Parallel Sorting
Collective Communication
Interconnect

Parallel Sorting

• Sorting is fundamental algorithm in data processing
  – Given an unordered set of keys $x_0, x_1, \ldots, x_{N-1}$
  – Return the keys in sorted order
• The keys may be character strings, floating point numbers, integers, or any object for which the relations $>$, $<$, and $==$ hold
• We’ll assume integers here
• Will talk about other algorithms later on
Compare and exchange sorts

• Simplest sort, based on the bubble sort algorithm
• The fundamental operation is compare-exchange
• Compare-exchange\( (x, y) \)
  – swaps its arguments if they are in decreasing order
  – returns FALSE if a swap was made
  – satisfies the post-condition that \( x \leq y \)

Compare and exchange sorts

• We cannot parallelize bubble sort in parallel owing to the *loop carried dependence* in the inner loop
• The value of \( a[j] \) computed in iteration \( j \) depends on the \( a[i] \) computed in iterations \( 0, 1, ..., j-1 \)

```plaintext
for i = N-1 to 1 by -1 do
    done = TRUE;
    for j = 0 to i-1 do
        done = Compare-exchange(a[j], a[j+1])
    end do
    if (done) break;
end do
```
Odd/Even sort

- If we re-order the comparisons we can parallelize the algorithm
  - number the points as even and odd
  - alternate between sorting the odd and even points
- This removes the no loop carried dependences
- All the odd (even) points are decoupled

Diagram:

\[ a_{i-1} \ a_i \ a_{i+1} \]

The algorithm

```
done = TRUE;
for i = 0 to n−1 do
    for j = 0 to n−1 by 2 do // Even
        done &= Compare-exchange(a[j], a[j+1]);
    end do
    for j = 1 to n−1 by 2 do // Odd
        done &= Compare-exchange(a[j], a[j+1]);
        if (done) break;
    end do
end do
```
Partitioning the data

• Partitioning is the process of splitting up the data
• We partition the data into intervals, assigning each to a unique processor

Data dependences

• The right end of each interval depend on a value “owned” by the left end of the succeeding interval
• We add “overlap” or “ghost” cells to hold a copy of the value
• Before each sorting sweep pass messages to refresh the copies
• We can use the original serial loop without having to put communication inside
Odd Even Transposition sort

- While odd/even sort parallelizes, it has a long running time
- Only one data element at a time is swapped between neighboring processes
- A more efficient algorithm moves blocks of data at each step
- Uses the odd/even ordering, but this time over block numbers (process IDs)
- The fundamental operation is a block compare-swap

The algorithm

- As a pre-processing step, each processor locally sorts its data using a fast serial algorithm like quicksort
- Processes exchange chunks in odd-even pairs using block compare and swap
- Each process applies a local merge sort to extract the smallest (largest) N/P values, discards the rest
Odd-even merge sort in action

N values to be sorted
Treat as four lists of M = N/4
Sort each separately
Compare and swap

Compare and swap

Final sorted list


Block compare and swap

-1 3 7 9 11
2 4 8 12 14

Processor 0 Processor 1

• Compare and swap
  – Each processor swaps data with its neighbor
    -1 3 7 9 11 2 4 8 12 14
  – Sorts the merged list
    -1 2 3 4 7 8 9 11 2 14
  – Processor 0 takes 5 smallest values: -1 2 3 4 7
  – Processor 1 takes 5 largest values: 8 9 11 12 14

The algorithm

done = false;

for i = 0 to P-1 do
    for p = 0 to P-1 by 2 do  // Even
        done &= Block-Compare-exchng(A_p, A_{p+1});
    end do

    for p = 1 to P-1 by 2 do  // Odd
        done &= Block-Compare-exchng(A_p, A_{p+1});
    end do
    if (done) break;
end do

Performance of odd-even merge sort

• There are at most P passes
• Normalize communication cost parameters $\alpha$ and $\beta$ to the cost of performing a compare
• Cost of a swap: $\alpha + \beta(N/P)$
• Running time is
  \[ (N/P)\log(N/P) + P(\alpha + \beta(N/P) + N/P) \]

Local sort swap merge
Collective Communication

Collective communication

• In the serial sorting algorithm we maintain a flag telling us when the sort has completed
• In a parallel sort, all local done flags must be set before the global sort has completed
• In the integration algorithm of the previous lecture, we used a simple linear time algorithm to accumulate of the local sums with point to point communication
• A more convenient abstraction is to view this communication as a global operation
• We use collective communication
Collective communication in MPI

- Collective operations are called by all processes in a communicator
  - Broadcast: distribute data from one process (the root) to all others in a communicator.
    \[ \text{MPI\_Bcast(buffer, count, datatype, root, comm);} \]
  - Reduce: combine data from all processes in communicator and returns it to one process.
    \[ \text{MPI\_Reduce(in, out, count, datatype, operation, root, comm);} \]
- Information flow in Reduce is the reverse of the information flow in a broadcast

Using collective communication in sorting

- Everyone needs to know everyone is done sorting
- To do this, we reduce the local flags, and then broadcast the result
- This operation is known as \text{Allreduce()}\()
- Everyone obtains a copy of the reduced result
  \[ \text{MPI\_Allreduce (&l\_done, &done, 1, MPI\_INT, MPI\_SUM, WORLD)} \]
Broadcast

- In a broadcast, one processor has \( m \) pieces of data to send to the \( p-1 \) other processors
- With the linear algorithm approach this processor performs \( p-1 \) sends of length \( m \)
  - Cost is \((p-1)(\alpha + \beta m)\)
- Another approach is to use the hypercube algorithm, which has a logarithmic running time

What is a hypercube?

- A hypercube is a \( d \)-dimensional graph with \( 2^d \) nodes
  - A 0-cube is a single node
  - A 1-cube is a line connecting two points
  - A 2-cube is a square, etc
Properties of hypercubes

- A hypercube with \( p \) nodes has \( \log_2(p) \) dimensions
- Inductive construction: we may construct a \( d \)-cube from two \((d-1)\) dimensional ones, connecting corresponding nodes

Node numbering

- We label node using a binary reflected grey code; see [http://www.nist.gov/dads/HTML/graycode.html](http://www.nist.gov/dads/HTML/graycode.html)
- Each node has \( d \) neighbors
- A neighbor’s label differs in exactly one bit position
Hypercube broadcast algorithm

- 2-cube case
- Processor 0 is the root

Hypercube exchange

- Processor 0 sends its data to its hypercube “buddy:” processor 2
- We repeat on the other hypercube dimension
Hypercube exchange

- Processor 0 and 2 send data to its their respective “buddies”

All Reduce

- We may use the hypercube algorithm to perform reductions as well as broadcasts
- Consider
  
  `Allreduce()`

- A clever algorithm performs an Allreduce in one phase rather than having perform separate reduce and broadcast phases
Interconnection network characteristics

- **Diameter**: maximum distance between any 2 points in the network
- **Bisection bandwidth**: collective bandwidth between two “halves” of the network; split the graph into two equal parts and measure the capacity of the cut edges

Some important interconnects

- Linear array
- What is the diameter?
- Bisection bandwidth?
- Broadcast running time
**Mesh**

- The Mesh is a $\sqrt{P} \times \sqrt{P}$ array
- Diameter is $2\sqrt{P}$
- Bisection bandwidth?
- Broadcast algorithm?

**Ring**

- P element array with end-around connection
- Diameter is cut in half
- Bisection bandwidth doubles
- Broadcast running time?
Toroidal mesh

- End around connections on rows and columns
- Diameter?
- Bisection bandwidth?
- Broadcast running time?

Crossbar

- Expensive – all points connected
- Diameter?
- Bisection bandwidth?
- Broadcast algorithm?
Multi-stage networks

- Switching is performed in stages
- Sometimes the stages are inductively constructed
- There are usually redundant paths

An Omega Network

- The network is constructed from switch modules
- A module can swap the inputs, or pass them through unchanged
Switch contention

- If two messages require the same modules at the same time, they contend for that module
- Performance penalty, since access is serialized