CSE 160
Lecture 13

Sorting

Announcements

- No Lab in APM this Friday
- Quiz return

Today's Lecture

- Parallel Sorting (II)
  - Bucket Sort
  - Sample Sort
  - Bitonic Sort

Parallel sorting

- We'll consider in-memory sorting of integer keys
  - Bucket sort
  - Sample sort
  - Bitonic sort
- In practice, we sort on external media, i.e. disk
  - See: http://sortbenchmark.org
  - **TritonSort (UCSD):** 0.725×10^{12} bytes/minute

Rank Sorting

- Compute the rank of each input value
- Move each value in sorted position according to its rank
- Makes idealizing assumptions
  - An ideal parallel computer with no memory contention and an infinite number of processors
  - The forall loops parallelize perfectly

```
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]
```

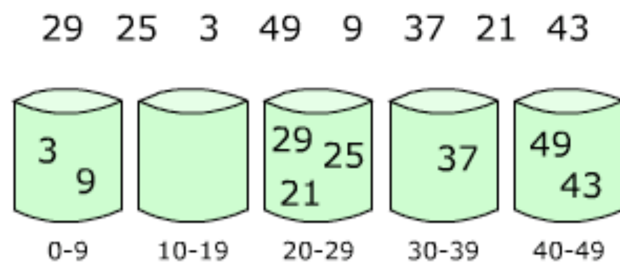
©2013 Scott B. Baden / CSE 160 / Winter 2013

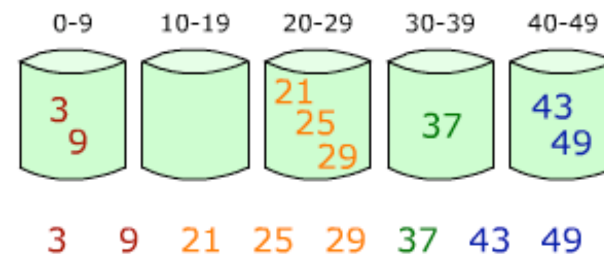
In search of a fast and practical sort

- Rank sorting is impractical on real hardware
- Let's borrow the concept: compute the thread owner for each key
- Shuffle data in sorted order in one step
- But how do we know which thread should be the owner?
- Subdivide the key space

1st attempt: bucket sort

- Divide the range of keys into equal subranges and associate a *bucket* with each range
- Each processor maintains p local buckets
  - Assigns each key to a bucket: $\lfloor \mathbf{p \times key/(K_{max}-1)} \rfloor$
  - Routes the buckets to the correct owner (each local bucket has ~ $\mathbf{n/p^2}$ elements)
  - Sort all incoming data into a single bucket





Wikipedia

Running time

- Assume that the keys are distributed uniformly over 0 to K_{max}-1
- Local bucket assignment: O(n/p)
- Route each local bucket to the correct owner O(n)
- Local sorting (using radix sort) : O(n/p))

www.csse.monash.edu.au/~lloyd/tildeAlgDS/Sort/Radix

Worst case behavior

- The assignment of keys to threads is based solely on the knowledge of K_{max}
- If the keys are integers in the range [0,Q-1] ….
  … thread k has keys in the range

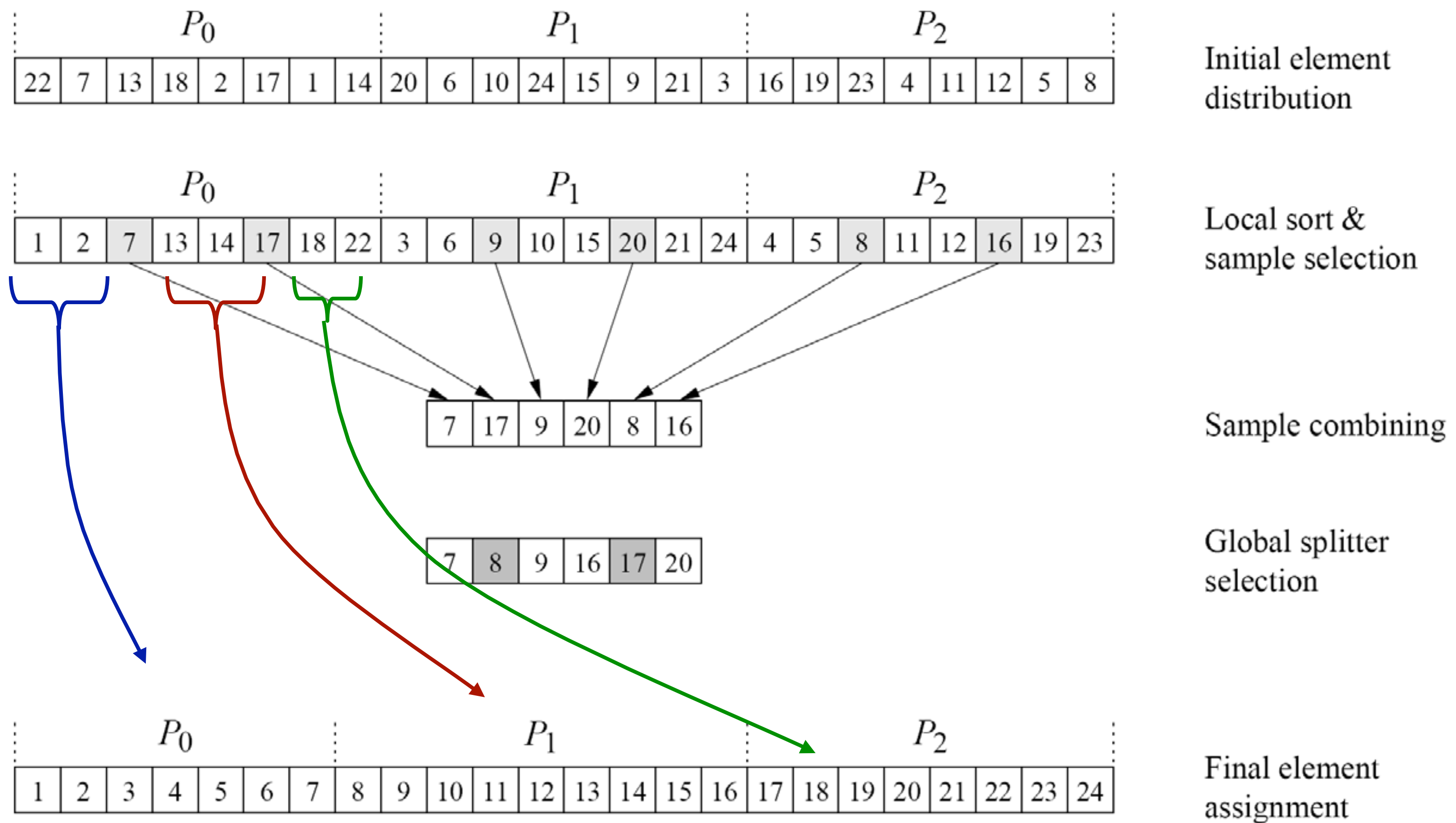
$$\left[k\frac{Q}{P},(k+1)\frac{Q}{P}\right]$$

- E.g. for $Q=2^{30}$, P=64, each thread gets $2^{24} = 16$ M elements
- For a non-uniform distribution, we need more information to balance keys (and communication) over the processors
- In the worst case, all the keys could go to one processor

Improving on bucket sort

- Sample sort
- Uses a heuristic to estimate the distribution of the global key range over the p threads
- Each processor gets about the same number of keys
- Sample the keys to determine a set of p-1 *splitters* that partition the key space into p disjoint regions (buckets)

Sample selection



Introduction to Parallel Computing, 2nd Ed,, A.Grama, A.l Gupta, G. Karypis, and V. Kumar, Addison-Wesley, 2003.

Splitter selection: regular sampling

- Shi and Schaeffer [1992]
- Each processor sorts its local keys, then selects *s* evenly spaced samples
- These candidate splitters are collected by one thread
  - Sorted
  - Sampled at uniform positions to generate a *p-1* element splitter list

Performance

- Assuming $n \geq p^3$ …
- $T_P = O((n/p) \lg n)$
- If $s = p$, each processor will will merge not more than $2n/p + n/s - p$ elements
- If s > p, each processor will will merge not more than $(3/2)(n/p) - (n/(ps)) + 1 + d$ elements
- Duplicates d do not impact performance unless $d = O(n/p)$
- Tradeoff: increasing s …
  - Spreads the final distribution more evenly over the processors
  - Increases the cost of determining the splitters
- For some inputs, communication patterns can be highly irregular with some pairs of processors communicating more heavily than others, lowering performance

Radix sort

- We need a **stable** sorting algorithm to do the local sorts: the output preserves the order of inputs having the same associated key
- ***radix sort*** meets our needs: sort the keys in passes, choosing an r-bit block at a time, O(n) running time
- Explanation with a demo www.csse.monash.edu.au/~lloyd/tildeAlgDS/Sort/Radix/

A simple example

- Following an example in the NIST *Dictionary of Algorithms and Data Structures* http://www.nist.gov/dads/
- Uses buckets to sort the keys in passes
- Running time is O(cn), c depends on size of the keys and the number of buckets

Radix sort in action

- Consider the input keys
  34, 12, 42, 32, 44, 41, 34, 11, 32, and 23
- Use 4 buckets
- Sort on each digit in succession, least significant to most significant

Radix sort in action

- Consider the input keys
  34, 12, 42, 32, 44, 41, 34, 11, 32, and 23
- Use 4 buckets
- Sort on each digit in succession, least significant to most significant
- After pass 1
  41 11   <u>12 42 32 32</u>   23   <u>34 44 34</u>

Radix sort in action

- Consider the input keys
  34, 12, 42, 32, 44, 41, 34, 11, 32, and 23
- Use 4 buckets
- Sort on each digit in succession, least significant to most significant
- After pass 1

  41 11 12 42 32 32 23 34 44 34
- After pass 2

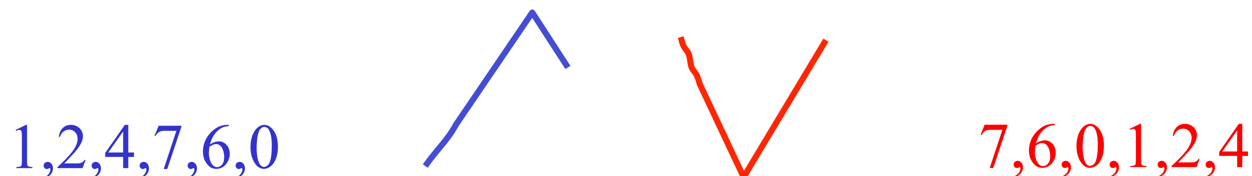
  11 12 23 32 32 34 34 41 42 44

Today's Lecture

- Parallel Sorting (II)
  - Bucket Sort
  - Sample sort
  - Bitonic Sort

Bitonic sort

- Classic parallel sorting algorithm: $O(\log^2 n)$ on n processors
- Also used in fast sorting on a GPU
- **Definition:** A *bitonic sequence* is a sequence of numbers a_0, $a_1 \ldots a_{n-1}$ with at most 1 local maximum and 1 local minimum (Endpoints wrap around)
  - There exists an index *i* where
    $a_0 \le a_1 \le a_1 \ldots \le a_i$ **and** $a_i \ge a_{i+1} \ge a_{i+1} \ldots \ge a_{n-1}$
  - We may cyclically shift the a_k while maintaining this relationship
- Merge property: We may merge two bitonic sequences in much the same way as we merge two *monotonic* sequences



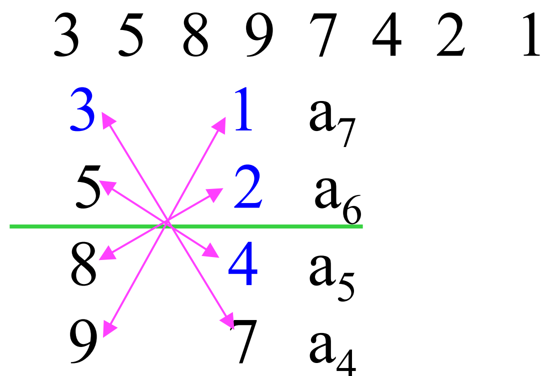
Splitting property of bitonic sequences

- We can split a bitonic sequence y into two bitonic sequences L(y) and R(y)

  $L(y) = \langle \min\{a_0,a_{n/2}\}, \min\{a_1,a_{n/2+1}\},\ldots, \min\{a_{n/2+1},a_{n-1}\}\rangle$

  $R(y) = \langle \max\{a_0,a_{n/2}\}, \max\{a_1,a_{n/2+1}\},\ldots,\max\{a_{n/2+1},a_{n-1}\}\rangle$

- See the notes for a proof



All values in L(y) < R(y)

L(y): 3 4 2 1

R(y): 7 5 8 9

©2013 Scott B. Baden / CSE 160 / Winter 2013

Sorting a bitonic sequence is easy

- Split the bitonic sequence y into two bitonic subsequences L(y) and R(y)
- Sort L(y) and R(y) recursively
- Merge the two sorted lists
  - Since all values in L(y) are smaller than all values in R(y) we don't need to exchange values in L(y) and R(y)
- When |L(.)| < 3, sorting is trivial
- We designate **S(n)** to be sort on of an n-element bitonic sequence

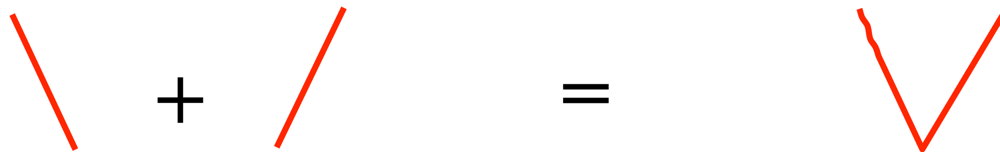
©2013 Scott B. Baden / CSE 160 / Winter 2013

Bitonic sort algorithm

- Create a bitonic sequence y from an unsorted list
- Apply the previous algorithm to sort the bitonic sequence
- We need an algorithm to create the bitonic sequence y

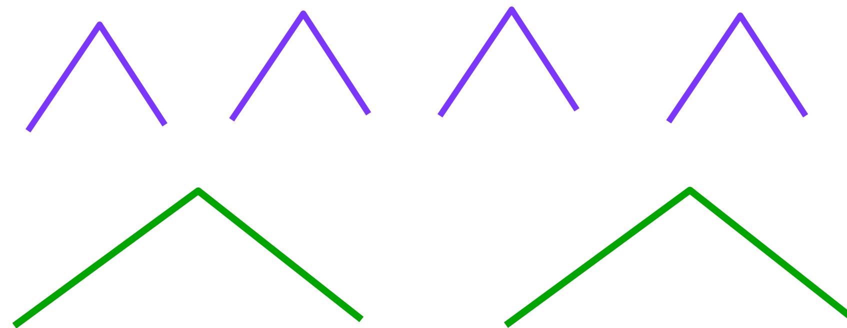
Additional properties of bitonic sequences

- Any 2 element sequence is a bitonic sequence
- We can trivially construct a bitonic sequence from two monotonic sequences, one sorted in increasing order, the other in decreasing order



Inductive construction of the initial bitonic sequence

- Form matched pairs of 2-element bitonic sequences, pointing up and down [B(2)]
- Trivially merge these into 4-element bitonic sequences
- Now form matched pairs of 4-element sequences [B(4)]
- Apply S(4) to each sequence, sorting the first upward, the second downward
- Trivially merge into an 8-element bitonic sequence
- Continue until there is just one sequence

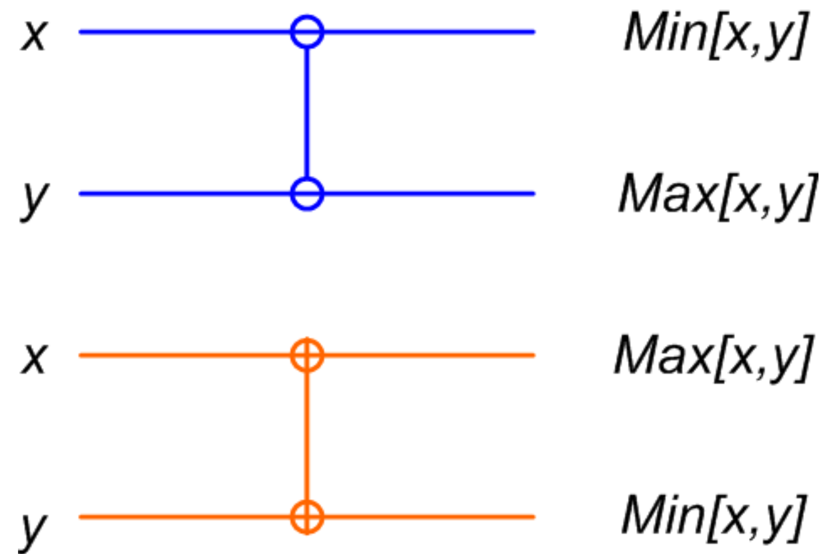


Implementing the bitonic sort algorithm

- Create a bitonic sequence *y* from an unsorted list, *B(n)*
- Apply the previous algorithm to sort the bitonic sequence, *S(n)*
- We use comparators to re-order data
- We use a shuffle exchange network to form *L(y)* and *R(y)*
  - This network shuffles an n-element sequence by interleaving $x_0, x_{n/2}, x_1, x_{n/2+1}, \ldots$

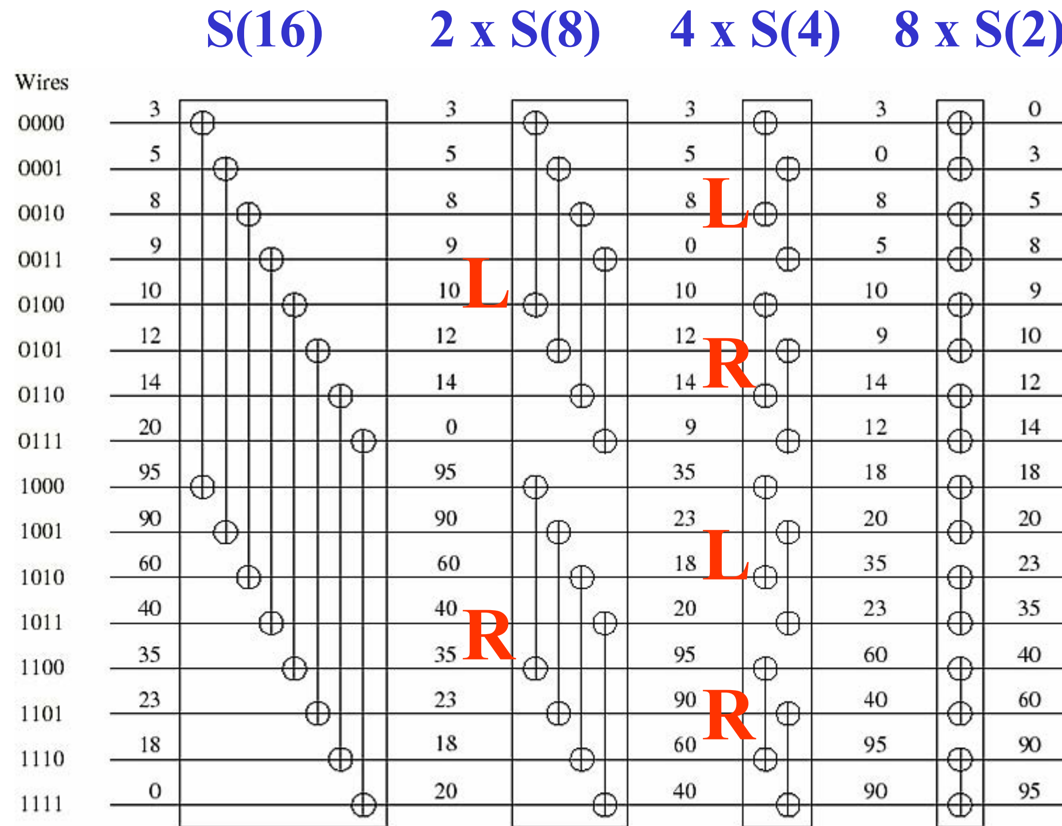
Comparators

- Given two values x & y, produce two outputs
- **For an increasing comparator, the output is**
  min[x,y], max[x,y]
- **For a decreasing comparator, the output is**
  max[x,y], min[x,y]



Bitonic merging network

- Converts a bitonic sequence into a sorted sequence

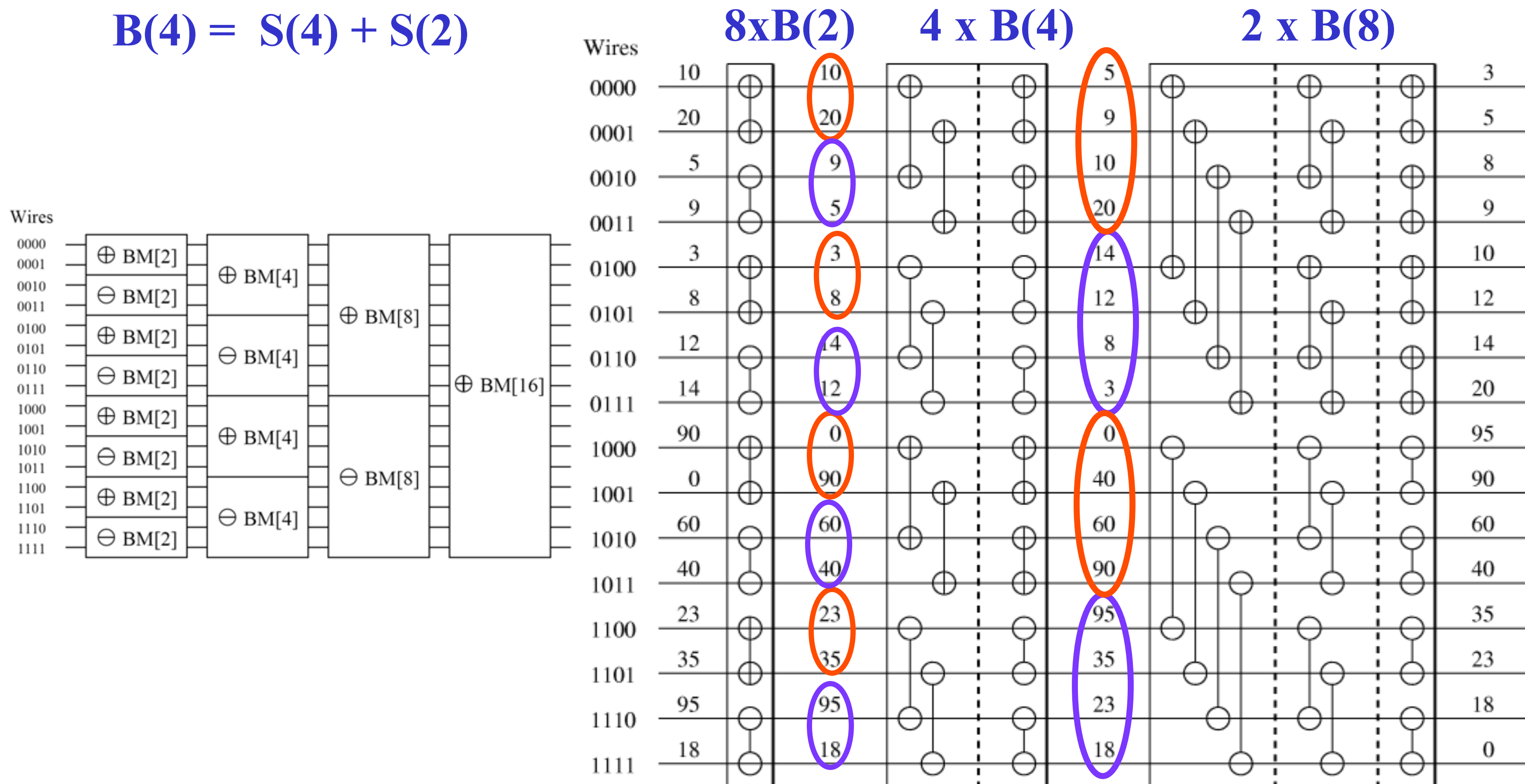


From *Introduction to Parallel Computing*, V. Kumar et al, Benjamin Cummings, 1994

©2013 Scott B. Baden / CSE 160 / Winter 2013

Bitonic conversion network

Converts an unordered sequence into a bitonic sequence



From *Introduction to Parallel Computing*, V. Kumar et al, Benjamin Cummings, 2003

Fin