Data Distribution

- Goal: distribute each array in a program so that completion time is minimized
- Minimizing completion time has two components
  - Minimize communication
  - Balance the computational workload

By "distribute" we mean finding a function that maps each array element to a node

Owner Computes Rule

- Each data element has an owner
  - Owner is responsible for all writes to that element
  - All other nodes can only reference the element
- Owner computes rule is most common strategy used
  - Exception: applications with irregular access patterns---not always possible to use this rule
    - E.g., indirection arrays---may not know what elements are being written until run time

Modeling Completion Time

Assume f(i) makes no array accesses, and n+1 elements in A, B

```
for i = 1 to n
   A[i] = B[i+1] + f(i)
```

Transformed to a parallel loop as follows:

```
for i = start to end by step
   A[i] = B[i+1] + f(i)
```

Modeling completion time, cont.

for i = start to end by step

A[i] = B[i+1] + f(i)

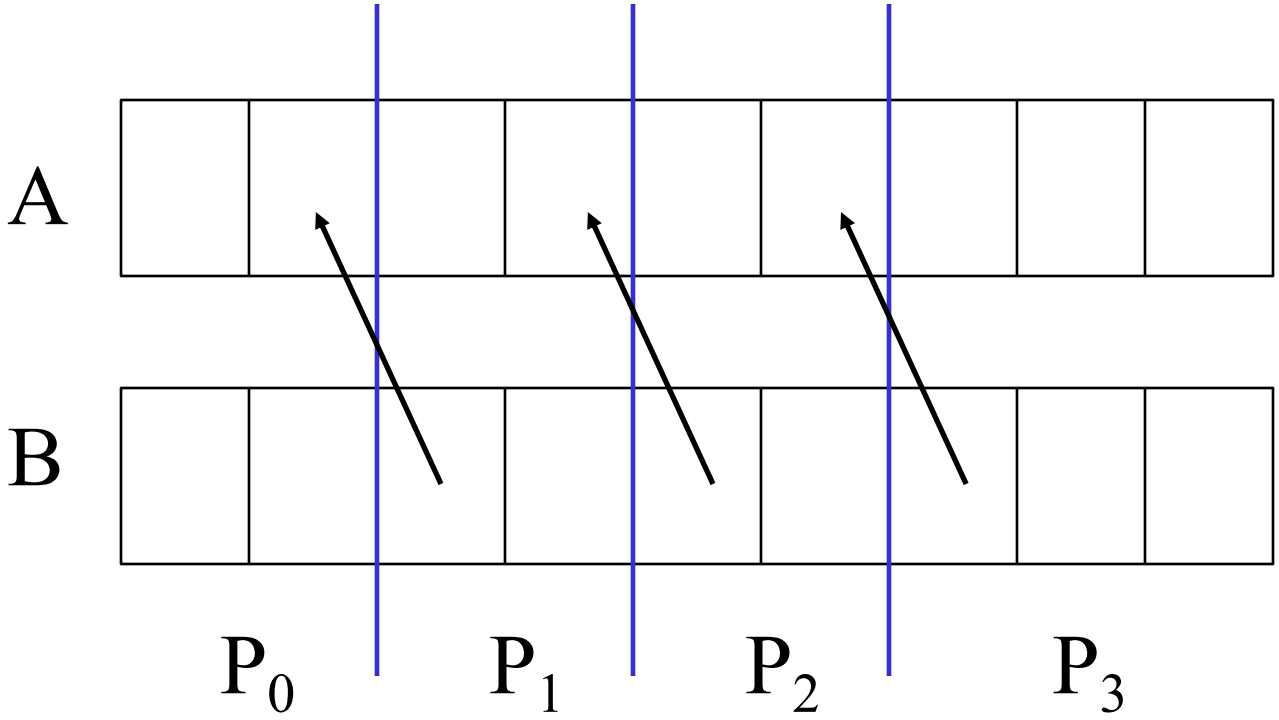
- Time for iteration i:
  - Time to compute f(i) + time to load B[i+1] + time for an addition + time to store A[i] + *time for communication, if any*
- Given a data distribution, we can determine what communication is needed

Modeling completion time, cont.

// Suppose A, B distributed as {BLOCK}

for i = start to end

A[i] = B[i+1] + f(i)



Modeling completion time, cont.

```
// Suppose A, B distributed as {BLOCK}
for i = start to end
   A[i] = B[i+1] + f(i)
```

- Communication will occur at the boundaries
- Communication time for iteration *i* is zero, except if *i* is equal to *start or end*
  - In the boundary case, an element of B must be sent to the previous node
  - So, *i* == *start* implies start; *i* == *end* implies receive
    - Unless process is the first or last

Modeling completion time, cont.

- What options do we have to determine the time for *f*(i), the load, addition, and store?
  - Statically analyze with a cost model
  - Execute and profile
  - Accept input from the programmer
  - Maybe it doesn't even matter
    - If computation is uniform across nodes
    - But this isn't always the case; *f*(i) might be non-uniform, i.e., dependent on *i*

Modeling completion time (in general)

- Each node's completion time is the sum of the times for all iterations that it performs
- Overall completion time is then the maximum completion time over the nodes

Group Exercise

- In groups: how do we express the optimal data distribution, given:
  - P is the number of nodes
  - D is the set of possible distributions
  - $I(p,d)$ is the set of iterations on a given timestep performed by node p when using distribution d
  - $C(i)$ is the completion time of iteration i

Group Exercise

- In groups: how do we express the optimal data distribution, given:
  - *P* is the number of nodes
  - *D* is the set of possible distributions
  - *I(p,d)* is the set of iterations on a given timestep performed by node *p* when using distribution *d*
  - *C(i)* is the completion time of iteration *i*

$$\min_{d \in \mathcal{D}} \left(\max_{p \in \mathcal{P}} \left(\sum_{i=1}^{I(p,d)} C(i) \right) \right)$$

Optimal Data Distribution

- Defined as the data distribution that leads to the smallest overall completion time
- Min-max problem
- NP-complete
  - Intuitively, there are “quite a few” ways to partition the work and the data

Good News

- In practice, there is often an efficient data distribution that can be easily found
  - *f*(i) may be independent of *i*, and communication restricted to the boundaries
    - For example, in your Jacobi program, *f*(i) is essentially computing the average, and there was communication only at the top/bottom rows
  - Maybe there is even *no* communication
    - For example, on matrix multiplication, there is no communication after the initial distribution

Possibly Bad News

- In practice, programs may have multiple phases; each phase may have a different optimal distribution
  - Need to consider *data redistribution* in such cases

Example

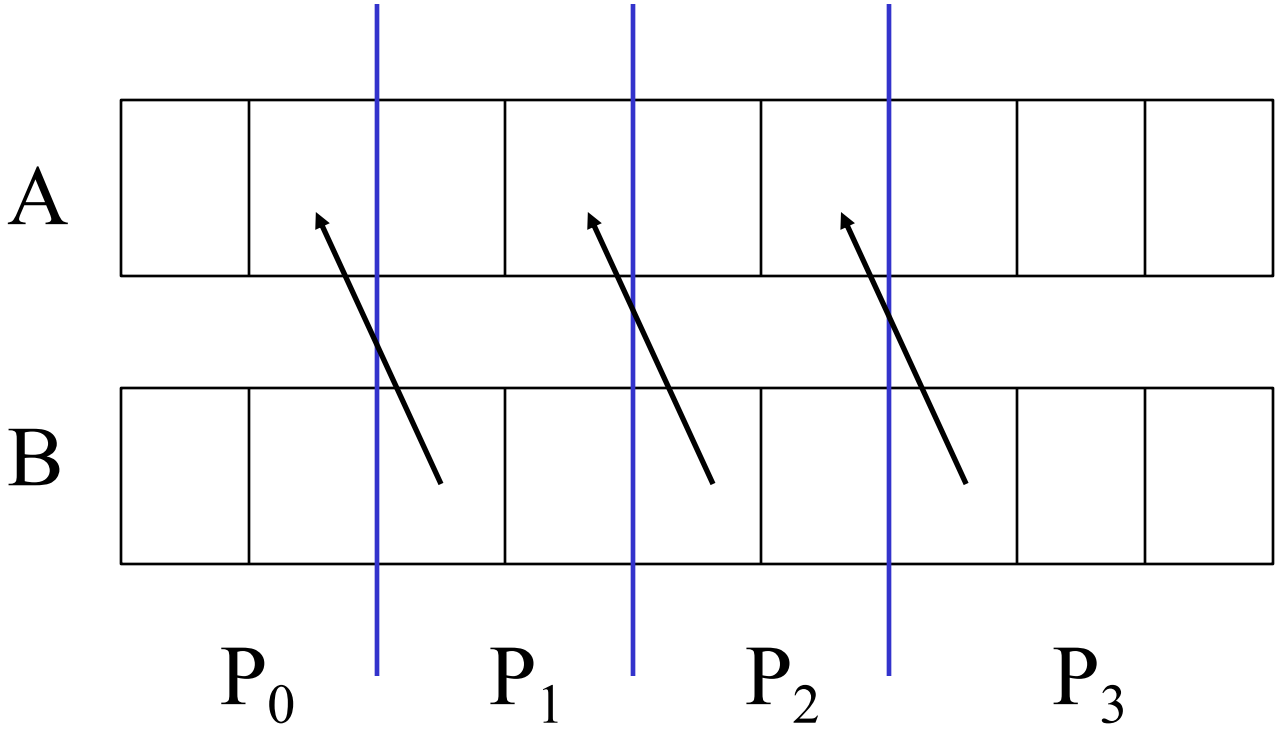
```
for t = 1 to timesteps {
  for i = 1 to n
    A[i] = B[i+1] + f(i)  // f(i) accesses no arrays

  for i = 1 to n
    B[i] = A[i] + g(i)  // g(i) accesses no arrays
}
```

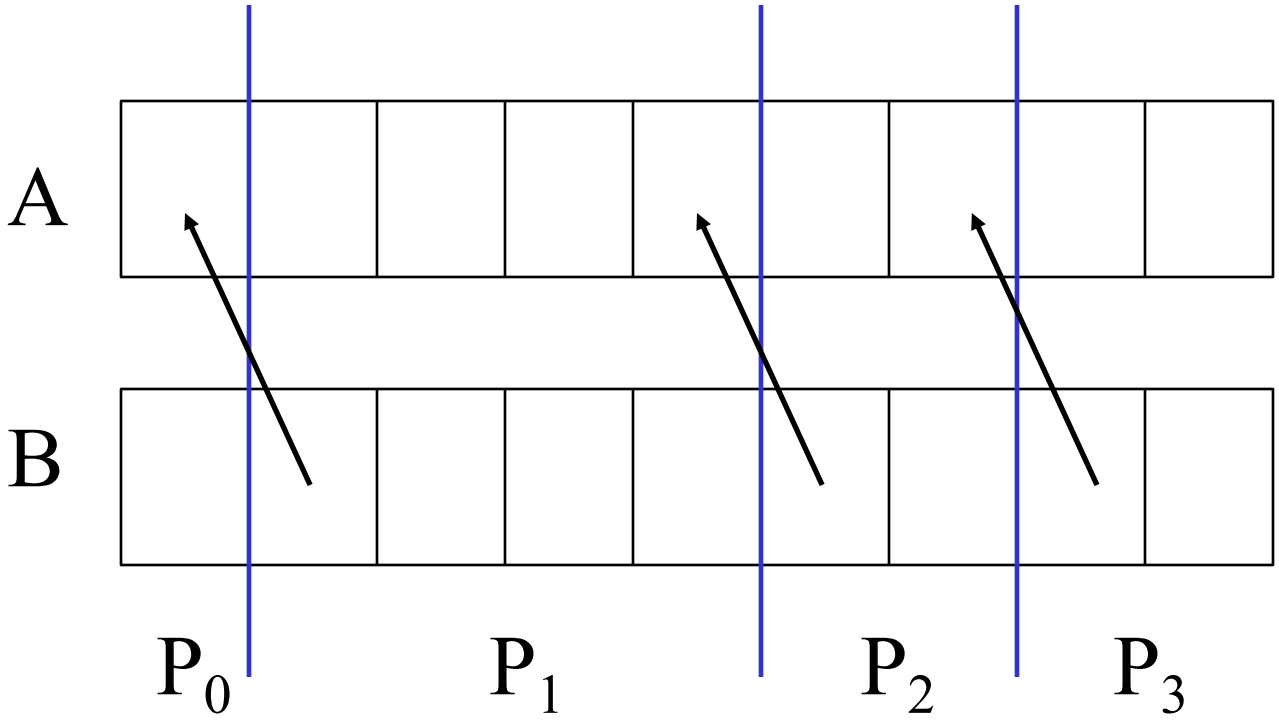
Example (parallelized)

```
for t = 1 to timesteps {
  for i = start to end by step
     A[i] = B[i+1] + f(i)  // f(i) accesses no arrays
  if (distribution needs to change)
     Redistribute( )
  for i = start' to end' by step'
     B[i] = A[i] + g(i)  // g(i) accesses no arrays
  if (distribution needs to change)
     Redistribute( )
}
```

Assume that BLOCK is optimal for the first phase (loop)



Assume that this “IRREGULAR BLOCK” is optimal for the second phase (loop)



How do we decide distributions per phase?

- First, we can model completion time for each loop in exactly the same way as before
- We have choices:
  - Optimal distribution per phase, with redistribution
    - In our example, BLOCK in first phase and IRREGULAR BLOCK in second phase
  - One (identical) distribution for both phases, and that distribution is optimal in one of the phases
    - In our example, *either* BLOCK *or* IRREGULAR BLOCK used in *both* phases
  - Suboptimal (identical) distribution for each phase
    - In our example, a different IRREGULAR BLOCK than the one shown on the previous slide

Group Exercise

- How do we decide between the options listed on previous slide? To recap, the options are:
  - Optimal distribution per phase, with redistribution
  - One (identical) distribution for both phases, which is optimal in one of the phases
  - Suboptimal (identical) distribution for each phase

How do we decide distributions per phase?

- Depends on several factors
  - Communication latency and bandwidth
    - Faster network makes redistribution more efficient
  - Computation speed
    - Faster processor makes the redistribution relatively more expensive
  - Amount of data that needs to be communicated
    - More data makes redistribution more expensive
  - How much computation is implied by the code (i.e., computation-to-communication ratio)
    - Higher ratio means balancing load is more important