Homework 2

José Javier González Ortiz

September 29, 2015

1 Efficiency Analysis

Given the Serial program we can analyze its Big-O notation. From the given information about initialize, statement 3, statement 5 and finalize we can calculate the serial time as follows:

$$\text{SerTime}(n) = \Theta(n) + n \cdot (\Theta(1) + n\Theta(1)) + \Theta(1) = \Theta(n^2) \tag{1.1}$$

Since we can perfectly parallelize the inner loop the Parallel time without any communication will be:

$$\text{ParTime}(n,p) = \Theta(n) + n \cdot \left(\Theta(1) + \frac{n}{p}\Theta(1)\right) + \Theta(1) = \Theta\left(\frac{n^2}{p}\right) + \Theta(n) \tag{1.2}$$

We cannot discard the $\Theta(n)$ term since for $p = \Theta(n)$ will as important as the first term. Introducing communication into the algorithm yields the following efficiencies:

a) Introducing a $\Theta(1)$ communication in the inner loop produces no changes in the big O representation of the Parallel Time

$$\text{ParTime}(n,p) = \Theta(n) + n \cdot \left(\Theta(1) + \frac{n}{p}\Theta(1)\right) + \Theta(1) = \Theta\left(\frac{n^2}{p}\right) + \Theta(n) \tag{1.3}$$

The efficiency will be:

$$\text{Efficiency}(n,p) = \frac{\Theta(n^2)}{p\left(\Theta\left(\frac{n^2}{p}\right) + \Theta(n)\right)} = \frac{\Theta(n^2)}{\Theta(n^2) + \Theta(np)} \tag{1.4}$$

This will only be constant as long as $np = O(n^2)$, namely if $p = O(n)$. If $p = \omega(n)$ the efficiency will drop to zero, however we know by definition that $p \leq n$ since there cannot be more processors than iterations of the inner loop.

b) Introducing a $\Theta(p)$ time in the outer loop the Parallel time will be:

$$\text{ParTime}(n,p) = \Theta(n) + n \cdot \left(\Theta(1) + \frac{n}{p}\Theta(1) + \Theta(p)\right) + \Theta(1) = \Theta\left(\frac{n^2}{p}\right) + \Theta(np) \tag{1.5}$$

With this expression the efficiency is:

$$\text{Efficiency}(n,p) = \frac{\Theta(n^2)}{p\left(\Theta\left(\frac{n^2}{p}\right) + \Theta(np)\right)} = \frac{\Theta(n^2)}{\Theta(n^2) + \Theta(np^2)} \tag{1.6}$$

As before, this ratio will tend to a constant given that $np^2 = O(n^2)$. Simplifying the expression we get $p = O(\sqrt{n})$. For $p = \omega(\sqrt{n})$ the efficiency will go to zero.

c) Introducing a $\Theta(\sqrt{n/p})$ time in the outer loop the Parallel time will be:

$$\text{ParTime}(n,p) = \Theta(n) + n \cdot \left(\Theta(1) + \frac{n}{p}\Theta(1) + \Theta\left(\sqrt{\frac{n}{p}}\right)\right) + \Theta(1) = \Theta\left(\frac{n^2}{p}\right) + \Theta(n) + \Theta\left(n\sqrt{\frac{n}{p}}\right) \quad (1.7)$$

The efficiency in this case follows a somewhat more complicate expression

$$\text{Efficiency}(n,p) = \frac{\Theta(n^2)}{p\left(\Theta\left(\frac{n^2}{p}\right) + \Theta(n) + \Theta\left(n\sqrt{\frac{n}{p}}\right)\right)} = \frac{\Theta(n^2)}{\Theta(n^2) + \Theta(np) + \Theta(\sqrt{n^3 p})} \quad (1.8)$$

To get a better idea of the asymptotic behavior we can divide by $\Theta(n^2)$ and get

$$\text{Efficiency}(n,p) = \frac{\Theta(1)}{\Theta(1) + \Theta\left(\frac{p}{n}\right) + \Theta\left(\sqrt{\frac{p}{n}}\right)} \quad (1.9)$$

From this we have to satisfy both:

$$\frac{p}{n} = \Theta(1) \qquad \sqrt{\frac{p}{n}} = \Theta(1) \quad (1.10)$$

Which will only happen if $p = O(n)$. Then the efficiency will tend to a constant. As we said before $p \leq n$ so this will always be satisfied

2 Reduce Sum

a) We want to minimize the communication in order to improve the efficiency, so the best approach will be summing the n/p values in each processor and then adding these intermediate sums. Since they are arranged in a 1-dimensional mesh the running time of the communication is $\Theta(p)$. The procedure would be something similar to the one described in Algorithm 1.

Algorithm 1 1D reduce(sum,A)

Require: Each processor has the common values p and n, as well as the local value id

procedure REDUCE(SUM,A)
  $s \leftarrow sum[A(id \cdot n/p : (id+1) \cdot n/p)]$
  if $id > 0$ **then**
    $receive(id-1, t)$
    $s \leftarrow s + t$
  if $id < p - 1$ **then**
    $send(id+1, s)$
  return s

b) As we can see from the previous question the algorithm will take $\Theta(n/p) + \Theta(p)$ where $1 \leq p \leq n$. It is easy to see that for both $p = \Theta(1)$ and $p = \Theta(n)$ the running time is $\Theta(n)$. We want to minimize so by making the terms being equal we will minimize the running time.

$$\Theta\left(\frac{n}{p}\right) = \Theta(p) \rightarrow p = \Theta(\sqrt{n}) \quad (2.1)$$

Therefore the best running time will happen when $p = \Theta(\sqrt{n})$ and will be $\text{ParTime} = \Theta(\sqrt{n})$

c) In a fully connected computer, given a binary operator the best communication will be $\Theta(\log p)$. In this case the running time is going to be $\Theta(n/p) + \Theta(\log p)$.

Since due to the communication constraint we know that there is a $\Omega(\log n)$ lower bound we can see that we cannot do better than getting rid of the linear term. This happens for $p = \Theta(n)$ which minimizes the time to $\text{ParTime} = \Theta(\log n)$

3 SIMD Hypercube

3.1 Analysis

As a first step is good to think about which is the minimum ideal runtime given the communication constraints of an hypercube. Let p_M be the processor with the maximum value $v_M \in V : v_M \geq v_i \quad \forall v_i \in V$. There will exist a processor $\overline{p_M}$ such that the minimum distance from p_M measured in edges will be $\log p$. To reason this, just think about this maximum point as a vertex in the hypercube and the processor $\overline{p_M}$ as the opposite vertex, since the hypercube has $\log p$ dimensions there is no shortest path between them. Furthermore, the coordinates $\overline{p_M}$ will be the bitwise complement of the coordinates of $\overline{p_M}$.

Thus the algorithm will only be correct if $v_{\overline{p_M}} = v_M$ which will need at least $\Omega(\log p)$ communications. We have found a logarithmic lower bound in our running time. An efficient algorithm that solves the problem can be thought in the following manner:

> In each timestep i , each processor exchanges its value with its neighbor in the i^{th} dimension and stores as its new value the maximum of the two. After $\log p$ steps all the processors have the maximum value in the array.

This algorithm is described in the Algorithm 2.

3.2 Algorithm

Algorithm 2 SIMD Hypercube AllReduce(max)

Require: Each processor has the common value p and the local values pid and V, as well as the variablew M and R

The controller has the variables dim and x

procedure ALLREDUCE(max, $V(0 : p-1)$)	
$M \leftarrow V$	This instruction is transmitted and executed in the processors
$x \leftarrow \lg p$	This instruction is executed in the Controller
for $dim = 0$ **to** $x - 1$ **do**	This instruction is executed in the Controller
$exchange(dim, M, R)$	This instruction is transmitted and executed in the processors
if $R > M$ **then**	This instruction is transmitted and executed in the processors
$M \leftarrow R$	This instruction is transmitted and executed in the processors
return M	This instruction is transmitted and executed in the processors

3.3 Correctness

For correctness first we state that given any processor p it has only one neighbor in the dimension *dim*, namely $p \oplus 2^{dim}$. At each time step the number of different values in the array is at least halved because the processors are saving the maximum value and each processor is *exchanging* the value with only one other processor. After $\log p$ steps we will have gone through all the dimensions and all the processors will have the maximum value v_M.

Namely, let the p_M be the processor with the maximum value $v_M \in V : v_M \geq v_i \quad \forall v_i \in V$, and be the bit representation of its *pid* $b_{x-1}b_{x-2}\ldots b_2b_1b_0$. After each time step *dim* the processors with *pids* of the form shown in Equation 3.1 have the maximum value because they have received it in any of the previous iterations. After $x = \log p$ steps all the processors will be within this set.

$$b_{x-1}b_{x-2}\ldots b_{dim+2}b_{dim+1} \underbrace{X\ldots XX}_{dim+1 \text{ times}} . \tag{3.1}$$

We can also prove that given any processor p there exists a chain of *exchanges* less than $\log p$ that goes from p_M to p.

3.4 Running Time

Finally, to calculate the running time we have to take into account that both the *exchange* statement and the *if* statement are $\Theta(1)$ because we are sending before receiving. Since there are $\log p$ steps the total running time will be $\Theta(\log p)$ which is the best we can do since there is a logarithmic lower bound as we saw due to the communication constraints.