing diagram

Figure 5.6 Pipeline processing 10 data elements.



(a) Processes with the same execution time

(b) Processes not with the same execution time

Figure 5.7 Pipeline processing where information passes to next stage before end of process.

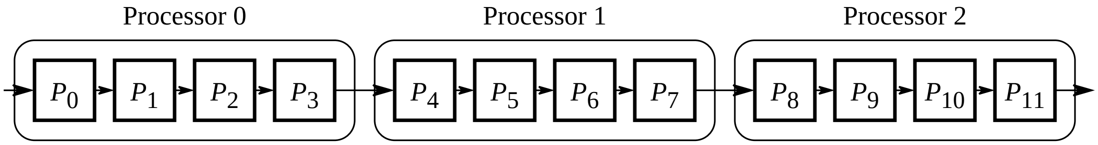


Figure 5.8 Partitioning processes onto processors.

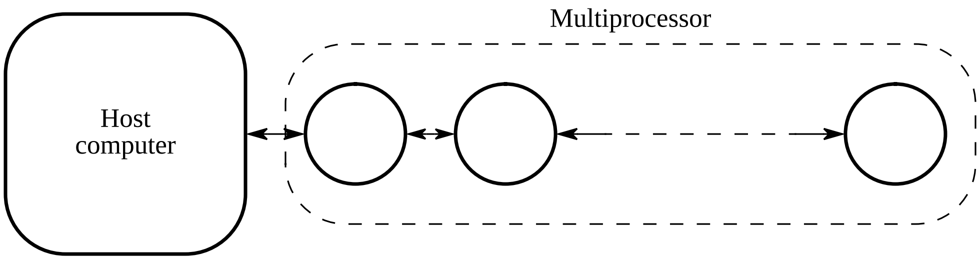


Figure 5.9 Multiprocessor system with a line configuration.

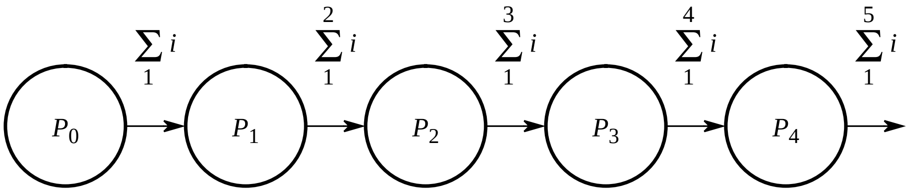


Figure 5.10 Pipelined addition.

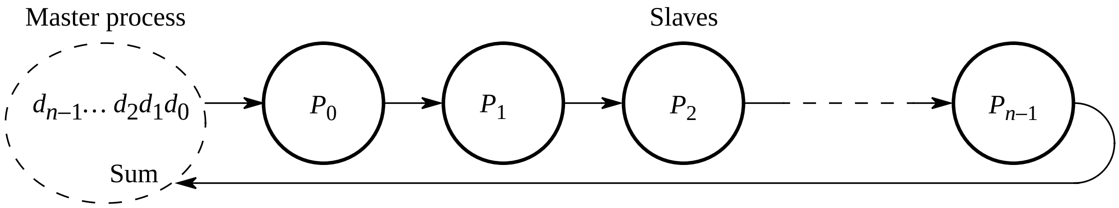


Figure 5.11 Pipelined addition numbers with a master process and ring configuration.

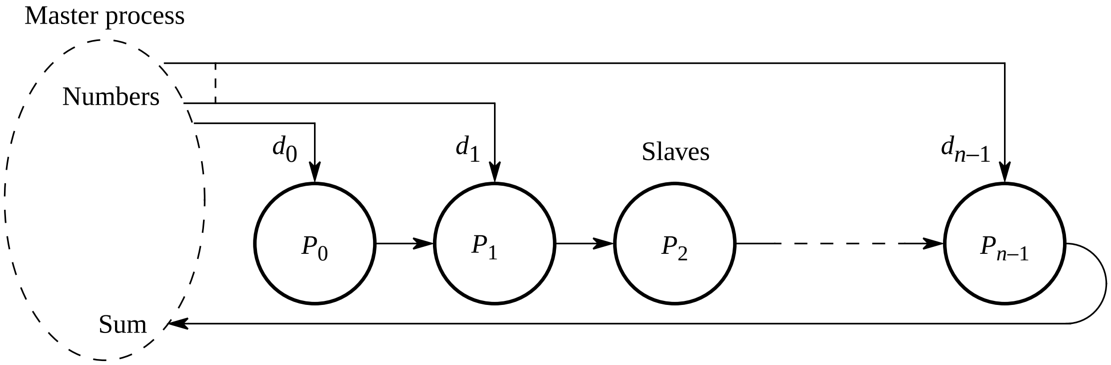


Figure 5.12 Pipelined addition of numbers with direct access to slave processes.

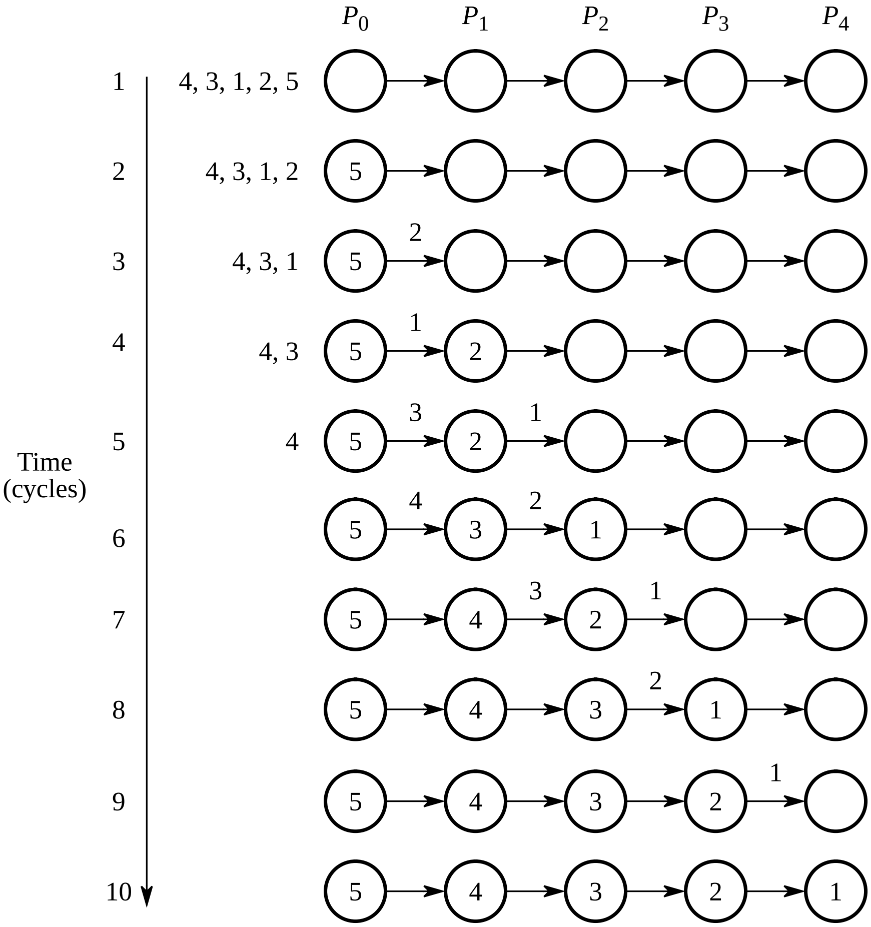


Figure 5.13 Steps in insertion sort with five numbers.

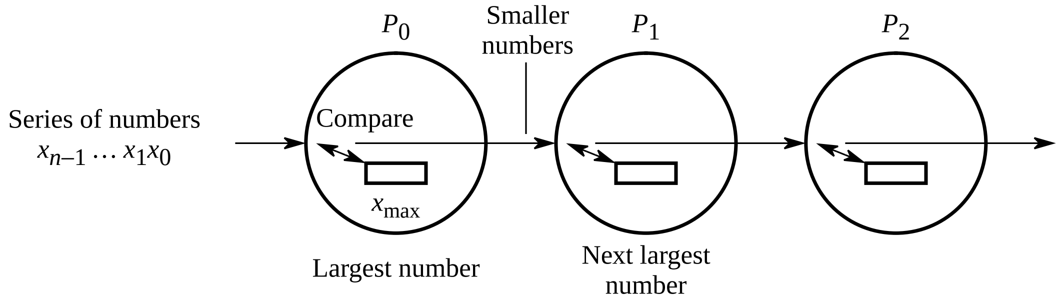


Figure 5.14 Pipeline for sorting using insertion sort.

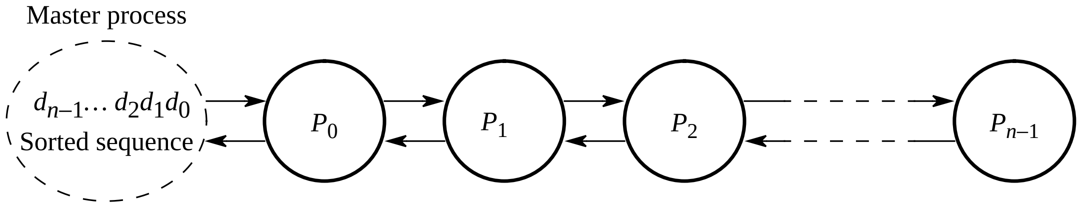


Figure 5.15 Insertion sort with results returned to the master process using a bidirectional line configuration.

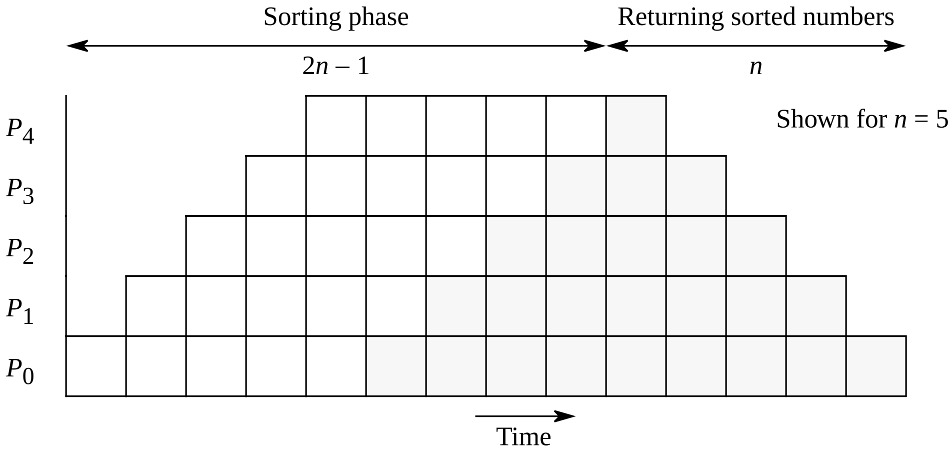


Figure 5.16 Insertion sort with results returned.

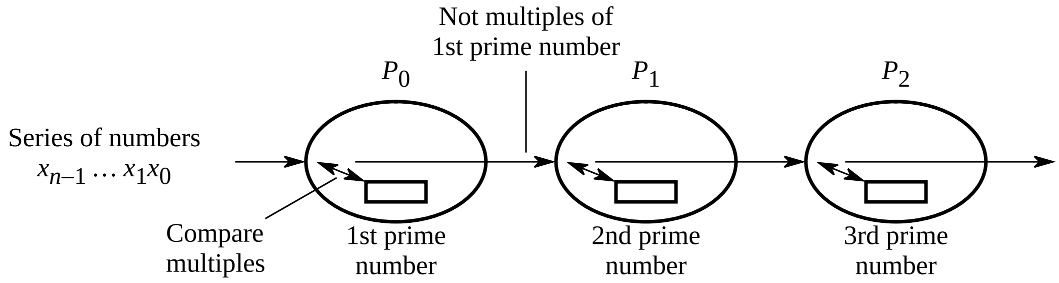


Figure 5.17 Pipeline for sieve of Eratosthenes.

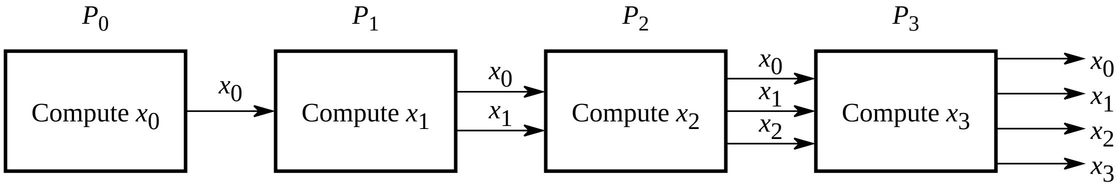


Figure 5.18 Solving an upper triangular set of linear equation using a pipeline.

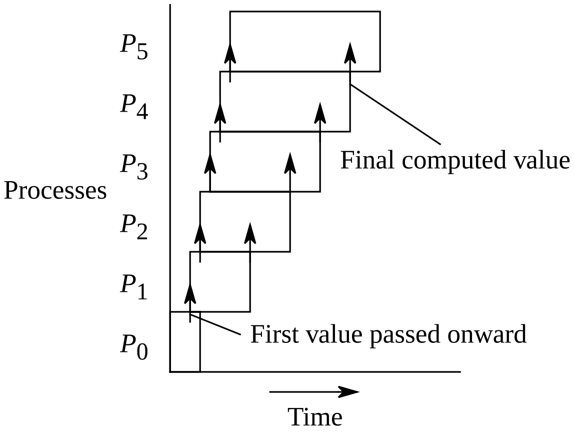


Figure 5.19 Pipeline processing using back substitution.

Time

P_0	P_1	P_2	P_3	P_4
divide				
send(x_0) ⇒	recv(x_0)			
end	send(x_0) ⇒	recv(x_0)		
	multiply/add	send(x_0) ⇒	recv(x_0)	
	divide/subtract	multiply/add	send(x_0) ⇒	recv(x_0)
	send(x_1) ⇒	recv(x_1)	multiply/add	send(x_1) ⇒
	end	send(x_1) ⇒	recv(x_1)	multiply/add
		multiply/add	send(x_1) ⇒	recv(x_1)
		divide/subtract	multiply/add	send(x_1) ⇒
		send(x_2) ⇒	recv(x_2)	multiply/add
		end	send(x_2) ⇒	recv(x_2)
			multiply/add	send(x_2) ⇒
			divide/subtract	multiply/add
			send(x_3) ⇒	recv(x_3)
			end	send(x_3) ⇒
				multiply/add
				divide/subtract
				send(x_4) ⇒
				end

Figure 5.20 Operations in back substitution pipeline.

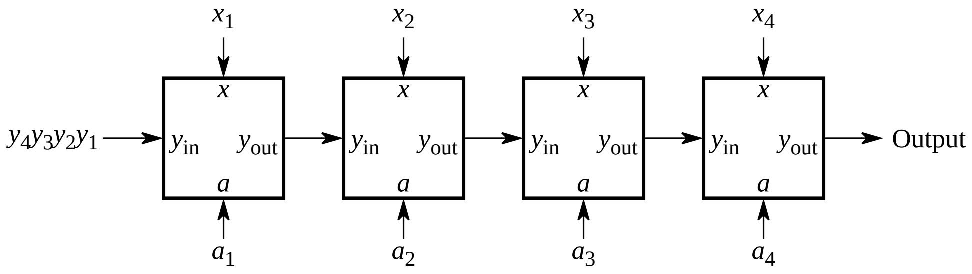
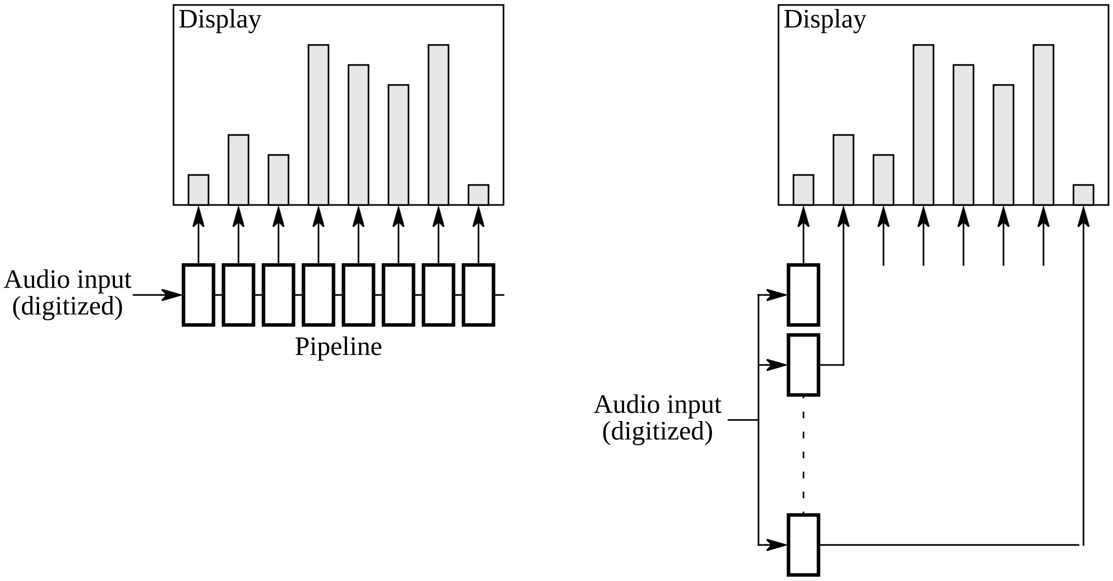


Figure 5.21 Pipeline for Problem 5-9.



(a) Pipeline solution

(b) Direct decomposition

Figure 5.22 Audio histogram display.

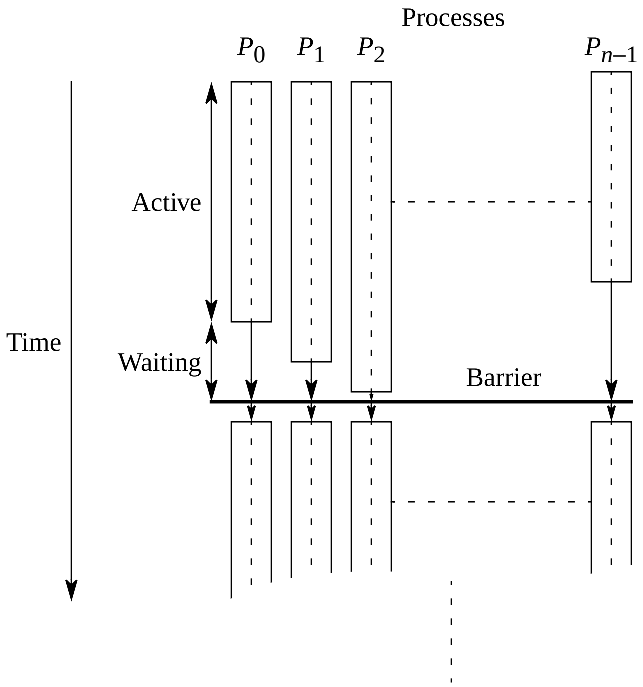


Figure 6.1 Processes reaching the barrier at different times.

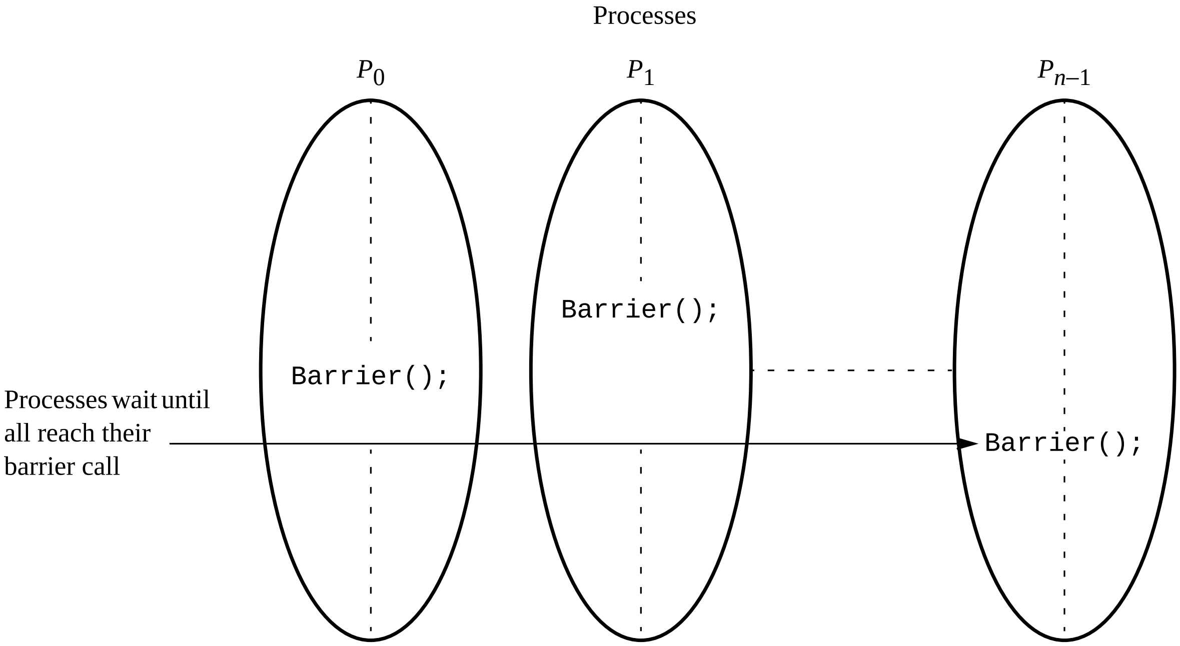


Figure 6.2 Library call barriers.

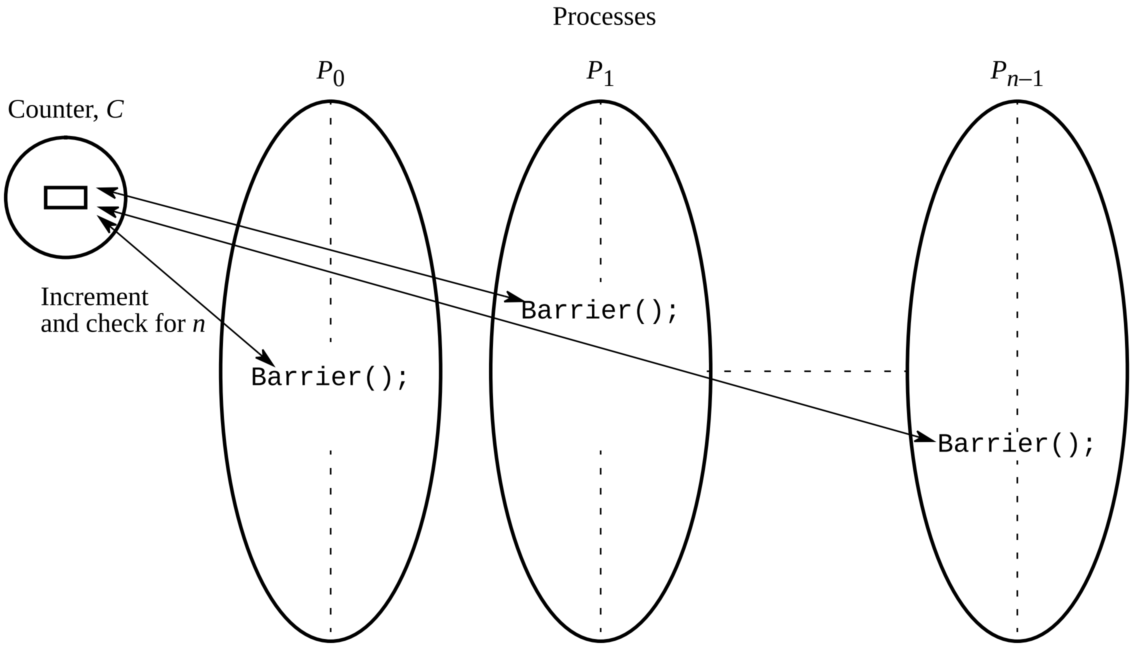


Figure 6.3 Barrier using a centralized counter.

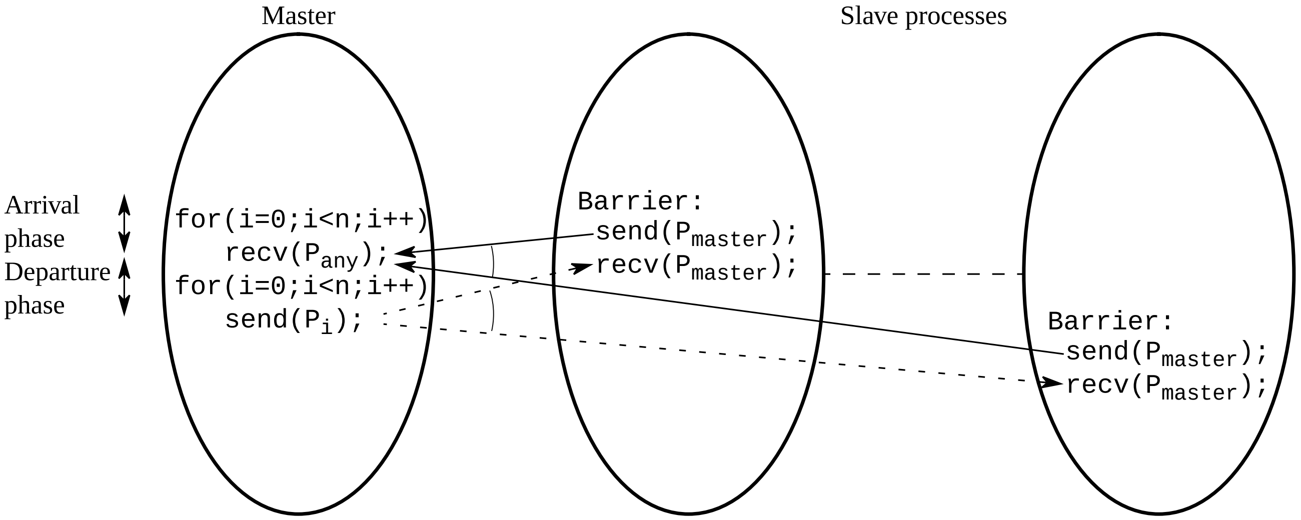


Figure 6.4 Barrier implementation in a message-passing system.

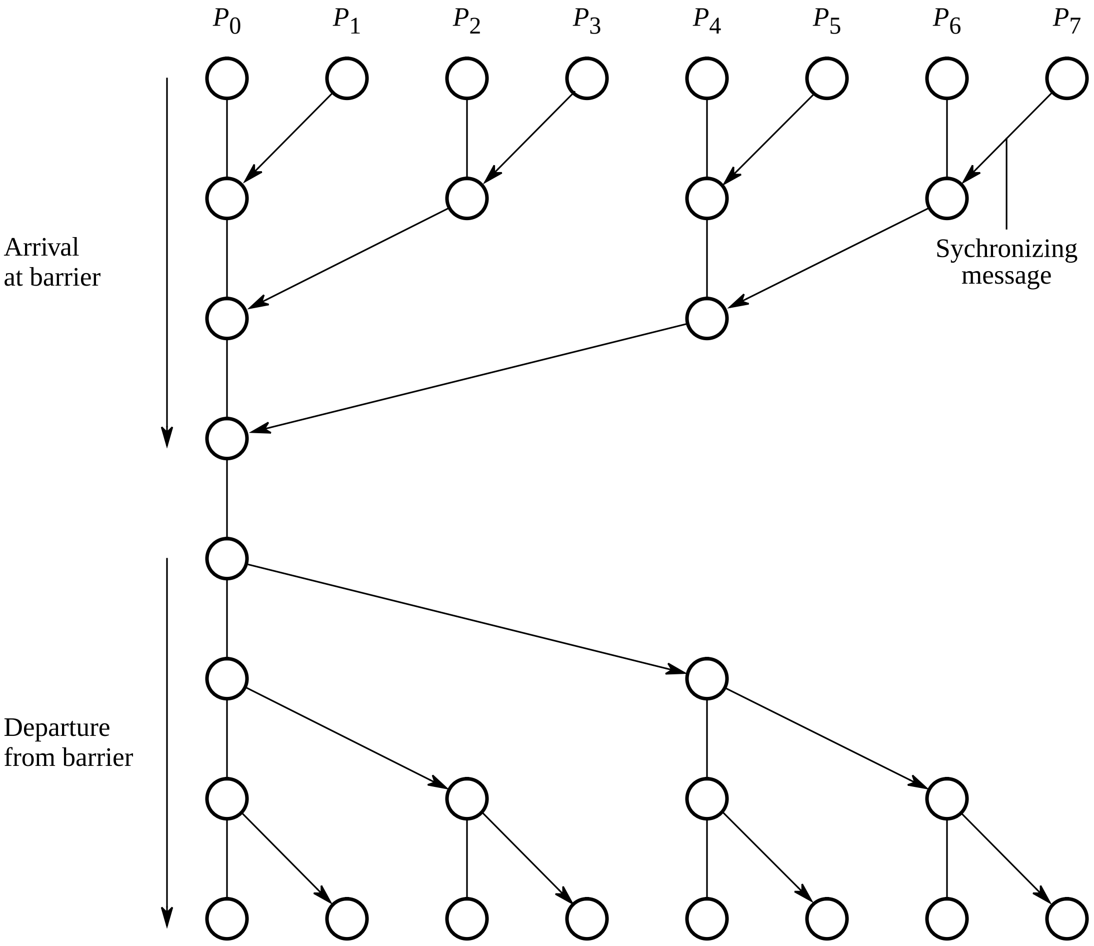


Figure 6.5 Tree barrier.

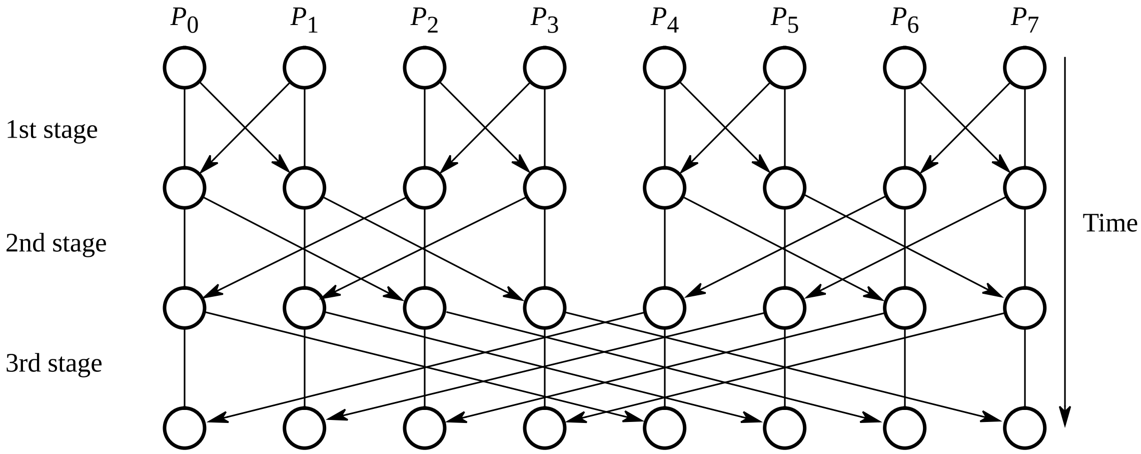


Figure 6.6 Butterfly construction.

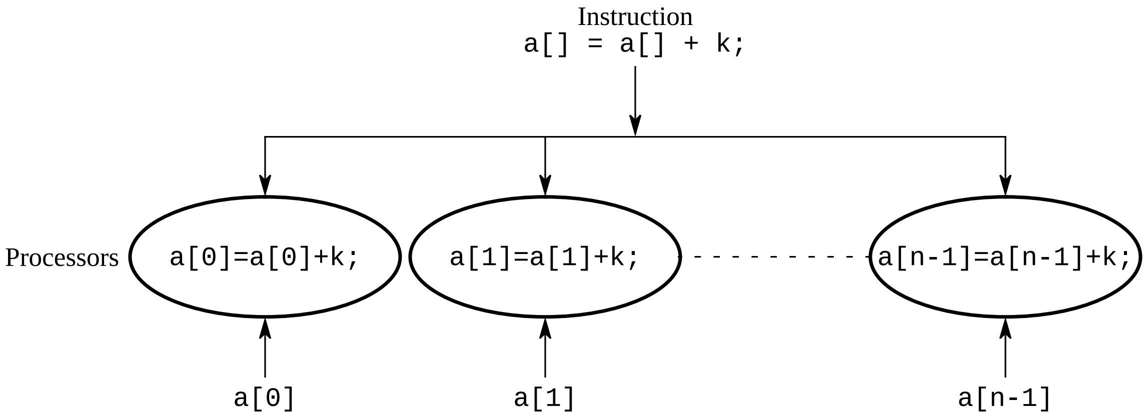


Figure 6.7 Data parallel computation.

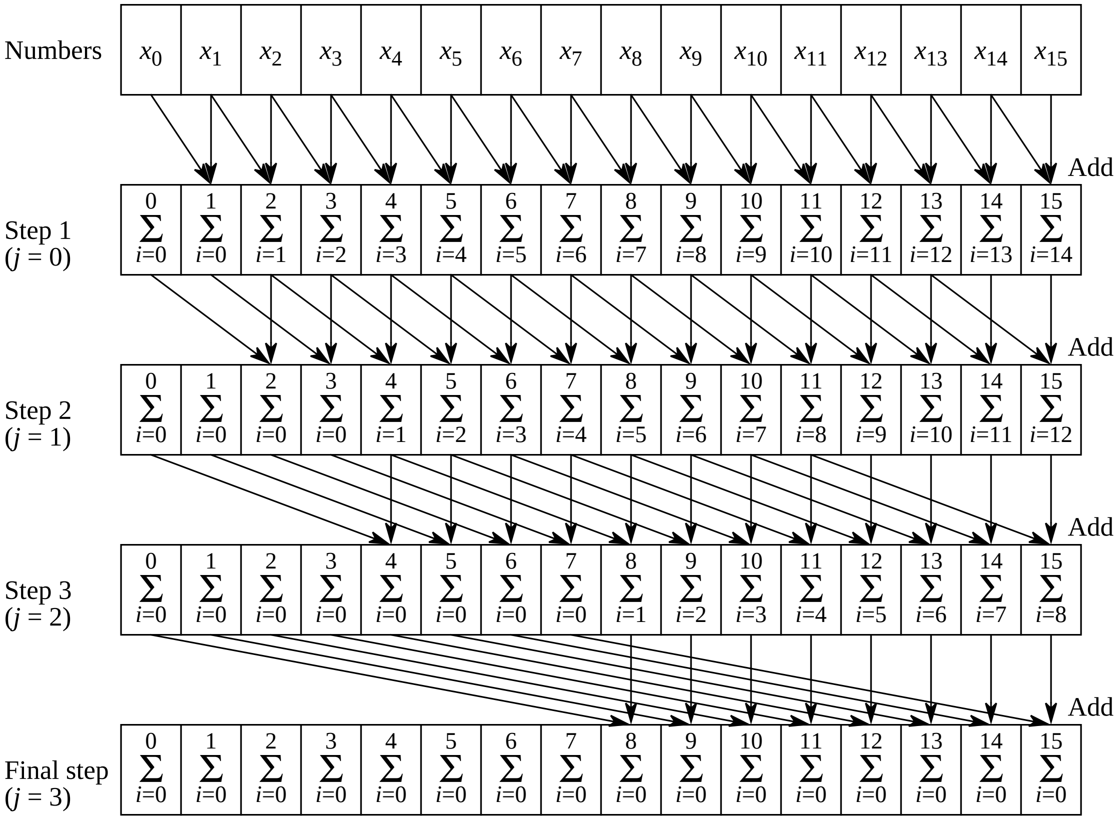

Numbers
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10} x_{11} x_{12} x_{13} x_{14} x_{15}
Add
Step 1
(j = 0)
$\sum_{i=0}^{0}$ $\sum_{i=0}^{1}$ $\sum_{i=1}^{2}$ $\sum_{i=2}^{3}$ $\sum_{i=3}^{4}$ $\sum_{i=4}^{5}$ $\sum_{i=5}^{6}$ $\sum_{i=6}^{7}$ $\sum_{i=7}^{8}$ $\sum_{i=8}^{9}$ $\sum_{i=9}^{10}$ $\sum_{i=10}^{11}$ $\sum_{i=11}^{12}$ $\sum_{i=12}^{13}$ $\sum_{i=13}^{14}$ $\sum_{i=14}^{15}$
Add
Step 2
(j = 1)
$\sum_{i=0}^{0}$ $\sum_{i=0}^{1}$ $\sum_{i=0}^{2}$ $\sum_{i=0}^{3}$ $\sum_{i=1}^{4}$ $\sum_{i=2}^{5}$ $\sum_{i=3}^{6}$ $\sum_{i=4}^{7}$ $\sum_{i=5}^{8}$ $\sum_{i=6}^{9}$ $\sum_{i=7}^{10}$ $\sum_{i=8}^{11}$ $\sum_{i=9}^{12}$ $\sum_{i=10}^{13}$ $\sum_{i=11}^{14}$ $\sum_{i=12}^{15}$
Add
Step 3
(j = 2)
$\sum_{i=0}^{0}$ $\sum_{i=0}^{1}$ $\sum_{i=0}^{2}$ $\sum_{i=0}^{3}$ $\sum_{i=0}^{4}$ $\sum_{i=0}^{5}$ $\sum_{i=0}^{6}$ $\sum_{i=0}^{7}$ $\sum_{i=1}^{8}$ $\sum_{i=2}^{9}$ $\sum_{i=3}^{10}$ $\sum_{i=4}^{11}$ $\sum_{i=5}^{12}$ $\sum_{i=6}^{13}$ $\sum_{i=7}^{14}$ $\sum_{i=8}^{15}$
Add
Final step
(j = 3)
$\sum_{i=0}^{0}$ $\sum_{i=0}^{1}$ $\sum_{i=0}^{2}$ $\sum_{i=0}^{3}$ $\sum_{i=0}^{4}$ $\sum_{i=0}^{5}$ $\sum_{i=0}^{6}$ $\sum_{i=0}^{7}$ $\sum_{i=0}^{8}$ $\sum_{i=0}^{9}$ $\sum_{i=0}^{10}$ $\sum_{i=0}^{11}$ $\sum_{i=0}^{12}$ $\sum_{i=0}^{13}$ $\sum_{i=0}^{14}$ $\sum_{i=0}^{15}$

Figure 6.8 Data parallel prefix sum operation.

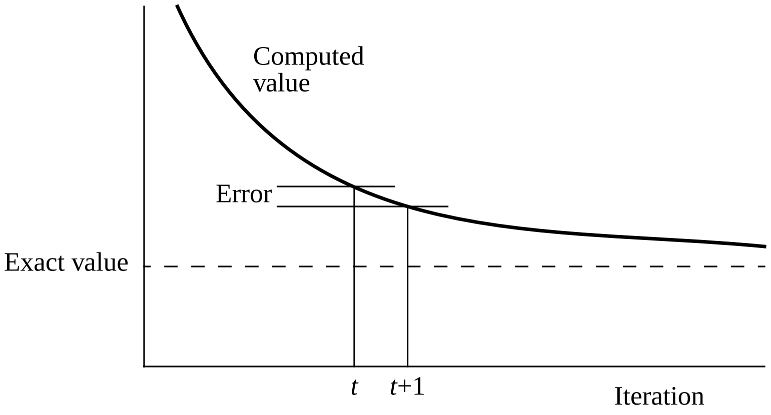


Figure 6.9 Convergence rate.

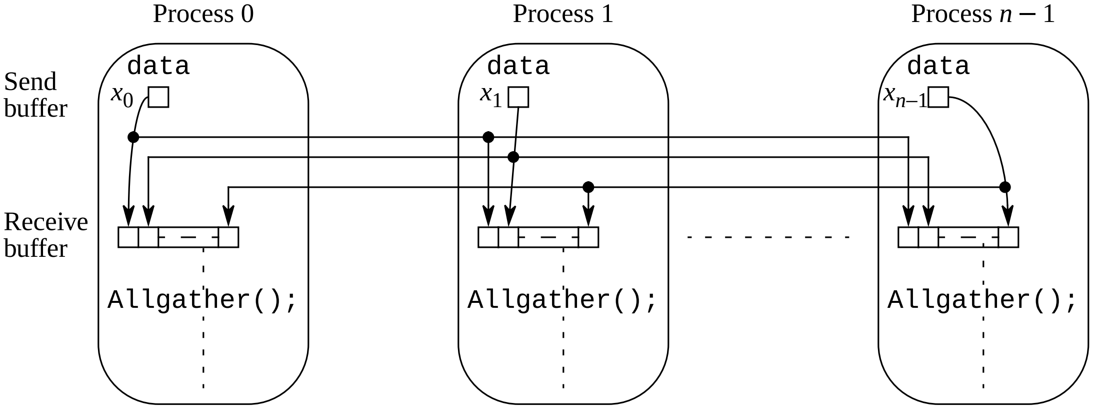


Figure 6.10 Allgather operation.

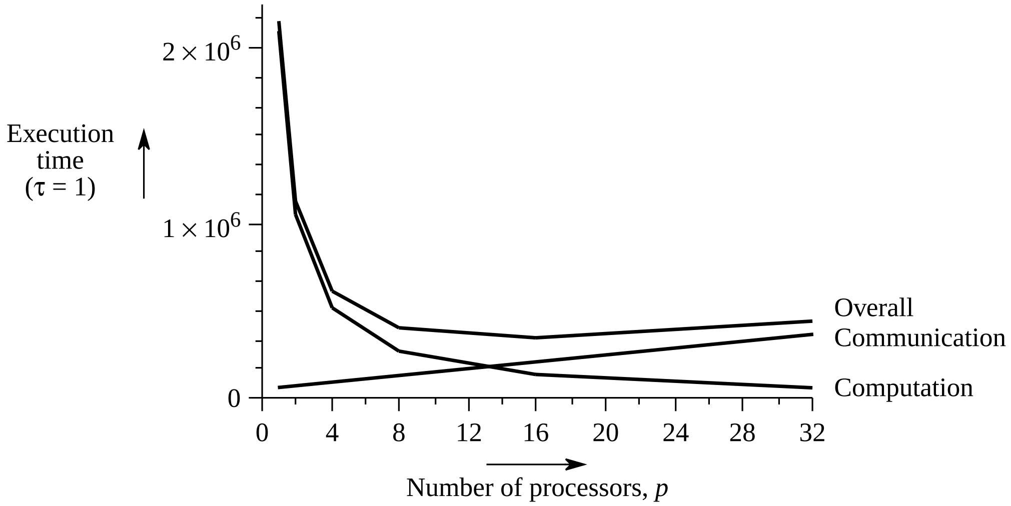


Figure 6.11 Effects of computation and communication in Jacobi iteration.

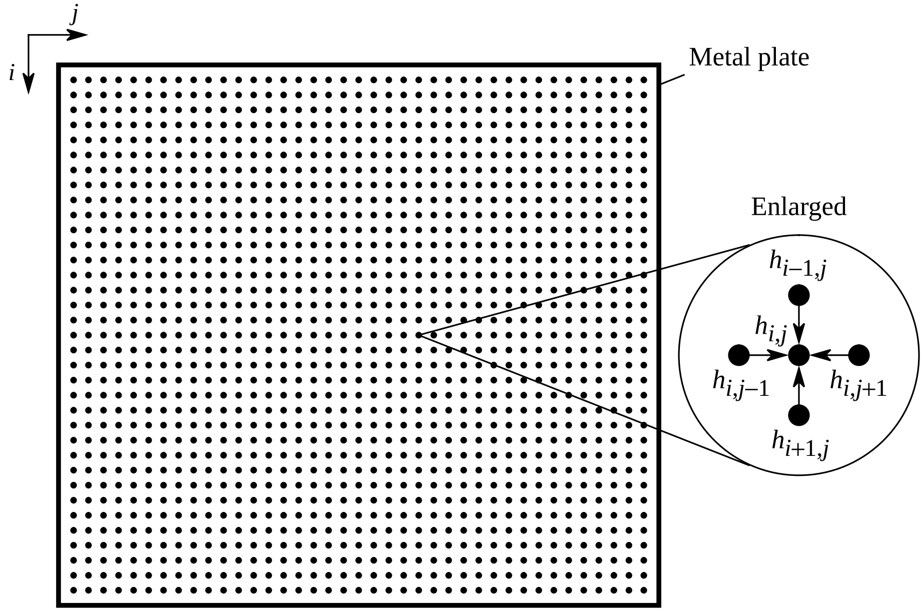


Figure 6.12 Heat distribution problem.

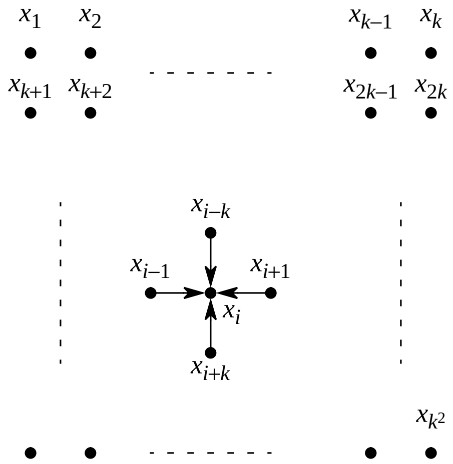


Figure 6.13 Natural ordering of heat distribution problem.

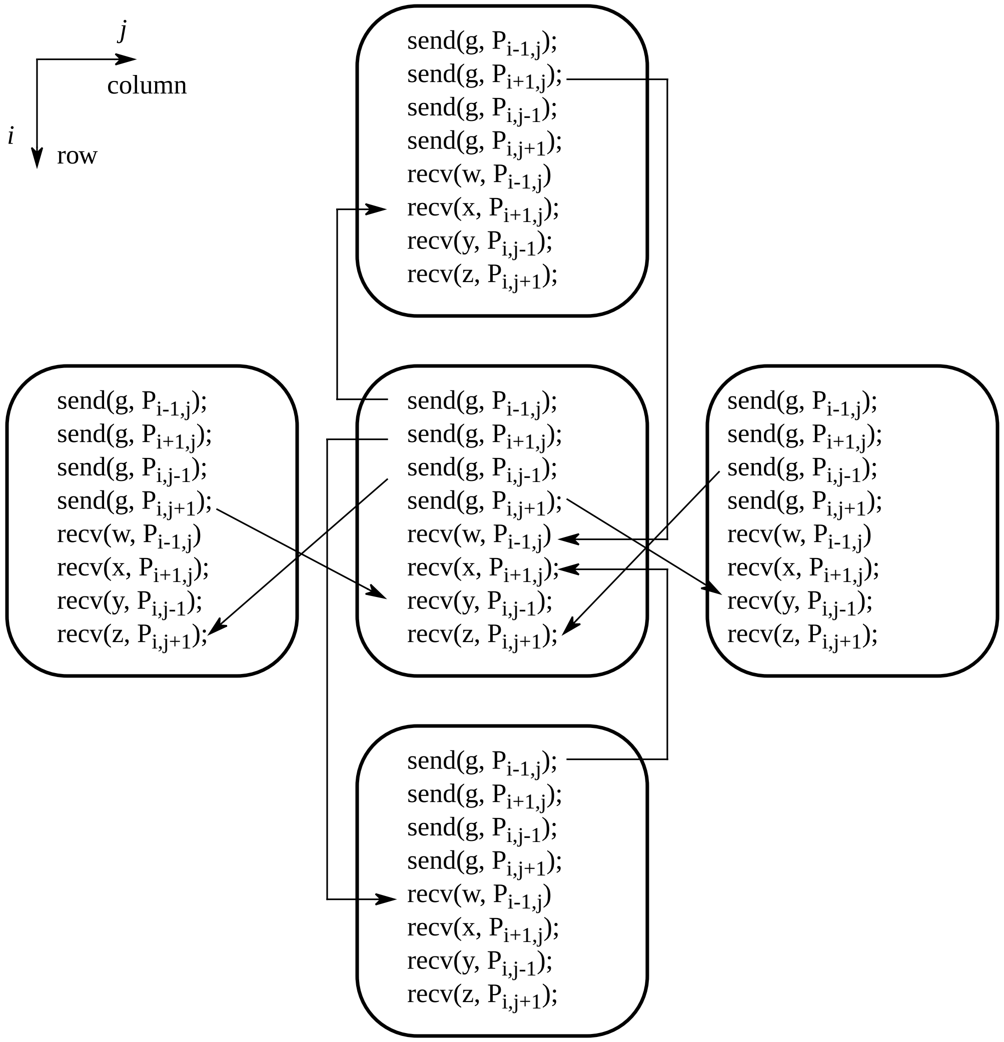


Figure 6.14 Message passing for heat distribution problem.

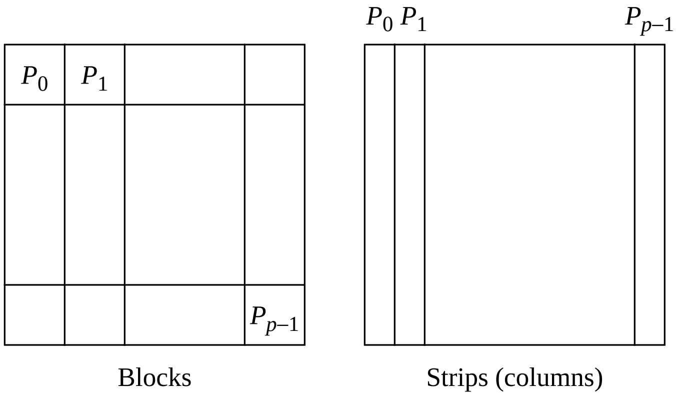


Figure 6.15 Partitioning heat distribution problem.

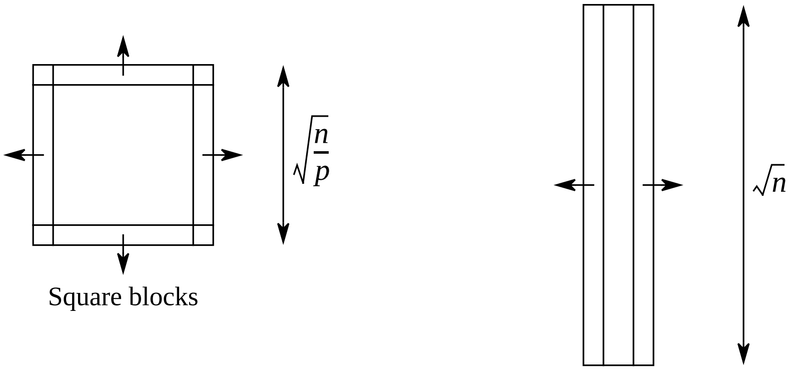


Figure 6.16 Communication consequences of partitioning.

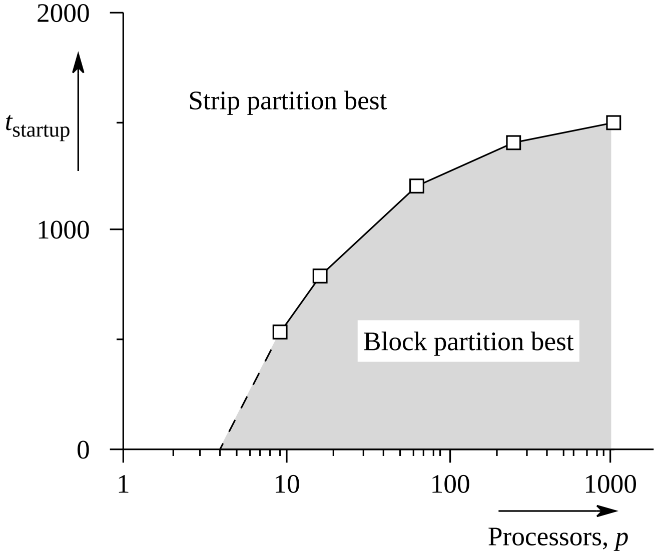


Figure 6.17 Startup times for block and strip partitions.

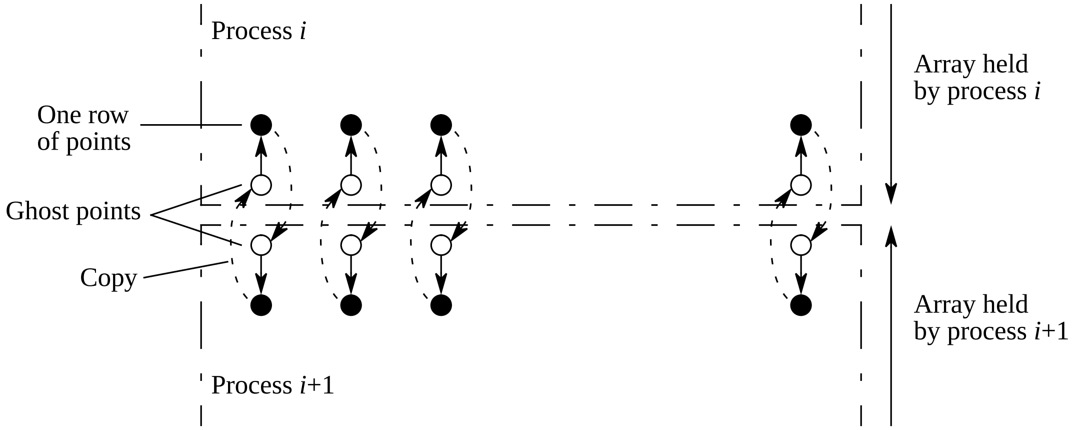


Figure 6.18 Configurating array into contiguous rows for each process, with ghost points.

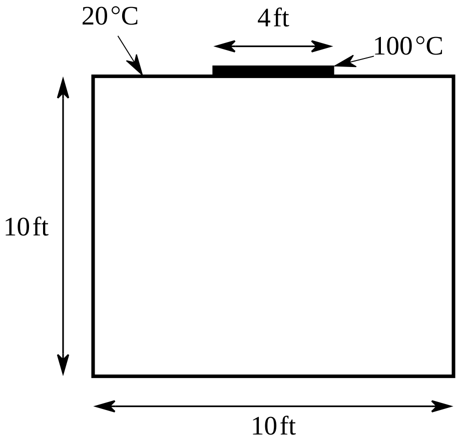


Figure 6.19 Room for Problem 6-14.

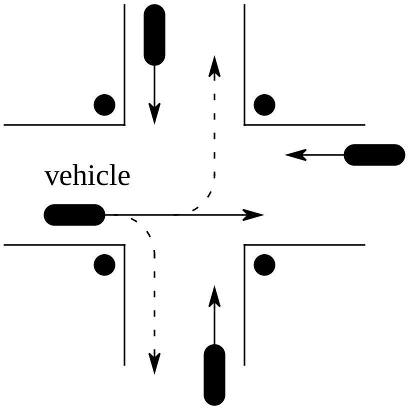


Figure 6.20 Road junction for Problem 6-16.

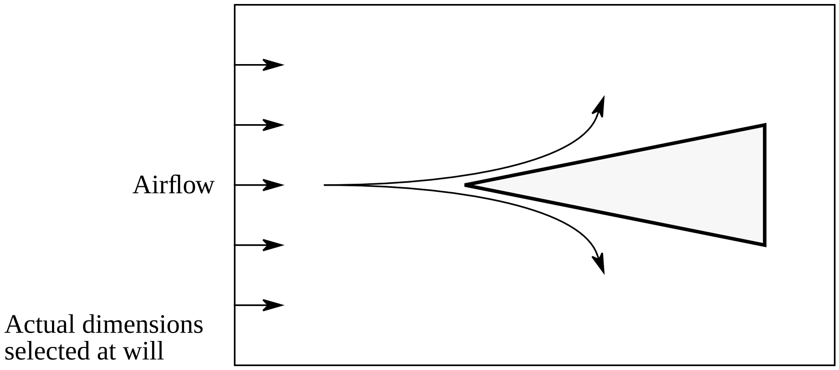
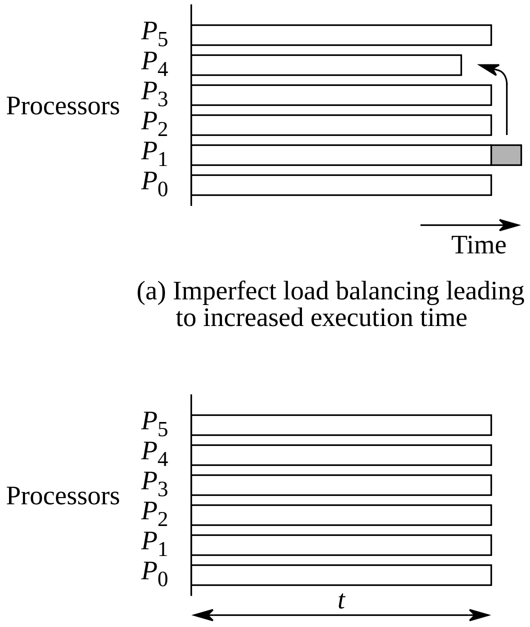


Figure 6.21 Figure for Problem 6-23.



(a) Imperfect load balancing leading to increased execution time

(b) Perfect load balancing

Figure 7.1 Load balancing.

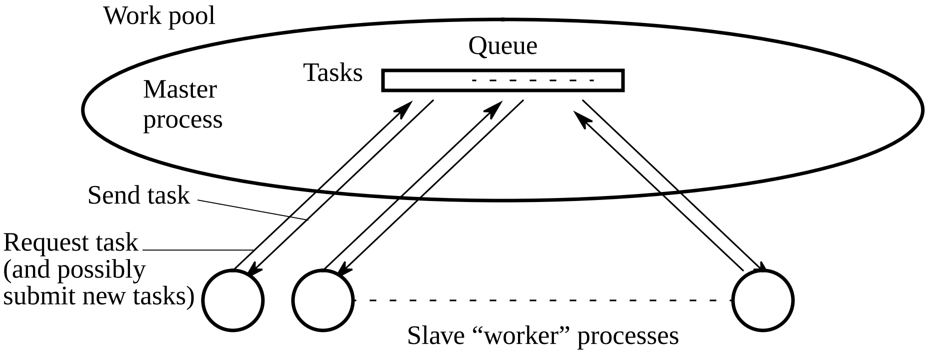


Figure 7.2 Centralized work pool.

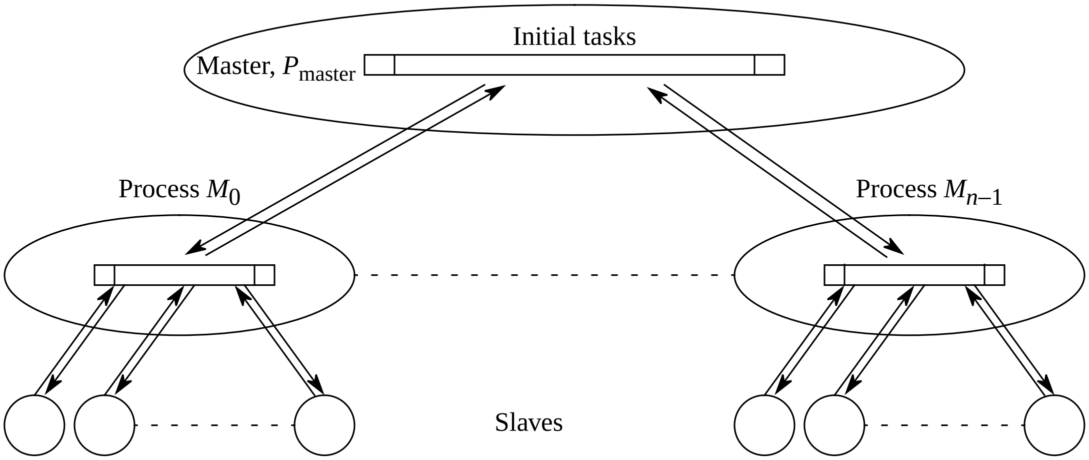


Figure 7.3 A distributed work pool.

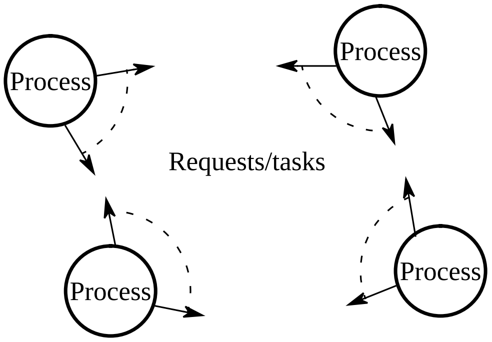


Figure 7.4 Decentralized work pool.

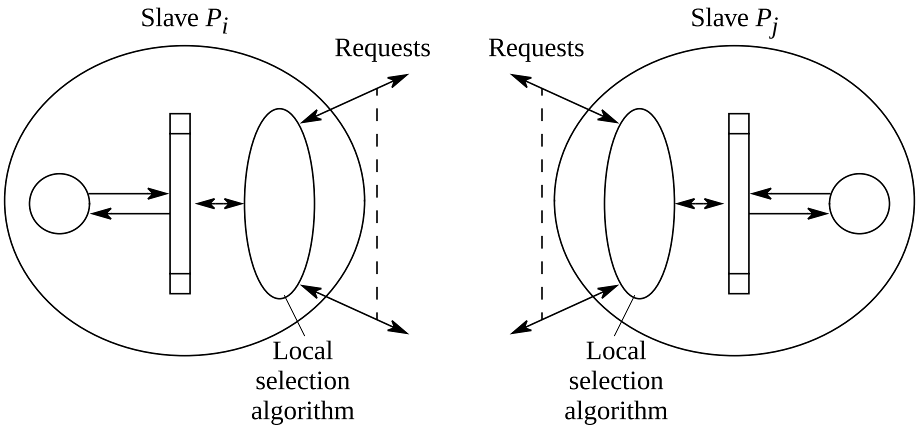


Figure 7.5 Decentralized selection algorithm requesting tasks between slaves.

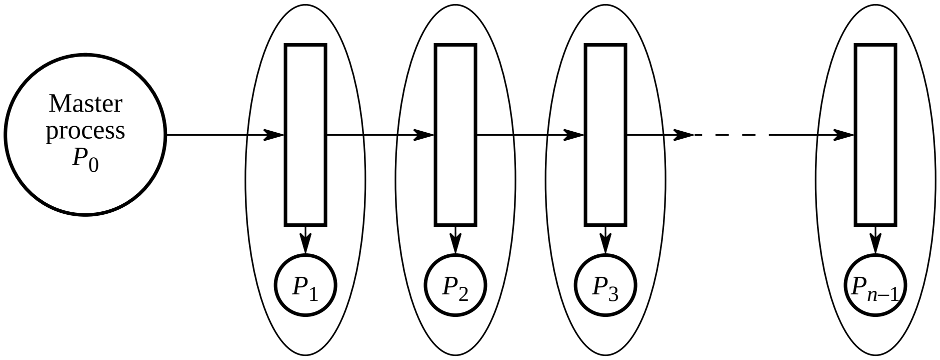
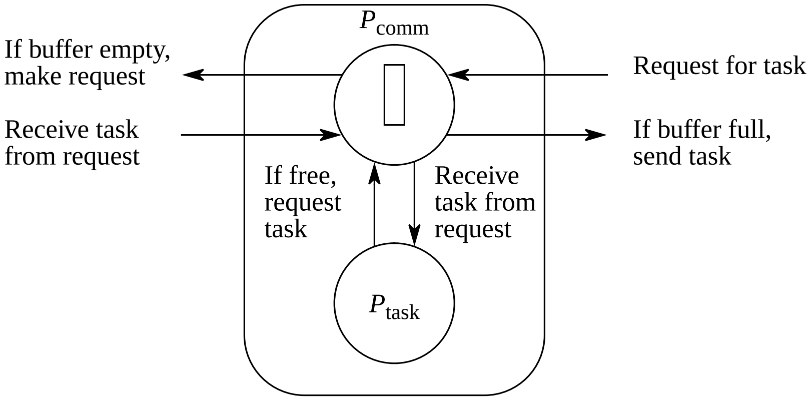


Figure 7.6 Load balancing using a pipeline structure.



P_{comm}
If buffer empty, make request
Receive task from request
Request for task
If buffer full, send task
If free, request task
Receive task from request
P_{task}

Figure 7.7 Using a communication process in line load balancing.

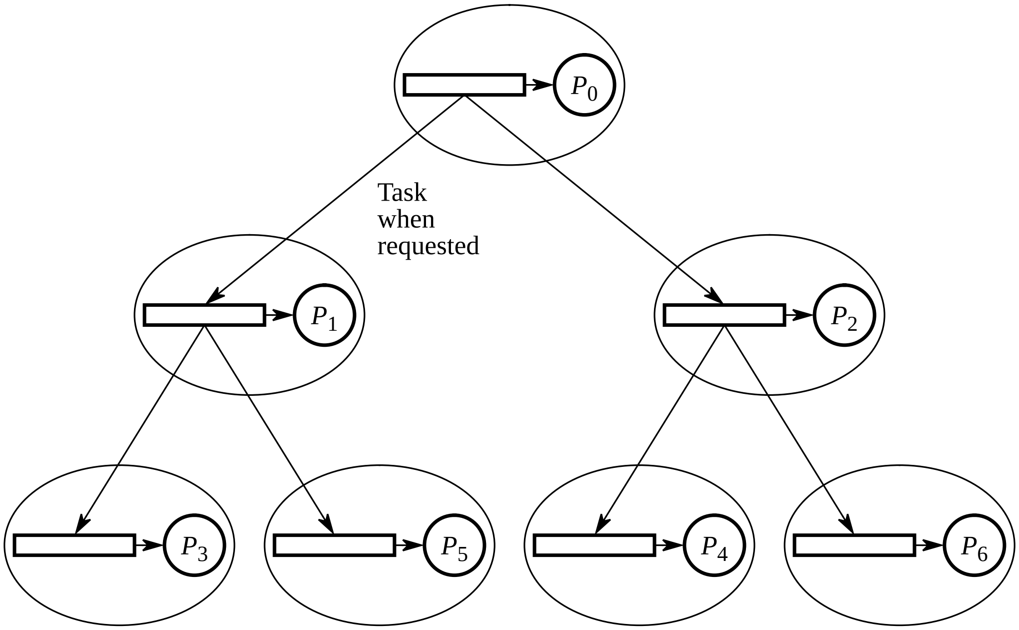


Figure 7.8 Load balancing using a tree.

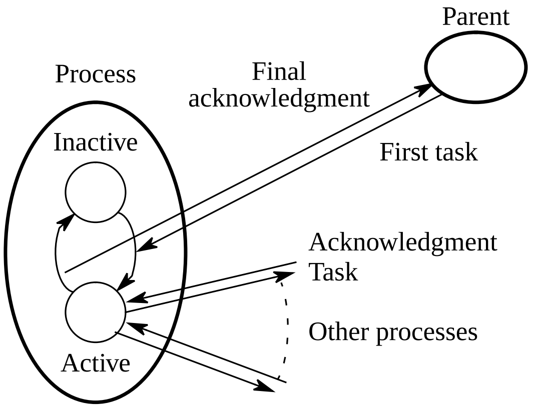


Figure 7.9 Termination using message acknowledgments.

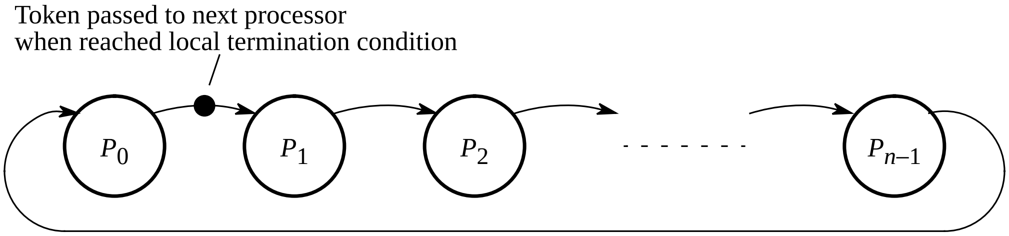

Token passed to next processor
when reached local termination condition
P_0
P_1
P_2
P_{n-1}

Figure 7.10 Ring termination detection algorithm.

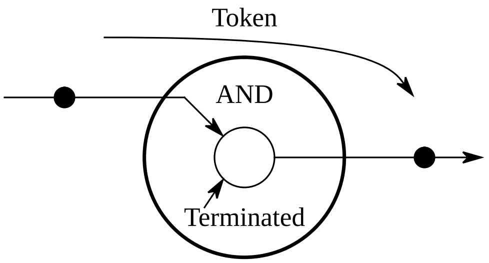


Figure 7.11 Process algorithm for local termination.

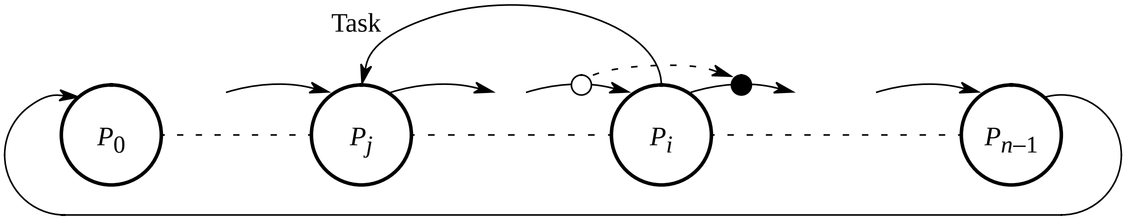


Figure 7.12 Passing task to previous processes.

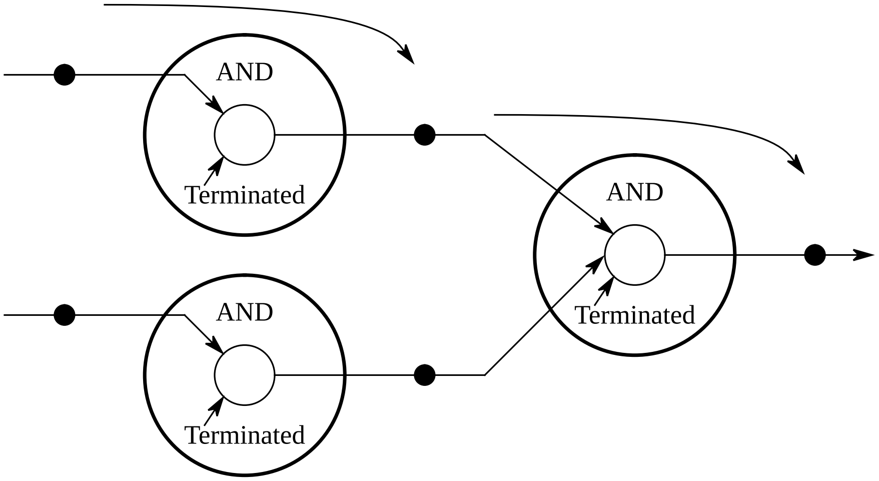


Figure 7.13 Tree termination.

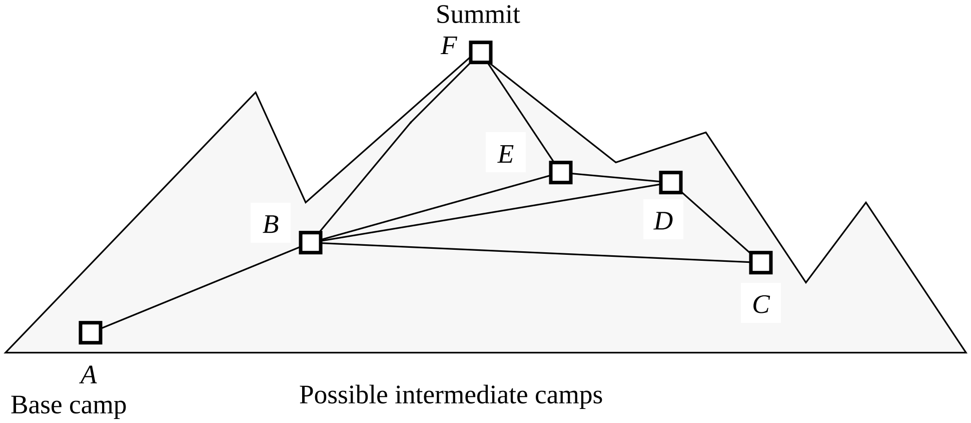


Figure 7.14 Climbing a mountain.

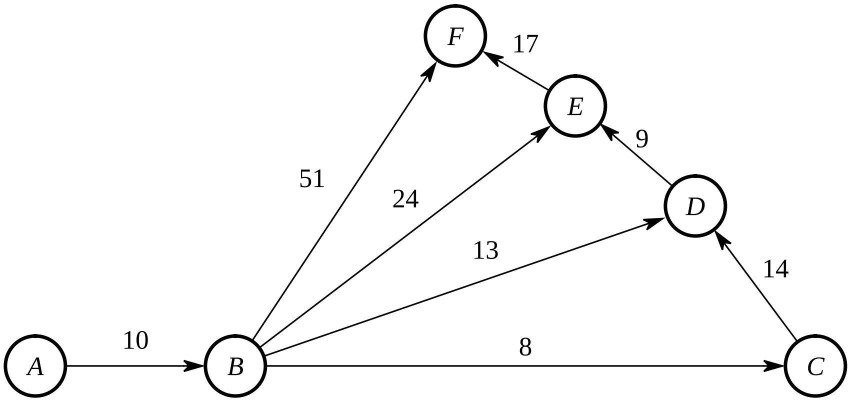
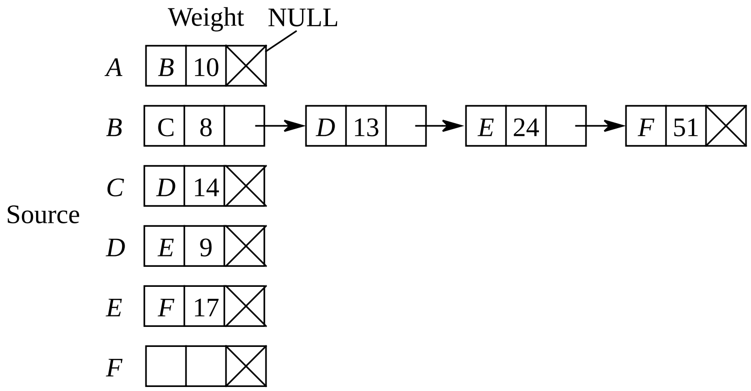


Figure 7.15 Graph of mountain climb.

		Destination					
		A	*B*	*C*	*D*	*E*	*F*
Source	*A*	∞	10	∞	∞	∞	∞
	B	∞	∞	8	13	24	51
	C	∞	∞	∞	14	∞	∞
	D	∞	∞	∞	∞	9	∞
	E	∞	∞	∞	∞	∞	17
	F	∞	∞	∞	∞	∞	∞

(a) Adjacency matrix



(b) Adjacency list

Figure 7.16 Representing a graph.

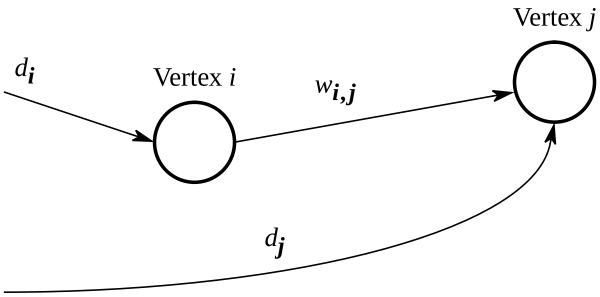


Figure 7.17 Moore's shortest-path algorithm.

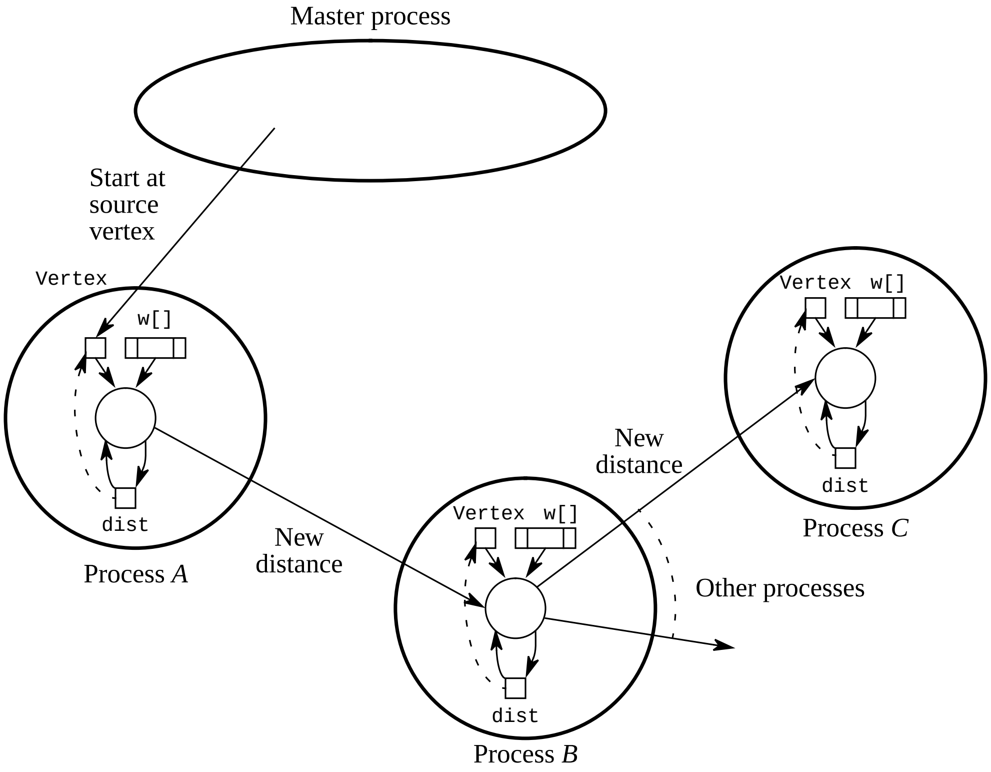


Figure 7.18 Distributed graph search.

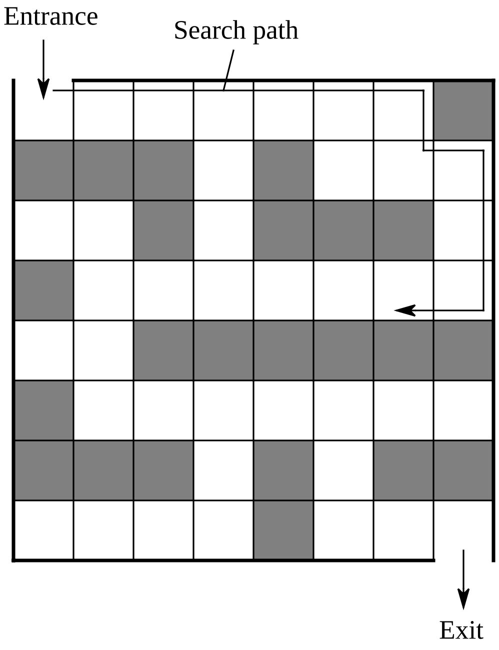


Figure 7.19 Sample maze for Problem 7-9.

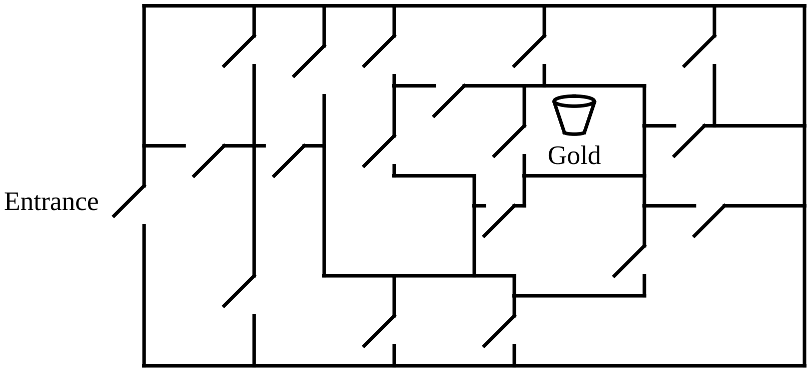


Figure 7.20 Plan of rooms for Problem 7-10.

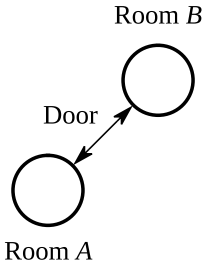


Figure 7.21 Graph representation for Problem 7-10.

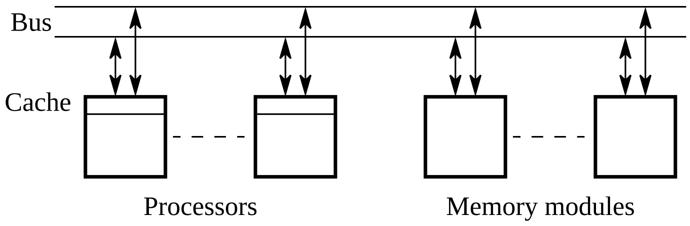


Figure 8.1 Shared memory multiprocessor using a single bus.

TABLE 8.1 SOME EARLY PARALLEL PROGRAMMING LANGUAGES

Language	Originator/date	Comments
Concurrent Pascal	Brinch Hansen, 1975[a]	Extension to Pascal
Ada	U.S. Dept. of Defense, 1979[b]	Completely new language
Modula-P	Bräunl, 1986[c]	Extension to Modula 2
C*	Thinking Machines, 1987[d]	Extension to C for SIMD systems
Concurrent C	Gehani and Roome, 1989[e]	Extension to C
Fortran D	Fox et al., 1990[f]	Extension to Fortran for data parallel programming

a. Brinch Hansen, P. (1975), "The Programming Language Concurrent Pascal," *IEEE Trans. Software Eng.*, Vol. 1, No. 2 (June), pp. 199–207.

b. U.S. Department of Defense (1981), "The Programming Language Ada Reference Manual," *Lecture Notes in Computer Science*, No. 106, Springer-Verlag, Berlin.

c. Bräunl, T., R. Norz (1992), *Modula-P User Manual*, Computer Science Report, No. 5/92 (August), Univ. Stuttgart, Germany.

d. Thinking Machines Corp. (1990), *C* Programming Guide, Version 6*, Thinking Machines System Documentation.

e. Gehani, N., and W. D. Roome (1989), *The Concurrent C Programming Language*, Silicon Press, New Jersey.

f. Fox, G., S. Hiranandani, K. Kennedy, C. Koelbel, U. Kremer, C. Tseng, and M. Wu (1990), *Fortran D Language Specification*, Technical Report TR90-141, Dept. of Computer Science, Rice University.

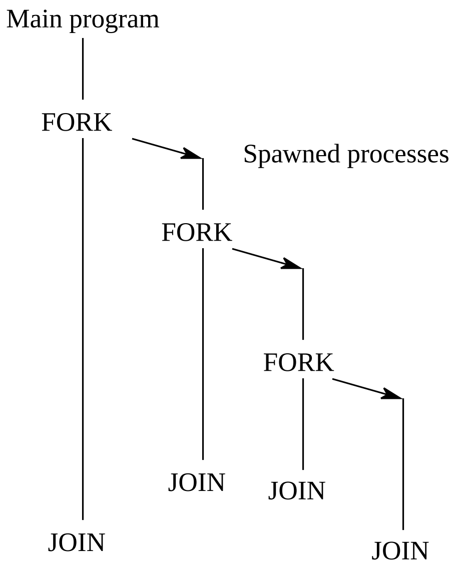
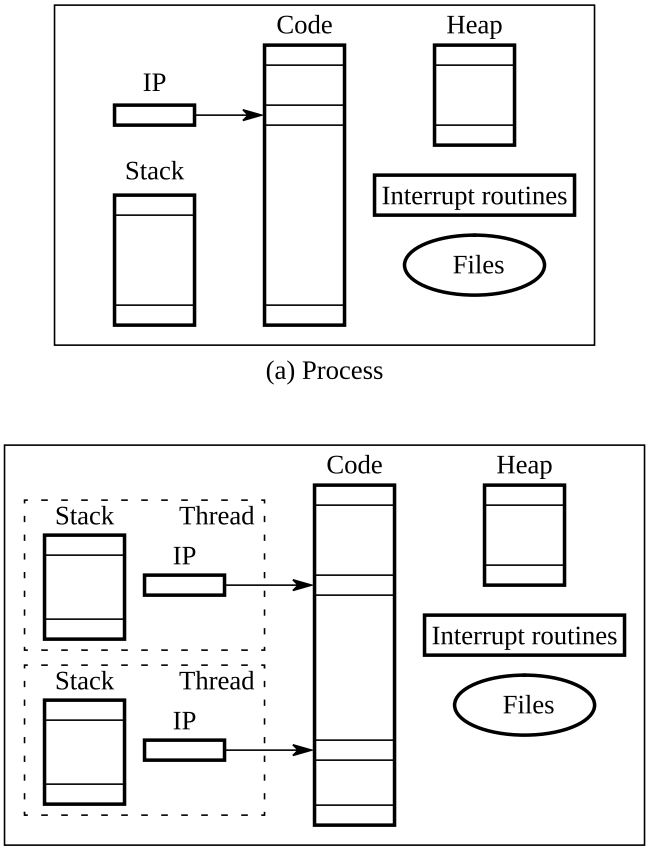


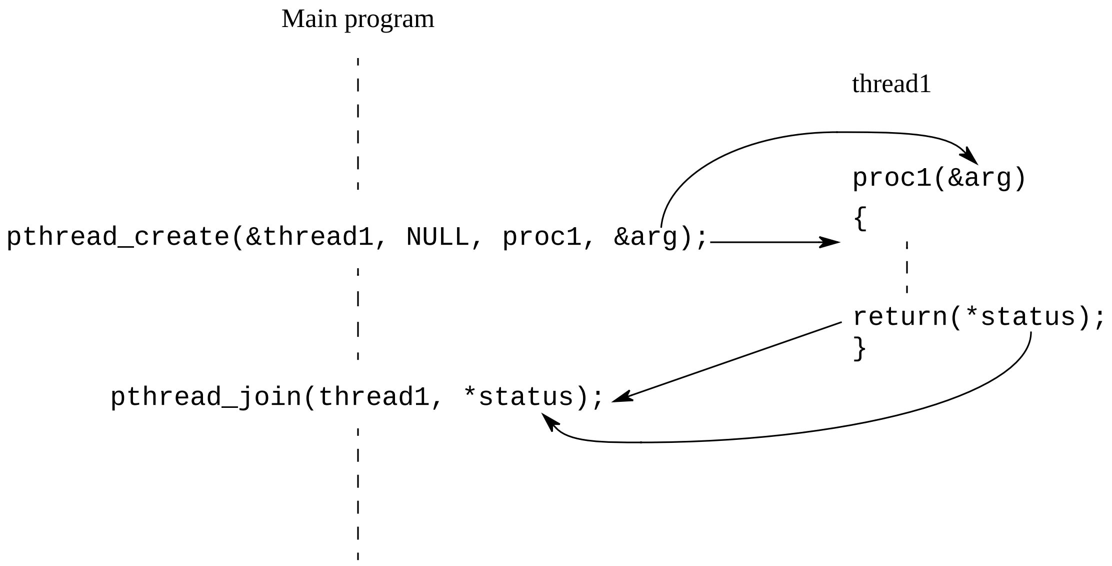
Figure 8.2 FORK-JOIN construct.



(a) Process

(b) Threads

Figure 8.3 Differences between a process and threads.



Main program

thread1

```
pthread_create(&thread1, NULL, proc1, &arg);
```

```
proc1(&arg)
{

return(*status);
}
```

```
pthread_join(thread1, *status);
```

Figure 8.4 `pthread_create()` and `pthread_join()`.

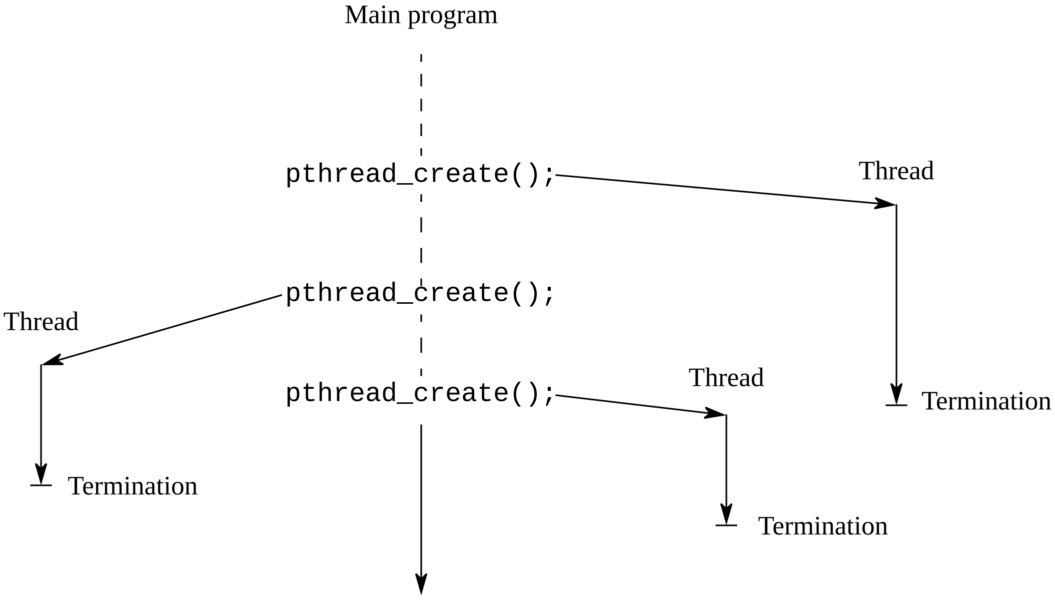


Figure 8.5 Detached threads.

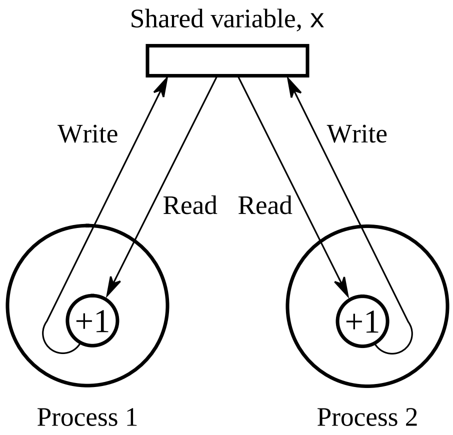
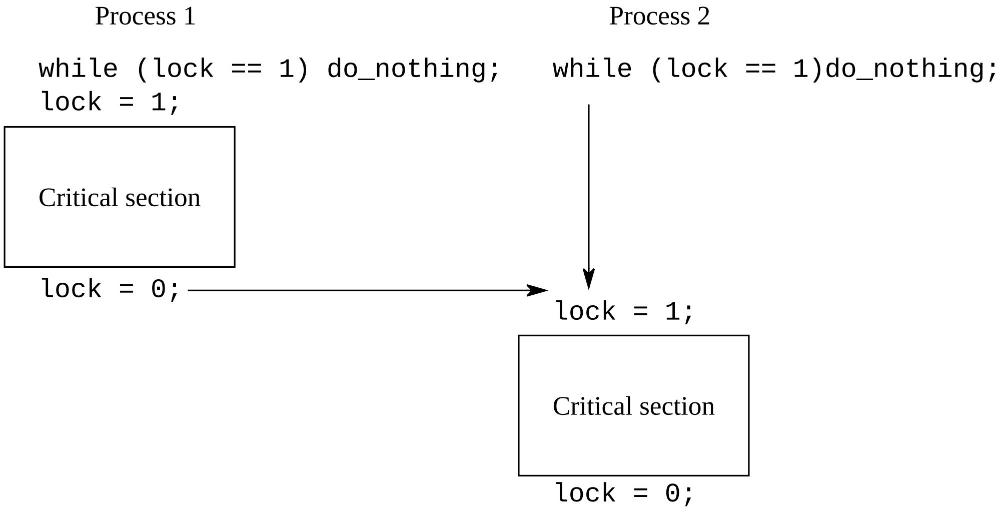


Figure 8.6 Conflict in accessing shared variable.



Process 1

```
while (lock == 1) do_nothing;
lock = 1;
```

Critical section

```
lock = 0;
```

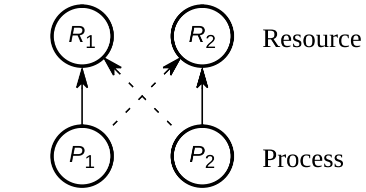
Process 2

```
while (lock == 1)do_nothing;
lock = 1;
```

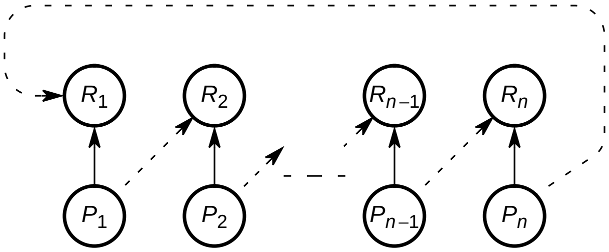
Critical section

```
lock = 0;
```

Figure 8.7 Control of critical sections through busy waiting.



(a) Two-process deadlock



(b) *n*-process deadlock

Figure 8.8 Deadlock (deadly embrace).

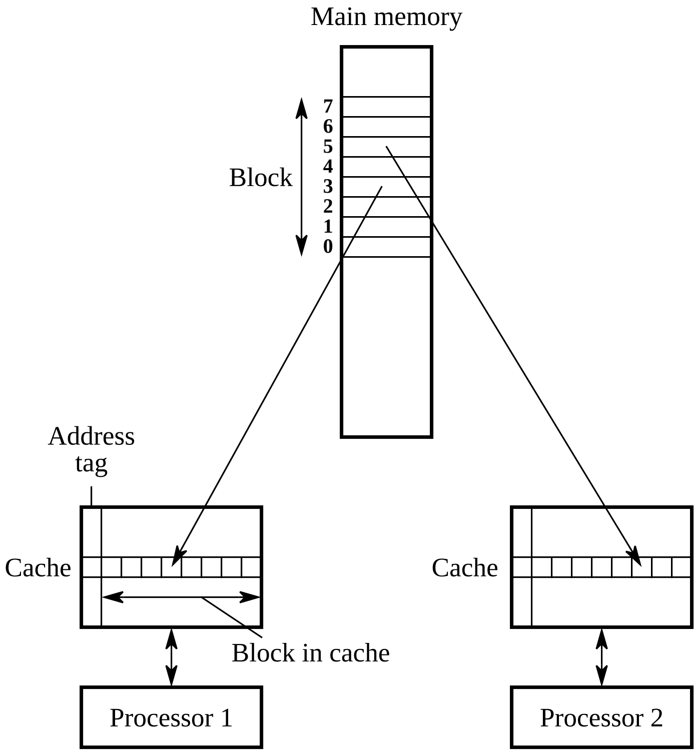


Figure 8.9 False sharing in caches.

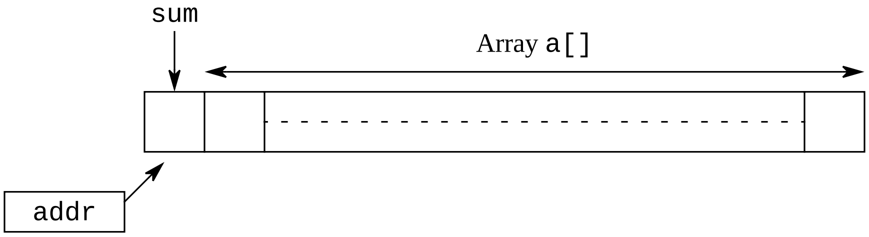


Figure 8.10 Shared memory locations for Section 8.4.1 program example.

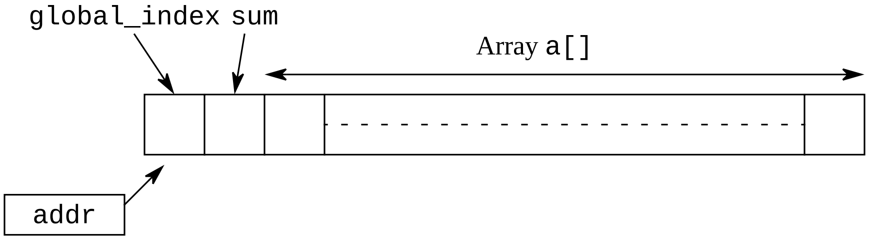


Figure 8.11 Shared memory locations for Section 8.4.2 program example.

TABLE 8.2 LOGIC CIRCUIT DESCRIPTION FOR FIGURE 8.12

Gate	Function	Input 1	Input 2	Output
1	AND	Test1	Test2	Gate1
2	NOT	Gate1		Output1
3	OR	Test3	Gate1	Output2

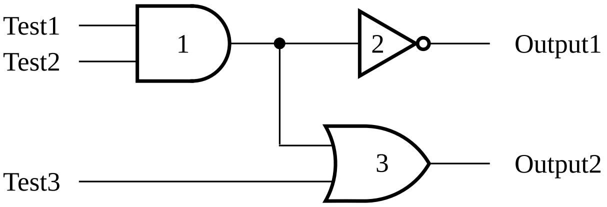


Figure 8.12 Sample logic circuit.

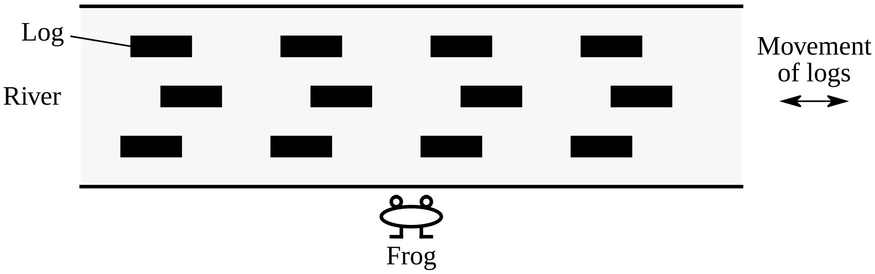


Figure 8.13 River and frog for Problem 8-23.

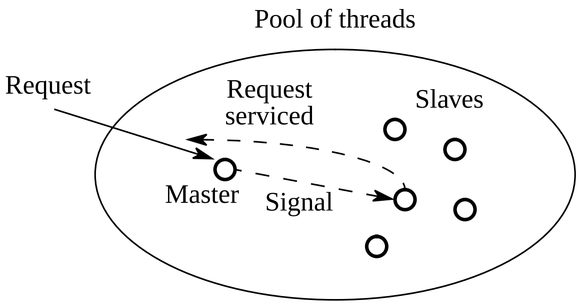


Figure 8.14 Thread pool for Problem 8-24.

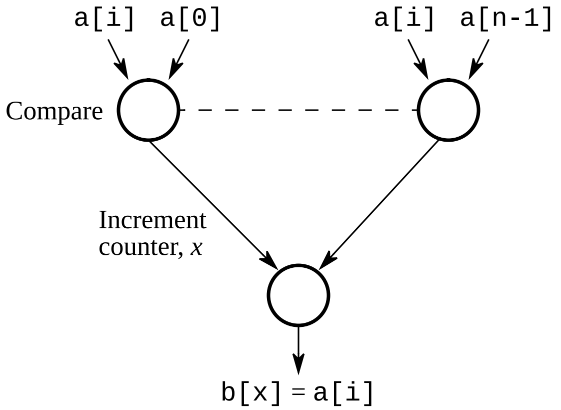


Figure 9.1 Finding the rank in parallel.

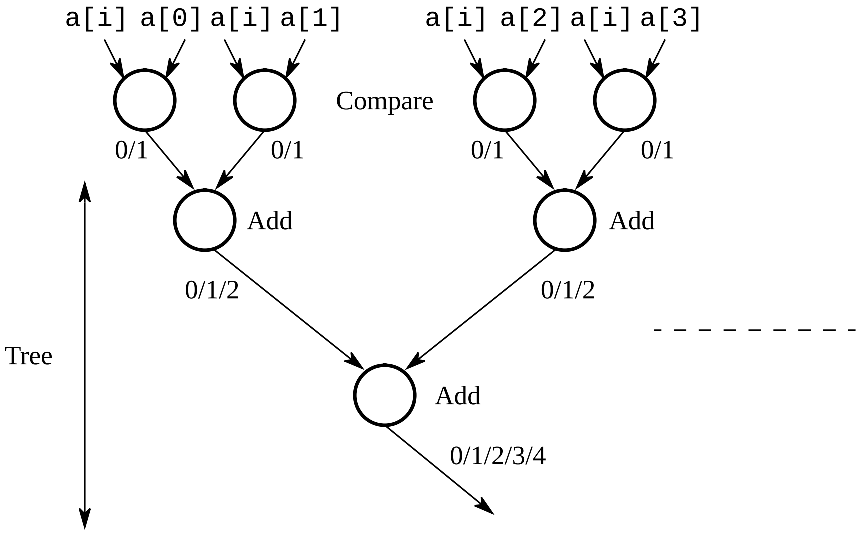


Figure 9.2 Parallelizing the rank computation.

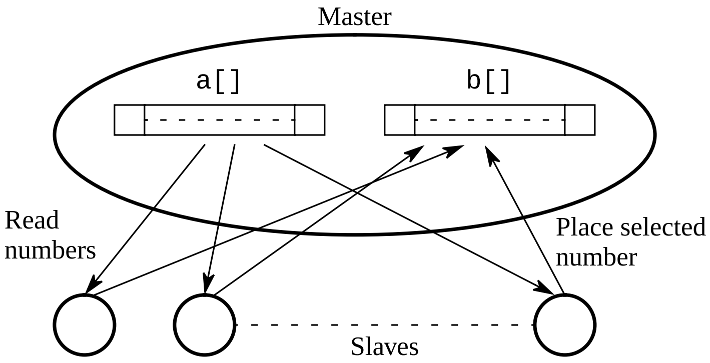


Figure 9.3 Rank sort using a master and slaves.

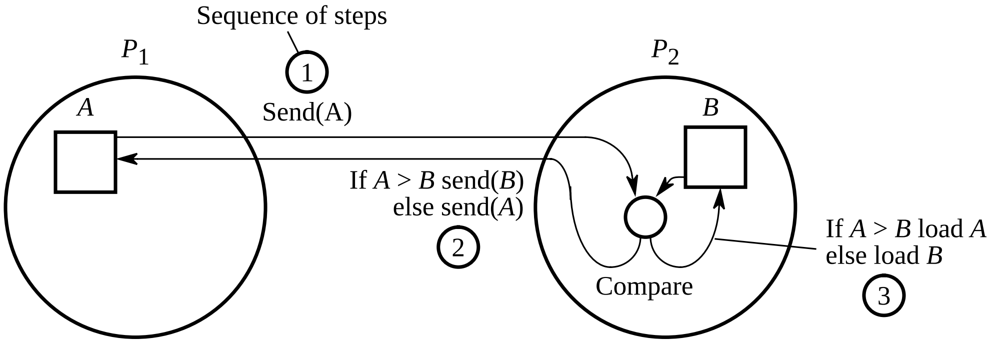


Figure 9.4 Compare and exchange on a message-passing system — Version 1.

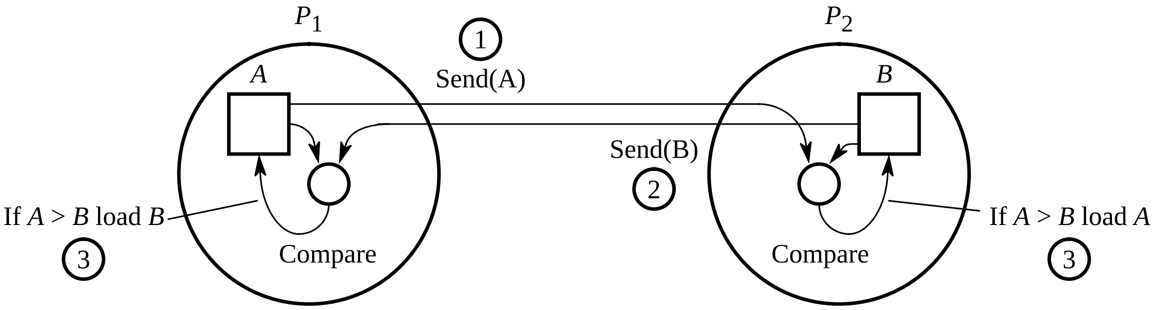


Figure 9.5 Compare and exchange on a message-passing system — Version 2.

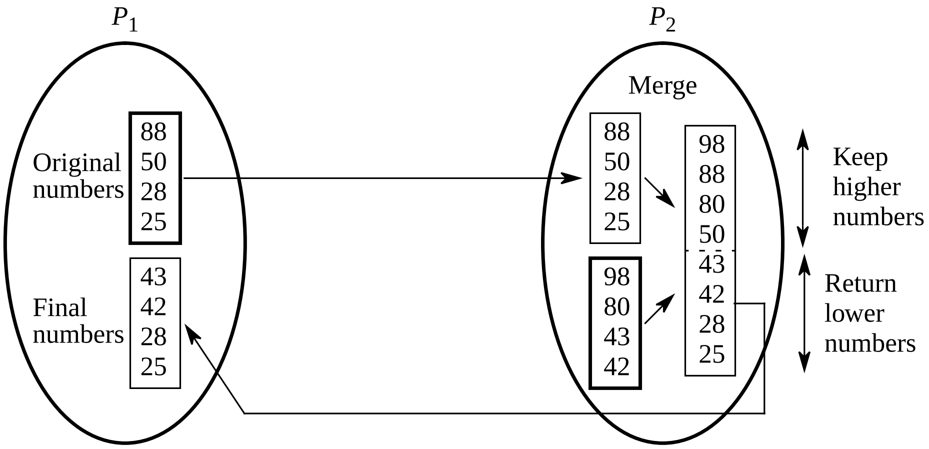


Figure 9.6 Merging two sublists — Version 1.

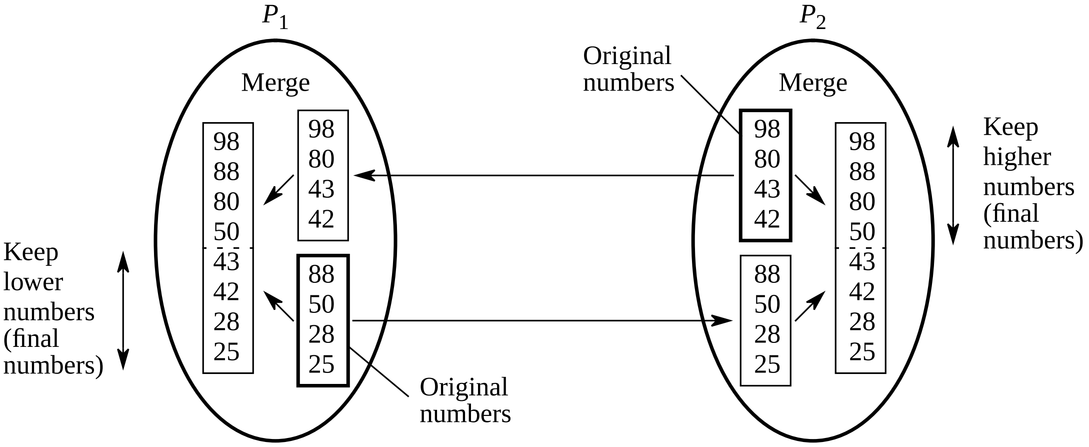


Figure 9.7 Merging two sublists — Version 2.

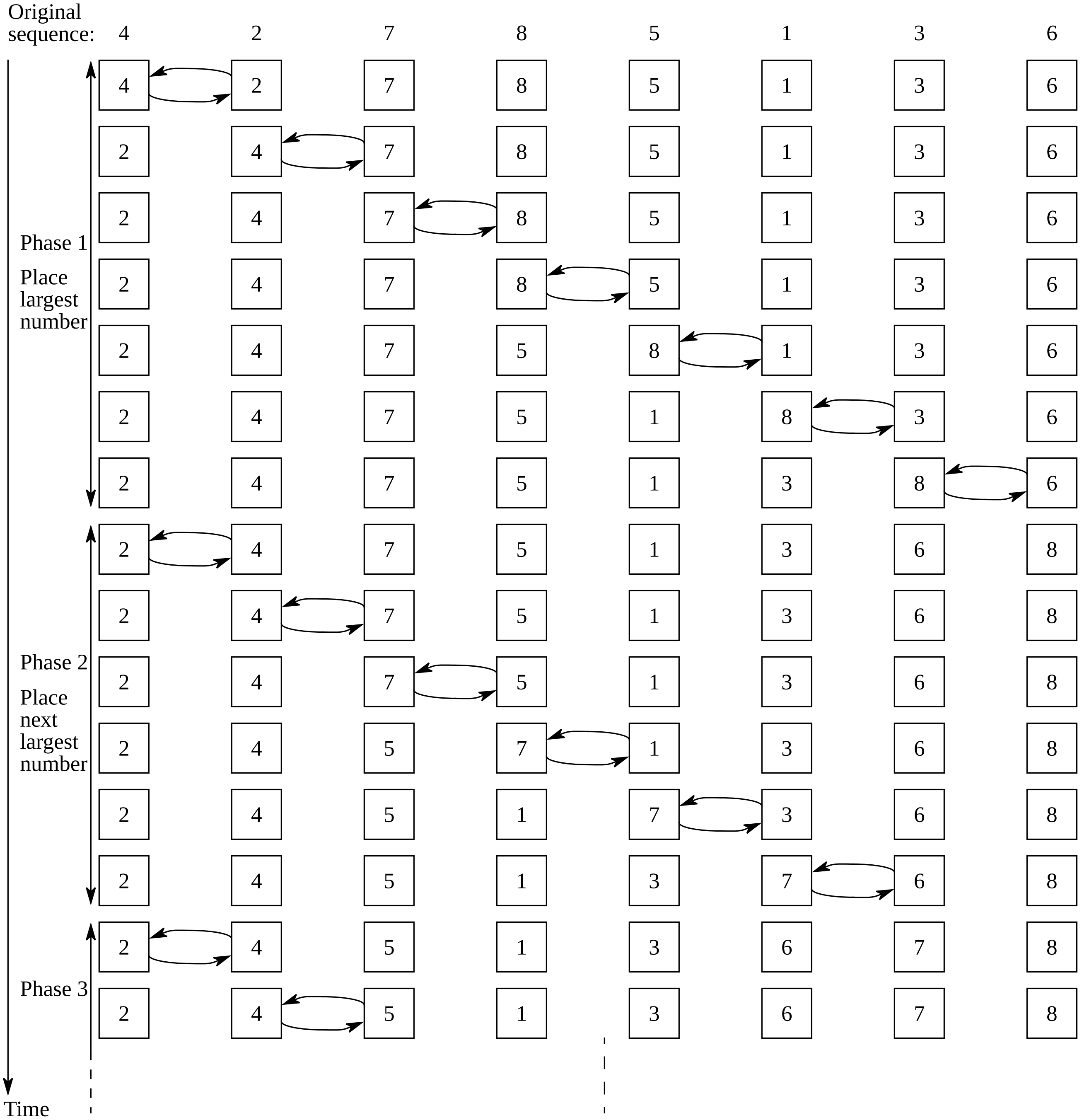


Figure 9.8 Steps in bubble sort.

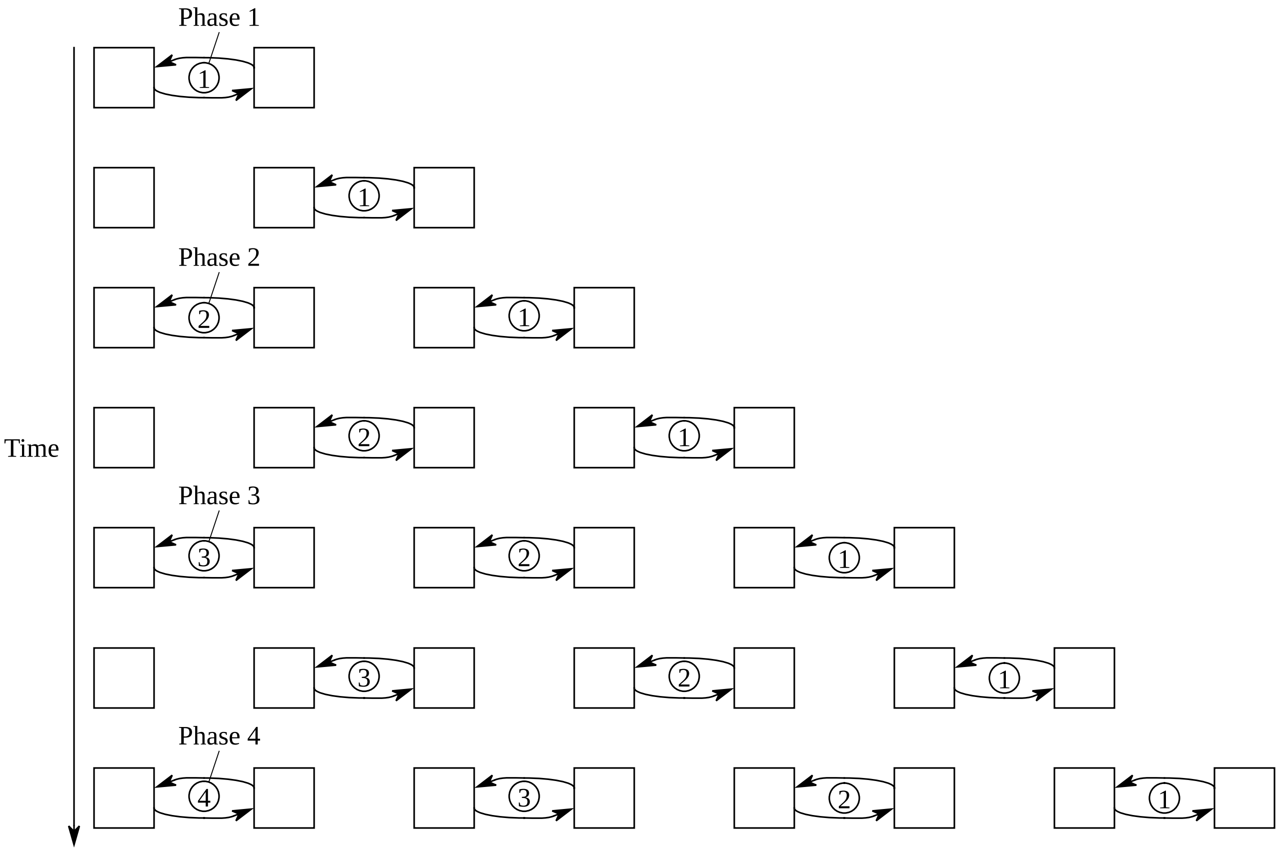
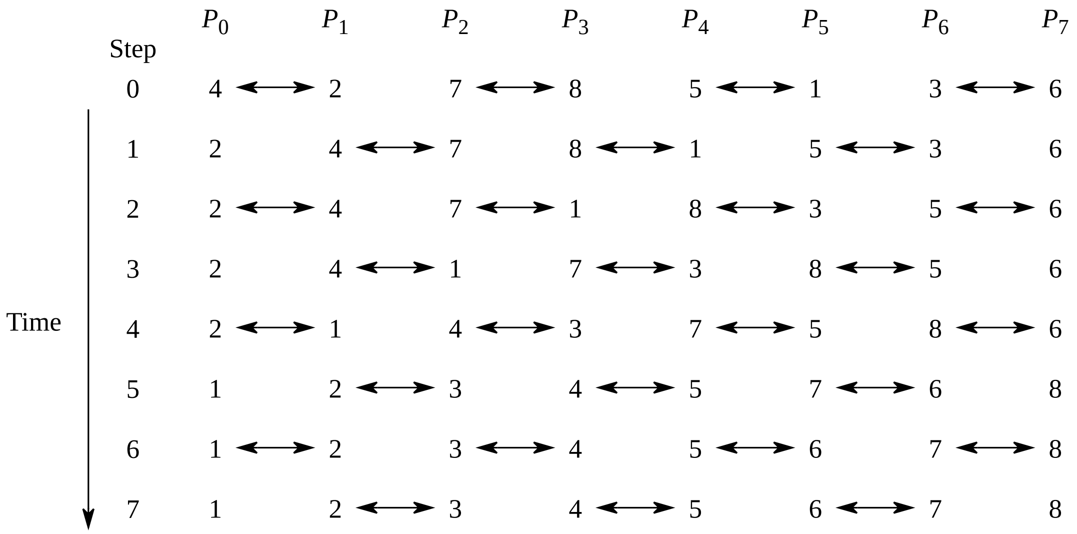


Figure 9.9 Overlapping bubble sort actions in a pipeline.



Step	P_0		P_1		P_2		P_3		P_4		P_5		P_6		P_7
0	4	↔	2		7	↔	8		5	↔	1		3	↔	6
1	2		4	↔	7		8	↔	1		5	↔	3		6
2	2	↔	4		7	↔	1		8	↔	3		5	↔	6
3	2		4	↔	1		7	↔	3		8	↔	5		6
4	2	↔	1		4	↔	3		7	↔	5		8	↔	6
5	1		2	↔	3		4	↔	5		7	↔	6		8
6	1	↔	2		3	↔	4		5	↔	6		7	↔	8
7	1		2	↔	3		4	↔	5		6	↔	7		8

Figure 9.10 Odd-even transposition sort sorting eight numbers.

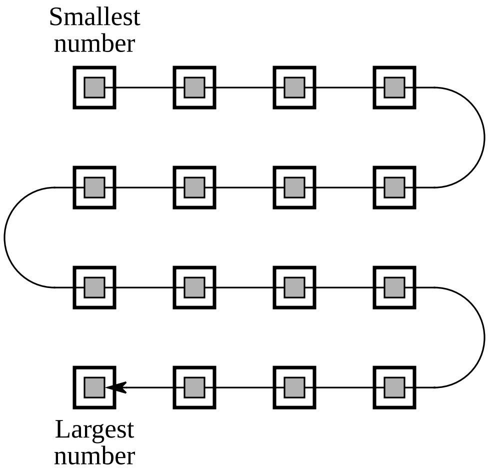
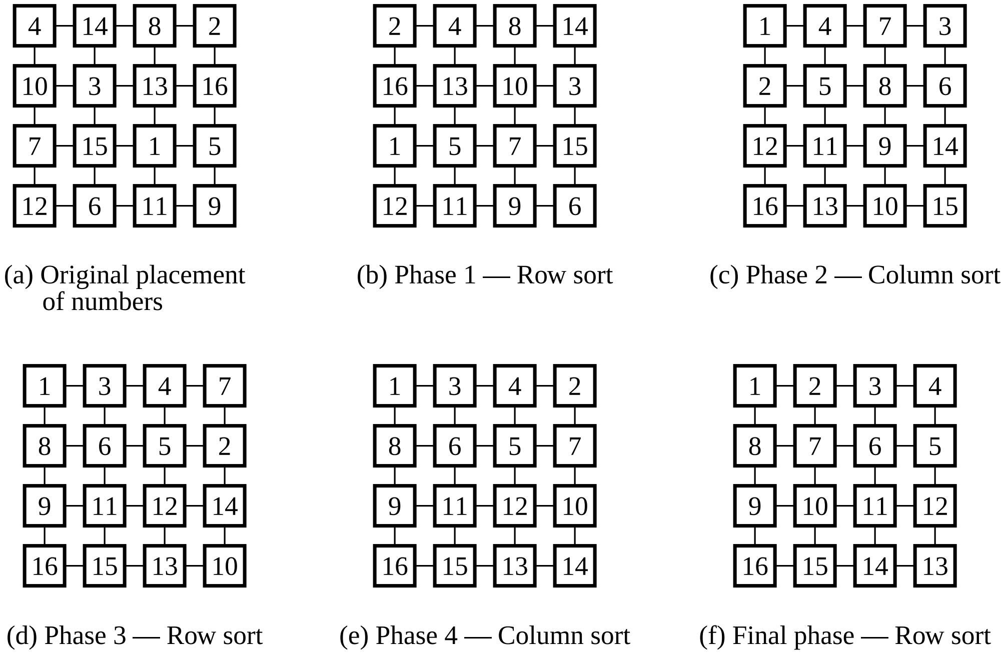


Figure 9.11 Snakelike sorted list.



(a) Original placement of numbers

(b) Phase 1 — Row sort

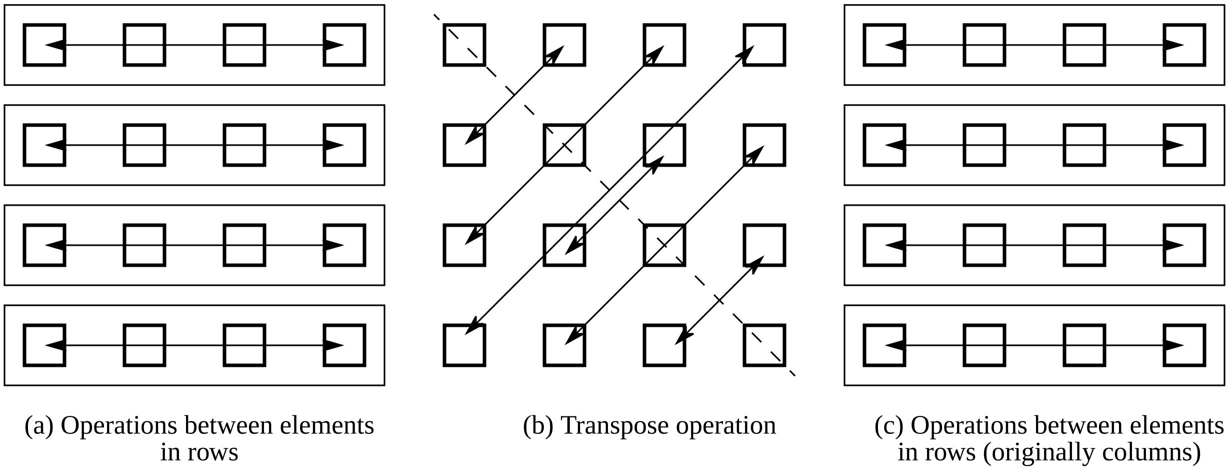
(c) Phase 2 — Column sort

(d) Phase 3 — Row sort

(e) Phase 4 — Column sort

(f) Final phase — Row sort

Figure 9.12 Shearsort.



(a) Operations between elements in rows

(b) Transpose operation

(c) Operations between elements in rows (originally columns)

Figure 9.13 Using the transpose operation to maintain operations in rows.

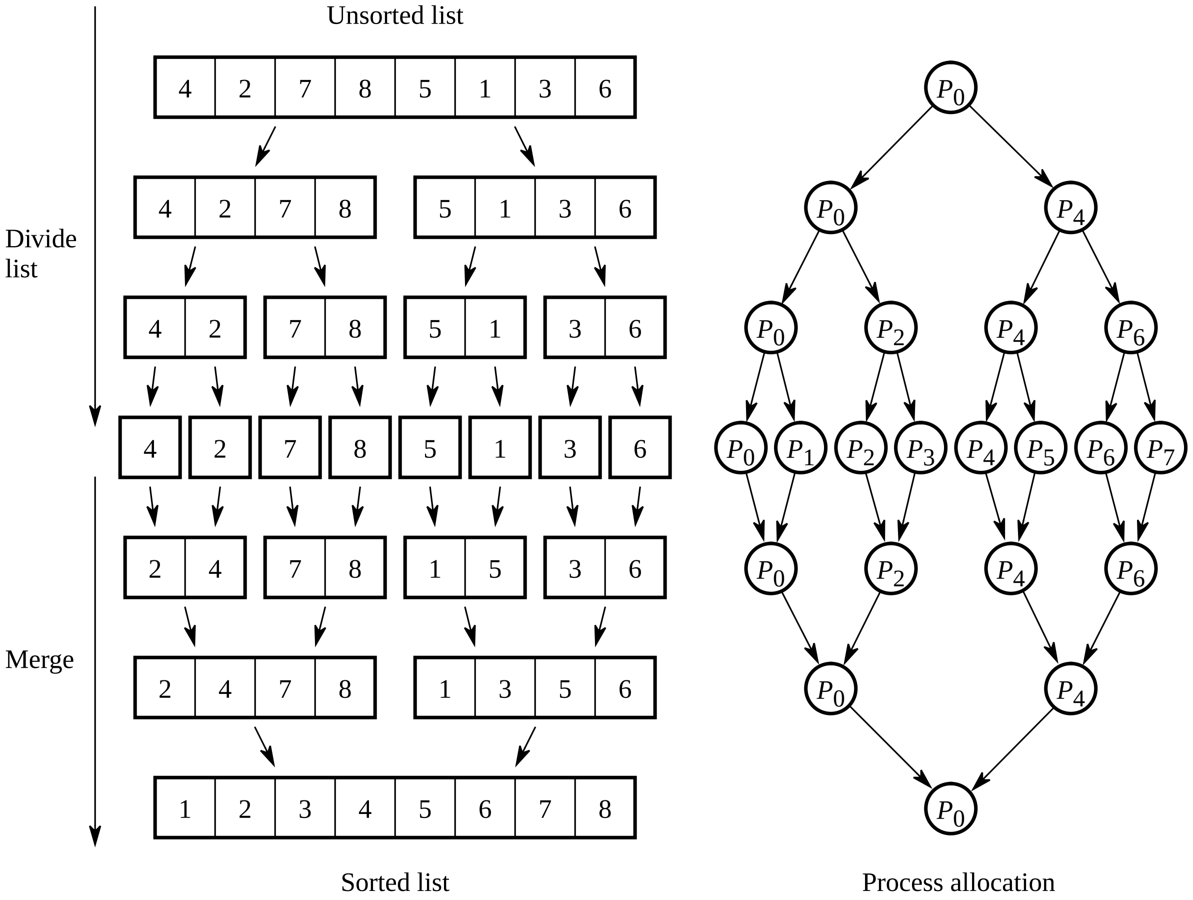


Figure 9.14 Mergesort using tree allocation of processes.

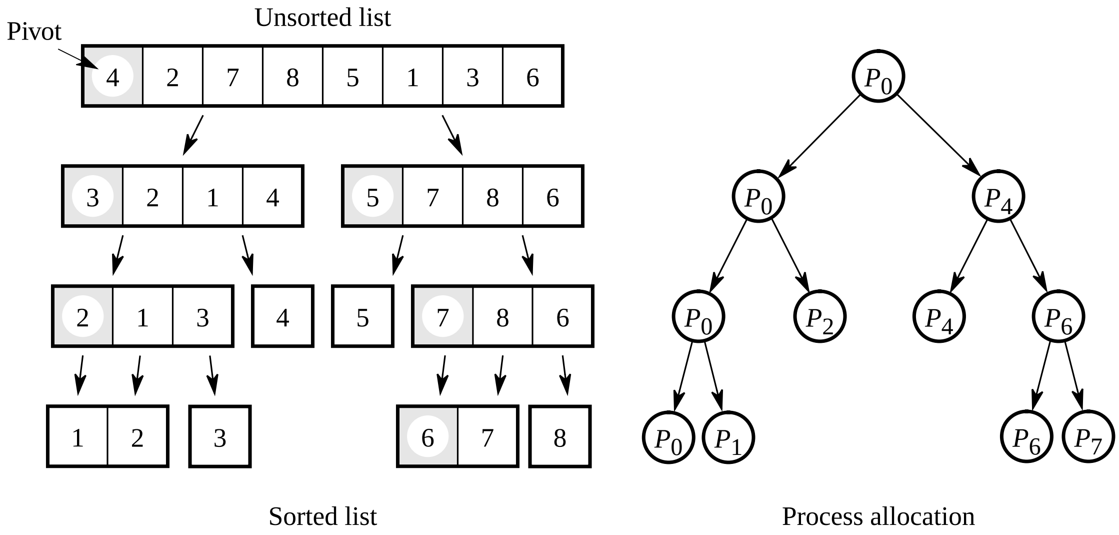


Figure 9.15 Quicksort using tree allocation of processes.

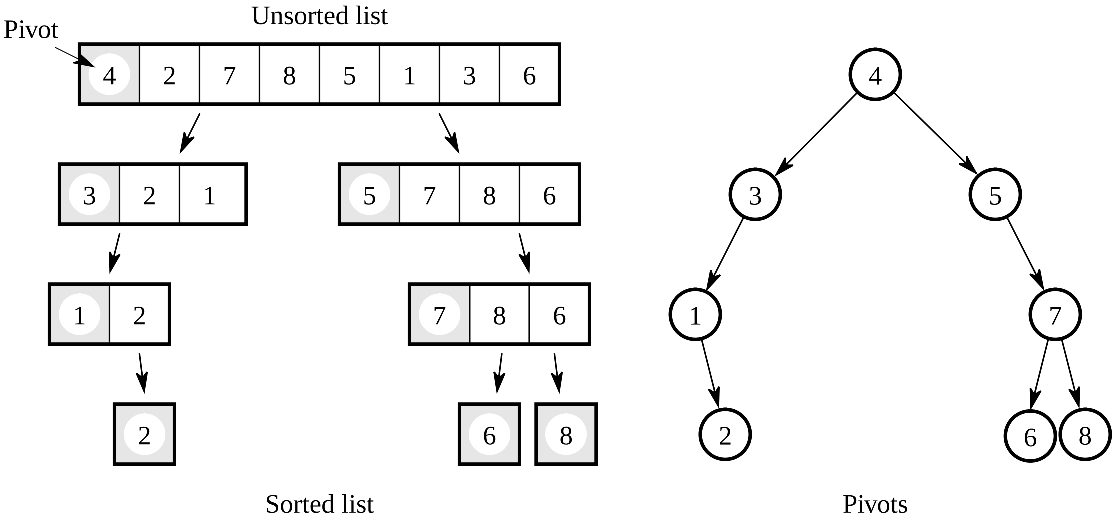


Figure 9.16 Quicksort showing pivot withheld in processes.

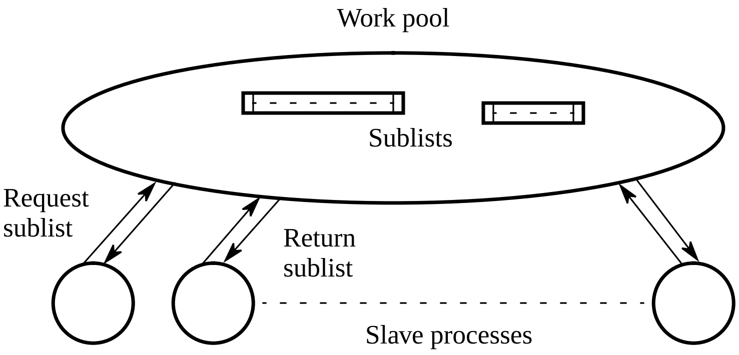


Figure 9.17 Work pool implementation of quicksort.

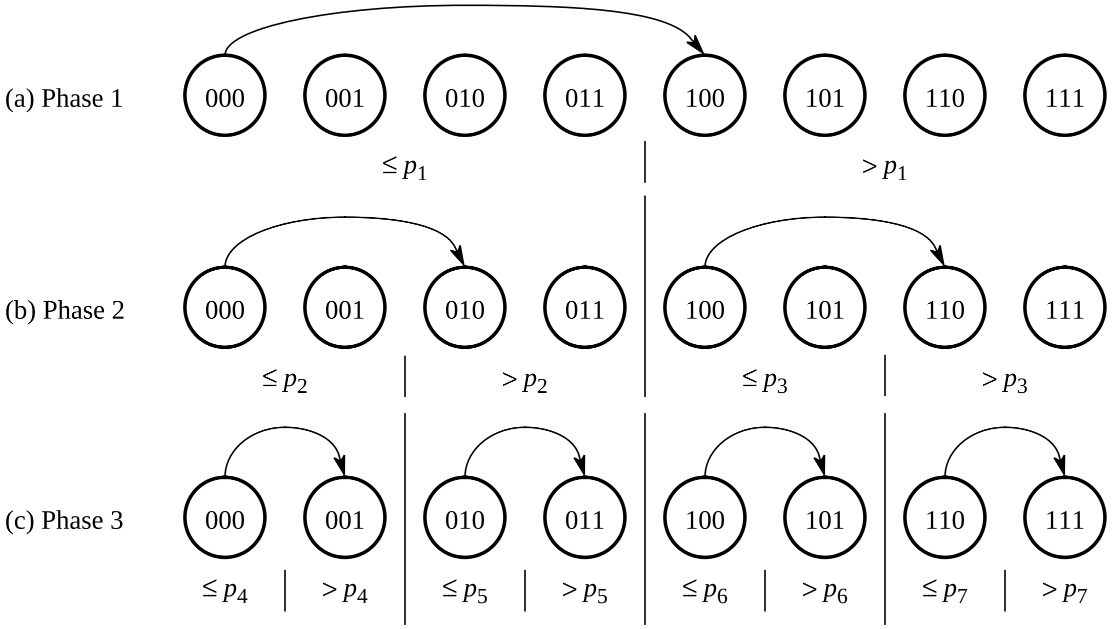


Figure 9.18 Hypercube quicksort algorithm when the numbers are originally in node 000.

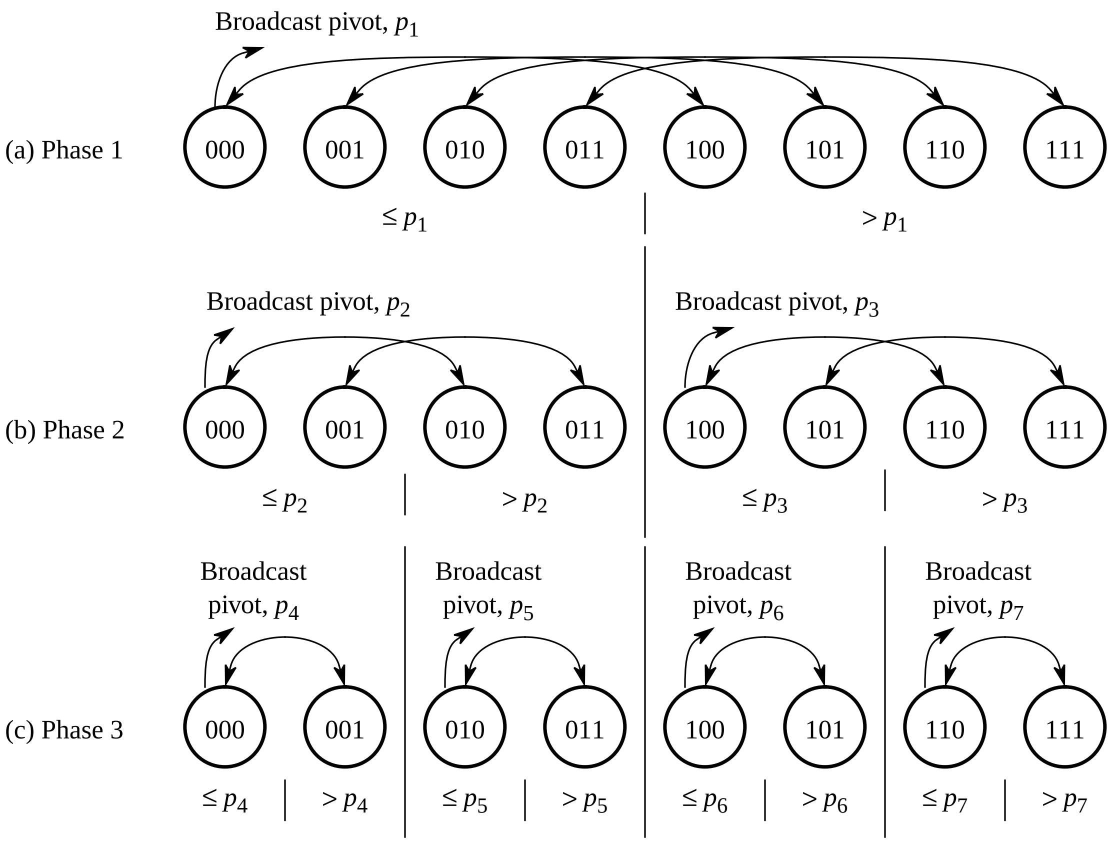


Figure 9.19 Hypercube quicksort algorithm when numbers are distributed among nodes.

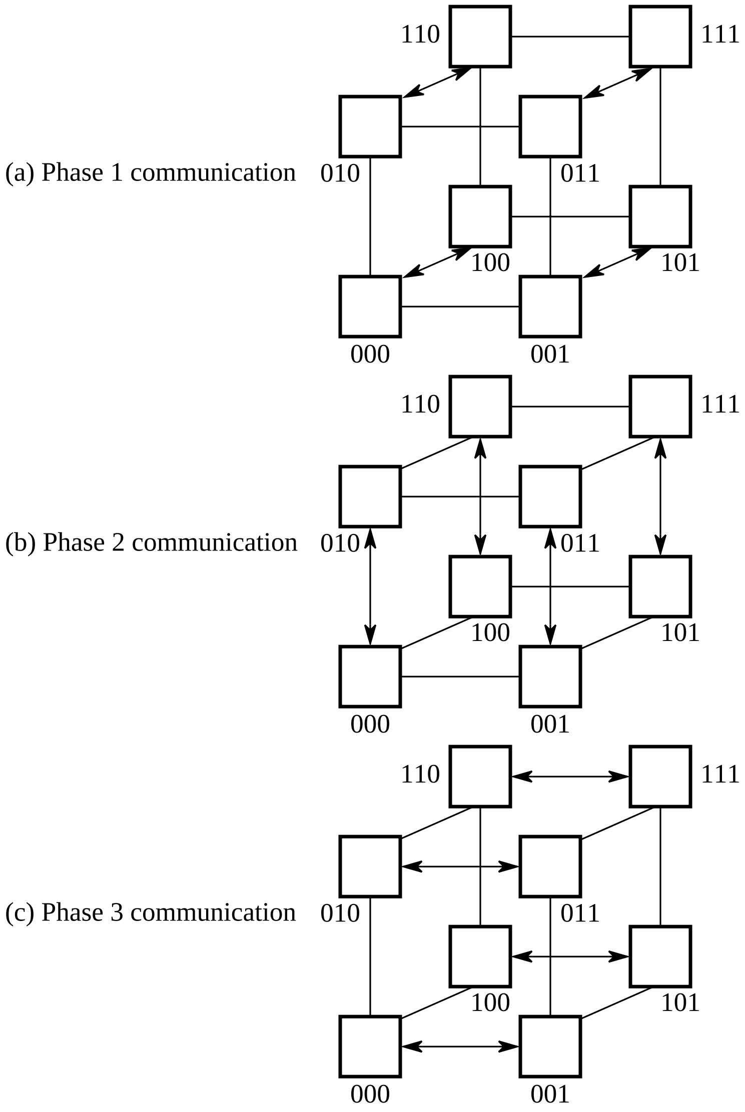

110
111
(a) Phase 1 communication
010
011
100
101
000
001
110
111
(b) Phase 2 communication
010
011
100
101
000
001
110
111
(c) Phase 3 communication
010
011
100
101
000
001

Figure 9.20 Hypercube quicksort communication.

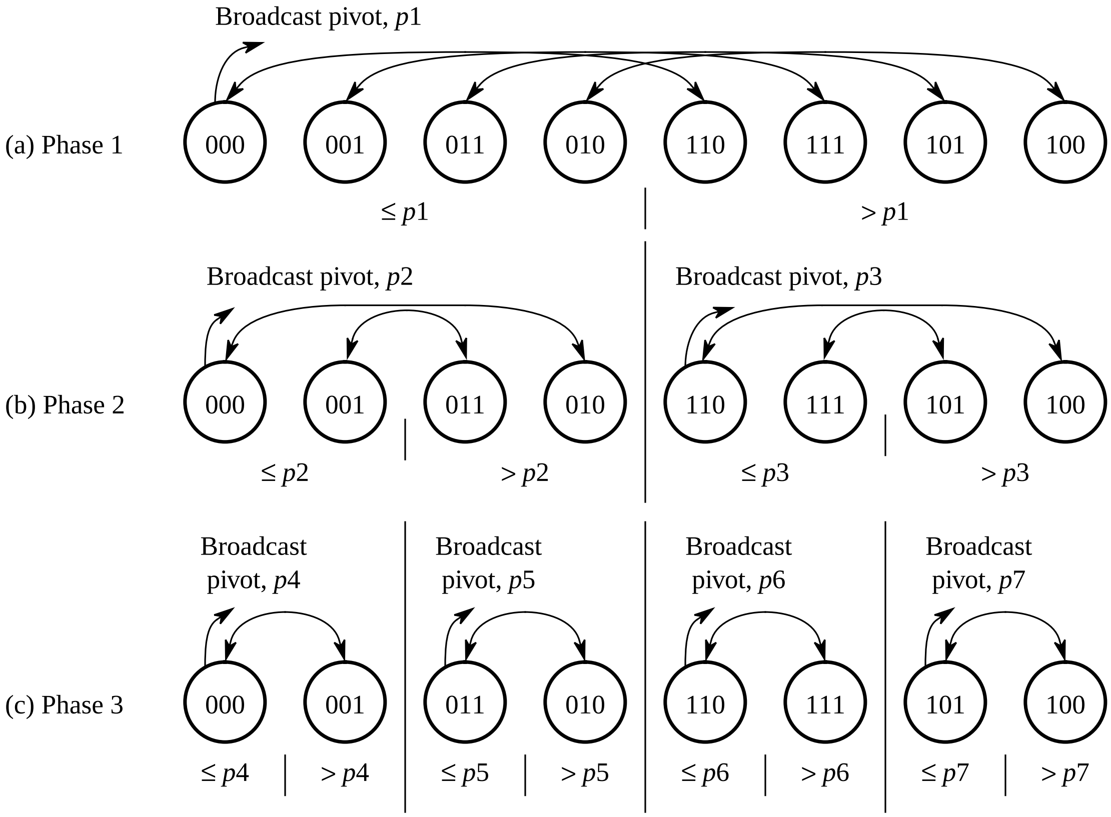


Figure 9.21 Quicksort hypercube algorithm with Gray code ordering.

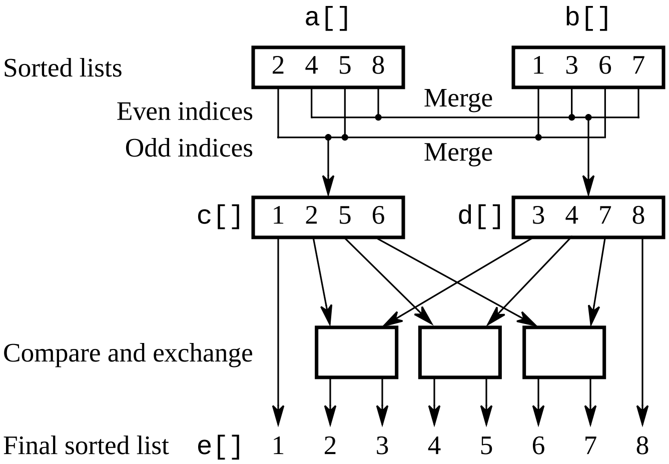


Figure 9.22 Odd-even merging of two sorted lists.

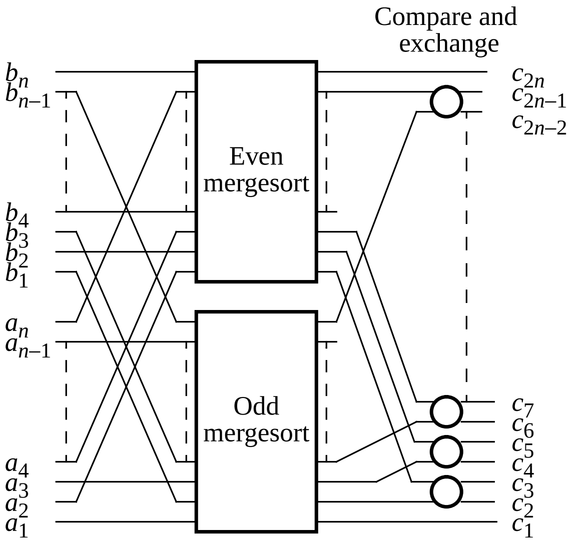
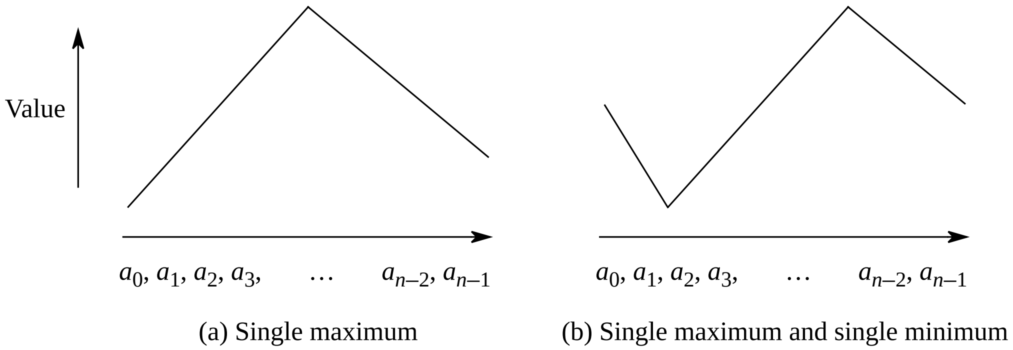


Figure 9.23 Odd-even mergesort.


Value
$a_0, a_1, a_2, a_3,$ … a_{n-2}, a_{n-1}
$a_0, a_1, a_2, a_3,$ … a_{n-2}, a_{n-1}

(a) Single maximum

(b) Single maximum and single minimum

Figure 9.24 Bitonic sequences.

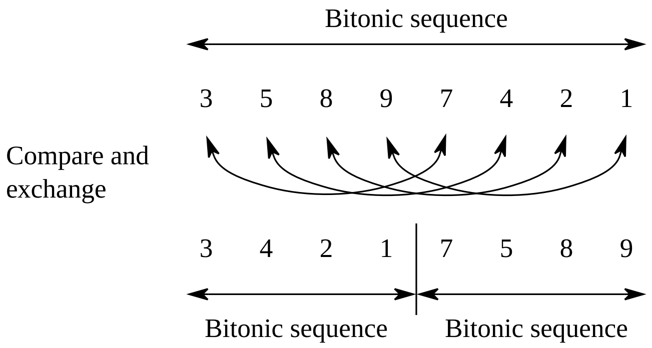


Figure 9.25 Creating two bitonic sequences from one bitonic sequence.

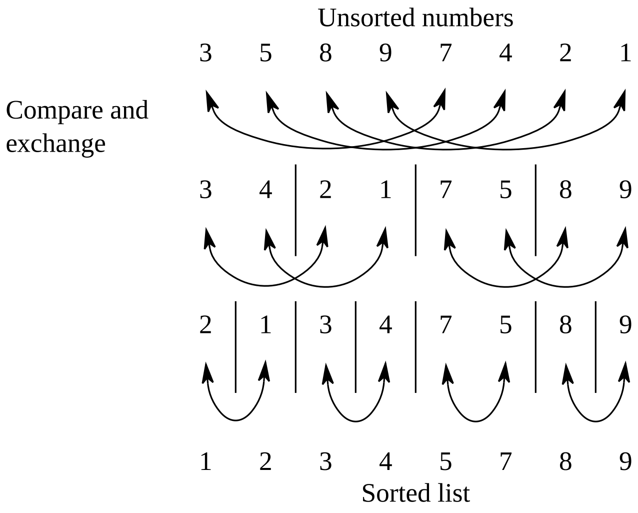


Figure 9.26 Sorting a bitonic sequence.

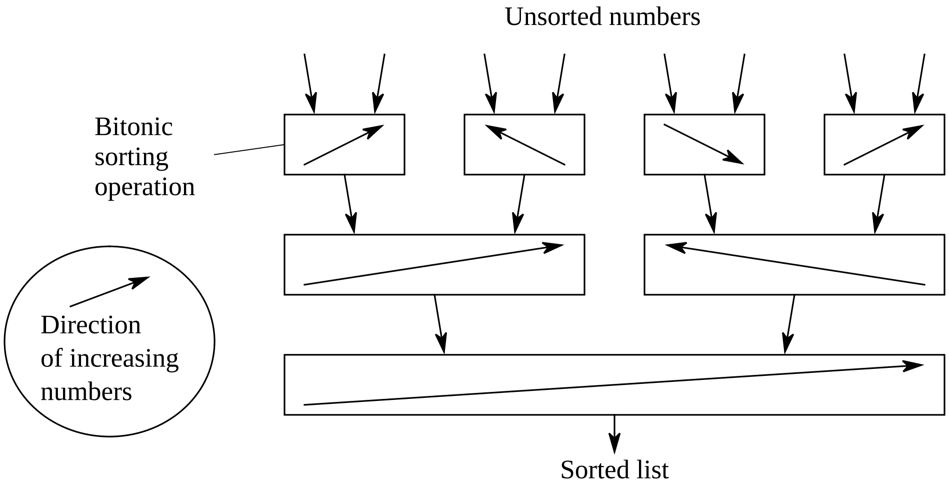


Figure 9.27 Bitonic mergesort.

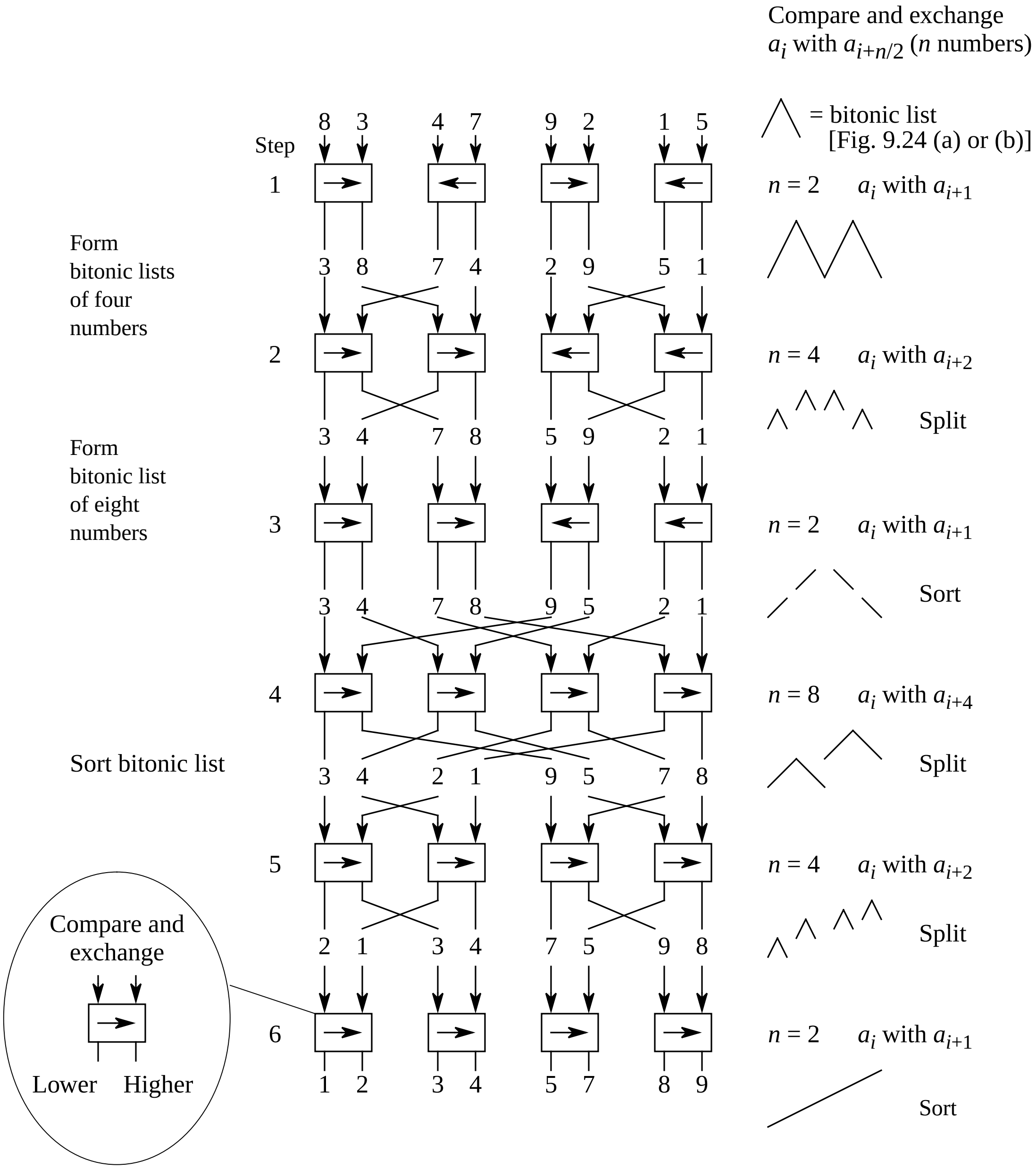
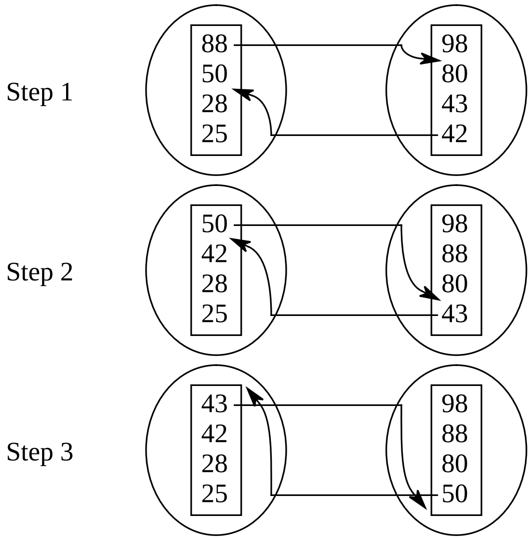


Figure 9.28 Bitonic mergesort on eight numbers.



Terminates when insertions at top/bottom of lists

Figure 9.29 Compare-and-exchange algorithm for Problem 9-5.

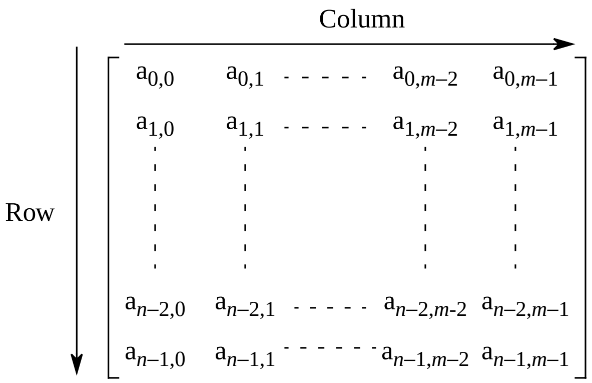

Column
Row
$a_{0,0}$ $a_{0,1}$ $a_{0,m-2}$ $a_{0,m-1}$
$a_{1,0}$ $a_{1,1}$ $a_{1,m-2}$ $a_{1,m-1}$
$a_{n-2,0}$ $a_{n-2,1}$ $a_{n-2,m-2}$ $a_{n-2,m-1}$
$a_{n-1,0}$ $a_{n-1,1}$ $a_{n-1,m-2}$ $a_{n-1,m-1}$

Figure 10.1 An $n \times m$ matrix.

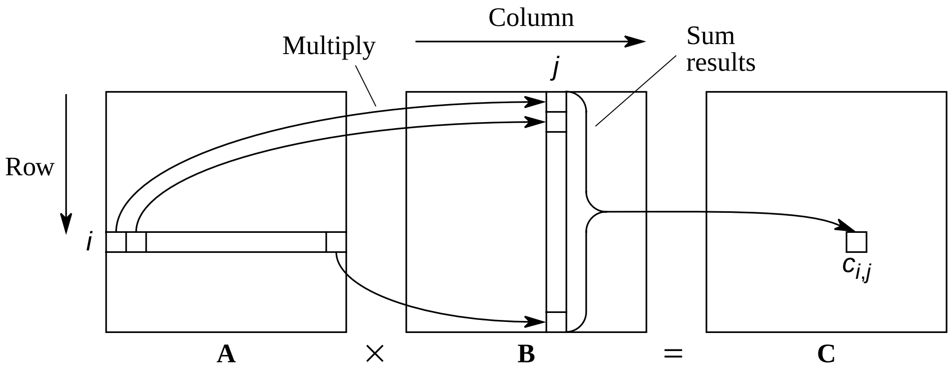


Figure 10.2 Matrix multiplication, **C** = **A** × **B**.

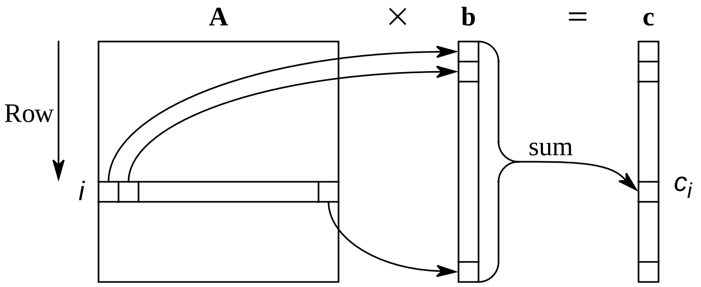


Figure 10.3 Matrix-vector multiplication
c = **A** × **b**.

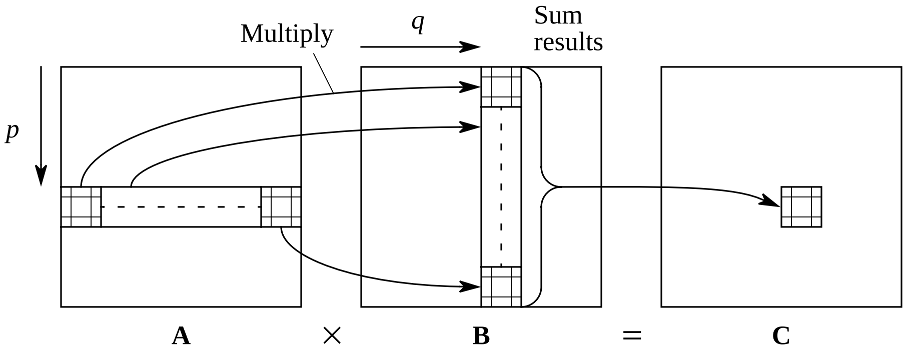


Figure 10.4 Block matrix multiplication.

$$\left[\begin{array}{cc|cc} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{array}\right] \times \left[\begin{array}{cc|cc} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ \hline b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{array}\right]$$

(a) Matrices

$$\overset{A_{0,0}}{\begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \end{bmatrix}} \times \overset{B_{0,0}}{\begin{bmatrix} b_{0,0} & b_{0,1} \\ b_{1,0} & b_{1,1} \end{bmatrix}} + \overset{A_{0,1}}{\begin{bmatrix} a_{0,2} & a_{0,3} \\ a_{1,2} & a_{1,3} \end{bmatrix}} \times \overset{B_{1,0}}{\begin{bmatrix} b_{2,0} & b_{2,1} \\ b_{3,0} & b_{3,1} \end{bmatrix}}$$

$$= \begin{bmatrix} a_{0,0}b_{0,0}+a_{0,1}b_{1,0} & a_{0,0}b_{0,1}+a_{0,1}b_{1,1} \\ a_{1,0}b_{0,0}+a_{1,1}b_{1,0} & a_{1,0}b_{0,1}+a_{1,1}b_{1,1} \end{bmatrix} + \begin{bmatrix} a_{0,2}b_{2,0}+a_{0,3}b_{3,0} & a_{0,2}b_{2,1}+a_{0,3}b_{3,1} \\ a_{1,2}b_{2,0}+a_{1,3}b_{3,0} & a_{1,2}b_{2,1}+a_{1,3}b_{3,1} \end{bmatrix}$$

$$= \begin{bmatrix} a_{0,0}b_{0,0}+a_{0,1}b_{1,0}+a_{0,2}b_{2,0}+a_{0,3}b_{3,0} & a_{0,0}b_{0,1}+a_{0,1}b_{1,1}+a_{0,2}b_{2,1}+a_{0,3}b_{3,1} \\ a_{1,0}b_{0,0}+a_{1,1}b_{1,0}+a_{1,2}b_{2,0}+a_{1,3}b_{3,0} & a_{1,0}b_{0,1}+a_{1,1}b_{1,1}+a_{1,2}b_{2,1}+a_{1,3}b_{3,1} \end{bmatrix}$$

$$= C_{0,0}$$

(b) Multiplying $A_{0,0} \times B_{0,0}$ to obtain $C_{0,0}$

Figure 10.5 Submatrix multiplication.

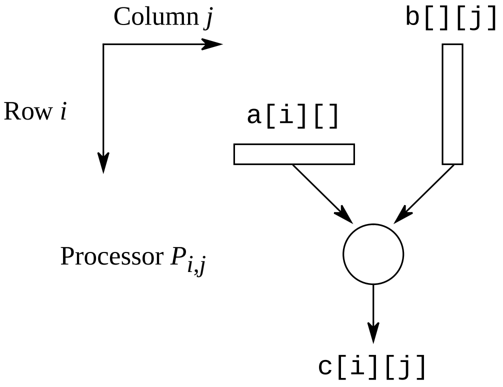


Figure 10.6 Direct implementation of matrix multiplication.

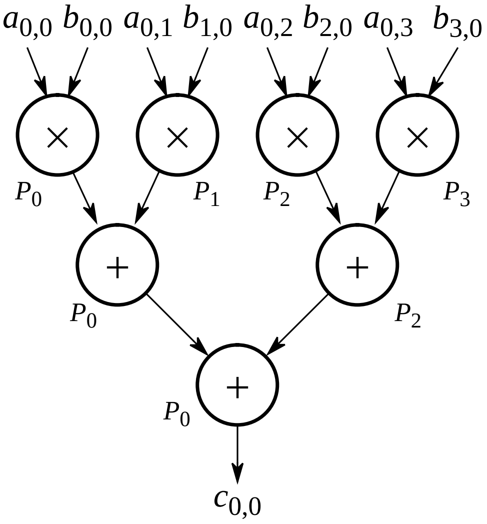


Figure 10.7 Accumulation using a tree construction.

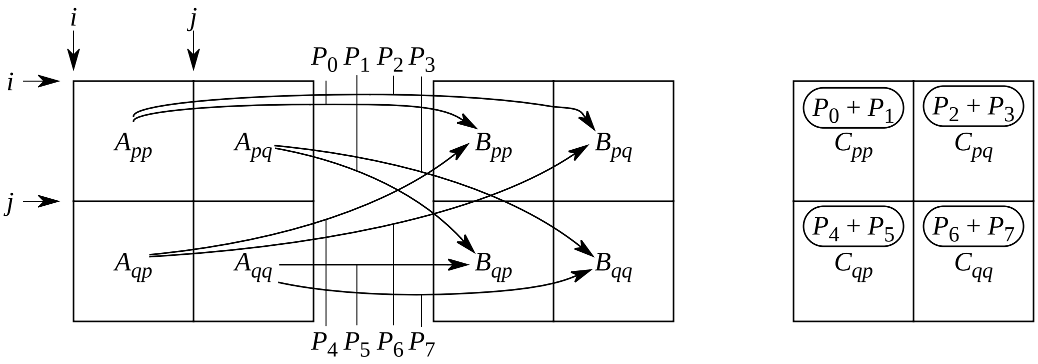


Figure 10.8 Submatrix multiplication and summation.

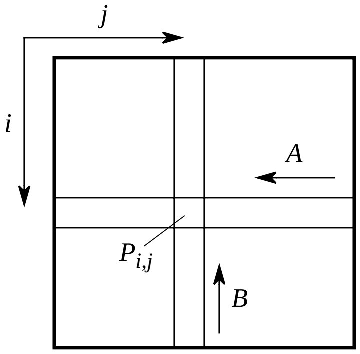


Figure 10.9 Movement of *A* and *B* elements.

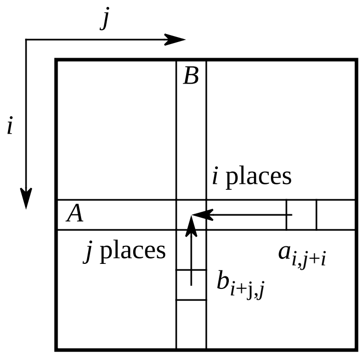


Figure 10.10 Step 2 — Alignment of elements of *A* and *B*.

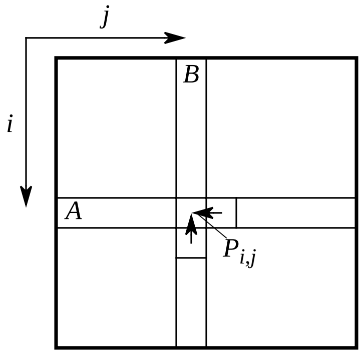


Figure 10.11 Step 4 — One-place shift of elements of *A* and *B*.

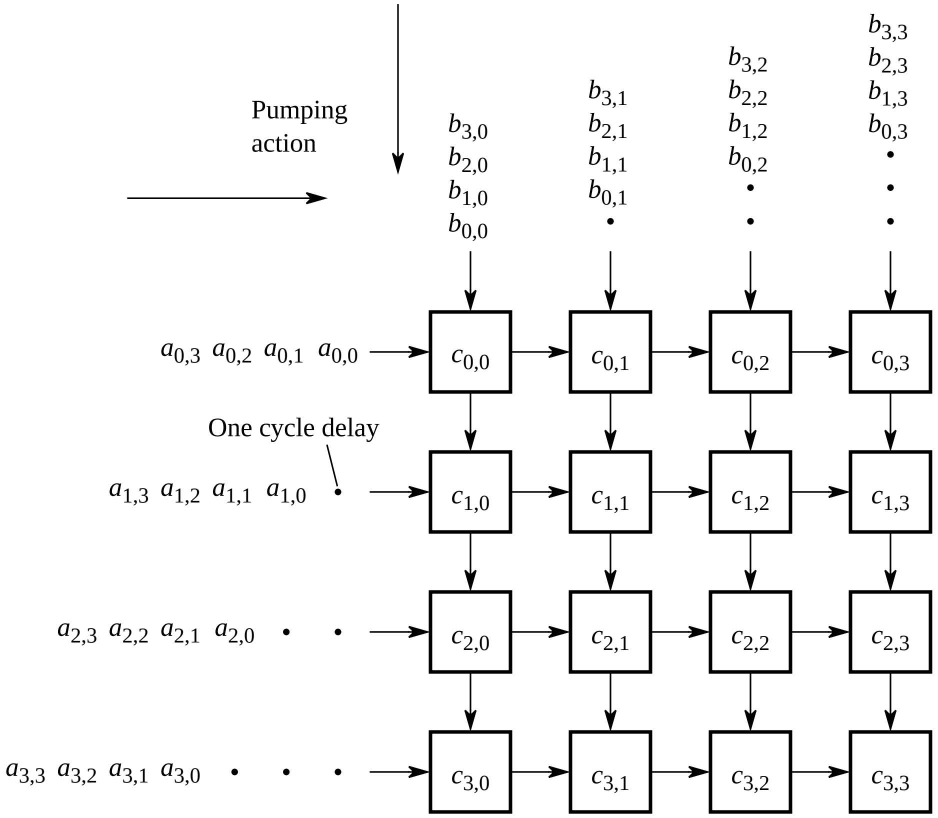

Pumping action
$b_{3,0}$ $b_{2,0}$ $b_{1,0}$ $b_{0,0}$
$b_{3,1}$ $b_{2,1}$ $b_{1,1}$ $b_{0,1}$
$b_{3,2}$ $b_{2,2}$ $b_{1,2}$ $b_{0,2}$
$b_{3,3}$ $b_{2,3}$ $b_{1,3}$ $b_{0,3}$
$a_{0,3}$ $a_{0,2}$ $a_{0,1}$ $a_{0,0}$
One cycle delay
$a_{1,3}$ $a_{1,2}$ $a_{1,1}$ $a_{1,0}$
$a_{2,3}$ $a_{2,2}$ $a_{2,1}$ $a_{2,0}$
$a_{3,3}$ $a_{3,2}$ $a_{3,1}$ $a_{3,0}$
$c_{0,0}$ $c_{0,1}$ $c_{0,2}$ $c_{0,3}$
$c_{1,0}$ $c_{1,1}$ $c_{1,2}$ $c_{1,3}$
$c_{2,0}$ $c_{2,1}$ $c_{2,2}$ $c_{2,3}$
$c_{3,0}$ $c_{3,1}$ $c_{3,2}$ $c_{3,3}$

Figure 10.12 Matrix multiplication using a systolic array.

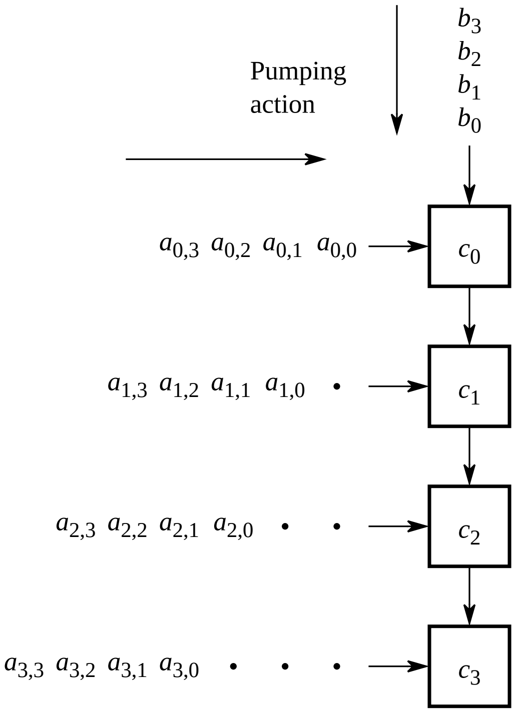

Pumping action
b_3
b_2
b_1
b_0
$a_{0,3}$ $a_{0,2}$ $a_{0,1}$ $a_{0,0}$
$a_{1,3}$ $a_{1,2}$ $a_{1,1}$ $a_{1,0}$ •
$a_{2,3}$ $a_{2,2}$ $a_{2,1}$ $a_{2,0}$ • •
$a_{3,3}$ $a_{3,2}$ $a_{3,1}$ $a_{3,0}$ • • •
c_0
c_1
c_2
c_3

Figure 10.13 Matrix-vector multiplication using a systolic array.

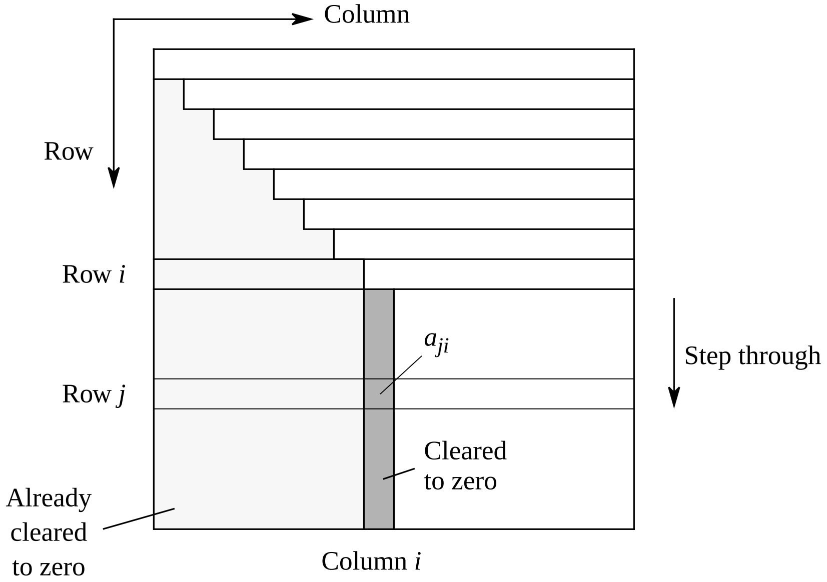


Figure 10.14 Gaussian elimination.

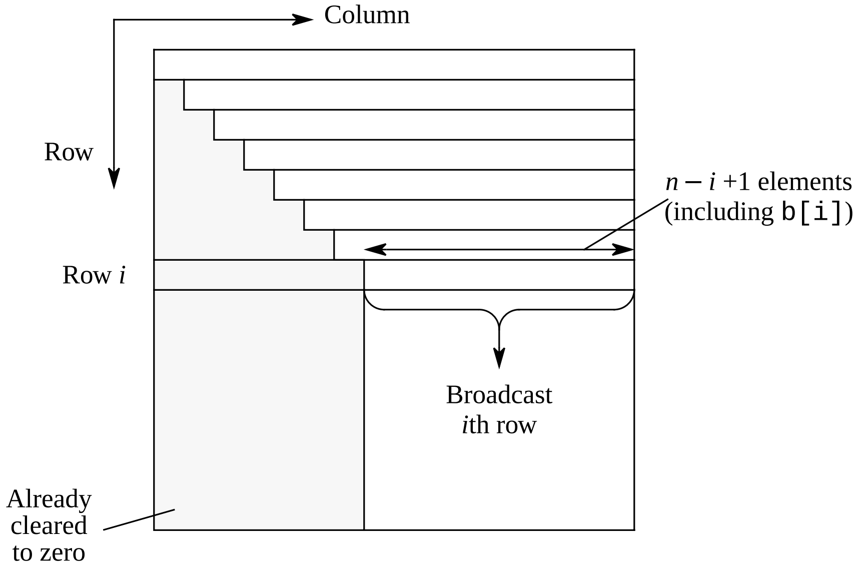


Figure 10.15 Broadcast in parallel implementation of Gaussian elimination.

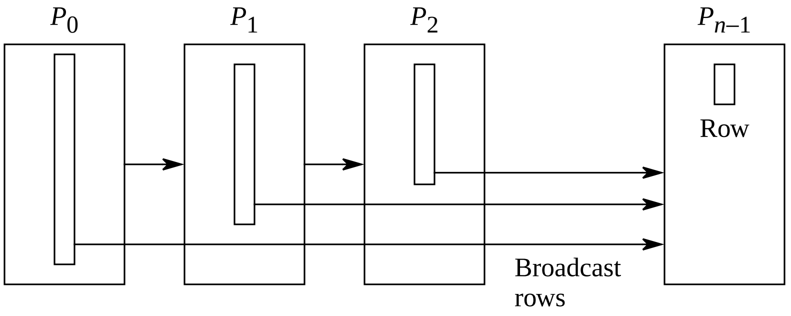


Figure 10.16 Pipeline implementation of Gaussian elimination.

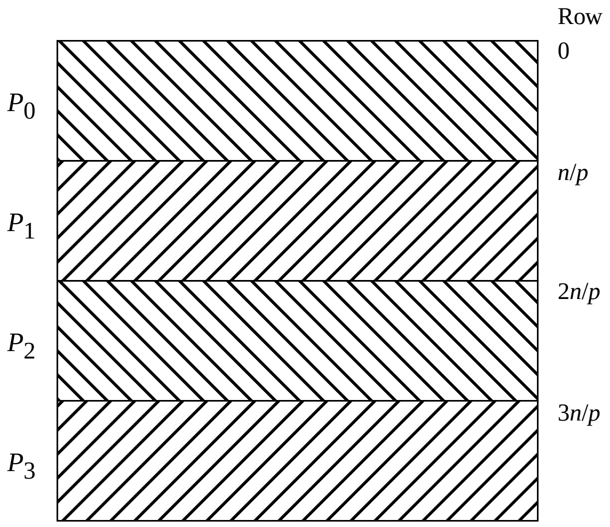


Figure 10.17 Strip partitioning.

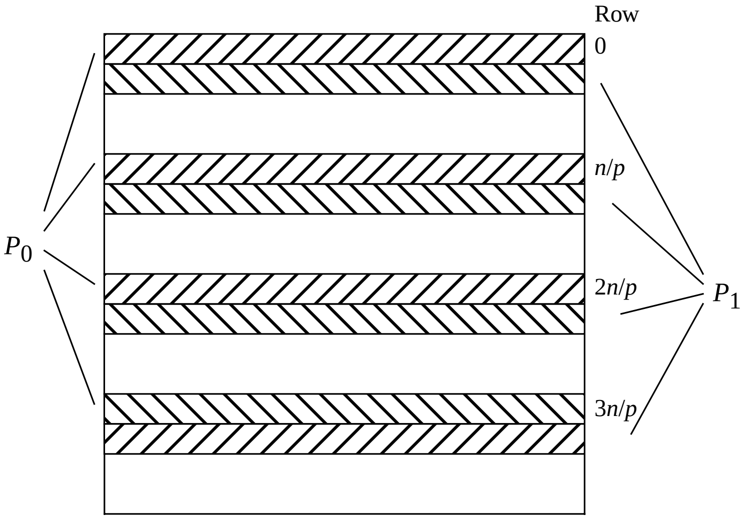


Figure 10.18 Cyclic partitioning to equalize workload.

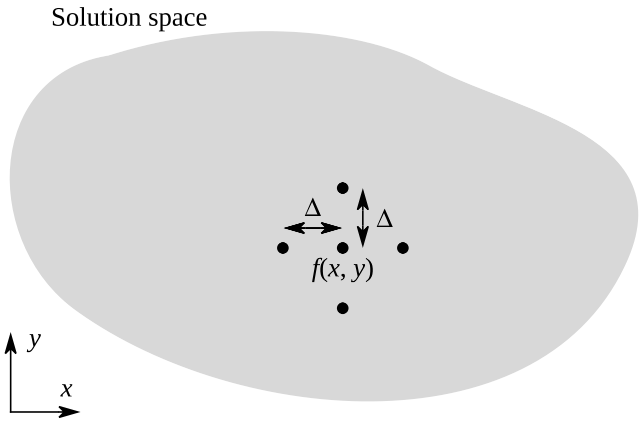


Figure 10.19 Finite difference method.

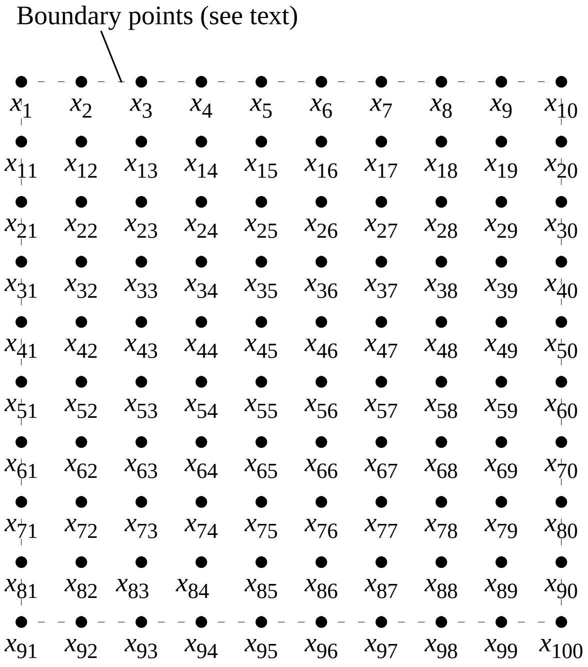

Boundary points (see text)
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9 x_{10}
x_{11} x_{12} x_{13} x_{14} x_{15} x_{16} x_{17} x_{18} x_{19} x_{20}
x_{21} x_{22} x_{23} x_{24} x_{25} x_{26} x_{27} x_{28} x_{29} x_{30}
x_{31} x_{32} x_{33} x_{34} x_{35} x_{36} x_{37} x_{38} x_{39} x_{40}
x_{41} x_{42} x_{43} x_{44} x_{45} x_{46} x_{47} x_{48} x_{49} x_{50}
x_{51} x_{52} x_{53} x_{54} x_{55} x_{56} x_{57} x_{58} x_{59} x_{60}
x_{61} x_{62} x_{63} x_{64} x_{65} x_{66} x_{67} x_{68} x_{69} x_{70}
x_{71} x_{72} x_{73} x_{74} x_{75} x_{76} x_{77} x_{78} x_{79} x_{80}
x_{81} x_{82} x_{83} x_{84} x_{85} x_{86} x_{87} x_{88} x_{89} x_{90}
x_{91} x_{92} x_{93} x_{94} x_{95} x_{96} x_{97} x_{98} x_{99} x_{100}

Figure 10.20 Mesh of points numbered in natural order.

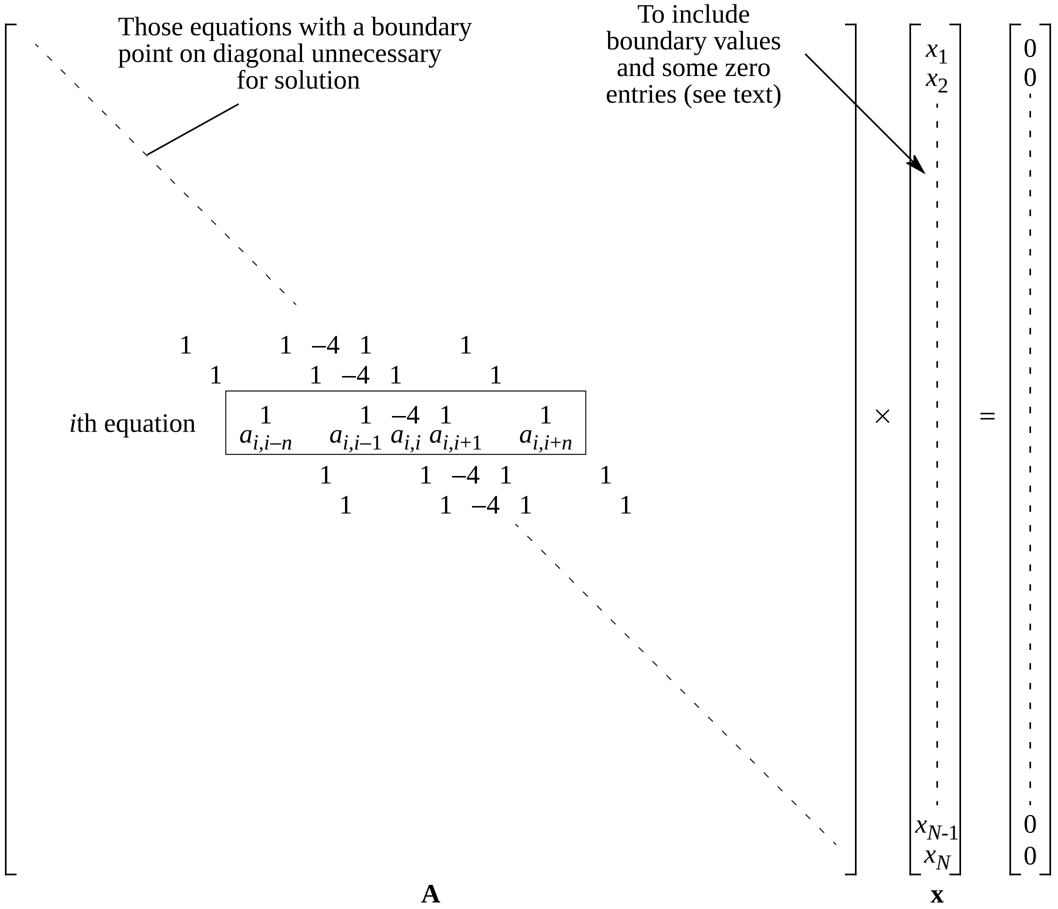

Those equations with a boundary point on diagonal unnecessary for solution
To include boundary values and some zero entries (see text)
1 1 −4 1 1
1 1 −4 1 1
ith equation
1 1 −4 1 1
$a_{i,i-n}$ $a_{i,i-1}$ $a_{i,i}$ $a_{i,i+1}$ $a_{i,i+n}$
1 1 −4 1 1
1 1 −4 1 1
×
=
x_1
x_2
$x_{N\text{-}1}$
x_N
0
0
0
0
A
x

Figure 10.21 Sparse matrix for Laplace’s equation.

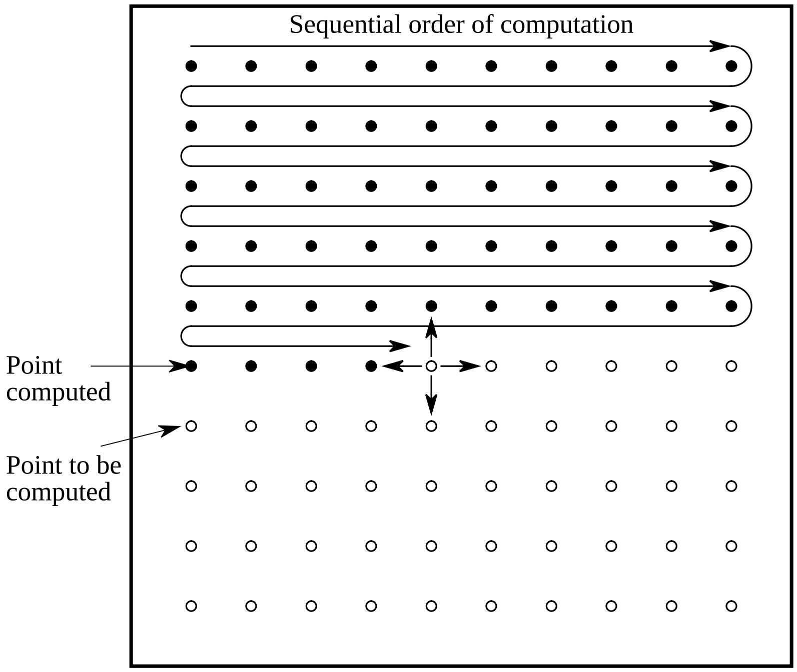


Figure 10.22 Gauss-Seidel relaxation with natural order, computed sequentially.

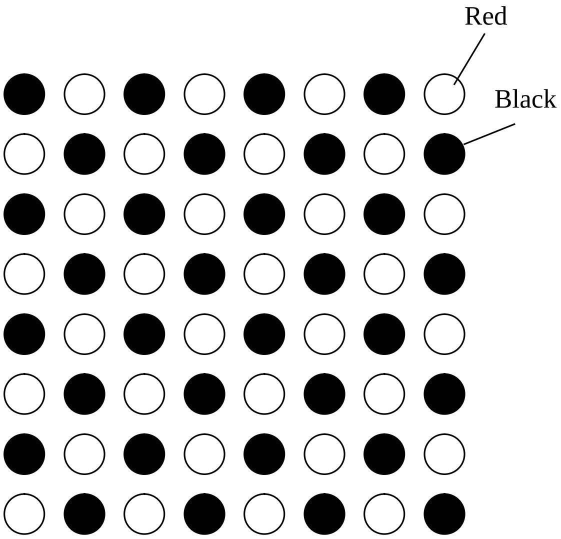


Figure 10.23 Red-black ordering.

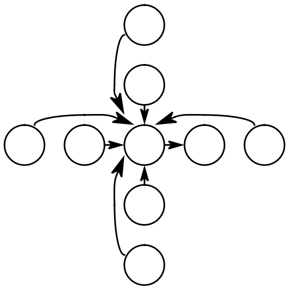


Figure 10.24 Nine-point stencil.

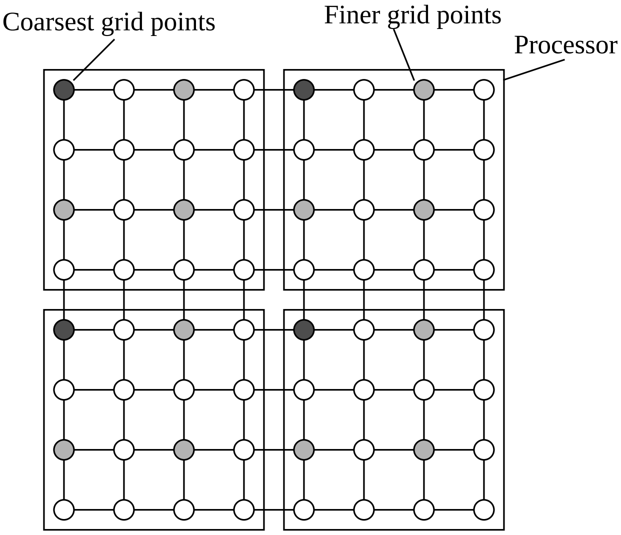
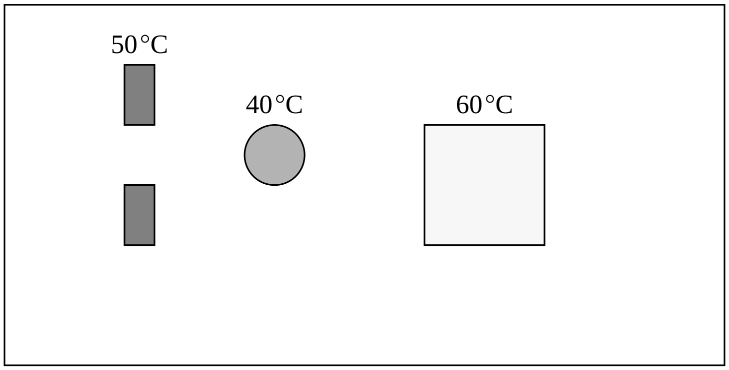


Figure 10.25 Multigrid processor allocation.



Ambient temperature at edges of board = 20°C

Figure 10.26 Printed circuit board for Problem 10-18.

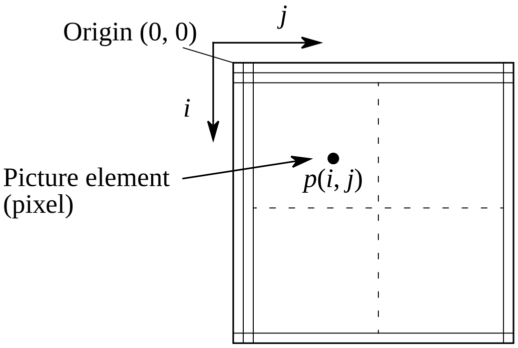


Figure 11.1 Pixmap.

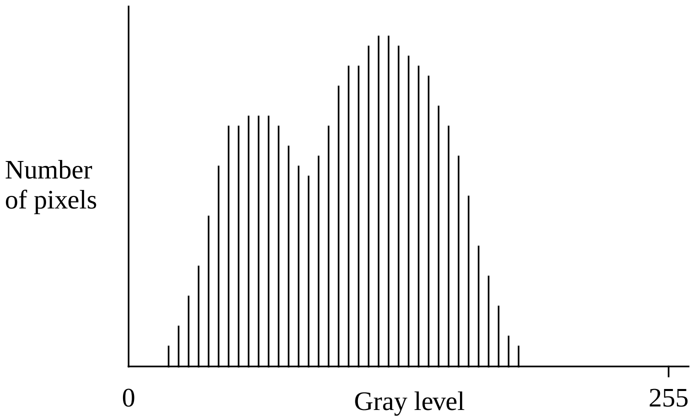
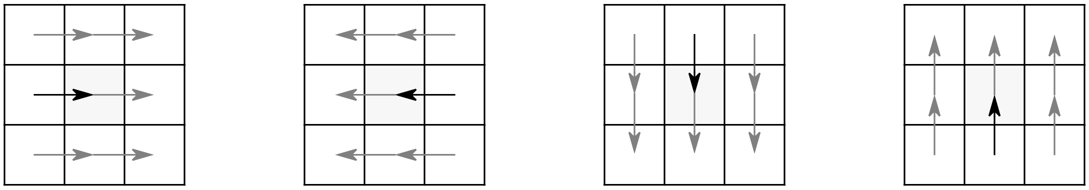


Figure 11.2 Image histogram.

x_0	x_1	x_2
x_3	x_4	x_5
x_6	x_7	x_8

Figure 11.3 Pixel values for a 3 × 3 group.



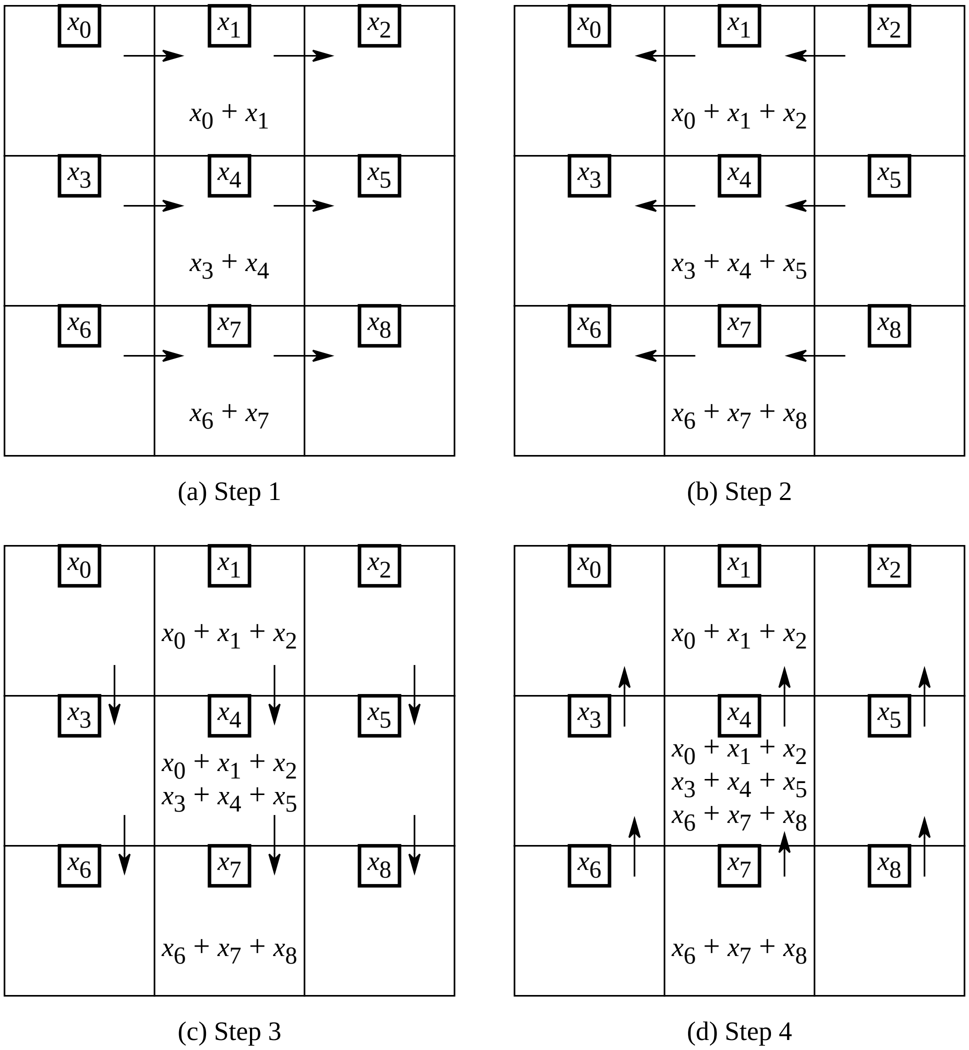
Step 1
Each pixel adds
pixel from left

Step 2
Each pixel adds
pixel from right

Step 3
Each pixel adds pixel
from above

Step 4
Each pixel adds pixel
from below

Figure 11.4 Four-step data transfer for the computation of mean.



(a) Step 1

(b) Step 2

(c) Step 3

(d) Step 4

Figure 11.5 Parallel mean data accumulation.

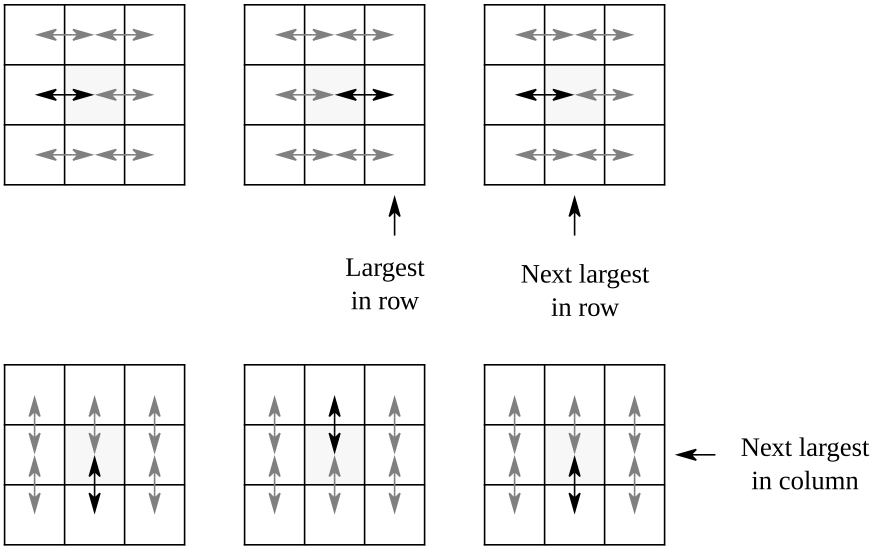


Figure 11.6 Approximate median algorithm requiring six steps.

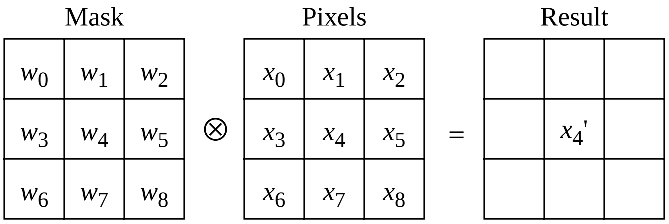


Figure 11.7 Using a 3 × 3 weighted mask.

$k = \frac{1}{9}$

1	1	1
1	1	1
1	1	1

Figure 11.8 Mask to compute mean.

$k = \frac{1}{16}$

1	1	1
1	8	1
1	1	1

Figure 11.9 A noise reduction mask.

$$k = \frac{1}{9}$$

−1	−1	−1
−1	8	−1
−1	−1	−1

Figure 11.10 High-pass sharpening filter mask.

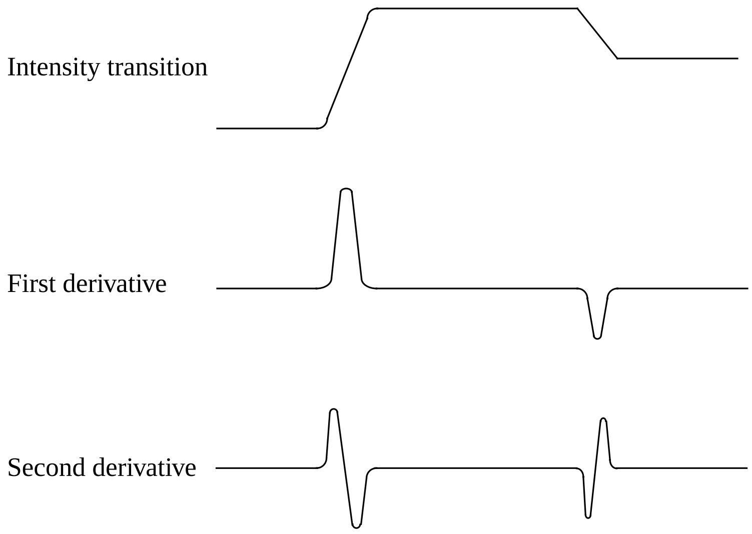


Figure 11.11 Edge detection using differentiation.

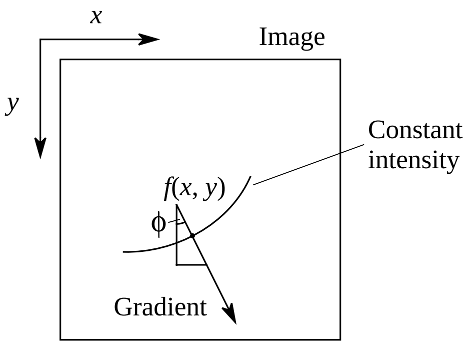


Figure 11.12 Gray level gradient and direction.

–1	–1	–1
0	0	0
1	1	1

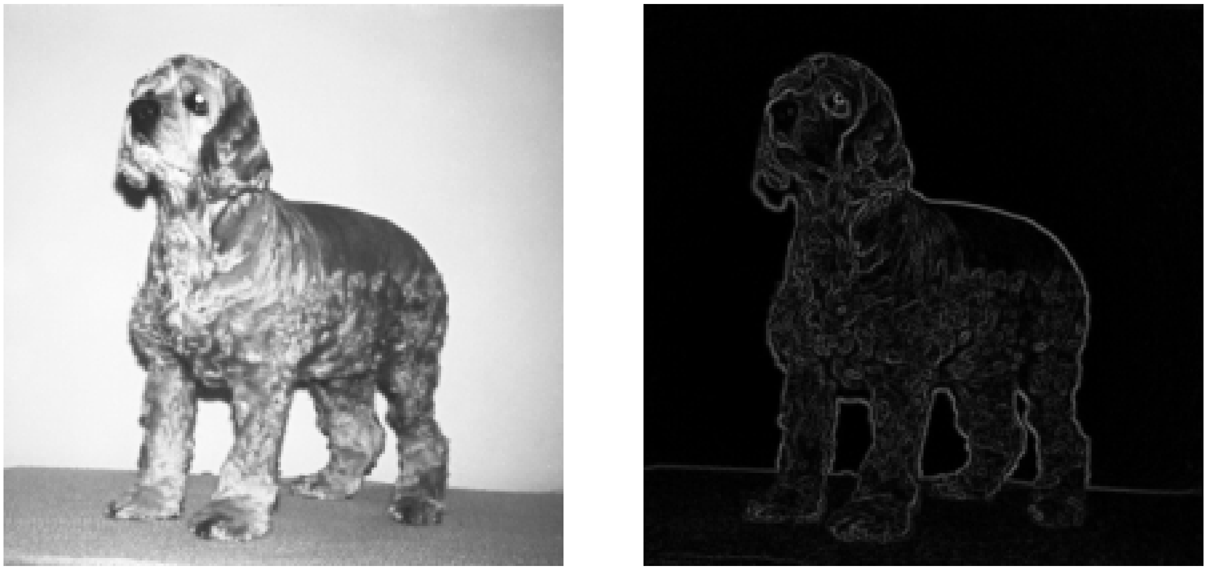
–1	0	1
–1	0	1
–1	0	1

Figure 11.13 Prewitt operator.

−1	−2	−1
0	0	0
1	2	1

−1	0	1
−2	0	2
−1	0	1

Figure 11.14 Sobel operator.



(a) Original image (Annabel)

(b) Effect of Sobel operator

Figure 11.15 Edge detection with Sobel operator.

0	–1	0
–1	4	–1
0	–1	0

Figure 11.16 Laplace operator.

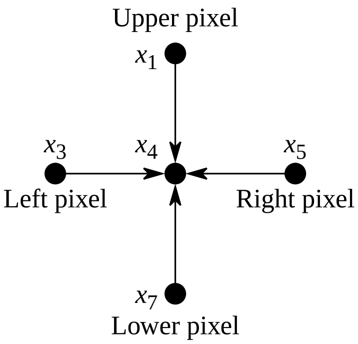


Figure 11.17 Pixels used in Laplace operator.

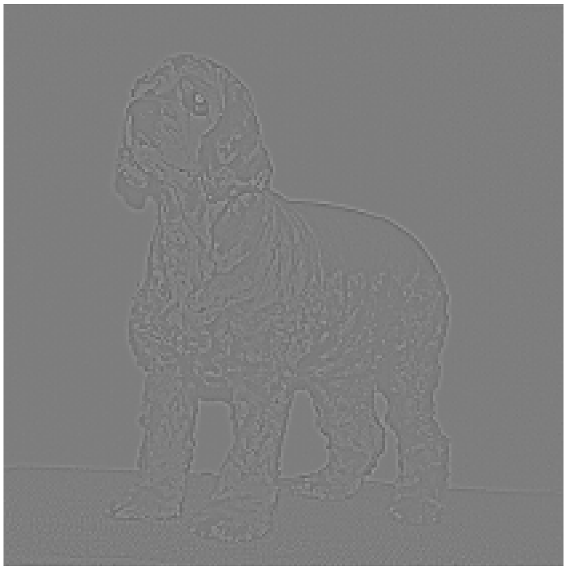


Figure 11.18 Effect of Laplace operator.

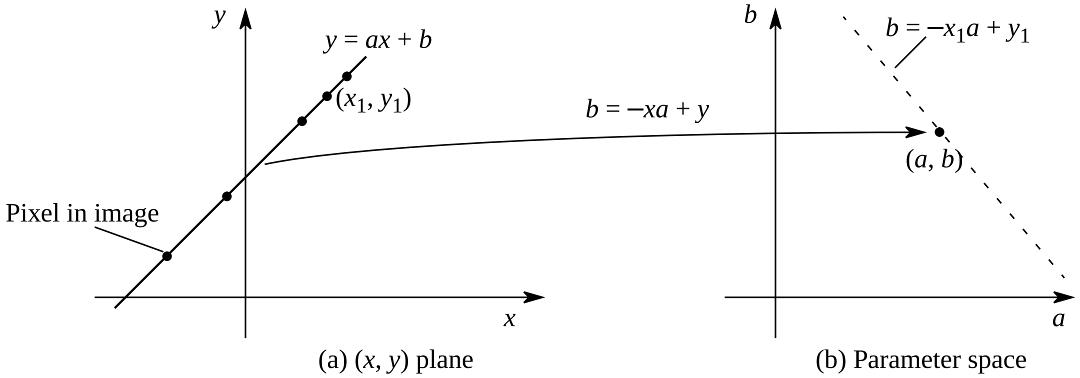


Figure 11.19 Mapping a line into (a, b) space.

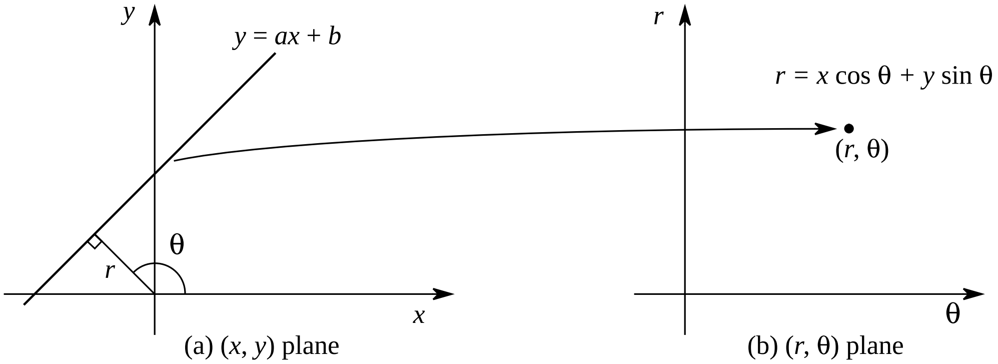


Figure 11.20 Mapping a line into (r, θ) space.

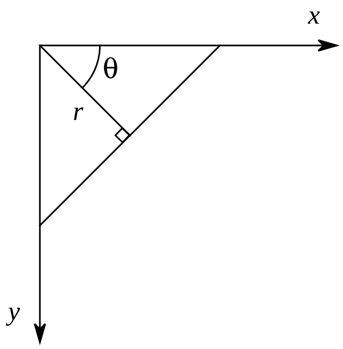


Figure 11.21 Normal representation using image coordinate system.

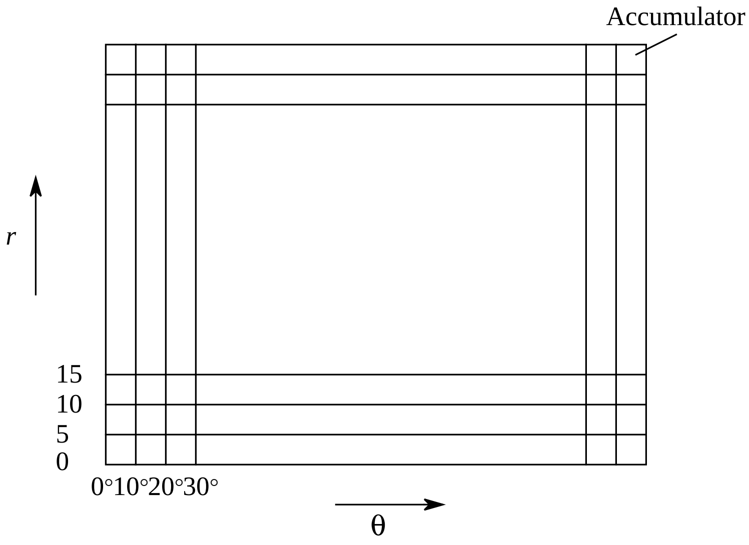


Figure 11.22 Accumulators, acc[*r*][θ], for the Hough transform.

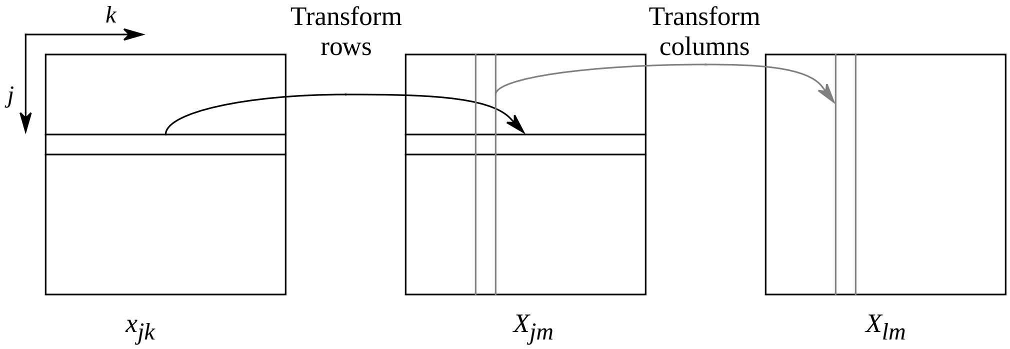
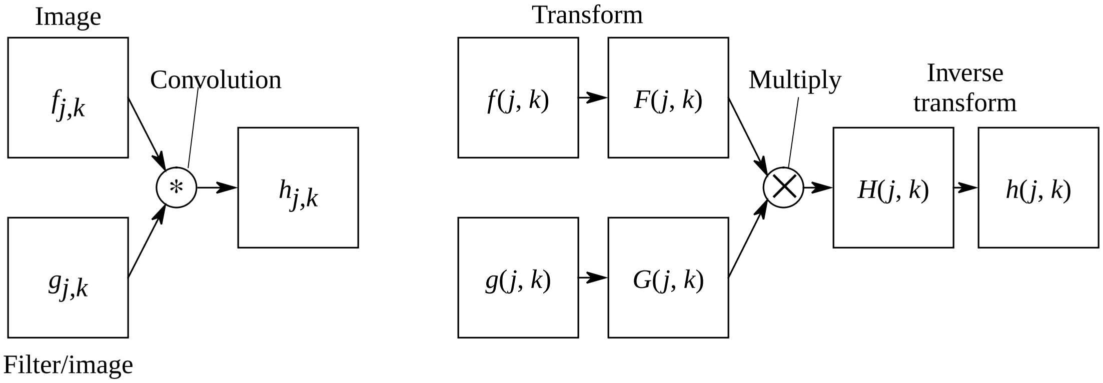


Figure 11.23 Two-dimensional DFT.



(a) Direct convolution

(b) Using Fourier transform

Figure 11.24 Convolution using Fourier transforms.

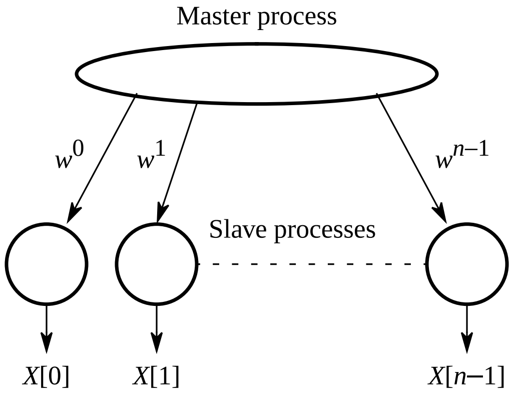


Figure 11.25 Master-slave approach for implementing the DFT directly.

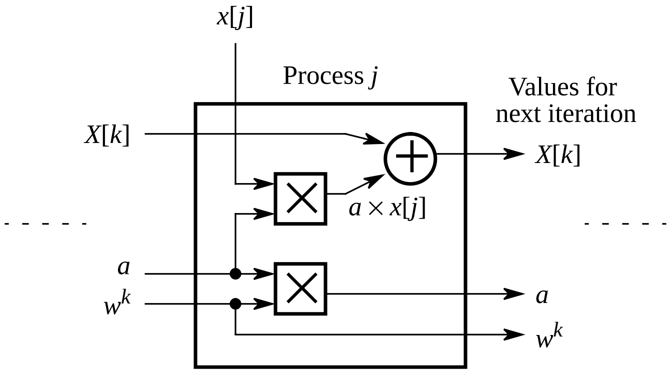
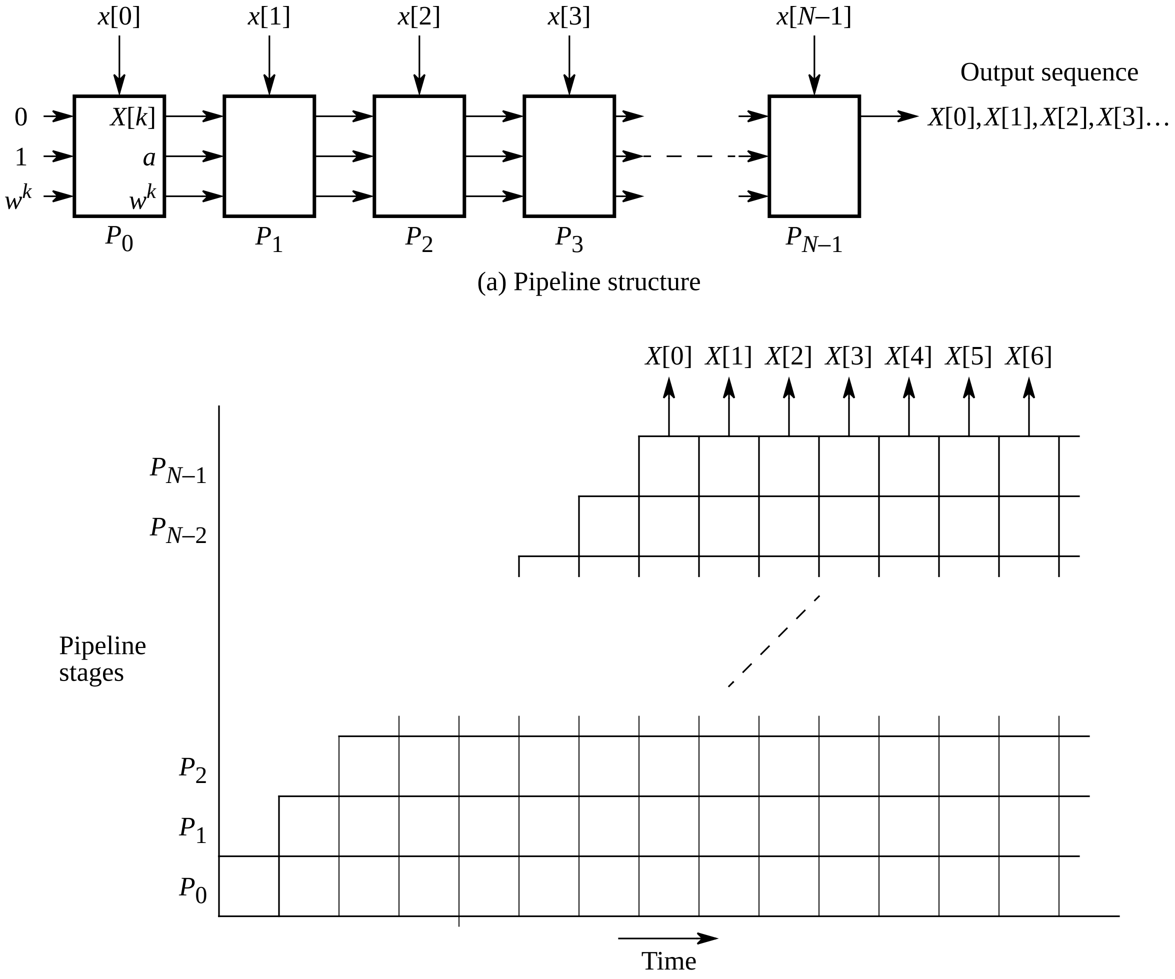


Figure 11.26 One stage of a pipeline implementation of DFT algorithm.



(a) Pipeline structure

(b) Timing diagram

Figure 11.27 Discrete Fourier transform with a pipeline.

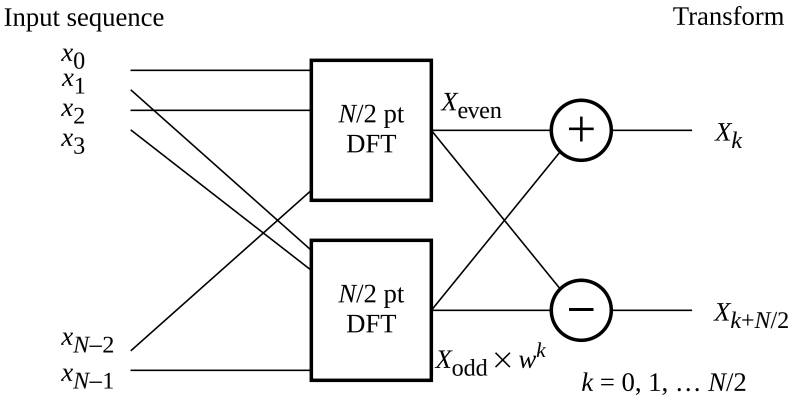


Figure 11.28 Decomposition of *N*-point DFT into two *N*/2-point DFTs.

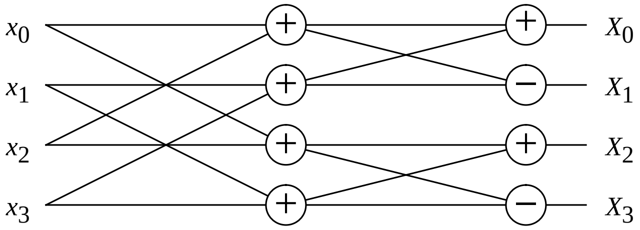


Figure 11.29 Four-point discrete Fourier transform.

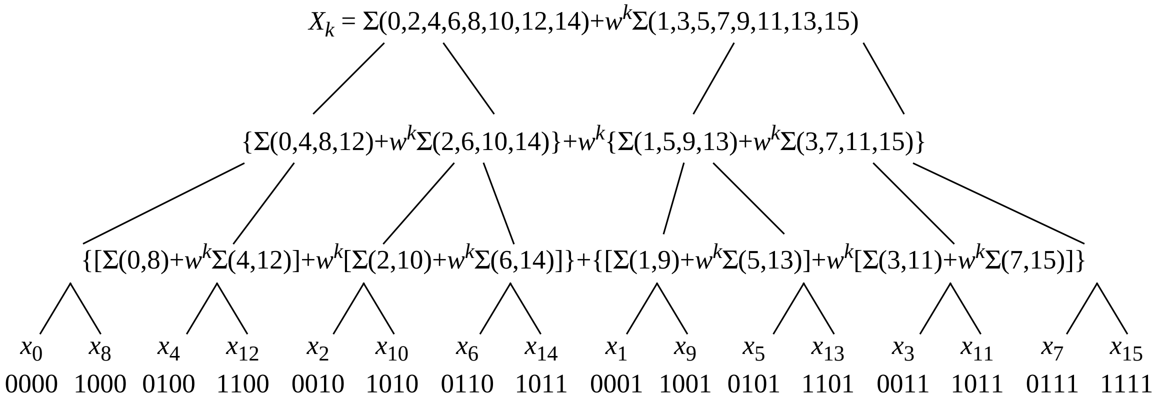

$X_k = \Sigma(0,2,4,6,8,10,12,14)+w^k\Sigma(1,3,5,7,9,11,13,15)$
$\{\Sigma(0,4,8,12)+w^k\Sigma(2,6,10,14)\}+w^k\{\Sigma(1,5,9,13)+w^k\Sigma(3,7,11,15)\}$
$\{[\Sigma(0,8)+w^k\Sigma(4,12)]+w^k[\Sigma(2,10)+w^k\Sigma(6,14)]\}+\{[\Sigma(1,9)+w^k\Sigma(5,13)]+w^k[\Sigma(3,11)+w^k\Sigma(7,15)]\}$
x_0 x_8 x_4 x_{12} x_2 x_{10} x_6 x_{14} x_1 x_9 x_5 x_{13} x_3 x_{11} x_7 x_{15}
0000 1000 0100 1100 0010 1010 0110 1011 0001 1001 0101 1101 0011 1011 0111 1111

Figure 11.30 Sixteen-point DFT decomposition.

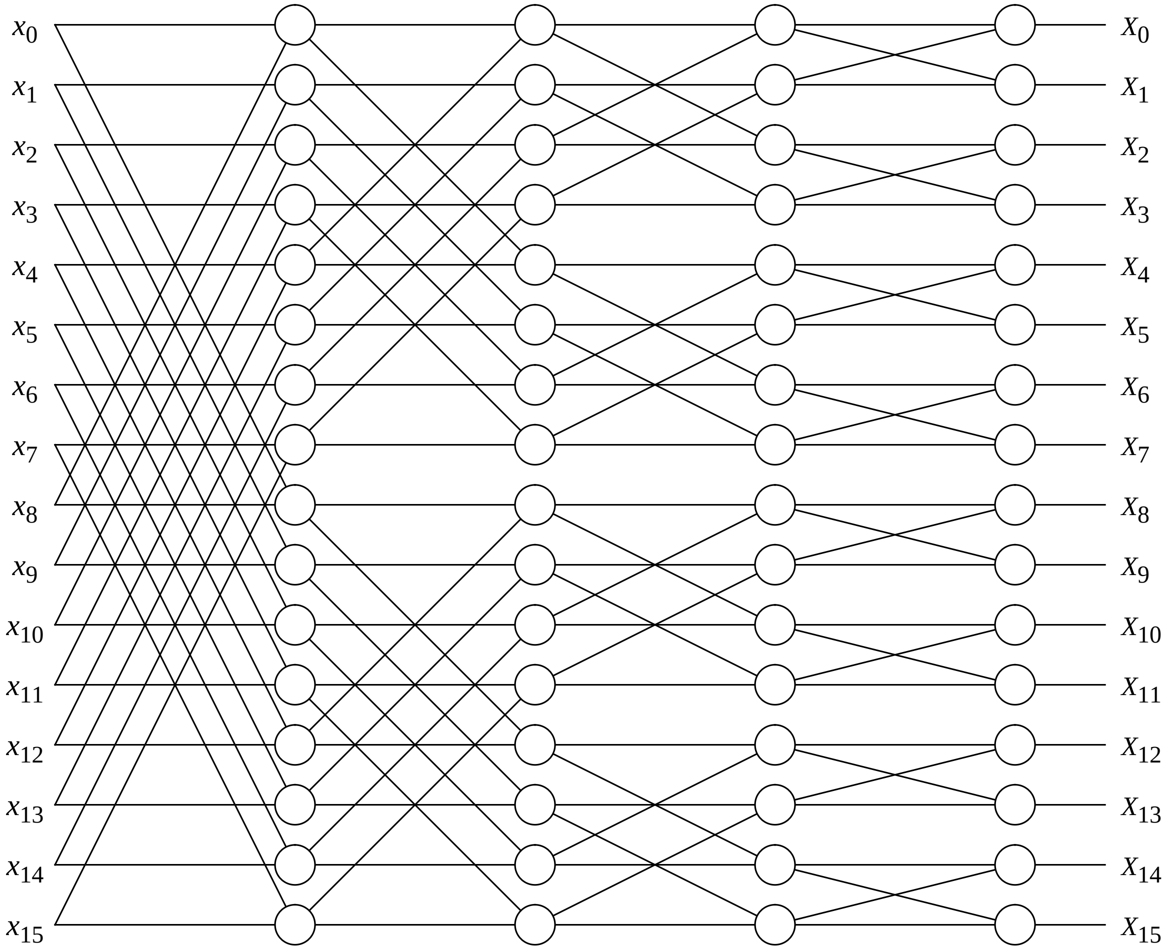


Figure 11.31 Sixteen-point FFT computational flow.

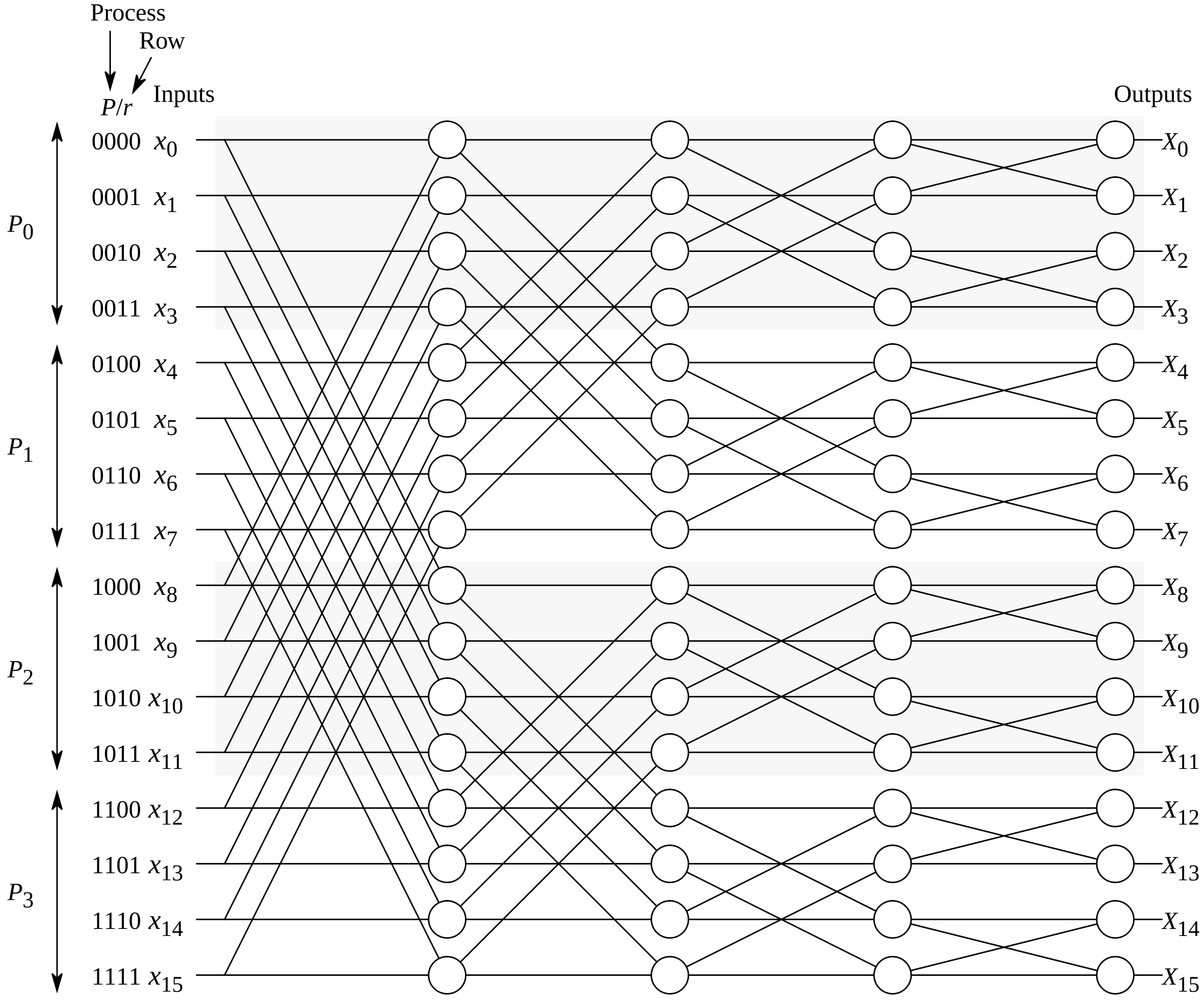


Figure 11.32 Mapping processors onto 16-point FFT computation.

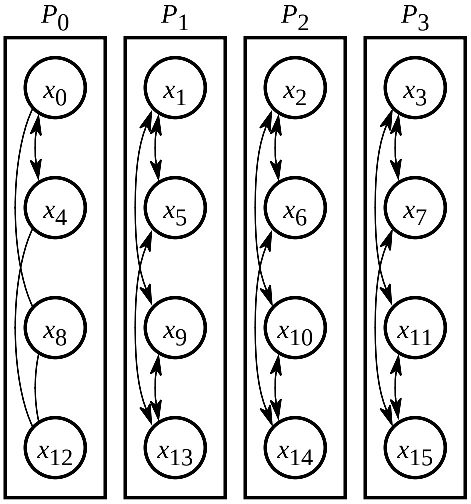


Figure 11.33 FFT using transpose algorithm — first two steps.

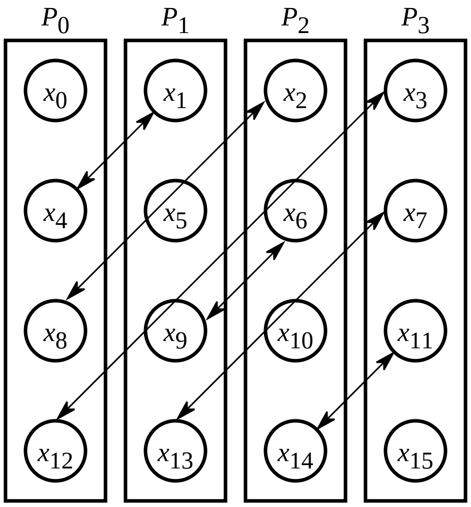


Figure 11.34 Transposing array for transpose algorithm.

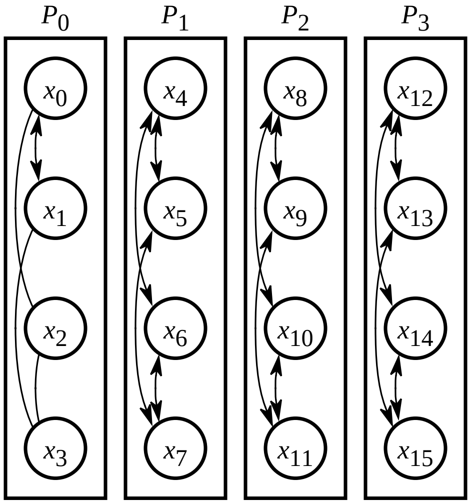


Figure 11.35 FFT using transpose algorithm — last two steps.

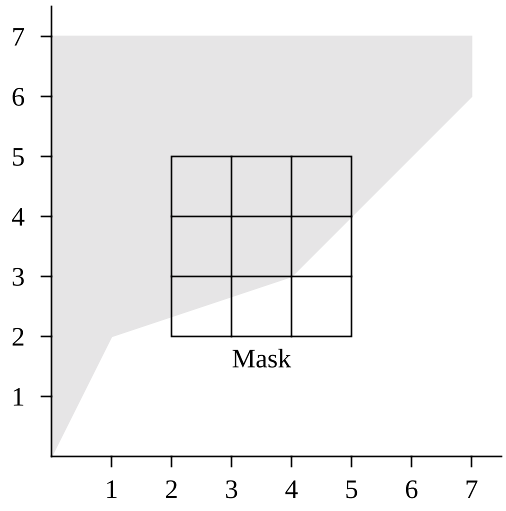


Figure 11.36 Image for Problem 11-3.

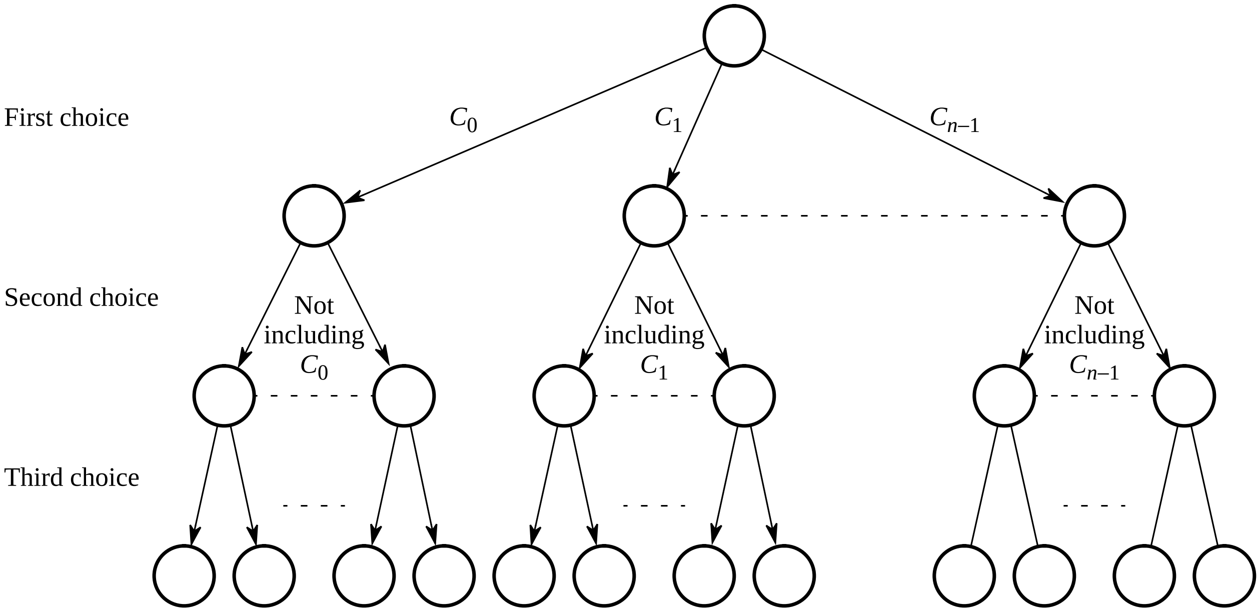


Figure 12.1 State space tree.

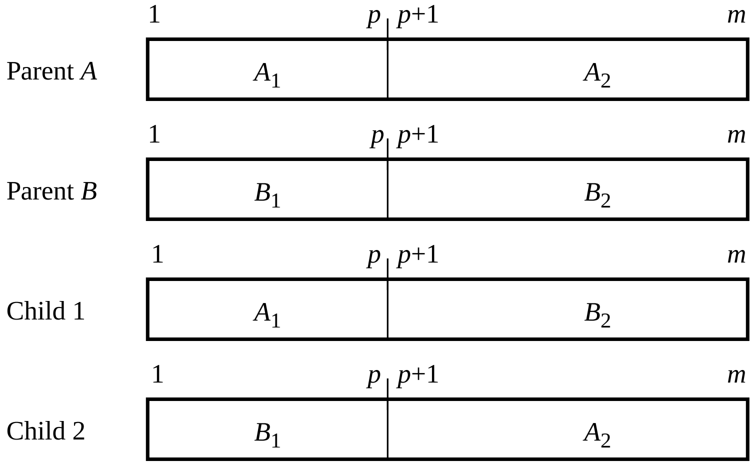


Figure 12.2 Single-point crossover.

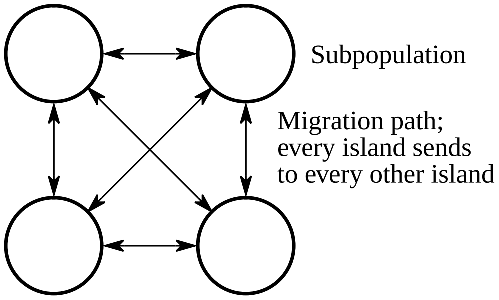


Figure 12.3 Island model.

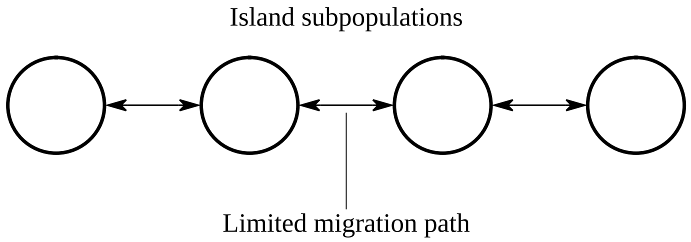


Figure 12.4 Stepping stone model

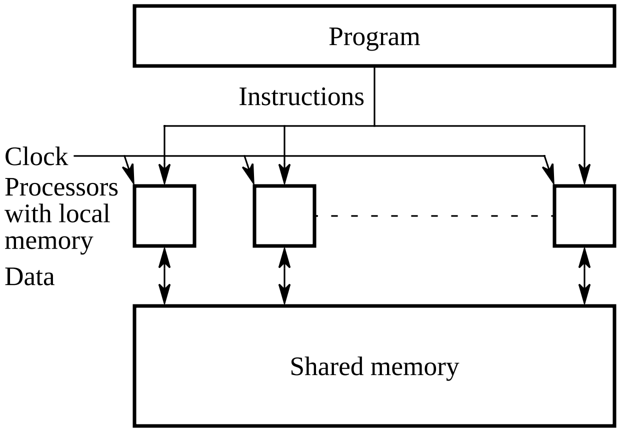


Figure D.1 PRAM model.

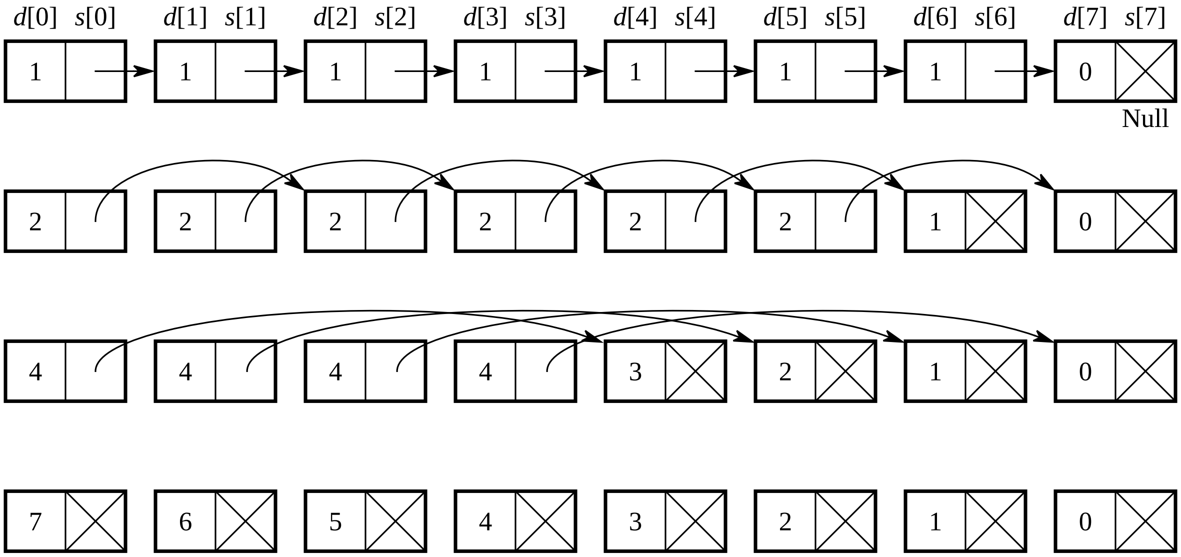


Figure D.2 List ranking by pointer jumping.

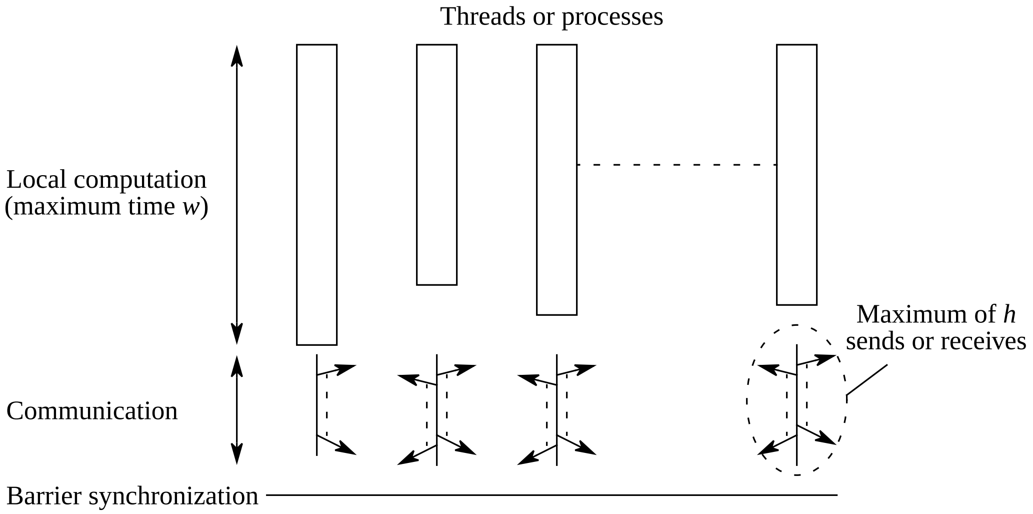


Figure D.3 A view of the bulk synchronous parallel model.

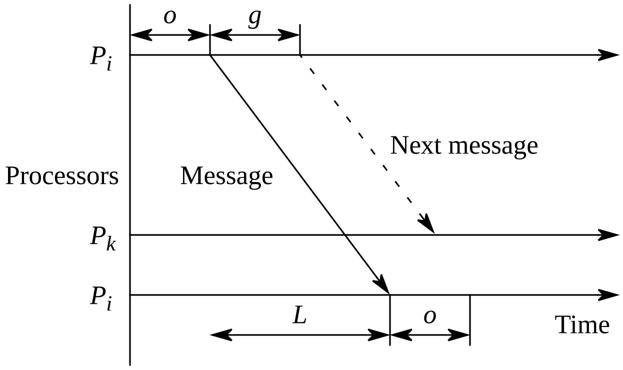


Figure D.4 LogP parameters.