CS475 Parallel Programming

Sorting

Wim Bohm, Colorado State University
Sorting Problem

- **Sorting**
  - Input: sequence \( S = (a_0, a_1, \ldots, a_{n-1}) \)
  - Output: \((b_0, b_1, \ldots, b_{n-1}) = \) permutation of \( S \) s.t. \( b_i \leq b_{i+1} \)

- **Sorting Algorithm Categories**
  - Internal sorting: \( S \) is small enough to fit in memory/network
    - We concentrate on this
  - External sorting: \( S \) partly stored on external device (disk)
  - Comparison sorting: Uses compares and exchanges
    - \( \Omega(n \log(n)) \) work
  - Non comparison sorting: Uses extra information about input data
    - Values lie in a small range (Radix Sort)
    - \( S \) is permutation of \( (1 .. N) \) (Pigeon hole sort)
    - Sometimes \( \Omega(n) \) work
Storage for Input and Output

- **Input sequence**
  - Distributed in equal blocks over processors unless specified otherwise

- **Output sequence**
  - Distributed over p processors unless specified otherwise
  - Ordering of blocks $S_1$ and $S_2$
    - $S_1 \leq S_2$ iff $\forall s_1 \in S_1$, $\forall s_2 \in S_2$: $s_1 \leq s_2$
    - requires enumeration of PEs
      - e.g. on hypercube: PE bit label or Gray code
  - Sorted output
    - $PE_i < PE_j$ (according to enumeration) $\Rightarrow$ block($PE_i$) $\leq$ block($PE_j$)
Discussion: Parallel Merge Sort

- A pipeline of sorters $S_0, S_1 \ldots S_n$
- $S_0$:
  - One input stream, two output streams
  - reads input stream and creates “sorted” subsequences of size 1
  - sends the subsequences to its outputs (alternating between the two)
- $S_i$: ($i = 1 \ldots n-1$)
  - Two input streams, two output streams
  - merges sorted subsequences from two input streams
  - sends double-sized, merged subsequences to its outputs (again alternating)
- $S_n$:
  - Two input streams, one output stream
  - merges sorted subsequences from two inputs into one result
Questions:
1. Given $n = 2^m$ input numbers, how many sorters are needed?
2. If a sorter can read one number in one time step, write one number in one time step, and store and compare in zero time steps, how many time steps does it take to sort $n$ numbers?
3. Is this algorithm cost optimal?
Compare Exchange, Compare Split

- **Compare exchange: $n = p$**
  - Ascending (+)
  - Descending (-)

- **Compare split: $n > p$**
  - $P_i$ and $P_j$ have blocks of data
  - Merge the two blocks
  - $P_i$ gets lower half, $P_j$ gets upper half
Sorting networks

- n numbers, n lines, M stages
- Each stage:
  - $\leq n/2$ compare-exchanges
  - Each compare exchange computes in $O(1)$ time
- Time complexity: $M$
- Cost: $M*n$
- Does this network sort?
Bitonic Sequence

- A sequence \( A = a_0, a_1, \ldots, a_{n-1} \) is bitonic iff
  1. There is an index \( i, 0 < i < n \), s.t.
     \[
     a_0 \ldots a_i \quad \text{is increasing}
     \]
     and
     \[
     a_i \ldots a_{n-1} \quad \text{is decreasing}
     \]
  or 2. There is a cyclic shift of \( A \) for which 1 holds.

Why BItonic? (vs MONOtonic)
A bitonic split divides a bitonic sequence in two:

\[
S_1 = (\min(bs_0, bs_{n/2}), \min(bs_1, bs_{n/2+1}), \ldots, \min(bs_{n/2-1}, bs_{n-1}))
\]

\[
\text{BitSplit}(BS) =
\]

\[
S_2 = (\max(bs_0, bs_{n/2}), \max(bs_1, bs_{n/2+1}), \ldots, \max(bs_{n/2-1}, bs_{n-1}))
\]

Theorem:

S1 and S2 are both bitonic and S1 < S2

Proof:

By careful consideration of all cases
Bitonic Merge

- Given: a Bitonic Sequence BS of size $n = 2^m$
- Sort BS using $m$ (parallel) Bitonic Split stages
Bitonic merge = $\log(n)$ bitonic split stages

- Can sort a bitonic sequence in $\log(n)$ steps
  - Increasing order: $+BM(n)$
    - use $+$ compare exchangers
  - Decreasing order: $-BM(n)$
    - use $-$ compare exchangers
Bitonic Sort

- Each 2 element subsequence is bitonic
- Merging 2 element subsequences, up and down, creates bitonic subsequences of size 4
  - Merging 2 elements up: +BM2
  - Merging 2 elements down: -BM2
- Merging these 4 sized subsequences up (+BM4) and down (-BM4) creates bitonic subsequences of size 8
- and so on......
Bitonic sort = $\log(n)$ bitonic merge stages
Bitonic Sort network
Bitonic sort: time and work

- Time: $O(\log^2(n))$

  Number of stages:
  
  $B2 + B4 + B8 + \ldots + B2^m = 1 + 2 + 3 + \ldots + m$
  
  where $m = \log(n)$

- Work: $O(n \log^2(n))$

  $O(n)$ per stage
Bitonic Sort on the hyper cube

- One element per processor
- Power of two distances in bitonic sort map perfectly on cube
- Question: perform +BMx or −BMx?
  - stage i, step j: compare bit-j to bit-i+1
    - equal: take minimum
    - unequal: take maximum
  - check it out yourself (do a pencil and paper exercise)

for i = 0 to d-1
  for j = i downto 0
    partner = flip bit (myLabel[j])
    exchange data with partner
    if (myLabel[i+1] = = myLabel[j] )
      keep min
    else keep max
Bitonic Sort on Mesh

- No ideal mapping; best: nearest = most used

Distance 1: used 7 times, Distance 2: used 3 times
Bitonic Sort $n > p$

- $n/p$ elements per PE
- Do local sorts at the beginning
- Use compare-split instead of compare-exchange
- Perfect load balance
- On hypercube: perfect nearest neighbor mapping
  - Cost optimal if $\log^2(p) / \log(n) = O(1)$
Parallel Bubble Sort

Odd-Even sort:
- sorts n elements in n/2 phases
- Each phase has two stages
  - first stage compares even element with next element
  - second stage compares odd element with next
- $O(n)$ time, $O(n^2)$ work
Count/Radix/Bucket family

- Enumeration Sort
  - Determine rank of every element
  - Sort A[0..n-1], using counters C[0..n-1]

\[
\text{forall } i \in 0..n-1 \text{ C}[i]=0
\]
\[
\text{forall } i \in 0..n-1, \text{forall } j \in 0..n-1
\]
\[
\]
\[
\text{forall } i \in 0..n-1 \text{ S}[C[i]]=A[i]
\]
Count sort: large number of small numbers

- **n** numbers in range 0..r-1
  - **n >> r**

  forall i in 0..r  C[i]=0
  forall i in 0..n-1 C[A[i]+1]++
  PPC = ParallelPrefixSum( C )
  forall i in 0..n-1 S[ PPC[A[i]]++ ]=A[i]

- One of the fastest sorts for this case
Partial sums, or Parallel Prefix

N numbers $V_1$ to $V_n$ stored in $A[1]$ to $A[n]$

Compute all partial sums $(V_1+..+V_k)$

d = 1

\[\text{do log(n) times}\]

\[\text{for all i in 1..n:}\]

\[\text{if } (i-d) > 0 \quad A[i] = A[i] + A[i-d]\]

d *= 2