COMP90025 Parallel and Multicore Computing

Sorting, Merging and other building blocks

Aaron Harwood

School of Computing and Information Systems
The University of Melbourne

2019 Semester II

- *Compaction* For array $X = [x_0 \ldots x_{n-1}]$ such that k elements of X have nonempty values, move all nonempty values into the first k consecutive locations of X, e.g. $[\emptyset, 1, \emptyset, \emptyset, 4]$ becomes $[1, 4, \emptyset, \emptyset, \emptyset]$.
- *Unique Counts* Given a sorted array X of n elements, return an array C of length n that contains tuples of the form $(Value, Count)$ for each distinct element (value) in X, consecutively in the lower portion of C and $\emptyset$ for all remaining unused elements of C.
- *Distribution* Let X contain elements such that some are empty and some have values, e.g. $[6, 3, \emptyset, \emptyset, \emptyset, 5, \emptyset, \emptyset]$, then the returned array has the value of all nonempty elements copied to itself and all consecutive nonempty positions: $[6, 3, 3, 3, 3, 5, 5, 5]$.

Description: For array $X = [x_0 \ldots x_{n-1}]$ such that k elements of X have nonempty values, moves all nonempty values into the first k consecutive locations of X, e.g. $[\emptyset, 1, \emptyset, \emptyset, 4]$ becomes $[1, 4, \emptyset, \emptyset, \emptyset]$.

Analysis: $\Theta\left(\frac{n}{p} + \log p\right)$

procedure $\textsc{Compaction}^{\star}_{\text{EREW}}(X, n, p)$
    for $i \leftarrow 0$ **to** $p - 1$ **do in parallel**
        processor i **does**
            for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do**   ▷ $\Theta\left(\frac{n}{p}\right)$ steps
                if $X[j] \neq \emptyset$ **then**
                    $R[j] \leftarrow 1$
                else
                    $R[j] \leftarrow 0$
                $A[j] \leftarrow \emptyset$ ▷ Setup a temporary array as well
    all processors do
        $R \leftarrow \textsc{PrefixSum}^{\star}_{\text{EREW}}(R, n, p)$ ▷ $\Theta\left(\frac{n}{p} + \log p\right)$ steps

for $i \leftarrow 0$ **to** $p-1$ **do in parallel**
processor i **does**
for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright\ \Theta\left(\frac{n}{p}\right)$ steps
if $X[j] \neq \emptyset$ **then**
$A[R[j]-1] \leftarrow X[j]$
for $i \leftarrow 0$ **to** $p-1$ **do in parallel**
processor i **does**
for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright\ \Theta\left(\frac{n}{p}\right)$ steps
$X[j] \leftarrow A[j]$
return X

Description: Given a sorted array X of n elements, return an array C of length n that contains tuples of the form $(\mathit{Value}, \mathit{Count})$ for each distinct element (value) in X, consecutively in the lower portion of C and $\emptyset$ for all remaining unused elements of C.

Require: $n > 0$

Analysis: $\Theta\left(\frac{n}{p} + \log p\right)$

procedure $\textsc{UniqueCounts}^{\bigstar}_{\text{EREW}}(X, n, p)$
    for $i \leftarrow 0$ **to** $p-1$ **do in parallel**
        processor i **does**
            for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps
                if $j = 0$ **then**
                    $R[0] \leftarrow 1$
                else if $X[j] \neq X[j-1]$ **then**
                    $R[j] \leftarrow 1$
    all processors do
        $R \leftarrow \textsc{PrefixSum}^{\bigstar}_{\text{EREW}}(R, n, p)$ $\triangleright$ $\Theta\left(\frac{n}{p} + \log p\right)$ steps

for $i \leftarrow 0$ **to** $p-1$ **do in parallel**
    processor i **does**
        for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps
            $C[j] = \emptyset$

for $i \leftarrow 0$ **to** $p-1$ **do in parallel**
    processor i **does**
        for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps
            if $j = 0$ **then**
                $C[0] \leftarrow (X[0], 0)$ $\triangleright$ $(Value, Rank)$
            else
                if $X[j] \neq X[j-1]$ **then**
                    $C[R[j]-1] \leftarrow (X[j], j)$

for $i \leftarrow 0$ **to** $p-1$ **do in parallel**
  processor i **does**
    for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p}-1$ **do** ▷ $\Theta\left(\frac{n}{p}\right)$ steps
      if $j < n-1$ **then**
        if $C[j+1] \neq \emptyset$ **then**
          $C[j] \leftarrow (C[j]_{Value}, C[j+1]_{Rank} - C[j]_{Rank})$ ▷ $(Value, Count)$
        else
          $C[j] \leftarrow (C[j]_{Value}, C[j]_{Rank} + n - j)$
      else
        $C[n-1] \leftarrow (C[n-1]_{Value}, 1)$ ▷ A single unique value at the very end of the array
return C

Description: Let X contain elements such that some are empty and some have values, e.g. $[6, 3, \emptyset, \emptyset, \emptyset, 5, \emptyset, \emptyset]$, then the returned array has the value of all nonempty elements copied to itself and all consecutive nonempty positions: $[6, 3, 3, 3, 3, 5, 5, 5]$.

Ensure: $X[0] \neq \emptyset$

Analysis: $\Theta\left(\frac{n}{p} + \log p\right)$

procedure $\textsc{Distribute}^{\star}_{\textsc{erew}}(X, n, p)$
    for $i \leftarrow 0$ **to** $p - 1$ **do in parallel**
        processor i **does**
            for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps
                if $X[j] \neq \emptyset$ **then**
                    $M[j] \leftarrow 1$
                else
                    $M[j] \leftarrow 0$
    all processors do
        $M \leftarrow \textsc{PrefixSum}^{\star}_{\textsc{erew}}(M, n, p)$ $\triangleright$ $\Theta\left(\frac{n}{p} + \log p\right)$ steps
        $CM \leftarrow \textsc{UniqueCounts}^{\star}_{\textsc{erew}}(M, n, p)$ $\triangleright$ $\Theta\left(\frac{n}{p} + \log p\right)$ steps

for $i \leftarrow 0$ **to** $p-1$ **do in parallel** $\triangleright$ At most p values remain to be (completely) distributed after this step, and no subarray has more than one unfinished value to distribute.

    processor i **does**

        $x \leftarrow \emptyset$

        for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p}-1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps

            if $X[j] \neq \emptyset$ **then**

                $x \leftarrow X[j]$

            else if $x \neq \emptyset$ **then**

                $X[j] \leftarrow x$

for $i \leftarrow 1$ **to** $p-1$ **do in parallel** $\triangleright$ Processor 0 is never active since $X[0] \neq \emptyset$

  processor i **does**

    if $X[i\frac{n}{p}] = \emptyset$ **then**

      if $X[i\frac{n}{p} - 1] \neq \emptyset$ **then**

        $x \leftarrow CM[M[i\frac{n}{p}] - 1]_{Value}$

$k = \frac{p}{n} CM[M[i\frac{n}{p}] - 1]_{Count} - 1$ **processor array** $i \ldots i+k-1$ **does**

        $\textsc{Broadcast}^{\star}_{\textsc{EREW}}(x, k)$ $\triangleright$ $\Theta(\log k)$ steps, $k \leq p$

      for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps

        if $X[j] = \emptyset$ **then**

          $X[j] \leftarrow x$

        else

          break

return X

Sorting

- Sorting algorithms are fundamental to many applications. Given an array of n data elements, $\{a_0, a_1, \ldots, a_{n-1}\}$, sorting rearranges the order of the elements to produce a sorted array, $\{b_0, b_1, \ldots, b_{n-1}\}$ such that $b_i \leq b_j$ for every $0 \leq i \leq j \leq n-1$.
- The worst-case time complexity of mergesort is $O(n \log_2 n)$.
- The average-case time complexity of quicksort is $O(n \log_2 n)$.
- Using n processors, at best we could expect a time complexity of $O(\log_2 n)$.

Mergesort without parallel merging

Mergesort proceeds from a single processor or process that holds an array of $n = 2^t$ elements. The divide-and-conquer approach is used to divide the array into two halves and give one half to another process. The subdivision continues until at most each of n processes holds exactly one element. Then the processes use Mergesort to generate the sorted array.

Assuming there are $p = n = 2^t$ processors. The first division phase of the Mergesort algorithm is essentially scattering the elements over the processors. Each processor receives one element of the array.
The total number of parallel computation steps is t, at each step, $i = 0, 1, \ldots, t-1$, two lists of size 2^i are merged with a single processor.
It takes $2n - 1$ steps in the worst case to merge two sorted lists each of n numbers.
The number of computational steps is then

$$2\sum_{i=0}^{t-1}\left(2^i - \tfrac{1}{2}\right) = 2^t - t - 2$$

which is $O(n)$.

Quicksort

Quicksort also parallelizes over n processors to obtain $O(n)$ parallel computational steps.

Recall that Quicksort selects a *pivot* for the elements in the array. All elements in the array that are less than the pivot are put into a lower array and all elements greater than the pivot are put into a higher array. The Quicksort algorithm is then recursively applied on the higher and lower arrays.

Selection of the pivot is not too important for the sequential algorithm however it is important for the parallel algorithm in order to keep the tree of processes reasonably balanced.

Mergsort with parallel merge

- Merge two sorted lists, $A = [a_0 \dots a_{n_1-1}]$ and $B = [b_0 \dots b_{n_2-1}]$, of length n_1 and n_2 resp., where $a_i, b_j \in \{1, 2, \dots, n\}$ for all $0 \leq i < n_1$, $0 \leq j < n_2$.
- Sort list $A = [a_0 \dots a_{n-1}]$ of length n where $a_i \in \{1 \dots n\}$ for all $0 \leq i < n$.

Description: Merge two sorted lists, $A = [a_0 \ldots a_{n_1-1}]$ and $B = [b_0 \ldots b_{n_2-1}]$, of length n_1 and n_2 resp., where $a_i, b_j \in \{1, 2, \ldots, n\}$ for all $0 \le i < n_1$, $0 \le j < n_2$. Based on Bahig and Bahig, 2007.

Require: $n_1 > 0, n_2 > 0,\ n = \max\{n_1, n_2\}$

Analysis: $\Theta\left(\frac{n}{p} + \log p\right)$

1: **procedure** $\textsc{Merge}^{\star}_{\textsc{EREW}}(A, B, n_1, n_2, p)$
2: **for** $i \leftarrow 0$ **to** $p - 1$ **do in parallel**
3: **processor** i **does**
4: **for** $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright$ $\Theta\left(\frac{n}{p}\right)$ steps
5: $X[j] \leftarrow \emptyset$
6: **all processors do**
7: $CA \leftarrow \textsc{UniqueCounts}^{\star}_{\textsc{EREW}}(A, n_1, p)$ $\triangleright$ $\Theta\left(\frac{n_1}{p} + \log p\right)$ steps, $n_1 \le n$
8: $CB \leftarrow \textsc{UniqueCounts}^{\star}_{\textsc{EREW}}(B, n_2, p)$ $\triangleright$ $\Theta\left(\frac{n_2}{p} + \log p\right)$ steps, $n_2 \le n$

for $i \leftarrow 0$ **to** n_1 **do in parallel**
  processor i **does**
    for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright\ \Theta\left(\frac{n}{p}\right)$ steps
      if $CA[j] \neq \emptyset$ **then**
        $X[CA[j]_{Value}] \leftarrow CA[j]$
for $i \leftarrow 0$ **to** n_2 **do in parallel** $\triangleright$ Aggregate the counts
  processor i **does**
    for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright\ \Theta\left(\frac{n}{p}\right)$ steps
      if $CB[j] \neq \emptyset$ **then**
        if $X[CB[j]_{Value}] = \emptyset$ **then**
          $X[CB[j]_{Value}] \leftarrow CB[j]$
        else

$X[CB[j]_{Value}] \leftarrow (CB[j]_{Value}, CB[j]_{Count} + X[CB[j]_{Value}]_{Count})$

all processors do
  $X \leftarrow \text{COMPACTION}^{\star}_{\text{EREW}}(X, n, p)$ $\triangleright\ \Theta\left(\frac{n}{p} + \log p\right)$ steps

for $i \leftarrow 0$ **to** $n-1$ **do in parallel**
    processor i **does**
        for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright\ \Theta\left(\frac{n}{p}\right)$ steps
            if $X[j] \neq \emptyset$ **then**
                $PX[j] \leftarrow X[j]_{Count}$
            else
                $PX[j] \leftarrow 0$

all processors do
    $PX \leftarrow \textsc{PrefixSum}^{\star}_{\text{EREW}}(PX, n, p)$ $\triangleright\ \Theta\left(\frac{n}{p} + \log p\right)$ steps

for $i \leftarrow 0$ **to** $n-1$ **do in parallel**
    processor i **does**
        for $j \leftarrow i\frac{n}{p}$ **to** $(i+1)\frac{n}{p} - 1$ **do** $\triangleright\ \Theta\left(\frac{n}{p}\right)$ steps
            if $j = 0$ **then**
                $M[0] \leftarrow X[0]_{Value}$
            else
                if $X[j] \neq \emptyset$ **then**
                    $M[PX[j]] \leftarrow X[j]_{Value}$

```
all processors do
    M ← DISTRIBUTE★_EREW(M, n, p)        ▷ Θ(n/p + log p) steps
return M
```

Description: Sort list $A = [a_0 \dots a_{n-1}]$ of length n where $a_i \in \{1 \dots n\}$ for all $0 \le i < n$. Based on Bahig and Bahig, 2007.

Analysis: $\mathcal{O}\left(\frac{n}{p}\log\frac{n}{p} + \left(\frac{n}{p} + \log p\right)\log p\right)$

procedure $\textsc{MergeSort}^{\diamondsuit}_{\text{EREW}}(A, n, p)$
  if $n = 1$ **then**
    return A
  if $p = 1$ **then**
    return $\textsc{SequentialSort}(A, n)$
  all processors do ▷ $\Theta\left(\frac{n}{p}\log\frac{n}{p} + \left(\frac{n}{p} + \log p\right)\log p\right)$ steps
    $\frac{p}{2}$ **processor array** $0 \dots \frac{n}{2} - 1$ **does**
      $L \leftarrow \textsc{MergeSort}^{\diamondsuit}_{\text{EREW}}(A[0 \dots \frac{n}{2} - 1], \frac{n}{2}, \frac{p}{2})$
    $\frac{p}{2}$ **processor array** $\frac{n}{2} \dots n - 1$ **does**
      $U \leftarrow \textsc{MergeSort}^{\diamondsuit}_{\text{EREW}}(A[\frac{n}{2} \dots n - 1], \frac{n}{2}, \frac{p}{2})$
  $k = \min\{p, \frac{n}{\log n}\}$ **processor array** $0 \dots k - 1$ **does**
    $A \leftarrow \textsc{Merge}^{\bigstar}_{\text{EREW}}(L, U, \frac{n}{2}, \frac{n}{2}, k)$ ▷ $\Theta\left(\frac{n}{k} + \log k\right)$ steps
  return A

Rank Sort

Rank sort algorithms count for each element, a_i, the number of of elements, c_i, that are smaller than a_i. Thus the sorted array elements $b_{c_i} = a_i$.

We only consider arrays of *unique* elements, however the algorithms can be modified to take into account arrays that contain non-unique elements. When all elements are unique then c_i is also unique over all $0 \leq i < n$.

```
for(i=0;i<n;i++){
 x=0;
 for(j=0;j<n;j++)
  if(a[i]>a[j]) x++;
 b[x]=a[i];
}
```

- Comparing each number against $n-1$ other numbers requires $n-1$ computation steps. There are n elements so there are $n(n-1)$ computational steps in total.
- For n processors, each computing the index of an element in parallel, sorting can be accomplished in $O(n)$ computational steps.
- Each processor needs access to the entire array of numbers and so this is convenient for shared memory architectures. The efficiency is $\frac{\log_2 n}{n} \times 100\%$.

Consider the use of n^2 processors. Each processor $p_{i,j}$ compares a_i with a_j. (processors $p_{i,i}$ are not actually required.) Comparison requires $O(1)$ computational steps.

Using a reduction across i, processors $p_{i,j}$ can compute the index, b_i, of element i in $O(\log_2 n)$ computational steps. In a final $O(1)$ computational step, the element a_i is written to index b_i.

The sorting is accomplished in $O(\log_2 n)$ steps. However the efficiency is $\frac{1}{n} \times 100\%$.

Using a CRCW memory architecture with concurrent writes being handled as additions, the reduction operation can be accomplished in $O(1)$ steps. Thus the sorting is accomplished in $O(1)$ steps. The efficiency is now $\frac{\log_2 n}{n} \times 100\%$.

Description: Merge two lists of size n unique elements using n processors and return merged list S of size $2n$ elements.

Analysis: $\Theta(\log n)$

Processors: n

procedure $\textsc{RankMerge}^{\diamondsuit}_{\text{CREW}}(A, B, n)$

  for $p \leftarrow 0$ **to** $n-1$ **do in parallel**

    processor p **does**

      $RA[p] \leftarrow \textsc{SequentialRank}(A, n, A[p]) + \textsc{SequentialRank}(B, n, A[p])$

      $RB[p] \leftarrow \textsc{SequentialRank}(A, n, B[p]) + \textsc{SequentialRank}(B, n, B[p])$

      $S[RA[p]] \leftarrow A[p]$

      $S[RB[p]] \leftarrow B[p]$

  return S

Bitonic Mergesort

The basis of the bitonic mergesort is the *bitonic sequence*, a list having specific properties that will be utilized in the sorting algorithm.

A monotonic increasing sequence is a sequence of increasing numbers. A bitonic sequence has two sequences, one increasing and one decreasing. Formally, a bitonic sequence is a sequence of numbers, $a_0, a_1, a_2, \ldots, a_{n-2}, a_{n-1}$, which monotonically increases in values, reaches a maximum, and then monotonically decreases in value:

$$a_0 < a_1 < a_2 < \cdots < a_i > a_{i+1} > \cdots > a_{n-2} > a_{n-1}.$$

for some $0 \leq i < n$. A sequence is also bitonic if the preceding can be achieved by shifting the number cyclically (left or right).

The bitonic sequence has an interesting property that if we compare and exchange a_i with $a_{i+n/2}$ for all $0 \leq i < n/2$, we get two bitonic sequences, where the numbers in one sequence are all less than the numbers in the other sequence. For example before:

$$3, 7, 9, 8, 6, 5, 4, 1$$

and after

$$3, 5, 4, 1, 6, 7, 9, 8.$$

The second list is now two bitonic sequences, $3, 5, 4, 1$ and $6, 7, 9, 8$.

Using this property, with $n = 2^t$ elements and n processors, after t parallel steps it is clear that a given bitonic list can be sorted. This is called a *bitonic sort operation*.

So sorting an unsorted list of numbers requires building bitonic lists and then sorting the bitonic lists.

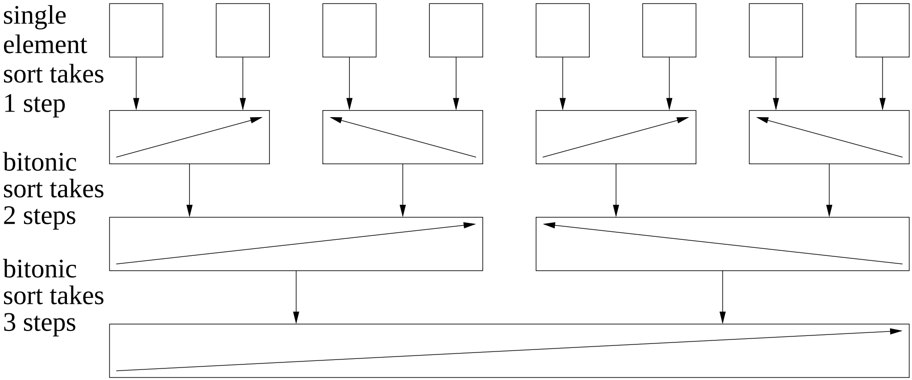

Figure: Bitonic Mergesort on 8 elements.

With $n = 2^t$ elements, there are t phases numbered $1, 2, \cdots, t$, each requiring a bitonic sorting operation (the first phase is simply sorting single elements) of t steps. Hence the total number of steps is given by

$$\sum_{i=1}^{t} i = \frac{t(t+1)}{2} = \frac{\log_2 n(\log_2 n + 1)}{2} = O(\log_2^2 n).$$

The speedup on n processors is thus $O\left(\frac{n}{\log_2 n}\right)$ and gives an efficiency of roughly $\frac{1}{\log_2 n} \times 100\%$.

Processor Optimal Parallel Merging

- From Huang and Kleinrock, 1990.
- Assume we wish to merge two sorted lists, L_1 and L_2, each of length N elements, or $2\,N$ elements in total. For this discussion we will assume that all the elements are distinct, but the approach will work in general.
- Assume we have $P = \sqrt{N}$ processors in total, or $N = P^2$.
- The optimal parallel merge algorithm will merge the lists in time $O(N/P) = O(\sqrt{N})$.

1. Divide L_1 into P sublists where the i-th sublist contains elements at locations $i, P+i, 2P+i, \ldots, N-P+i$. Also divide L_2 into P sublists similarly.
2. Have P_i merge the i-th sublist from L_1 and the i-th sublist from L_2 and put the result back to the locations originally occupied by these sublists $1 \leq i \leq P$. All P processors work simultaneously.
3. Group the resulting list after Step 2 into $2\,P$ groups with P consecutive elements in each group. Number these groups from 1 to $2\,P$. Have P_i merge groups $2i-1$ and $2i$ and put the result back to the locations originally by these two groups for $1 \leq i \leq P$. All P processors work simultaneously.
4. Group the resulting list after Step 3 into $2\,P$ groups with P elements in each group. Number these groups from 1 to $2\,P$. Have P_i merge groups $2i$ and $2i+1$ and put the result back to the locations originally occcupied by these two groups for $1 \leq i \leq P-1$. All $P-1$ processors work simultaneously.

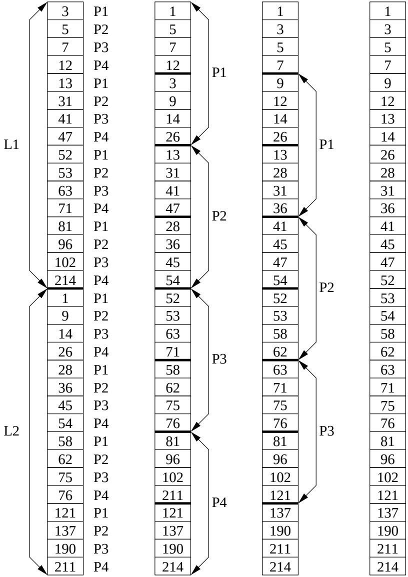

Figure: Example merging.

Intuitive understanding

- Lemma: After Step 2, every element is within $\pm P$ positions from its final position.
- Lemma: After Step 2, every group is sorted.
- Lemma: After Step 2, every element in group i is smaller than all elements in group j for $j \geq i + 2$.
- Theorem: After Step 4, the entire list is sorted.
- Since each step 1 is constant time and steps 2 to 4 take $O(\sqrt{N})$ operations, the algorithm takes $O(\sqrt{N})$ steps in total, using $P = \sqrt{N}$ processors.

Multiway Parallel Sorting Algorithm

- Consider using $P = \sqrt{N}$ processors to sort $2\,N$ elements. Assume $P = \sqrt{N} = 2^k$.
- In Phase 1, assign $2\sqrt{N}$ elements to each processor and have each processor sort those elements independently, using the best known sequential algorithm.
- After Phase 1 we have P sorted lists.
- In Phase 2, recursively merge two sorted lists into one large sorted list until there is only one list which is totally sorted.
  - After Phase 1 there are $P = 2^k$ sorted lists; therefore we need to perform k merge runs to finish the mergesort.
  - At the beginning of the i-th step, $i = 1, 2, \ldots, k$, there are $2^k/2^{i-1}$ sorted lists each with size $N_i = 2^{k+i}$ elements.
  - The number of processors available to sort two lists at the i-th step is $P_i = 2^i$. Since $2^{k+i} \geq 2^{i+i} = (2^i)^2$, $N_i \geq P_i^2$.

Analysis of the runtime

- In the first phase each processor sorts two lists of length $\sqrt{N}$, which takes $O(\sqrt{N}\log N)$ or $O(\frac{N\log N}{P})$.
- In the second phase if N_i is the length of the lists to merge at the i-th step and P_i is the number of processors then it takes $O(N_i/P_i) = O(2^{k+i}/2^i) = O(2^k) = O(N/P)$ operations at the i-th step.
- There are k steps in phase 2 and $k = \log P = \log\sqrt{N} = \frac{1}{2}\log N$. Therefore phase 2 takes $O(k\,N/P) = O(\frac{N\log N}{P})$ steps in total.
- Since phase 1 and phase 2 have the same complexity, the total algorithm takes $O(\frac{N\log N}{P})$ steps.

Huang and Kleinrock go on to show that for $P = N^{(2^k-1)/2^k}$ processors, the complexities of the merging algorithm and the sorting algorithm are $O(3^k N/P)$ and $O(3^k(N\log N)/P)$ respectively.