Lecture 10

Midterm review

Announcements

• Assignment #3 due today in class
• Assignment #4 has been posted
  Sorting
• Midterm will be given Tuesday in class
  – Closed book, closed notes
  – Calculators will not be permitted in the exam.
  – You are responsible for all material covered
    through today, including all assigned readings
    and the homework
Problem set: problem #1

- d-dimensional hypercube
- Hamming distance
  \[ H(A,B) = \# \text{ of non-zero bits in } A \otimes B \]
- \( P(A,B) \) Number of parallel paths (non-overlapping) between A and B
- Prove the following
  1. \( H(A,B) = \# \text{ hops in a path from } A \text{ to } B \)
  2. \( d = \text{total } \# \text{ of parallel paths between } A \text{ & } B \)
  3. \( H(A,B) = \# \text{ parallel paths of length } H(A,B) \)
  4. Length of remaining || paths  \( \geq \text{at least } H(A,B) + 2 \)

Solution to problem #1

1. \( H(A,B) = \# \text{ hops in a path from } A \text{ to } B \)
   - It is trivial to show that if we toggle one bit in A, then we move closer to B by one hop
   - We continue this process until we have reached A
   - Clearly there are \( H(A,B) \) hops
   - How many different paths are there?
Solution to problem #1-- continued

- We next prove a lemma that will be used to prove the remaining statements
- **Lemma:**
  If A and B differ in \( d-e \) bit positions, then they lay within the same \( d-e \) dimensional hypercube
- **Proof:**
  drop the bits that are identical

Proof of remaining results

2. \( d \) = total # of parallel paths between A & B
   - There are \( d \) edges incident to B
   - So there is a possibility of \( d \) parallel paths between A and B
   - We need to show that
     - None of the paths intersect
     - That there aren’t more than \( d \) paths
Proof of remaining results

• Let’s work backward from B along one of the paths toward A
• Call this node B_{-1}
• B_{-1} differs from A in $d-1$ bit positions
• Therefore, the A & B_{-1} are in the same $d-1$ dimensional hypercube
• By the previous result, the number of hops between A and B_{-1} is $d-1$
• In general there will be more than one path between A and B_{-1}, but we can use only one of these. Why?

Proof of remaining results

• Apply the same procedure to each of the $d$ incident edges
• We have $d$ different paths
• How do we know they do not intersect one another?
• Let’s construct $d$ paths that are guaranteed not to intersect
• To do this, we first pick a path
• Let the path be designated by a list of the dimension along which we route each link $0 \rightarrow 1 \rightarrow 2$
• The values must be unique within a given list (why?)
• The remaining $d-1$ paths are constructed by rotating the values in the original list: $1 \rightarrow 2 \rightarrow 0$ and $2 \rightarrow 0 \rightarrow 1$
• A 4-cube: $0\ 1\ 2\ 3\ 1\ 2\ 3\ 0\ 2\ 3\ 0\ 1\ 3\ 0\ 1\ 2$
Demonstrating the result

\[ \begin{array}{ccc}
0 \rightarrow 1 \rightarrow 2 \\
2 \rightarrow 0 \rightarrow 1 \\
000 & 001 & 100 & 101 \\
010 & 011 & 110 & 111 \\
1 \rightarrow 2 \rightarrow 0
\end{array} \]

Remainder of #1

3. \( H(A,B) = \# \) parallel paths of length \( H(A,B) \)
   - We’ve already proved this result for the case that \( H(A,B) = d \)
   - When \( H(A,B) < d \), we know from our lemma that \( A \) and \( B \) lie in some \( d-e < d \) dimensional hypercube
   - We can trivially use proof 2 to show that the statement is true
   - What about \( H(A,B) > d? \)
Remainder of #1

4. Length of remaining \(||\) paths is at least \(H(A,B) + 2\)
   - We know that there are paths of length \(H(A,B)\)
   - There can be no shorter paths (why?)
   - There can be longer paths (easy to show in 3d)
   - To construct such a path, we will need to route \(A\) away from \(B\) at least once
   - We will eventually have to undo the effect of this “wandering;” for each link that wandered off the path, we have to traverse a similar link back in the other direction
   - The minimum cost of wandering is 2 links

Problem set: problem #2

- All-to-all broadcast: each process broadcasts its data to all others.
- Text discusses a ring algorithm
- Another approach is to have the processes perform concurrent one-to-all broadcasts.
- Analyze the performance of this second approach using the ring algorithm for the one-to-all broadcasts.
- Compare the resultant performance model against that of the all-to-all ring algorithm given on page 164 of the text.
Ring algorithm for all-all broadcast

- Each processor starts with 1 element
- Send the element to the downstream neighbor
- Pick off the element from the upstream neighbor and store locally
- Forward that element in the next message to the downstream neighbor
- Message length remains constant in all \( p-1 \) steps
  \[ (p-1)(\alpha + \beta) \]
- Another approach is to have the processes perform concurrent one-to-all broadcasts.
- Recall that in a 1-all broadcast, the root starts with a single value and it sends along the chain
- The communication algorithm is precisely the same as before! But if we have bidirectional links, we can cut the running time in half

More all to all

- When messages are longer than one data element, 2 approaches are taken in practice
- Hypercube: \( \alpha \lg p + \beta m(p-1) \)
- Linear: (previously discussed): \( (p-1)(\alpha + m\beta) \)
- Linear method is preferable when \( m\beta \) is large relative to \( \alpha \)
- Hypercube is better when \( \alpha \) is large relative to \( m\beta \)
Problem #3

- All to all reduction on a hypercube
- Problem 4.8 of the text [p 191]
- Each node has a vector of $P$ elements
- The $i^{th}$ node gets the sum of the $i^{th}$ element on each processor
- An element contains $m$ values
- This is like a broadcast, except the elements can be combined
- $(\log P) \times (\alpha + \beta m + t_{\text{add}} \times m)$

AllReduce

- The simplest approach is to perform a reduction followed by a broadcast (using the hypercube algorithm)
  
  $(\log P) \times (2(\alpha + \beta m) + t_{\text{add}} \times m)$

- Another approach is to use a modified variant of the all-all broadcast algorithm [p.161-5]
  
  $\log P (\alpha + (\beta + t_{\text{add}}) m)$
  
  if the interconnect supports full duplex communication
**AllReduce algorithm**

result := my_val;
for i := 0 to d-1
    partner := my_id \otimes 2^i
    send result to partner
    receive incoming from partner
    result = result + incoming
end for

**Problem #4: k-to-all bcast**

- Let k be a power of 2
- Transmits a different m-word message from k > p sources.
- Give an algorithm with a running time of
  \( \alpha \lg p + \beta m \left( k \lg \left( \frac{p}{k} \right) + k - 1 \right) \)
  on a d-cube
- Assume that the messages cannot be split
Problem #4: k-to-all bcast

- There are two cases
  - the k processors form a lower dimensional hypercube (lg k < d)
  - the k processors do not form a hypercube
- The first case is a little easier to prove, so we’ll start with it
- We perform an all-to-all bcast within the lg k-dimensional hypercube
- Then we send messages of size km to the p-k remaining processors
- \[ \sum_{i=1}^{\lfloor \text{lg } k \rfloor} (\alpha + \beta \cdot m \cdot 2^{i-1}) + \sum_{i=\text{lg } k+1}^{\text{lg } p} (\alpha + \beta \cdot m \cdot 2 \cdot \text{lg } k) \]
  \[ = \alpha \cdot \text{lg } k + \beta \cdot m \cdot (k-1) + \beta \cdot m \cdot (k \cdot \text{lg } (p/k)) \]

Problem #5: Amdahl’s law

- Problem size = W
- Serial component = W_S
- Prove that W/W_S is an upper bound on speedup as P \rightarrow \infty
- Speedup = W / T_P
- T_P = (W - W_S)/P + W_S
  (What idealizing assumptions are we making?)
- S_P = W / ((W - W_S)/P + W_S)
- As P \rightarrow \infty, (W - W_S)/P vanishes, and we have
  S_P = W / W_S
Problem #6: Scaling

- Plot efficiency for the problem of adding n numbers on p processors
  - $t_{\text{add}} = 10$, time to communicate = 1
  - $p = 1, 4, 16, 64, 256$
- Fixed workload
  - Let $n = 256$, $W = 255$
  - Speedup = $W/T_p(W,P)$
- Scaled workload, base case $n = 256$, $p = 1$
  - $W = \Theta(P)$
  - Scaled speedup = $PW/T_p(PW,P)$
- “Isoefficient scaled workload”
  - $W = \Theta(P \log P)$
  - Isoefficient scaled speedup = $PW \log P / T_p(PW \log P, P)$

Results

- Running time for adding up n numbers of p processors:
  - $n/p - 1 + 11 \log p$
- Fixed workload
  - Let $n = 256$
  - $W = 255$
  - Speedup = $W/T_p(W,P)$
Results

- Scaled workload
- base case
  \( n=256, p=1 \)
- \( W = \Theta (P) \)
- Scaled speedup = \( PW/T_p(PW,P) \)

![Scaled Workload Graph]

Results

- Iso-scaled workload
- base case \( n=256, p=1 \)
- \( W = \Theta (P \log P) \)
- Isoefficient scaled speedup:
  \( PW \log P / T_p(PW \log P,P) \)
Problem #7: Enumeration sort

- Given implementations on
  - A CREW PRAM
  - A hypercube
- Recall the CRCW implementation

```plaintext
forall i=0:n-1, j=0:n-1
  if ( x[i] > x[j] ) then rank[i] = 1 end if
forall i=0:n-1
  y[rank[i]] = x[i]
```

CREW implementation

- CRCW implementation

```plaintext
forall i=0:n-1, j=0:n-1
  if ( x[i] > x[j] ) then rank[i] = 1 end if
forall i=0:n-1
  y[rank[i]] = x[i]
```

- CREW implementation can’t set rank[i] inside the doubly nested loop
- How do we avoid contention?
Avoiding contention

- CREW implementation uses a 2D array of ranks
  \[
  \text{forall } i=0:n-1, j=0:n-1 \quad \text{if } (x[i] > x[j]) \quad \text{then } \text{rank2d}[i,j] = 1 \quad \text{end if}
  \]
- Efficient algorithm to reduce each row of rank2d, rankd2d[i, :], resulting in a 1D rank array
  \[
  \text{forall } i=0:n-1 \quad \text{rank}[i] := \text{reduce}(\text{rank2d}[i,j])
  \]
- We can then employ the final step of the CRCW algorithm
  \[
  \text{forall } i=0:n-1 \quad y[\text{rank}[i]] = x[i]
  \]
- What limitations are there?

Hypercube implementation

- Assume that each process can send a single message to just one neighbor on the hypercube interconnect
- In the hypercube implementation we could require that each node have a copy of the complete input
- But this approach is not scalable (why?)
- Each node can hold only $O(1)$ values
  Remember that each node can communicate with nearest neighbors only
Hypercube implementation

- Recall the CRCW implementation

\[
\text{forall } i=0:n-1, j=0:n-1 \\
\text{if } (x[i] > x[j]) \text{ then rank}[i] = 1 \text{ end if} \\
\text{forall } i=0:n-1 \\
y[\text{rank}[i]] = x[i]
\]

- A simple approach is to use a ring algorithm
- After n-1 steps, each processor will have seen all n values, and can determine the rank of its element
- We then route the elements to their correct position using the same algorithm
- Running time is O(n). Can we do better?

Midterm review
Terms and concepts

- Know the definition and significance of ….  
- Parallel speedup, super-linear speedup, scaled speedup, efficiency, scalability, cost, cost-efficient, isoefficiency  
- Amdahl's law, serial bottlenecks  
- Granularity, surface-to-volume effect  
- Blocked data decompositions (uniform and irregular) cyclic decomposition  
- SPMD, MIMD, SIMD, Client server  
- Multiprocessors and multicomputers; shared memory, message passing  
- CRCW, CREW PRAM  
- Hypercube, ring interconnect, broadcast and reduction algorithms  
- Message startup, half power point $n_{1/2}$, peak bandwidth

Implementation techniques

- Message passing with MPI  
  - asynchronous, blocking and non-blocking communication  
  - collective communication: reduction, broadcast, barrier synchronizations  
  - communicators  
- Performance modeling and performance measurement techniques
Algorithms

• Know the purpose of the following algorithms, and the significant implementation issues affecting performance. Be familiar with performance models for each and be prepared to analyze performance and scalability

• **ODE solver**
  – The algorithm entails repeatedly updating a set of points on 1-dimensional. Each point is updated using nearest neighbors only.
  – To enable parallelization, we separate the points into even and odd sets. We maintain *ghost cells* holding off processor values.

• **Particle method**
  – Ring and chaining mesh strategy (you’ll only be asked to analyze performance of the ring algorithm)

• **Matrix multiplication**
  – Canon’s algorithm

• **Sorting**
  – Enumeration sort, odd-even sort, odd-even transposition sort, shell sort, bucket sort, sample sort (just higher level view)
  Be prepared to analyze performance for all but sample sort