Distributed Sorting Algorithms

Assumptions

- There are $p$ processors sorting $n$ numbers.
- Each processor begins with $n/p$ numbers stored in the array $x$.
- All numbers are in the range $0, \ldots, M - 1$.
- When the sorting algorithm ends, each processor has a sorted list of numbers and for $i < j$ every number in processor $i$ is less than every number in processor $j$.
- Each processor can compare two numbers and swap them (if necessary) in time $\chi$.
- Each machine can send data to one other machine at a time (but cannot send and receive at the same time).
- It takes $\lambda + k/\beta$ time to send $k$ numbers to another machine.
- $id$ specifies the identifier of the current machine.
- Local sort of $k$ elements takes time approximately $C \chi k \log_2 k$.

Programming assignment (Due 4/20) Write a short program to estimate $\chi$ on the Linux Lab machines. (This requires no parallelism)

Programming assignment (Due 4/20) Write a short program using MPI to estimate $\lambda$ and $\beta$ in the Linux Lab.
1 Quicksort

Algorithm 1 Distributed Quicksort

Assume $p = 2^d$

Copy all of the numbers to processor 0

if $id == 0$ then
    for $i = 1$ to $p - 1$ do
        Receive $n/p$ numbers from processor $i$ and append them to $x$
    end for
else
    Send the contents of $x$ to processor 0
end if

Partition the numbers and send them to the appropriate processor

for $i = 0$ to $d - 1$ do
    if $id \% (p/2^i) == 0$ then
        $m \leftarrow x[0]$
        Partition $x$ so that all elements less than $m$ appear before $m$ and all elements greater appear afterward
        Transmit the portion of $x$ after $m$ to processor $id + p/2^{i+1}$
        Replace $x$ by the portion not transmitted
    else if $id \% (p/2^i) == p/2^i + 1$ then
        Replace $x$ by the numbers transmitted by processor $id - p/2^{i+1}$
    end if
end for

Perform a local sort on $x$

Analysis  Assuming that each partition is optimal (i.e. splits into equal size halves)

Collecting numbers at node 0: $\log_2 p$ rounds of transmission sending $\frac{n}{p}$ numbers in the first round, $2^{\frac{n}{p}}$ in the second, $4^{\frac{n}{p}}$ in the third. The total numbers transmitted in the sum of this geometric series which is approximately $n$. So, the time to collect the numbers is approximately $\lambda \log_2 p + n/\beta$.

There are $\log_2 p$ rounds of partition. Partitioning $k$ numbers involves $k$ comparisons (and possible exchanges) so computation takes approximately $k\chi$ time. Transmission after each stage take $\lambda + k/2\beta$ (since half the data is being sent). The size of arrays being partitioned (assuming optimal splits) are $n, n/2, n/4, \ldots$

The total computation time in the sum of these which is approximately $2n\chi$ and the communication time is approximately $\lambda \log_2 p + n/\beta$ (since approximately $n$ numbers are being transmitted).

The final step (local sorting) is off approximately $\frac{n}{p}$ numbers and takes approximately $C\chi \frac{n}{p} \log_2 \frac{n}{p}$ time

<table>
<thead>
<tr>
<th>Step</th>
<th>Computation Time</th>
<th>Communication Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Collecting numbers</td>
<td>$2n\chi$</td>
<td>$\approx \lambda \log_2 p + n/\beta$</td>
</tr>
<tr>
<td>Partitioning</td>
<td>$C\chi \frac{n}{p} \log_2 \frac{n}{p}$</td>
<td>$\approx \lambda \log_2 p + n/\beta$</td>
</tr>
</tbody>
</table>

Note: An MPI implementation will be posted on the website
2 Hyperquicksort

Algorithm 2 Hyperquicksort

{Assume \( p = 2^d \)}
Run local sort on \( x \)
for \( i = 0 \) to \( d - 1 \) do
if \( id \% p/2^i = 0 \) then
\( m \leftarrow \text{median value of } x \)
Transmit \( m \) to processors \( id + 1, \ldots, id + p/2^i - 1 \)
else
Receive \( m \) from processor \( id - (id \% p/2^i) \)
end if
Partition \( x \) so that all elements less than \( m \) appear before \( m \) and all elements greater appear afterward
if \( id \% p/2^i < p/2^{i+1} \) then
Transmit the portion of \( x \) greater than \( m \) to processor \( id + p/2^{i+1} \)
Replace \( x \) by the portion not transmitted
Receive numbers from processor \( id + p/2^{i+1} \) and merge them with \( x \) in increasing order
else
Receive numbers from processor \( id - p/2^{i+1} \) and store them in array \( y \)
Transmit the portion of \( x \) less than \( m \) to processor \( id - p/2^{i+1} \)
Replace \( x \) by the portion not transmitted and merge it with \( y \) in increasing order
end if
end for

Analysis Here, we also assume that the partition sizes are perfect.

Local sort takes \( C\chi \frac{n}{p} \log_2 \frac{n}{p} \) time
Transmitting the median occurs \( \log_2 p \) times and sends one number each time. So this step takes a total of \( \lambda \log_2 p + \log_2 p/\beta \) time.
Partition \( k \) sorted elements takes \( \chi \log_2 k \) time. There are \( \log_2 p \) partitions of approximately \( \frac{n}{p} \) elements, which takes \( \log_2 \frac{n}{p} \chi \) time. Sending and receiving the partition takes \( 2\lambda + \frac{\chi}{p} \) time. So the partitioning steps take a total of \( \chi \log_2 p + \log_2 \frac{n}{p} \chi + 2\lambda + \frac{\chi}{p}/\beta \) time. This occurs \( \log_2 p \) times.
Merging two sorted lists of size approximately \( \frac{n}{p} \) together takes approximately \( 2\chi \frac{n}{p} \) time. This also occurs \( \log_2 p \) times.
So the total computation time is \( C\chi \frac{n}{p} \log_2 \frac{n}{p} + (\log_2 \frac{n}{p} \chi + 2\lambda \frac{n}{p}) \log_2 p \). And the communication time is \( (3\lambda + (1 + \frac{n}{p})/\beta) \log_2 p \) time.

Note: An MPI implementation will be posted on the website
3 Sorting by Regular Sampling

Algorithm 3 Sorting by Regular Sampling

{Assume $n$ is a multiple of $p^2$}
Run local sort on $x$
{Calculate $p$ partitions}
Let $s$ be the numbers in $x$ stored at indices $(n/p^2), 2(n/p^2), (p - 1)(n/p^2)$
if $id != 0$ then
    Transmit $s$ to processor 0
else
    Receive $p$ numbers from each of the other processors and append them to $s$
    Run local sort on $s$
    Create array $c$ consisting of the following data: $\{0, s[p - 1], s[2(p - 1)], s[3(p - 1)], \ldots, s[(p - 1)^2], \infty\}$
end if
{Transmit partition information}
if $id == 0$ then
    Transmit $c$ to all of the other processors
else
    Receive $c$ from processor 0
end if
{Send data to appropriate machines}
Create empty array $y$
for $i = 0$ to $p - 1$ do
    if $i != id$ then
        Transmit the portion of $x$ between $c[i]$
        Receive data from processor $i$ and append it to $y$
    else
        Append the portion of $x$ between $c[i]$ and $c[i + 1]$ to $y$
    end if
end for
Sort $y$ and replace $x$ by its contents

Note: An MPI implementation will be posted on the website
4 Merge Sort

Algorithm 4 Merge Sort

\{Assume \( p = 2^d \) and \( p \) divides \( n \)}

Perform local sort on \( x \)

\[\text{for } i = d \text{ to } 1 \text{ do}\]

\[\text{if } \text{id} \mod 2^i = 0 \text{ then}\]

\[\text{Receive array } y \text{ from machine } \text{id} + 2^i - 1\]

\[\text{Merge } x \text{ and } y \text{ into array } x \text{ stored in increasing order}\]

\[\text{else if } \text{id} \mod 2^i = 2^i - 1 \text{ then}\]

\[\text{Transmit } x \text{ to machine } \text{id} - 2^{i-1} \text{ and clear } x\]

\[\text{end if}\]

\[\text{end for}\]

Homework assignment (Due 4/17)  Estimate the running time of this algorithm. You may assume that it takes \( \chi \) time to compare two numbers and copy one of them into another array or swap them.

Programming assignment (Due 4/29)  Implement merge sort using MPI.

Note:  An Boost threading and MPI implementations will be posted on the website
5 Rank Sort

The algorithm below is designed for multithreaded machines

**Algorithm 5 Rank Sort**

(Processor 0 only) Initialize the array $c$ of length $n$ with all zeros.

(Processor 0 only) Allocate an array $y$ of length $n$.

for $i = n \ast id/p$ to $n \ast (id + 1)/p - 1$ do

for $j = 0$ to $n - 1$ do

if $x[j] \leq x[i]$ then

$c[i]++$

end if

end for

$y[c[i]] \leftarrow x[i]$

end for

(Processor 0 only) Replace $x$ with $y$

---

**Homework assignment (Due 4/17)** How long does this algorithm take to run on $p$ processors. (You may assume that it takes 1 time unit to do a comparison, incrementing or assigning values. Ignore the time it takes to allocate memory for the arrays.)

**Homework assignment (Due 4/17)** What is the optimal number of processors to run this algorithm on?

**Homework assignment (Due 4/17)** Write pseudocode for the distributed version of this algorithm.

**Programming assignment (Due 5/1)** Implement rank sort using Boost threading.

**Note:** An serial implementation will be posted on the website.
6 Bucket Sort

Algorithm 6 Bucket Sort

{Assume all numbers in the range 0 to $M - 1$}
Create a $d$ dimensional array of lists (or arrays) $y$
for $i = 0$ to $n - 1$ do
    Append $x[i]$ to $y[x[i]/(M/d)]$
end for
Replace $x$ by $y[id]$
for $i = 0$ to $p - 1$ do
    if $i == id$ then
        Transmit $y[i]$ to processor $i$
        Receive data from processor $i$ and append to $x$
    end if
end for
Perform a local sort on $x$

Homework assignment (Due 4/17) Estimate the running time of this algorithm assuming that the size of each bucket is approximately the same size.

Homework assignment (Due 4/17) Assuming equal sized buckets, how fast does the algorithm run on $n$ processors. Can this be improved with a slightly different communication scheme?

Programming assignment (Due 5/4) Implement bucket sort using MPI.

Note: An serial and Boost threading implementations will be posted on the website
7 Bitonic Sort

Algorithm 7 Bitonic Sort

Homework assignment (Due 4/24) Examine the bitonic sorting algorithm we used for multithreading. Write the pseudocode for the distributed version of the algorithm.
8 Odd-Even Transposition Sort

Algorithm 8 Odd-Even Transposition Sort

Perform a local sort on $x$

for $i = 0$ to $p^2 - 1$ do

if $id \% 2 == i \% 2$ and $id < p - 1$ then

Transmit $x$ to processor $id + 1$

Receive array $y$ of length $n/p$ from processor $id + 1$

Merge $x$ and $y$ into a sorted array $z$

Replace $x$ by the first half of the array $z$

else if $id > 0$ then

Receive array $y$ of length $n/p$ from processor $id - 1$

Transmit $x$ to processor $id - 1$

Merge $x$ and $y$ into a sorted array $z$

Replace $x$ by the second half of the array $z$

end if

end for

Analysis The local sort takes $C_p n/p \log_2 n/p$ time. Each round of comparison requires sending $n/p$ numbers each direction. This communication time is $2(\lambda + n/p \beta)$. Merging two sorted arrays (of size $n/p$) into a sorted array takes $\chi n/p$ time. These last two values need to be multiplied by $p^2$. So the total run time is $C_p n/p \log_2 n/p + 2\lambda n + 2\lambda p^2 + np/\beta$. 