Homework 1

Hai Zhu

due date: September 13, 2017

Question

Write programs to compute `scan(sum)`, that is, to have processor i end up with the value $\sum_{j=0}^{i} v(j)$, where $v(j)$ denotes the v value in processor j, for the two computers listed below

1. Write an efficient program, in pseudo-code, to compute `scan(sum)`. You must make it clear that your program is correct. By an efficient program I mean one that minimizes worst-case running time, as measured in O-notation.
2. Produce an O-notational analysis of the running time of your algorithm as a function of p. Assume that all processors start at the same time.

Do this for

1. A completely connected computer (also called fully connected), where every processor is connected to every other one.
2. A 2-dimensional mesh. For this computer p is a perfect square. For processor i, it is only connected to its up, down, left and right neighbors.

Solution

1. Completely connected computer.

   (a) (pseudo-code) We want to decompose the problem into parts. We could try to do something like finding the `scan(sum)` on the first half, and simultaneously on the second half. The first half is correct, and the second half need to be corrected with the last partial sum value from the first half. And this can goes recursively back to the problem of size 2. The whole algorithm can be illustrated as the following procedure

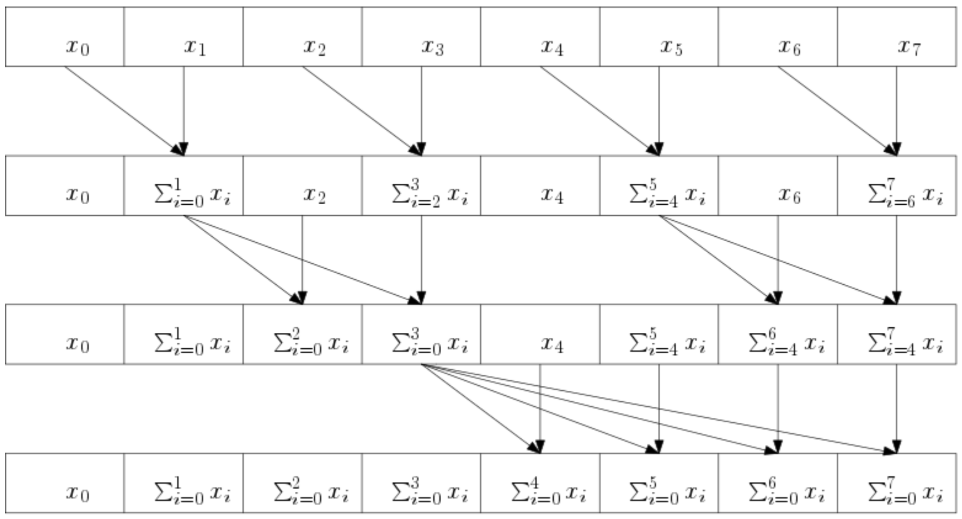
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
x_0 $\sum_{i=0}^{1} x_i$ x_2 $\sum_{i=2}^{3} x_i$ x_4 $\sum_{i=4}^{5} x_i$ x_6 $\sum_{i=6}^{7} x_i$
x_0 $\sum_{i=0}^{1} x_i$ $\sum_{i=0}^{2} x_i$ $\sum_{i=0}^{3} x_i$ x_4 $\sum_{i=4}^{5} x_i$ $\sum_{i=4}^{6} x_i$ $\sum_{i=4}^{7} x_i$
x_0 $\sum_{i=0}^{1} x_i$ $\sum_{i=0}^{2} x_i$ $\sum_{i=0}^{3} x_i$ $\sum_{i=0}^{4} x_i$ $\sum_{i=0}^{5} x_i$ $\sum_{i=0}^{6} x_i$ $\sum_{i=0}^{7} x_i$

Figure 1: One possible parallelism

One issue with this algorithm is that we could see immediately that the processor in the middle is taking too much responsibility in sending message to the second half, while most of other processors in the first half are in idle. And eventually this parallelism would cost the processors in the middle of their corresponding halves to do sending working of size $\Theta(p)$. Therefore even the number of steps is about $\Theta(\log p)$, the work needs to be done in a single processor in each step is $\Theta(2^i)$, where 2^i will eventually be about size of p.

We can make less processors waiting in idle in each step by letting them each do a constant amount of sending work to minimize message sending time. Just like before, processors send message to other processors that further and further away at each step.

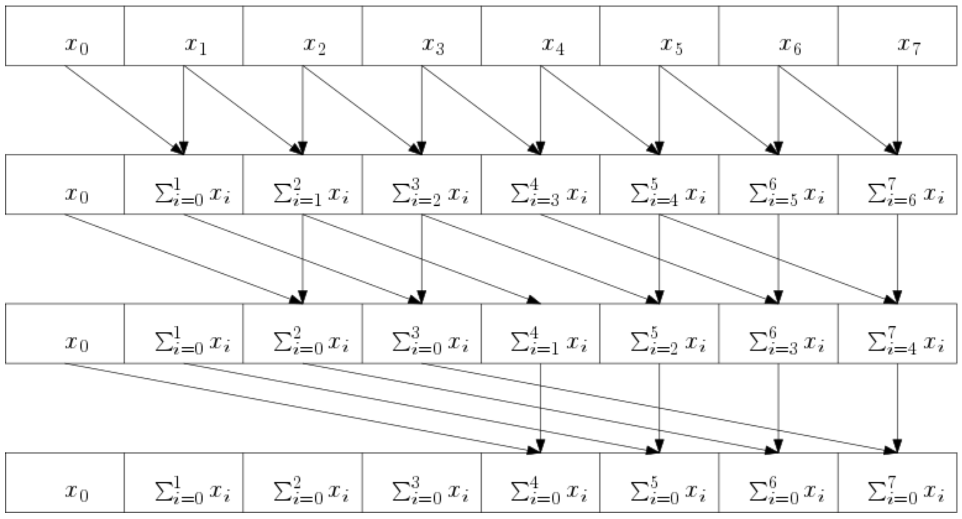
x_0 x_1 x_2 x_3 x_4 x_5 x_6 x_7
x_0 $\sum_{i=0}^{1} x_i$ $\sum_{i=1}^{2} x_i$ $\sum_{i=2}^{3} x_i$ $\sum_{i=3}^{4} x_i$ $\sum_{i=4}^{5} x_i$ $\sum_{i=5}^{6} x_i$ $\sum_{i=6}^{7} x_i$
x_0 $\sum_{i=0}^{1} x_i$ $\sum_{i=0}^{2} x_i$ $\sum_{i=0}^{3} x_i$ $\sum_{i=1}^{4} x_i$ $\sum_{i=2}^{5} x_i$ $\sum_{i=3}^{6} x_i$ $\sum_{i=4}^{7} x_i$
x_0 $\sum_{i=0}^{1} x_i$ $\sum_{i=0}^{2} x_i$ $\sum_{i=0}^{3} x_i$ $\sum_{i=0}^{4} x_i$ $\sum_{i=0}^{5} x_i$ $\sum_{i=0}^{6} x_i$ $\sum_{i=0}^{7} x_i$

Figure 2: Another parallelism with less message sending time

And the distance they are communicating depends on the recursive step they are currently in (i.e. 2^i). It also works when p is not a power of 2. For example, if we have one more processors in the above case, then we would have an last step, where `proc0` sends x_0 to `proc8`. The pseudo-code is as below:

Algorithm 1 completely connected

```
procedure SCAN(SUM)
    xsum ← $x$
    for $j = 0$ to $\lceil \log_2 p - 1 \rceil$ do        ▷ about log($p$) steps for all proc
        if $id < p - 1 - 2^j$ then
            send( xsum, $id + 2^j$)        ▷ pass partial sum to the right of distance $2^j$
        end if
        if $id \geq 2^j$ then
            receive( xin, $id - 2^j$)        ▷ receive partial sum from the left of distance $2^j$
            xsum ← xsum + xin
        end if
    end for
end procedure
```

(b) (run time analysis) Assume that all processors start at the same time, sending and receiving one message costs constant time c_1, adding values costs constant time c_2. It would take about $\log_2 p$ levels of steps. And at each step, every processor is only doing constant amount of sending, receiving, and adding work. Therefore we have the max running time is

$$T(p, p) = \Theta((2 * c_1 + c_2) \log_2 p) = \Theta(\log_2 p) \tag{1}$$

2. 2-d mesh

(a) (pseudo-code) In a completely connected machine, Algorithm 1 achieves maximum run time is order of $\log p$. In 2-d mesh case, message passing is going to cost more. The best we can hope for is probably order of $\sqrt{p}$, because in order for our last processor to know about the value stored in `proc0`, a length $2\sqrt{p}$ path needs to be passed by x_0.

Since the naive serial `scan(sum)` has run time cost of $\Theta(p)$, we can do the this along each line of the square mesh, which would cost $\Theta(\sqrt{p})$. Each processor receive the partial sum, add its own value, and then pass this updated partial sum to its right processor. And then only for the last column, from top to bottom, we do the same thing. So now the last column has the correct partial sum. The last thing we do is to broadcast partial sum received by the last column to all its left processors. In pseudo-code. These 3 steps are as follows:

Algorithm 2 2-d mesh

H

```
procedure Scan(sum)
    xsum ← $x$
    row ← $\lfloor id/\sqrt{p} \rfloor$, column ← $id(mod\sqrt{p})$
    if column = 0 then                          ▷ step1: scan sum row-wise from left to right
        $send$( xsum, $id + 1$)
    else if column < $\sqrt{p} - 1$ then
        $receive$( xin, $id - 1$)
        xsum ← xsum + xin
        $send$( xsum, $id + 1$)
    else
        $receive$( xin, $id - 1$)
        xsum ← xsum + xin
    end if
    if $((id + 1)mod\sqrt{p}) = 0$ then         ▷ step2: scan sum of right most column
        if row= 0 then
            $send$( xsum, $id - \sqrt{p}$)
        else if $row < \sqrt{p} - 1$ then
            $receive$( xin, $id - \sqrt{p}$)
            xsum ← xsum + xin
            $send$( xsum, $id + \sqrt{p}$)
            $send$( xin, $id - 1$)
        else
            $receive$( xin, $id - \sqrt{p}$)
            xsum ← xsum + xin
            $send$( xin, $id - 1$)
        end if
    else                                        ▷ step3: pass partial sum from right to left
        if column > 0 then
            $receive$( xin, $id + 1$)
            xsum ← xsum + xin
            $send$( xin, $id - 1$)
        else
            $receive$( xin, $id + 1$)
            xsum ← xsum + xin
        end if
    end if
end procedure
```

Take a 4×4 mesh for example, the following figure illustrates all 3 steps:

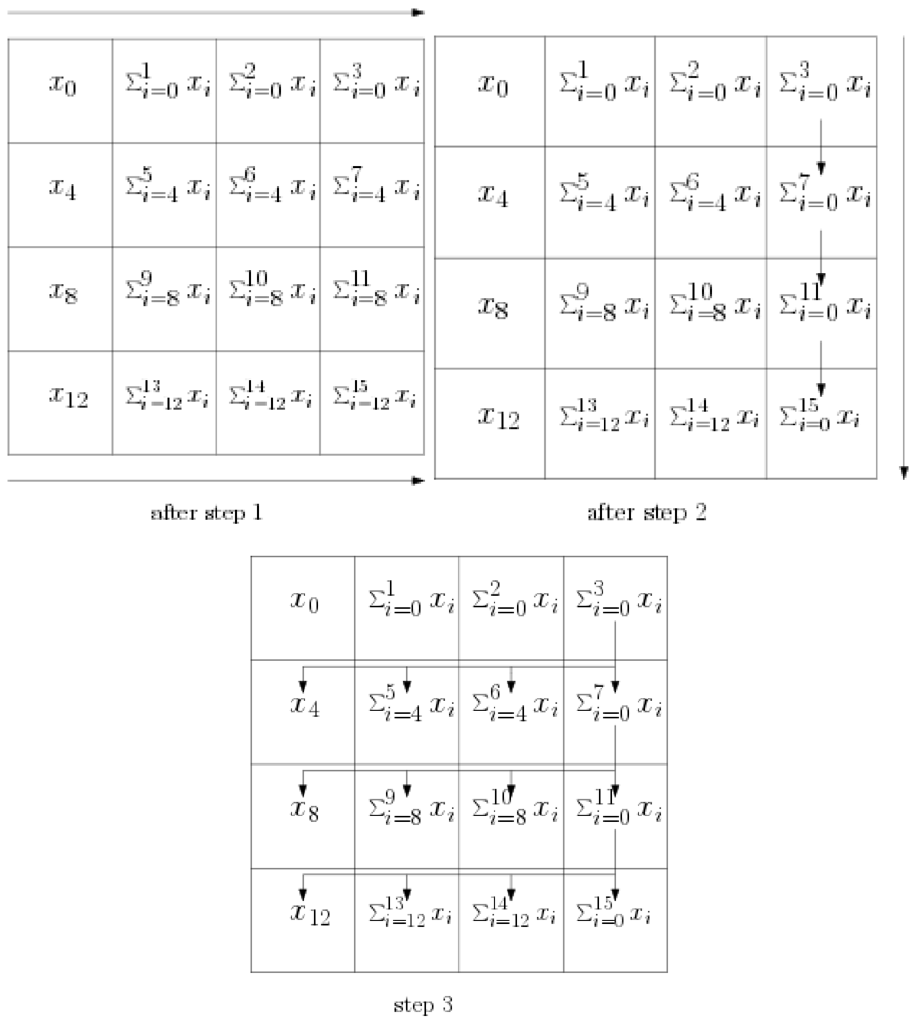

Figure 3: 2d mesh scan sum

(b) (run time analysis) In step 1, processors in the first column won't start unless they receive value from zeroth column, and then do one adding. This goes on until the $(\sqrt{p}-1)-th$ column. Therefore it is no difference compare to serial `scan(sum)`. It will cost $\Theta(\sqrt{p})$ amount of time. In step 2, processors at the last column are working. And using the same analysis as in step 1, we get the run time is also $\Theta(\sqrt{p})$. In step 3, processors in the right most column is sending value from right to left processor by processor. And each processor from left $\sqrt{p}-1$ columns is just receiving once, and then do one more adding. The time cost is also $\Theta(\sqrt{p})$. Therefore the total run time cost is

$$T(p,p) = \Theta(\sqrt{p}) \tag{2}$$