Parallel Programming Model

- Shared memory
  - Writes to a shared location are visible to all
- Distributed memory
  - No shared memory; data is partitioned and must be shared through explicit communication (message passing)
- Problems
  - Additional challenges to programmers: explicit communications, data partitioning
  - Slower inter-process communications
- Why distributed memory parallel programming?

Why Distributed Memory Parallel Programming?

- Less requirement on hardware support
  - Cluster of machines connected by commodity network
  - Better take advantage of newest processors
- More portable
  - Can run on shared memory multiprocessors
  - Can run on hybrid distributed/shared-memory platforms
- No worry of multiprocessor memory consistency
  - No memory consistency-related errors
- Better locality of parallel programs
  - Programmer knows her/his program best

Parallel Programming Steps

- Converting a sequential application to a parallel one

  - Decomposition into tasks
  - Assign tasks to processors
    - Also partition the data
  - Orchestrate data access and synchronization
    - Orchestrate message passing and synchronization
### Message Passing Interface

- **De facto standard programming interface for message passing-based parallel programs**
  - think of pthread for shared memory parallel programming
- You write a single program, multiple copies of which will run on multiple processors
  - Assumption: all processes do mostly similar things
  - Different parts distinguish through process ID
- Communications
  - Point-to-point: send/receive
  - Group communications: broadcast, gather, scatter, reduce, barrier
- It is a programming interface, not an implementation specification!

### MPI Send/Receive

- Matching send/receive:
  - Process x sends a message to process y
  - Process y receives a message from process x
- Nonblocking receive
- Nonblocking send
- Synchronous send

### MPI Group Communications

- Barrier
  - All processes wait until all have arrived
- Broadcast
  - One process (root) sends a message to be received by others
- Reduce
  - A function (MAX, SUM, …) is applied to data supplied by all processes; result is returned at one process (root)
  - Function is evaluated following process rank order
  - Can be optimized if associative and/or commutative
- Group communication lead to group wait
  - only when necessary, on smallest group necessary

### Performance Issues

- Load balance (dynamic task assignment difficult)
  - careful data partition and task assignment
- Synchronization/communication wait
  - block as late as possible
- Long communication latency (vs. high bandwidth)
  - our Ethernet cluster: 250us latency, 80MB/sec bandwidth
  - if 1KB per synchronization, effective bw is 4MB/sec
  - synchronize/wait as few times as possible
- Group communication lead to group wait
  - only when necessary, on smallest group necessary
Deadlocks

- Why does my MPI program gets stuck?
- Possible reasons
  - Receive without a matching send (or a matching send cannot be reached)
  - Group communications are not called by all in the group
  - Send blocked by insufficient buffer space
- Debugging
  - First find out where each process is blocked at
  - How does it conflict with design? What’s wrong with implementation?

Parallel Programming Example: Successive Over Relaxation

for (i=1; i<n; i++)
for (j=1; j<n; j++)
temp[i][j] = 0.25 * (grid[i-1][j]+grid[i+1][j]+grid[i][j-1]+grid[i][j+1]);
for (i=1; i<n; i++)
for (j=1; j<n; j++)
gird[i][j] = temp[i][j];

Parallel Programming Example: Successive Over Relaxation

for (i=1; i<n; i++)
for (j=1; j<n; j++)
tmp[i][j] = 0.25 * (grid[i-1][j]+grid[i+1][j]+grid[i][j-1]+grid[i][j+1]);
for (i=1; i<n; i++)
for (j=1; j<n; j++)
gird[i][j] = tmp[i][j];

Gaussian Elimination

- Reduce an equation matrix into an equivalent upper-diagonal matrix
- Partial pivoting to maintain numerical stability

Task decomposition and data partitioning:
- Communication/synchronization in row partitioning
- Find the maximum pivot, distribute the major row
- Communication/synchronization in column partitioning
- Distribute the major column
Gaussian Elimination

- Reduce an equation matrix into an equivalent upper-diagonal matrix
- Partial pivoting to maintain numerical stability

\[ \mathbf{A} \mathbf{x} = \mathbf{b} \]

- Task decomposition and data partitioning:
  - Block vs. cyclic partitioning?