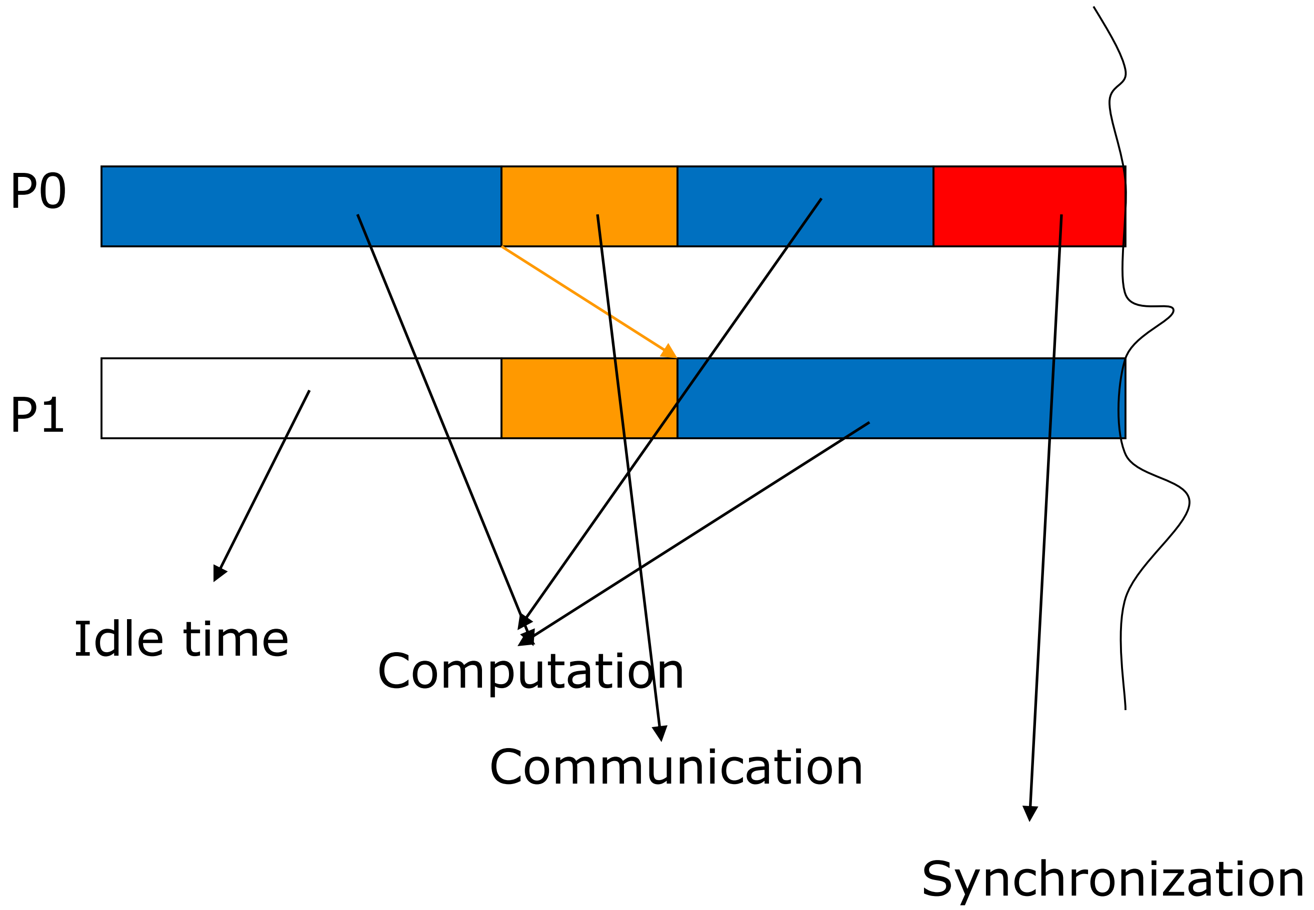
Parallel Programming Models

Sathish Vadhiyar
Department of Computational and Data Sciences
Supercomputer Education and Research Centre
Indian Institute of Science, Bangalore, India

Parallel Programming and Challenges

- Recall the advantages and motivation of parallelism
- But parallel programs incur overheads not seen in sequential programs
  - Communication delay
  - Idling
  - Synchronization

Challenges


P0
P1
Idle time
Computation
Communication
Synchronization

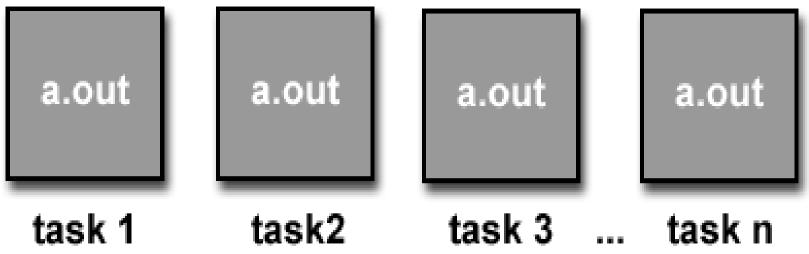
How do we evaluate a parallel program?

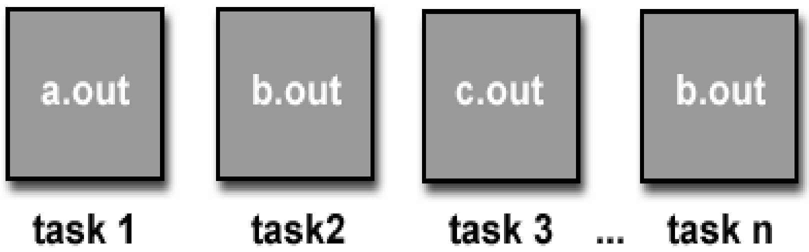
- Execution time, T_p
- Speedup, S
  - $S(p, n) = T(1, n) / T(p, n)$
  - Usually, $S(p, n) < p$
  - Sometimes $S(p, n) > p$ (superlinear speedup)
- Efficiency, E
  - $E(p, n) = S(p, n)/p$
  - Usually, $E(p, n) < 1$
  - Sometimes, greater than 1
- Scalability – Limitations in parallel computing, relation to *n* and *p*.

PARALLEL PROGRAMMING CLASSIFICATION AND STEPS

Parallel Program Models

- Single Program Multiple Data (SPMD)
- Multiple Program Multiple Data (MPMD)





Courtesy: http://www.llnl.gov/computing/tutorials/parallel_comp/

Programming Paradigms

- Shared memory model – Threads, OpenMP, CUDA
- Message passing model – MPI

Parallelizing a Program

Given a sequential program/algorithm, how to go about producing a parallel version

Four steps in program parallelization

1. Decomposition

   Identifying parallel tasks with large extent of possible concurrent activity; splitting the problem into tasks

2. Assignment

   Grouping the tasks into processes with best load balancing

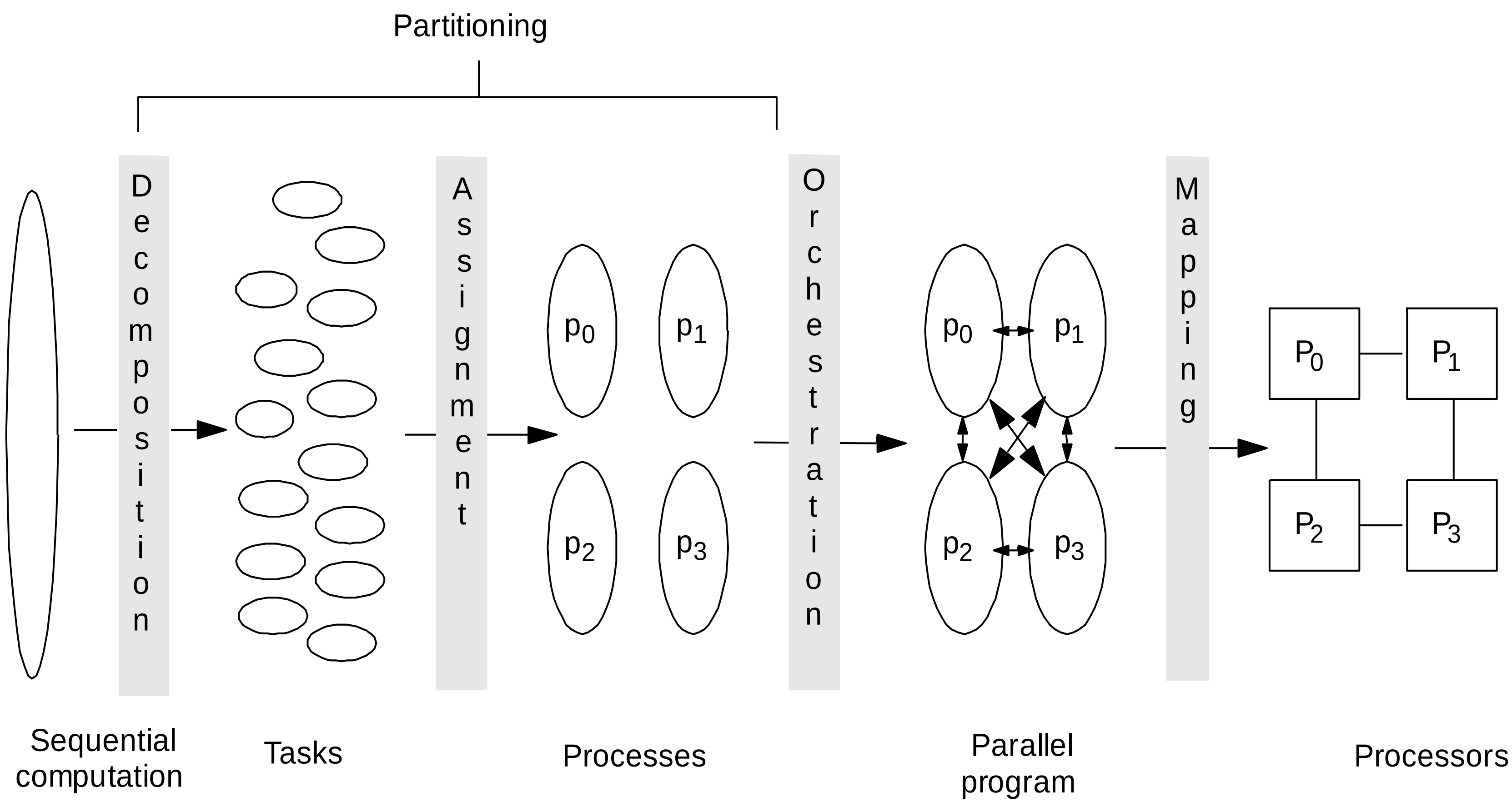
3. Orchestration

   Reducing synchronization and communication costs

4. Mapping

   Mapping of processes to processors (if possible)

Steps in Creating a Parallel Program



Decomposition and Assignment

- Specifies how to group tasks together for a process
  - Balance workload, reduce communication and management cost
- In practical cases, both steps combined into one step, trying to answer the question "What is the role of each parallel processing entity?"

Data Parallelism and Domain Decomposition

- Given data divided across the processing entitites
- Each process owns and computes a portion of the data – owner-computes rule
- Multi-dimensional domain in simulations divided into subdomains equal to processing entities
- This is called **domain decomposition**

Domain decomposition and Process Grids

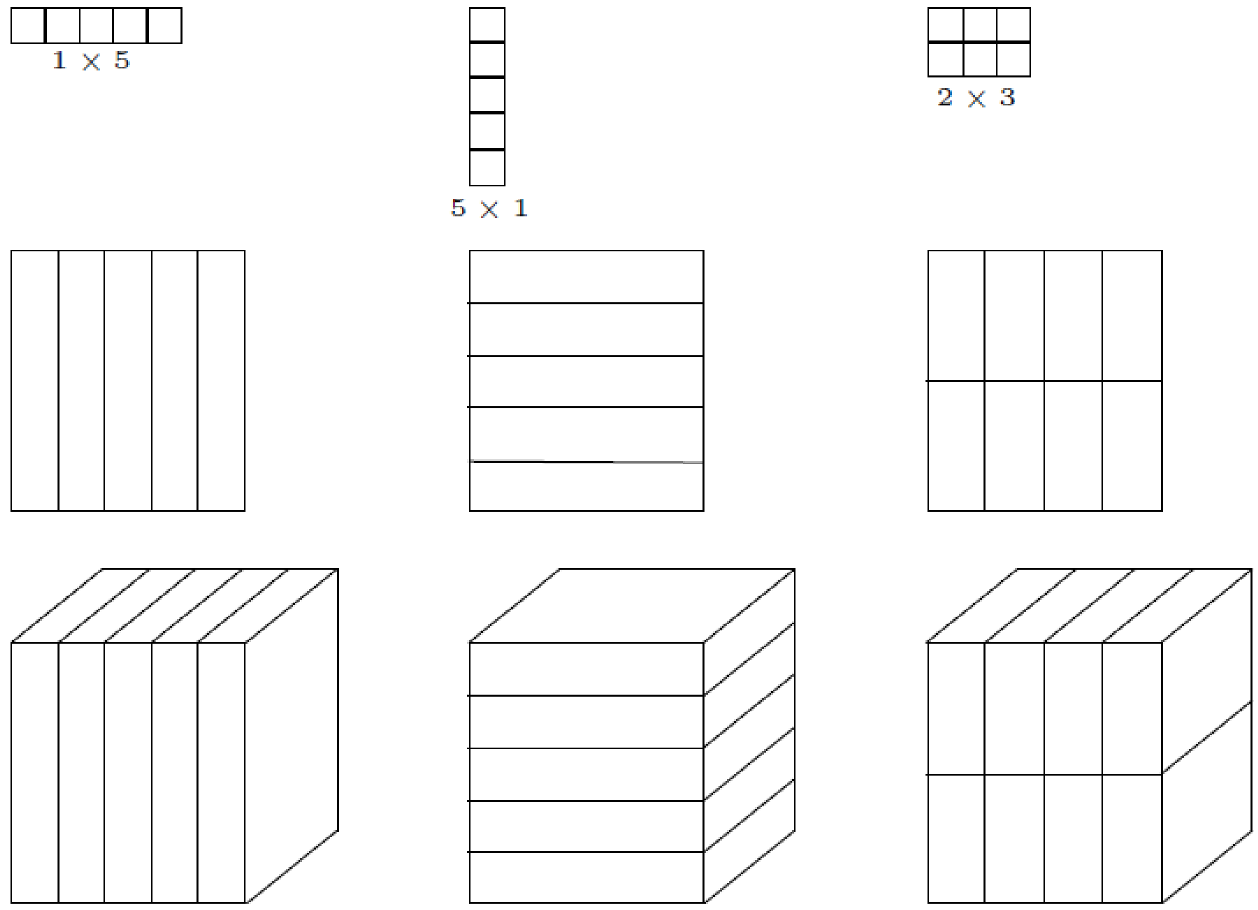
- The given P processes arranged in multi-dimensions forming a **process grid**
- The domain of the problem divided into process grid

Illustrations

Process grid

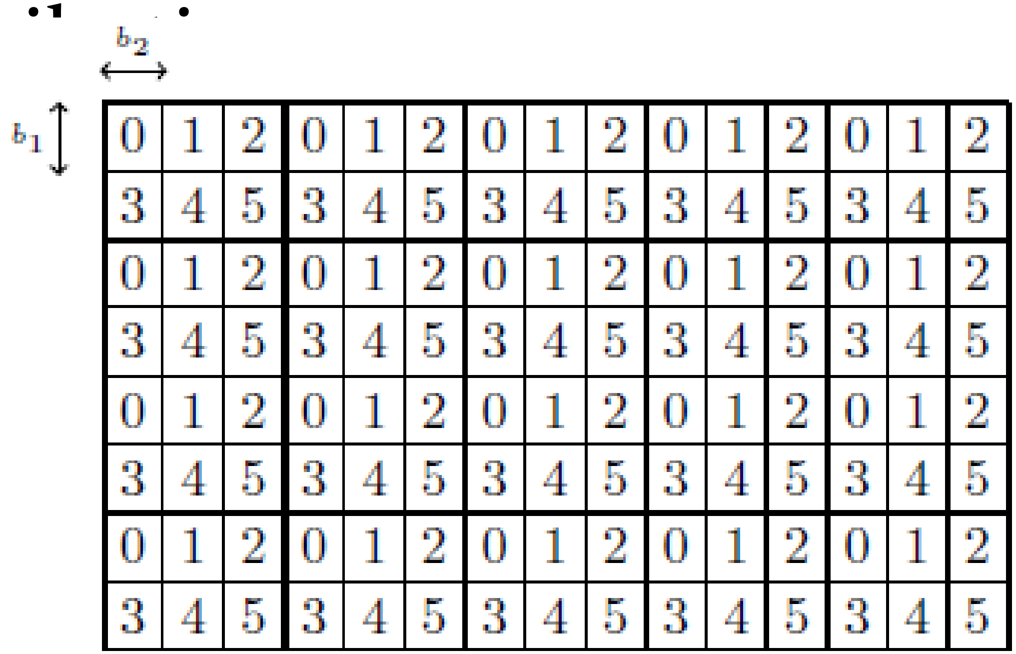
2-D domain decomposed using the process grid

3-D domain decomposed using the process grid



Data Distributions

- For dividing the data in a dimension using the processes in a dimension, **data distribution** schemes are followed
- Common data dist
  - Block: for regular computations
  - Block-cyclic: when there is load imbalance across space

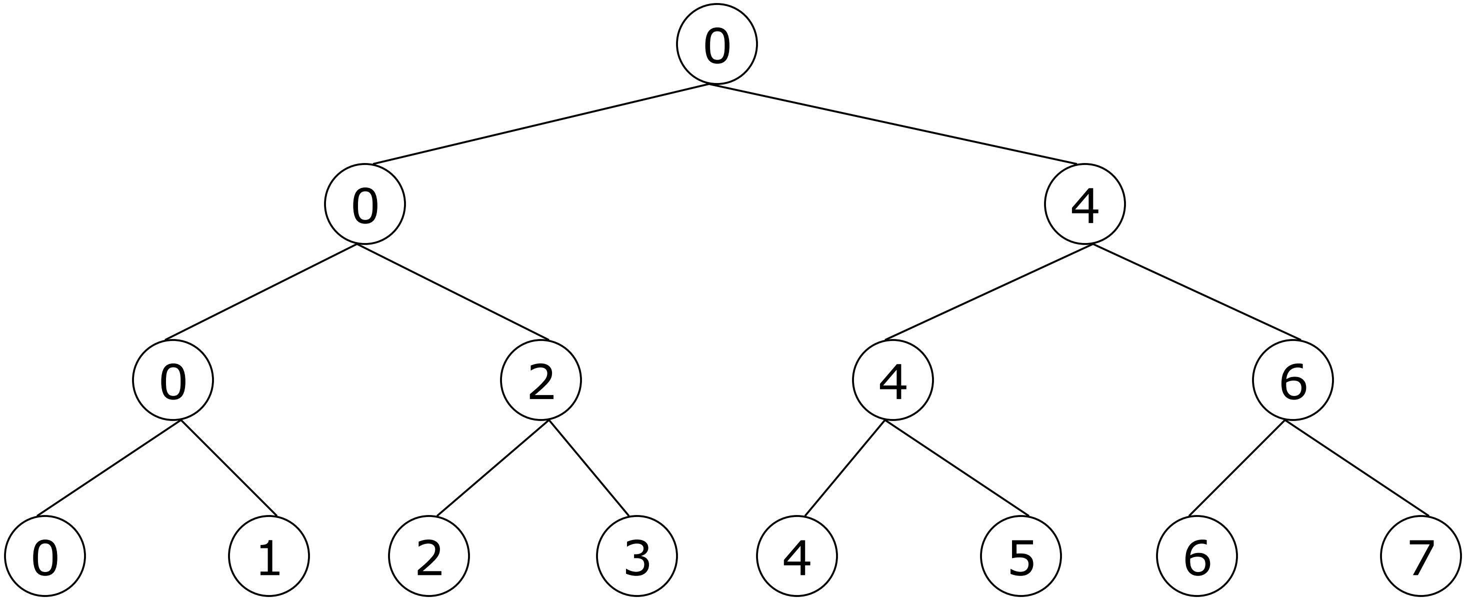


Task parallelism

- Independent tasks identified
- The task may or may not process different data

Based on Task Partitioning

- Based on **task dependency graph**



- In general the problem is NP complete

Orchestration

- Goals
  - Structuring communication
  - Synchronization
- Challenges
  - Organizing data structures – packing
  - Small or large messages?
  - How to organize communication and synchronization ?

Orchestration

- Maximizing data locality
  - Minimizing volume of data exchange
    - Not communicating intermediate results – e.g. dot product
  - Minimizing frequency of interactions - packing
- Minimizing contention and hot spots
  - Do not use the same communication pattern with the other processes in all the processes
- Overlapping computations with interactions
  - Split computations into phases: those that depend on communicated data (type 1) and those that do not (type 2)
  - Initiate communication for type 1; During communication, perform type 2
- Replicating data or computations
  - Balancing the extra computation or storage cost with the gain due to less communication

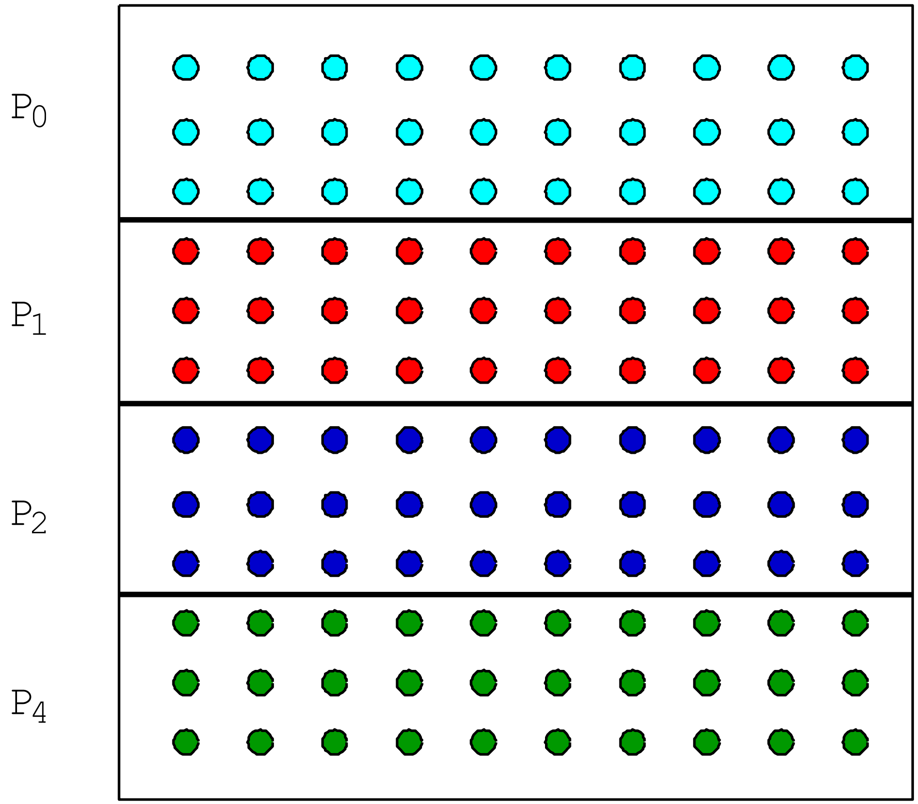
Mapping

- Which process runs on which particular processor?
  - Can depend on network topology, communication pattern of processes
  - On processor speeds in case of heterogeneous systems

Mapping

- Which process runs on which particular processor?
  - Can depend on network topology, communication pattern of processes
  - On processor speeds in case of heterogeneous systems

Assignment -- Option 3



Orchestration

- Different for different programming models/architectures
  - Shared address space
    - Naming: global addr. Space
    - Synch. through barriers and locks
  - Distributed Memory /Message passing
    - Non-shared address space
    - Send-receive messages + barrier for synch.

SAS Version – Generating Processes

```
1.   int n, nprocs;        /* matrix: (n + 2-by-n + 2) elts.*/
2.   float **A, diff = 0;
2a.  LockDec (lock_diff);
2b.  BarrierDec (barrier1);
3.   main()
4.   begin
5.       read(n) ;   /*read input parameter: matrix size*/
5a.      Read (nprocs);
6.       A ← g_malloc (a 2-d array of (n+2) x (n+2)  doubles);
6a.      Create (nprocs -1, Solve, A);
7.       initialize(A);     /*initialize the matrix A somehow*/
8.       Solve (A);         /*call the routine to solve equation*/
8a.      Wait_for_End (nprocs-1);
9.   end main
```

SAS Version -- Solve

```
procedure Solve (A)   /*solve the equation system*/
        float **A;              /*A is an (n + 2)-by-(n + 2) array*/
begin
        int i, j, pid, done = 0;
        float temp;
                mybegin  = 1 + (n/nprocs)*pid;
                myend = mybegin + (n/nprocs);
        while (!done) do     /*outermost loop over sweeps*/
          diff = 0;  /*initialize difference to 0*/
              Barriers (barrier1, nprocs);
            for  i ←  mybeg to myend do/*sweep for all points of grid*/
               for j ← 1 to n do
                    temp = A[i,j];          /*save old value of element*/
                    A[i,j] ← 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
                                A[i,j+1] + A[i+1,j]);   /*compute average*/
                    diff += abs(A[i,j] - temp);
               end for
            end for
            if (diff/(n*n) < TOL) then done = 1;
        end while
end procedure
```

SAS Version -- Issues

- SPMD program
- Wait_for_end – all to one communication
- How is diff accessed among processes?
  - Mutex to ensure diff is updated correctly.
  - Single lock $\Rightarrow$ too much synchronization!
  - Need not synchronize for every grid point. Can do only once.
- What about access to A[i][j], especially the boundary rows between processes?
- Can loop termination be determined without any synch. among processes?
  - Do we need any statement for the termination condition statement

SAS Version -- Solve

```
procedure Solve (A)   /*solve the equation system*/
            float **A;   /*A is an (n + 2)-by-(n + 2) array*/
begin
            int i, j, pid, done = 0;
            float mydiff, temp;
            mybegin  = 1 + (n/nprocs)*pid;
            myend = mybegin + (n/nprocs);
            while (!done) do          /*outermost loop over sweeps*/
           mydiff = diff = 0;          /*initialize local difference to 0*/
           Barriers (barrier1, nprocs);
                for  i ← mybeg to myend do/*sweep for all points of grid*/
                    for j ← 1 to n do
                              temp = A[i,j];               /*save old value of element*/
                              A[i,j] ← 0.2 * (A[i,j] + A[i,j-1] + A[i-1,j] +
                                          A[i,j+1] + A[i+1,j]);       /*compute average*/
                              mydiff += abs(A[i,j] - temp);
                      end for
                  end for
                  lock (diff-lock);
                  diff += mydiff;
                unlock (diff-lock)
       barrier (barrier1, nprocs);
                  if (diff/(n*n) < TOL) then done = 1;
       Barrier (barrier1, nprocs);
            end while
end procedure
```

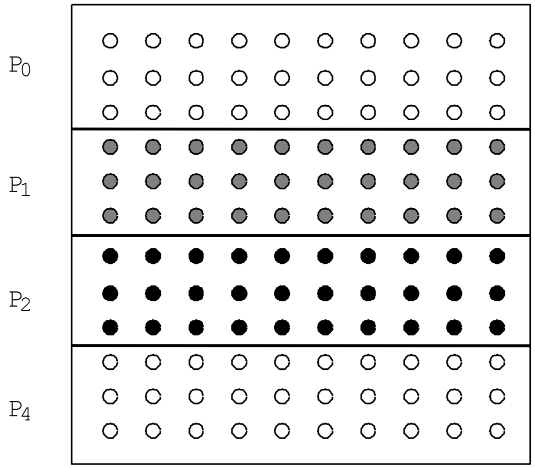
SAS Program

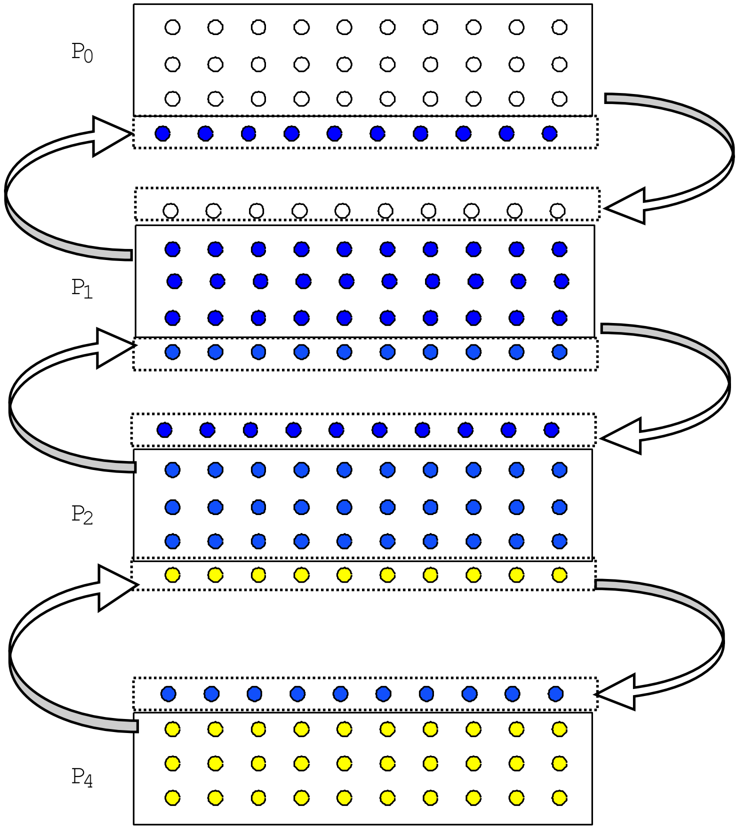
- **done** condition evaluated redundantly by all
- Code that does the update identical to sequential program
  - each process has private mydiff variable
- Most interesting special operations are for synchronization
  - accumulations into shared diff have to be mutually exclusive
  - why the need for all the barriers?
- Good global reduction?
  - Utility of this parallel accumulate??

Message Passing Version

- Cannot declare A to be global shared array
  - compose it from per-process private arrays
  - usually allocated in accordance with the assignment of work -- owner-compute rule
    - process assigned a set of rows allocates them locally
- Structurally similar to SPMD SAS
- Orchestration different
  - data structures and data access/naming
  - communication
  - synchronization
- Ghost rows

Data Layout and Orchestration





Data partition allocated per processor

Add ghost rows to hold boundary data

Send edges to neighbors

Receive into ghost rows

Compute as in sequential program

Message Passing Version – Generating Processes

```
1.    int n, nprocs;         /* matrix: (n + 2-by-n + 2) elts.*/
2.    float **myA;
3.    main()
4.    begin
5.        read(n) ;   /*read input parameter: matrix size*/
5a.       read (nprocs);
/* 6. A ← g_malloc (a 2-d array of (n+2) x (n+2) doubles); */
6a.       Create (nprocs -1, Solve, A);
/* 7.  initialize(A);          */ /*initialize the matrix A somehow*/
8.        Solve (A);         /*call the routine to solve equation*/
8a.       Wait_for_End (nprocs-1);
9.    end main
```

Message Passing Version – Array allocation and Ghost-row Copying

```
10.   procedure Solve (A)   /*solve the equation system*/
11.        float **A;             /*A is an (n + 2)-by-(n + 2) array*/
12.   begin
13.        int i, j, pid, done = 0;
14.        float mydiff, temp;
14a.       myend = (n/nprocs) ;
6.         myA = malloc  (array of (n/nprocs) x n floats );
7.        initialize (myA);   /* initialize myA LOCALLY */
15.        while (!done) do     /*outermost loop over sweeps*/
16.        mydiff = 0;            /*initialize local difference to 0*/
16a.     if (pid != 0) then
                 SEND (&myA[1,0] , n*sizeof(float), (pid-1), row);
16b.    if (pid != nprocs-1) then
                 SEND (&myA[myend,0], n*sizeof(float), (pid+1), row);
16c.    if (pid != 0) then
                 RECEIVE (&myA[0,0], n*sizeof(float), (pid -1), row);
16d.    if (pid != nprocs-1) then
                 RECEIVE (&myA[myend+1,0], n*sizeof(float), (pid -1),
                                  row);
```

Message Passing Version – Solver

```
begin
… … …
        while (!done) do          /*outermost loop over sweeps*/
 … … …
            for i ← 1 to myend do/*sweep for all points of grid*/
                for j ← 1 to n do
                        temp = myA[i,j];          /*save old value of element*/
                        myA[i,j] ← 0.2 * (myA[i,j] + myA[i,j-1] +myA[i-1,j] +
                                    myA[i,j+1] + myA[i+1,j]);   /*compute average*/
                        mydiff += abs(myA[i,j] - temp);
                end for
            end for
            if (pid != 0) then
                SEND (mydiff, sizeof (float), 0, DIFF);
                RECEIVE (done, sizeof(int), 0, DONE);
            else
                for k ← 1 to nprocs-1 do
                        RECEIVE (tempdiff, sizeof(float), k   ,  DIFF);
                        mydiff += tempdiff;
                endfor
                If(mydiff/(n*n) < TOL) then done = 1;
               for k ← 1 to nprocs-1 do
                        SEND (done, sizeof(float), k   ,  DONE);
                endfor
        end while
end procedure
```