• Reading
  – Skillicorn, Hill, McColl
    » Questions and Answers about BSP (pp 1-25)
Sequential radix sort

**Input:** A[0 : N-1], with b-bit elements
Radix s = 2^r, r ≥ 1

**Result:** A[0 : N-1] in sorted order

```
for d := 1 to ⌈b/r⌉ do

    -- construct histogram T[0 : s-1] of digit values in digit position d of A[0 : N-1]
    T[0:s-1] := 0
    for j := 0 to N-1 do  T[digit in position d of A[j]]++  end do

    -- cumulative histogram W[0 : s-1]
    W[0:s-1] := exclusive_scan(T[0:s-1], +)

    -- construct permutation H[0 : N-1] that sorts A[0 : N-1] into increasing order in digit position d
    for j := 0 to N-1 do  H[j] := W[digit in position d of A[j]]++  end do

    -- permute A[0 : N-1]

end do
```

**Complexity:**

\[
T_s(N) = \left\lceil \frac{b}{r} \right\rceil \left( O(2^r) + O(N) \right)
\]
Parallel radix sort

- For each digit position \( d \) from least to most significant
  - use sequential algorithm to compute local histogram \( T^{(j)}[0:s-1] \) for digit position \( d \) at each processor \( 1 \leq j \leq p \)
  - construct cumulative histogram defined as
    \[
    W^{(j)}[i] = \left( \sum_{i'=0}^{i-1} \sum_{j'=1}^{p} T^{(j')}[i'] \right) + \sum_{j'=1}^{j-1} T^{(j')}[i]
    \]
  - use \( W^{(j)} \) to determine the local portion of permutation \( H \) at each processor \( 1 \leq j \leq p \)
    and apply permutation in parallel to rearrange \( A \)

- Example
  - \( N = 20, p = 4, N/p = 5, \) \( b = 2, r = 2, s = 2^r = 4 \)
  - \( A = [3, 1, 0, 1, 3, 2, 0, 0, 2, 0, 2, 2, 2, 0, 1, 3, 2, 2] \)
  - \( H = [17, 5, 0, 6, 18, 8, 1, 2, 9, 3, 10, 11, 12, 13, 14, 4, 7, 19, 15, 16] \)

### Table

<table>
<thead>
<tr>
<th>( i/j )</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i ) ( 0 \leq i &lt; s )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>00</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>2</td>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( i/j )</th>
<th>00</th>
<th>01</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( i ) ( 0 \leq i &lt; s )</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>00</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>01</td>
<td>5</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>15</td>
</tr>
<tr>
<td>11</td>
<td>17</td>
<td>19</td>
<td>19</td>
<td>19</td>
</tr>
</tbody>
</table>
Parallel computation of cumulative histogram

• **Two alternatives**

  (1) Perform exclusive parallel prefix-sums across the \( s \) successive rows of \( T \)
  
  • Each sum starts with the ending value on the previous row

  » total BSP cost: \( s \) \( (2 \log p) \) \((1 + g + L)\)

(2) Multiscan method

• partition digits \( 0..s-1 \) into \( p \) contiguous intervals of size \( k = \frac{s}{p} \) and construct \( T^{(i)}[ik: (i+1)k -1] \) for \( i \) in \( 0..p-1 \)

• transpose \( T \) across processors
  
  • BSP cost: \( s(1 + g + L) \)

• compute local sum, one parallel prefix sum across processors, followed by a local prefix sum
  
  • BSP cost: \( 2s + (\log p)(1 + g + L) \)

• transpose result to yield \( W^{(i)}[ik: (i+1)k -1] \) for \( i \) in \( 0..p-1 \)
  
  • BSP cost: \( s(1 + g + L) \)

» total BSP cost: \( (4s + \log p) \) \((1 + g + L)\) superior when \( p > 2 \)
Example of multiscan

- **Problem parameters**
  - N = 20, p = 4, N/p = 5 and r = 2, s = 2^r = 4
  - A = [3, 1, 0, 1, 3, 2, 0, 0, 2, 0, 2, 2, 2, 2, 2, 0, 1, 3, 2, 2]

- 4 processors →

- # of 0: 1 3 0 1
- # of 1: 2 0 0 1
- # of 2: 0 2 5 2
- # of 3: 2 0 0 1

Input: local histogram – count of occurrences of each “digit” 0 .. 3 within each processor’s local section of A (N/p = 5)

- (1) transpose
- (2) local sum
- (3) global excl prefix sum
Multiscan example (contd)

- **Problem parameters**
  - \( N = 20, p = 4, N/p = 5 \) and \( r = 2, s = 2^r = 4 \)
  - \( A = [\begin{array}{cccc} 3, & 1, & 0, & 1, \\ 2, & 0, & 0, & 2, \\ 2, & 2, & 2, & 2, \\ 0, & 1, & 3, & 2, \end{array} \] \)

  - 4 processors

  - 4 processors →

  - Memory

  - Memory

  - (4) local exclusive prefix sum

  - (5) transpose

  - 4 processors →

  - Cumulative histogram \( W^{(i)}[j] \)

  - the destination index of the first occurrence of digit \( i \) held within processor \( j \)
Parallel radix sort - Analysis

• Algorithm
  – \( \lceil b/r \rceil \) iterations
    – each iteration (\( s = 2^r \)) (using multiscan)
      » construct histogram \( O(N/p + s) \)
      » transpose histograms \( sg+L \)
      » local sum \( O(s) \)
      » global prefix sum \( (\lg p)(1 + g + L) \)
      » local prefix sum \( O(s) \)
      » transpose cumulative histogram \( sg + L \)
      » compute destinations \( O(N/p) \)
      » permute values \( O((N/p)(b/64))g + L \)

  – BSP cost (\( b = 64 \))

\[
C^{RADIX}(N, p, r) = \left\lceil \frac{b}{r} \right\rceil \Theta \left( \frac{N}{p} + 2^r \right) + \left\lceil \frac{b}{r} \right\rceil \left( \frac{N}{p} + 2^r \right) \cdot g + \left\lceil \frac{b}{r} \right\rceil (\lg p) \cdot L
\]

  – How to find optimum choice of radix \( r \)?
    » \( r \) small means \( N/p \) dominates \( 2^r \)
    » \( r \) large means \( b/r \) is small
Predicted and measured times for radix sort

Figure 1.10  Predicted and measured execution time per key of radix sort on the CM-5.
Breakdown of radix sort running times

Figure 1.11  Predicted and measured execution times per key of various phases in radix sort on 512 processors.
Probabilistic sorting algorithms

- **Definitions**
  - An unordered collection $H$ with $N$ disjoint values is partitioned by splitters $S = S_1 < \ldots < S_{p-1}$ into $p$ disjoint subsets $H_1 \ldots H_p$ such that

  $$H_i = \{h \mid h \in H \text{ and } S_{i-1} \leq h < S_i\} \quad \text{(define } S_0 = -\infty, \text{ and } S_p = +\infty \text{)}$$

  - The skew $W(S)$ of a partition $S$ is the ratio of the maximum partition size to the optimal partition size $(N/p)$

  $$W(S) = \max_{1 \leq i \leq p} \left( \frac{|H_i|}{N/\ p} \right)$$
Determining good splitters through sampling

- Determining a set of splitters through sampling
  - sample $k \cdot p$ elements at random from $H$
    - $k \geq 1$ is the oversampling ratio
  - sort this sample into order $b_1 < b_2 < \ldots < b_{k \cdot p}$ and choose $S_i = b_{k \cdot i}$

- Probabilistic bounds on $W(S)$ of a sampled set of splitters $S$
  - given some maximum skew $W$ and a failure probability $0 < r < 1$

$$\Pr(W(S) > W) \leq r \quad \text{when} \quad k \geq \frac{2 \ln(p / r)}{(1 - 1/W)^2 W} \quad \text{(provided} \quad p > 1, \quad W > 1.3)$$

- so if we oversample sufficiently in choosing a set of splitters, the chance of a large skew can be made arbitrarily small
Oversampling ratio $k$ as a function of $p$

- Example
  - for $p = 100$ processors, we need to sample $k = 4 \ln \left( \frac{p}{r} \right) = 74$ values per processor to bound the skew $W(S) < 2$ with failure probability $r = 10^{-6}$
Parallel samplesort

**Algorithm**
1. sample k values at random in each processor to limit skew to W w.h.p.
   \[O(k)\]
2. sort kp sampled keys, extract p-1 splitters, and broadcast to all processors
   a) by sending all samples to one processor and performing a local sort
      \[O(kp) + (k+2)p \cdot g + 2 \cdot L\]
   a) by performing a bitonic sort with k values per processor
      \[O(k \log^2 p) + k(1+2 \log p) \cdot g + (1+\log p) \cdot L\]
3. compute destination processor for each value by binary search in splitter set
   \[O(N/p \log p)\]
4. permute values
   \[WN/p \cdot g + L\]
5. perform local sort of values in each processor
   \[O(Ts(WN/p))\]

**BSP cost**
\[
C^{\text{SAMPLE}}(N, p, W) = \Theta(W + \log p) \left(\frac{N}{p}\right) + W \left(\frac{N}{p}\right) \cdot g + (\log p) \cdot L \\
+ O(k \log p)(\log p \cdot g + L)
\]
Samplesort: predicted and measured times

![Graph showing predicted and measured times](image)

**Figure 1.12** Estimated and measured execution time of parallel sample sort on the CM-5.
Samplesort: breakdown of execution time

Figure 1.13 Estimated and measured execution times of various phase of parallel sample sort on 512 processors.
Parallel sorting: performance summary

- 32 bit values
  - for small N/p (not shown), bitonic sort is superior

Figure 1.14 Estimated execution time of four parallel sorting algorithms under LogP with the performance characteristics of the CM-5.
Samplesort issues

• Implementing the permutation
  – What is the destination address of a given value? Two strategies:
    » Send-to-queue operation
      • don’t care, maintain queue at destination
    » Compute unique destination for each value
      • planning cost: $O(p) + 2pg + 2L$

– In what order should the values be sent?
  » Global rearrangement defines a permutation, but piecewise implementation may yield poor performance
Samplesort issues

• **How to handle duplicate keys**
  – make each key unique
    » (key, original index)
      • increases comparison cost and network traffic

  – random choice of possible destinations
    » suppose p = 5 and splitters are
      10, 20, 20, 30
      where should we send key 20?

• **What about restoring load balance?**
  – Worst-case communication cost?
Two-phase sample sort

- **Objectives**
  - scramble input data to create a random permutation
  - highly supersample input to minimize skew

- Randomly distribute keys into $p$ buckets
- Transpose buckets and processors
  - expected bucket size $N/p^2$
- Local sort
- Proc 1 selects and broadcasts splitters
  - oversampling ratio $k = N/p^2$
- Partition local keys into sorted sections according to splitters
  - expected bucket size $N/p^2$
- Transpose sorted sections and processors
- Local $p$-way merge
Two-phase samplesort

1. Randomly distribute local keys into $p$ local buckets

2. Transpose buckets and processors

3. Local sort

4. Processor 1 selects $(p-1)$ splitters

5. Broadcast splitters

6. Local partitioning of values into $p$ sorted sections

7. Transpose sorted sections and processors

8. Local $p$-way merge of sorted sections

\[ C^{2\text{ph}}(N, p) = O\left(\frac{N}{p} \lg N\right) + 2\left(\frac{N}{p}\right) \cdot g + L \]

\[ + O(p \lg \left(\frac{N}{p}\right)) + 2p \cdot g + 3 \cdot L \]