CS-131:Hw5 Fun with Sorting

Matthew McLaughlin

May 22, 2018

Contents

Assumptions And Notation

I use one based indexing. S is the input sequence $S_{i,j}$ is the subsequence $S_i, \ldots S_j$. P_i denotes the ith proccessor in the network. $P_{i,j}$ is the jth value stored in proccessor i. If T is a list of values the $T \Leftarrow x$ adds x to the end of list T. Let M be the name of the network.

1 Shear Sort

For this problem I am assuming that sequence S is of length $\sqrt{n} = k$ where k is positive integer representing the number of proccessors. This will make it possible to treat the sequence like a square matrix by dividing the sequence into rows and assigning each proccessor to a row. The assumption that k is a factor of n serves to simply the pseudocode and analysis. However, it does not affect the complexity of the problem. That is to say, with the assumption that n is a square number and n is divisible by k, we could still in effect treat the sequence as a square matrix as before when $k * k = n$, yet now we would need to simulate this with the procecssors we had avaiable. For example, suppose $n = 16$ and $k = 2$. In this case we could have P_1 divide its portion of the sequence in half so that the first 4 values and last 4 are treated as two different subsequences allowing P_1 to simulate the effect of having an extra proccessor.

1.1 Implementation

Below I outline the procedure *Parallel-shear-sort* for preforming shear sort on sequence S. This procedure uses helper functions described below. We are required to use a comparison exchange sorting algorithm. Assume that the subroutine *sort* used by *Parallel-shear-sort* is mergesort and that it takes an additional argument specifying wether to sort in increasing order or decreasing order

Algorithm is-even: outputs true if the agument n is even

:

Input: n an integer
Result: true if n is even false otherwise
return $n \bmod 2 = 0$

The crucial part of the *Transpose* routine is line 4 where the ith proccessor sends it's ith row to all of the other proccessors. The effect is that the old entry ith entry in P_j is overwritten with $P_{i,i}$

Algorithm Distribute-sequence: Stores a subsequence of S in the proccessors

:

Input: A network of a collection of proccessor $P_1 \ldots P_k$, Sequence S, n is number of values in S

Result: Each proccessor has a portion of S in its memory. P_i stores the subsequence $S_{a,b}$ where $a = (i-1)(\sqrt{n}) + 1$ and $b = (i)\sqrt{n}$. Thus, the routine effectively transforms S into a square matrix and distribute it amongst the proccessors so that P_i stores the ith row of the matrix

$i \leftarrow 1$
$s \leftarrow \sqrt{n}$
while $i \leq s$ **do**
    $a \leftarrow (i-1)s + 1$
    $b \leftarrow i \cdot s$
    $P_i \leftarrow S_{a,b}$
    $i \leftarrow i + 1$
end

Algorithm Transpose: Simulates a matrix transpose on the network

:

Input: A network of a collection of proccessor $P_1 \ldots P_k$, m is the number of values that each proccessor keeps in memory ($m = sqrtn$ (see *Parallel-shear-sort*)

Result: If we think of M as a matrix where each proccessor P_i in M stores the ith row in M then after this routine returns each P_i will store the ith column of M

foreach $P_i \in M$ **do**
    for $j \leftarrow 1$ *to* m **do**
        if $j \neq i$ **then**
            $P_{j,i} \leftarrow P_{i,j}$
        end
    end
end

Algorithm Output-sequence: Output the sequence obtained by scanning over the network using snake like indices

:

Input: A network of a collection of proccessor $P_1 \ldots P_k$, m is the number of values that each proccessor keeps in memory ($m = sqrtn$ see *Parallel-shear-sort*)

Output: Produces a sequence T from M according to the following rules. If i is even then scan P_i from left to right otherwise scan from right to left

```
foreach P_i ∈ M do
    T ← empty-list() if is-even(i) then
        for j ← 1 to m do
            T ⇐ P_{i,j}
        end
    else
        for j ← m down-to 1 do
            T ⇐ P_{i,j}
        end
end
```

Algorithm Parallel-sort: Produces a sorted version of S

:

Input: A network of a collection of proccessor $P_1 \ldots P_k$, Sequence S, n is the number of values in S

```
Distribute-sequence(M,S,n)
cycle_num ← 0
m ← √n
max_cycles ← log(m)
while cycle_num ≤ max_cycles do
    if is-even(cycle_num) then
        foreach P_i ∈ M do
            if is-even(i) then
                sort(P_i,increasing)
            else
                sort(P_i,decreasing)
        end
    else
        Transpose(M,m)
        sort(P_i,increasing)
        Transpose(M,m)
    cycle_num ← cycle_num + 1
end
Output-sequence(M,m)
```

1.2 Analysis

1.2.1 Amount of compare exchange operations performed by each proccessor

Recall that we use merge sort to preform compare exchange operations. Thus, the total number of comparisons performed between all the proccessors in a given cycle is $O(n \log n)$. We know by the theorem stated in the slides that the number of iterations until *Parallel-shear-sort* converges is at most $O(\log \sqrt{n}) = O(\frac{1}{2}\log(n))$. This means the total number of compare exchange operations performed over *all* cycles is at most
$(\frac{1}{2}\log n)(n \log n) = \frac{n}{2}\log^2(n)$

1.2.2 Amount of memory needed by each proccessor

Each proccessor needs to be able to store $\sqrt{n}$ items. In total $O(n)$ storage is needed.

1.2.3 Number of Cross-Bar communications cycles

The only time communications between proccessors takes place is during the transpose phase of *Parallel-shear-sort* in lines $13, 15$. This means only $\frac{1}{2}$ the time (during an odd cycle) communications will take place.

Tranpose is invoked twice in the else block. Each inovations requires n communications which mean there will be $2n$ during an odd cycle.

In 1.2.1 we showed that there are no more than $\frac{1}{2}\log n$ cycles.

Putting all these ideas together, we see the total number of cross bar communications performed by the algorithm is ...

$$(\frac{1}{2})(2n)(\frac{1}{2}\log n) = \frac{n \log n}{2}$$

which is $O(n \log n)$

2 Bitonic Sort

I assume the $n = 2^j$ for some positive integer j and that n is divisible by k, the number of proccessor I assume the existence of two function, *send* and *read* responsible for communication between proccessors on the network. These functions are not defined explicitally but the headers are given. Also, note that *Compare-exchange* is defined to work when communication is between two different proccessors or just within the same proccesor. *Output-sequence* here simply scans over each proccessors memory starting at lower index proccesors and going up outputing the contents of there memory. Since it is such a simple function I have not included it. Lastly, I assume that the input sequence S is a bitonic sequence where the first half of the sequence is nondecreasing and the second half nonincreasing. This assumption was permitted by the instructor. Futhermore, even if it wasnt, it wouldnt be very hard to trasnform an arbitrary sequence so that it was bitonic

2.1 Implementation

Algorithm send: Sends data from one proccessor to another

Input: The sending procceesor P_i, the element to be sent x, the receiving proccesor P_j

Result: Element x is sent from P_i to P_j. Whatever is in P_j read buffer is automatically overwritten with x

Algorithm read: procceesor P_i reads its read buffer and returns whatever value is in there.

Input: The proccessor whose read buffer is to be read

Result: The value inside of P_i read buffer. There is only a valid value here if something was sent to P_i from another proccessor. Otherwise the buffer is garbage

Algorithm Distribute-sequence: Stores a subsequence of S in the proccessors

Input: A network of a collection of proccessor $P_1 \ldots P_k$, Sequence S, n is number of values in S, k is the number of proccessors

Result: Calculates how many values to load into each proccessor based on n then reads that many consecutive values into the correct processor

$i \leftarrow 1$
$s \leftarrow \frac{n}{k}$
while $i \leq k$ **do**
    $a \leftarrow (i-1)(s)+1$
    $b \leftarrow i \cdot s$
    $P_i \leftarrow S_{a,b}$
    $i \leftarrow i+1$
end

Algorithm Compare-exchange: Compares the elements between two processors and swaps them if the first proccessors element is greater than the seconds

Input: A proccessor P_i, the index of the element for comparision in P_i called a, A second proccessor P_j, the index of the element for comparison in P_j called b

Result: We assume that P_i should store smaller values and P_j larger. Thus, P_i and P_j send eachother values (if they are different proccesors) and P_i keeps the minimum of the two values whereas P_j keeps the maximum of the two

if $i \neq j$ **then**
    send$(P_i, P_{i,a}, P_j)$
    send$(P_j, P_{j,b}, P_i)$
    $x \leftarrow \text{read}(P_i)$
    $P_{i,a} \leftarrow \min(P_{i,a}, x)$
    $x \leftarrow \text{read}(P_j)$
    $P_{j,b} \leftarrow \min(P_{j,b}, x)$
else
    $x \leftarrow P_{i,a}$
    if $x > P_{i,b}$ **then**
        $P_{i,a} \leftarrow P_{i,b}$
        $P_{i,b} \leftarrow x$
    end

Algorithm Single-proccessor-bitonic-sort: Given a proccessor P whose memory stores a bitoncic sequence.

:

Input: A single proccessor from the network P_i, n number of proccessors in network that P_i is from, k number proccessor in network that P_i is from

$list\_size \leftarrow \frac{n}{k}$
$j \leftarrow 1$
$offset \leftarrow list\_size \cdot 2^{-j}$
while $offset \geq 1$ **do**
  for $i = 1$ *to* $(list\_size \cdot 2^{-1})$ **do**
    Compare-exchange(P_i, i, P_i, $i + offset$)
  end
end

Algorithm Parallel-bitonic-sort: Produces a sorted sequence given a bitonic sequence S where the first half of S is nondecreasing and the second half is nonincreasing

:

Input: A network of a collection of proccessor $P_1 \ldots P_k$, Bitonic sequence S, n is number of values in S, k is the number of proccessors

Distribute-sequence(M,S,n,k)
$list\_size \leftarrow \frac{n}{k}$
$j \leftarrow 1$
$offset \leftarrow list\_size \cdot 2^{-j}$
while $offset \geq 1$ **do**
  for $i = 1$ *to* $\frac{k}{2}$ **do**
    for $c = 1$ *to* $(list\_size \cdot 2^{-1})$ **do**
      $a \leftarrow i$
      $b \leftarrow i + offset$
      Compare-exchange(P_a, c, P_b, c)
    end
  end
end
foreach P_i **do**
  Single-procceesor-bitoncic-sort(P_i)
end
Output-sequence(M)

2.2 Analysis

2.2.1 Amount of compare exchange operations performed by each proccessor

There will be $\frac{n}{4} \log n$ iterations for the while loop in *parallel-bitonic-sort*. *Single-proccesor-bitonic-sort* is called once for each proccessor after the while loop. There are $n \log n$ total comparisons as a result of this function. Thus in total there are $O(n \log n)$ communications amongst proccessors.

2.2.2 Amount of memory needed by each proccessor

A given proccessor will need to store $\frac{n}{k}$ elements. In total of course the storage is $O(n)$

2.2.3 Number of Cross-Bar communications cycles

In this case the analaysis of the cross bar communication is idnetical to 2.2.1 thus there are $O(n \log n)$ cross bar communications cycles