

CSCE 5160 Parallel Processing

Midterm Examination: Thursday October 8, 2009
in F 219

Help Session: Tuesday October 6, 2009

NO CLASS Thursday Oct.1, 2009

Homework #4 (due Oct. 6, 2009)

4.12 (for one-to-one, broadcast)

4.14 (for one-to-one, broadcast)

4.19

Chapter 4:

Cost of communication

Cost of broadcast

Cost of scatter

Cost of all-to-all broadcast

Chapter 5: Performance and complexity analysis

Execution time

Computation time, communication time,
synchronization time, etc

Speed up, Efficiency

Amdahl's law

CSCE 5160 Parallel Processing

Amdahl's law.

Speed up is significantly affected by the serial fraction

$$S = T_s / T_p$$

Let f be the serial fraction of the total program.

$$\text{So, } S = 1 / ((1-f)/p + f) = (p / [1 + p*(f-1)])$$

Parallel overhead.

Note the only useful work = T_s

With p processors if the execution time is T_p , work done = $p * T_p$

$$\text{So, overhead time} = T_0 = p * T_p - T_s$$

Note that the overhead may be a function of the number of processors.

Efficiency = (Speed-Up) / #processors

Ideal Efficiency = 100%

Super Linear Speed Up That is, Efficiency greater than 100%

or if we 5 processors, we may get 6 times speed up. Is it possible?

CSCE 5160 Parallel Processing

Iso-Efficiency

n	p=1	p=4	p=8	p=16	p=32
64	1	0.8	0.57	0.33	0.17
192	1	0.92	0.8	0.6	0.38
320	1	0.95	0.87	0.71	0.5
512	1	0.97	0.91	0.8	0.62

For example, in order to maintain a fixed efficiency off say 0.8,
if we have $p=4$ processors, we need $n = 64$
if we have $p=8$ processors, we need $n = 192$
if we use $p=16$ processors, we need $n=512$
If we use $p=32$ processors, we need $n = 1280$

Here our goal was to maintain a fixed efficiency -- we call this ISO-Efficiency.

CSCE 5160 Parallel Processing

Scalability Issues:

Gustavson's Law. Here instead of comparing the execution times we will try to see how much more work a multiprocessing system can perform in the same amount time.

Suppose in some fixed time, uni-processor can perform W work,
Of which f fraction is serial.

Then p processors can perform: $W*f + (1-f)*p*W$ in the same fixed time.

Scalability = (work-by-p-processors)/(work-by-uni-processor)

$$= [f*W + (1-f)*W*p] / W = f + (1-f)*p$$

At least this looks more positive than Amdhal's law as we add processor (p is in the numerator)

There are other ideas on how to measure scalability

For example, can we show that the increased memory capacity can increase scalability (rather than super-linear speed up)

CSCE 5160 Parallel Processing

The extra work in a parallel implementation that we do not do in a serial implementation is defined as overhead.

For adding n numbers on a single node, we do not need any communication, but we need communication for parallel implementation.

Overhead depends on the problem size as well as on the number of processors

$$T_o = p * T_p - T_s = f(W, p)$$

Now we can define the execution time on p processors as

$$T_p = [W + T_o(W, p)] / p \quad \text{-- } W \text{ is execution time on one processor}$$

Speed up $S = W / T_p = (W * p) / [W + T_o(W, p)]$

Efficiency $E = S / p = 1 / [1 + T_o(W, p) / W]$

As W increases, efficiency should increase for fixed p -- overhead, hopefully does not increase exponential with W

As p increases, efficiency decreases for a fixed W , since overhead increases with p

CSCE 5160 Parallel Processing

We need to find a function which tells us how to increase W with increasing number of processors p , to maintain the same efficiency. Such a function is called **iso-efficiency function**.

$$E = S/p = 1 / [1 + T_o(W,p)/W]$$

$$W = (E/1-E) * T_o(W,p) = K * T_o(W,p) \quad \text{-- This is the function we are looking for.}$$

For a given Efficiency and p , we can find W

Consider performing matrix multiplication on a 2-D mesh.

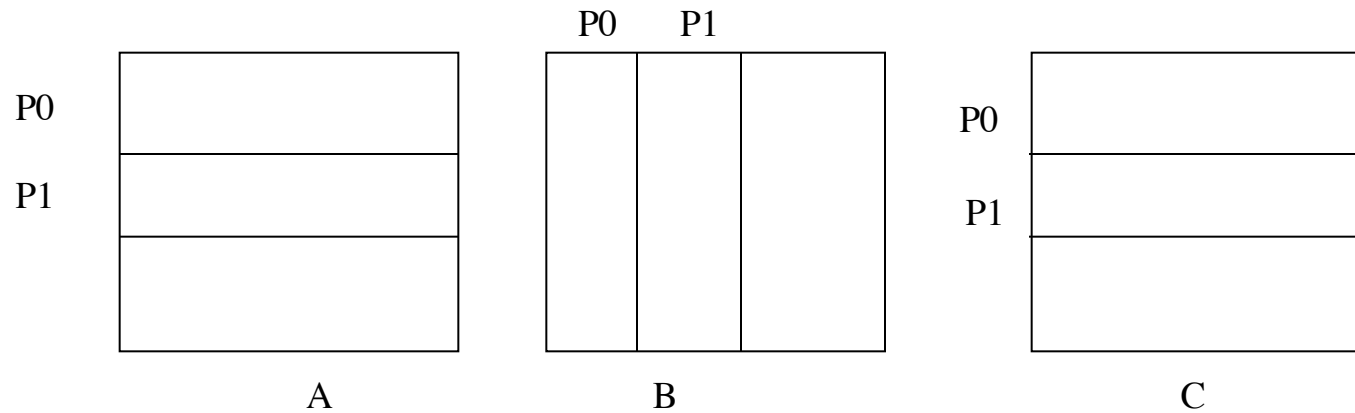
Suppose we set the number of processors = p and we are looking at matrices of $n*n$.

How can we distribute the matrices to processors?

How about n/p rows of A to each processor, and n/p columns B to each processor

Each processor will compute n/p rows of C .

CSCE 5160 Parallel Processing



To Compute the elements of C do we need to communicate?

Consider the C elements to be computed by processor P0

$C[0,0], C[0,1], \dots, C[0, n/p-1],$ $C[0, n/p] \dots C[0, n-1]$
 $C[1,0], C[1,1], \dots, C[1, n/p-1],$ $C[1, n/p] \dots C[1, n-1]$

 $C[n/p-1,0], C[n/p-1,1], \dots, C[n/p-1, n/p-1],$ $C[n/p-1, n/p] \dots C[n/p-1, n-1]$

CSCE 5160 Parallel Processing

Need the columns of B from other processors.

Each processor needs the columns assigned to the all other processors.

We can use all to all broadcast on 2-D mesh of $p^{1/2} * p^{1/2}$

To broadcast one message we have $2 * [t_s + t_w * m * (\sqrt{p}/2)]$

But we need to broadcast p messages, (all to all)

– actually for all to all we need only p-1

But, we need to send n/p columns from each processor to each processor

We can consider the message size as $(n/p)*n$ words., and $t_w = t_s = 1$.

So, the actual cost is $2 * (p^{1/2} - 1) + (p - 1)*(n^2/p)$

If we ignore the first term we have the time for communication = $(p - 1)*(n^2/p) = O(n^2)$

Each processor also computes $(n/p)*n = (n^2 / p)$ elements of C.

How many computational steps are needed?

For each element we need n multiplications and n additions

So the total number of computations per processor = $2(n) * (n^2 / p) = 2* n^3 / p$

CSCE 5160 Parallel Processing

Total Execution time= $(n^2) + 2 * n^3 / p$

Can we write this as $[W + T_o(W,p)] / p$

$$T_p = [n^3 + (n^2 * p) / 2] / p = [W + T_o(W,p)] / p$$

Note for matrix multiplication, $W = n^3$

$$T_o(W,p) = (n^2 * p) / 2 = (W^{2/3} * p) / 2$$

The isoefficiency function is $K * T_o(W,p) = K * (W^{2/3} * p) / 2$

$$\text{Speedup} = W / T_p = p / [W + (W^{2/3} * p) / 2]$$

$$E = \text{Speedup} / p = 2 / [2 + p * W^{-1/3}]$$

$$\text{Or } W = [(p * E) / (2 - E)]^3$$

So to maintain the same efficiency, if use the above equation to find the work that must be increased as p increases.

CSCE 5160 Parallel Processing

E	Processors					
	4	8	16	32	64	128
0.6	5.04	40.30	322.43	2579.41	20635.24	165081.94
0.75	13.82	110.59	884.74	7077.89	56623.10	452984.83
0.8	18.96	151.70	1213.63	9709.04	77672.30	621378.37
0.9	35.05	280.43	2243.41	17947.31	143578.49	1148627.95

Note that these numbers reflect values based on assumptions on cost of communication
Computation – we assumed the same time units for communication and computation

Also the number are for W but Work is proportional to n^3 .
So, you need to convert these work numbers into size of matrices.

CSCE 5160 Parallel Processing

Consider our example of bucket sort

If we decompose the problem based on input data, each processor will sort n/p inputs. What is the computation time to sort the numbers into buckets?

$O(n/p)$

How long will it take to sort each bucket?

$O(n \log(n))$

Note that it possible to have all n elements fall in one bucket

How long it takes to sort the numbers sequentially

We will not use bucket sort for this purpose

Use the best possible sequential algorithm (quick sort)

So, sequential complexity = $O(n \log(n))$

NO Speed up!

CSCE 5160 Parallel Processing

Let us consider a different decomposition – based on outputs

We allocate r/p buckets per processor.

Each processor looks at all n inputs and picks only those that belong to its bucket

Complexity = $O(n)$

But, again, we may end up all n elements in the same bucket so it takes

$O(n \log(n))$ to sort each bucket

Can we use more than r processors (more than one processor per bucket)?

So, why should we be interested in this algorithm?

If we assume that the numbers are randomly distributed, we expect each bucket to end up with approximately n/r numbers and sorting them will take $O((n/r) \log(n/r))$

CSCE 5160 Parallel Processing

Ideally, we would like a linear function for Iso-efficiency -- if we double the number of processors, we double the problem size to keep same efficiency

Is there a minimum bound?

Suppose the problem size is W .

We can only use a certain number of processors to compute this work in parallel.

If W grows slower than p (that is iso-efficiency function is sub-linear) then eventually, we have some processors doing no work and cannot be used effectively.

So, the lowest possible value for the iso-efficiency = $O(p)$

Concurrency vs Isoefficiency

The maximum parallelism in an algorithm – degree of concurrency
= the maximum number of processors we can use

CSCE 5160 Parallel Processing

Note the degree of parallelism, say $C(W)$ is the maximum number of operations that can be performed in parallel.

Consider matrix multiplication.

If there is not cost for communication or synchronization, what is degree of parallelism?

Can we use n^3 operations in parallel?

Yes, but then you need to accumulate results. If we can accumulate the results somehow in parallel, then $C(W) = O(n^3)$

Note that Work for matrix multiplication = work done by a single processor = $O(n^3)$

In this ideal case, the more work we have, more is the concurrency!

CSCE 5160 Parallel Processing

In reality, since we cannot accumulate in parallel, $C(W) = O(n^2)$.

So, as the problem W increases, the number of processors that we can use increases as $O(W^{2/3})$

Or as we increase the number of processors p , the work we can do grows as $O(p^{3/2})$
to maintain the same efficiency (iso-efficiency)

What is minimum time in which we can solve a problem?

What is the minimum value of T_p ?

And what is the number of processors we need to to achieve this minimum time?

Minimum number of processors is obtained using first order derivative

$$d(T_p)/dt = 0$$

CSCE 5160 Parallel Processing

Consider the example of accumulating the results of some parallel computations

Or say adding n numbers using p processors

$$T_p = O\{ (n/p) + 2*\log p\}$$

note the “2” is because of communication cost + computation cost
assuming that communication and computation takes the same time

What is the minimum value as we increase p ?

$$T_{p=\text{infinity}} = 2*\log p.$$

Before we move to specific parallel algorithms for various applications,
Let us discuss costs of synchronization in shared memory systems.

(we will return to Chapter 2 and discuss cache memory coherency issues
And implementation of barriers).

Revisit performance limited by memory and caches.

CSCE 5160 Parallel Processing

Consider problem 2.2. (HW 1). We bring 4 words on each access to DRAM.

Consider the loop

```
for(i=0; i< dim; i++) dot_prod += a[i]*b[i];
```

We need to get both a and b elements. We get 4 a and 4 b elements in 2 DRAM accesses or in 200 cycles (=200ns).

But we complete 4 multiplications and 4 additions using the data (or 8 operations) or 25ns per operation or 40MFLOPS

Without memory delay, we should expect 1GFLOP (or 1000 MFLOPs)

Also, even though not part of the problem, what is the cache miss rate for this application?

Elements of a and b are used only once, and we go back to DRAM once every 4 accesses, or the miss rate = 25%

Effective Access time = $0.75*1 + 0.25*100 = 25.75$ cycles or 25.75ns

Effective performance = 38.83 MFLOPS

CSCE 5160 Parallel Processing

Consider problem 2.3. Now we are dealing with two loop matrix-vector product

```
for (i=0; i<dim; i++)  
    for (j=0; j<dim; j++) c[i] += a[i][j]*b[j]
```

Note that $b[j]$ will need to be brought into cache only once (assuming no conflicts)
Since we are dealing $4K \times 4K$ matrices b should be $4K$ words
So, we need $4K/4 = 1K$ DRAM accesses to get all b elements.

However, we need $4K$ row elements of a for each “ i ” iteration
or $4K * 1K$ DRAM accesses total

If we ignore the time to access b elements, we can have 8 operations for every one access of DRAM (to get 4 elements of a). Or we have 8 operations in 100ns
Or 80MFLOP.

But, we cannot fit both a and b in the $32K$ cache.

CSCE 5160 Parallel Processing

Consider problem 2.4. We are now dealing with matrix multiplication (3 loops)

```
for (i=0; i<dim; i++)  
    for (j=0; j<dim; j++)  
        for (k=0; k<dim; k++) c[i][j] += a[i][k]*b[k][j]
```

Both a and b elements will be reused

If we have sufficiently large cache and there are no cache conflicts,

We need to fit 4K*4K a; 4K*4K b and 4K*4K c elements into cache

it takes $4K * 1K + 4K * 1K + 4K * 1K$ memory accesses = 12M DRAM accesses
 $= 12 * 10^6 * 100 = 12 * 10^8 \text{ns}$

How many operations do we complete for the complete matrix multiplication?

For every 4 k loops we complete 8 operations; and each outer loop is executed 4K times

operations = $8 * 1K * 4K * 4K = 128 * 10^9$ operations

or 106.6 operations per ns.

Of course we only have a clock rate of 1GHz, so we cannot exceed the 1GFLOP

But in order to get all a, b and c elements in $12 * 10^8 \text{ns}$, we need 3*64Mbytes of cache

CSCE 5160 Parallel Processing

Alternate way of computing this problem

However, if we assume that we need to bring a, b and c elements repeatedly, in k loop, for each 4 iterations, we need to access 4 elements of a (because of row major) in 100ns, but we need to access 4 different rows of b (or 400ns) and 1 element of c

and we are completing 8 operations in 4 iterations

takes 600 ns or 13.33MFLOPS

This should illustrate the impact of caches on the performance in a single processor. To understand the impact of caches in shared memory multiprocessors, consider the problem 2.6.

We assume that memory is distributed (even if shared) and thus we have local memory accesses and remote memory accesses.

10ns to access cache, 100ns to access local DRAM and 400ns to access remote DRAM

80% hit on local cache, 10% to local DRAM and 10% to remote DRAM

Access time = $0.8 * 10ns + 0.1 * 100 + 0.1 * 400 = 58ns$

CSCE 5160 Parallel Processing

If we have one processor (no remote DRAM), but now the cache hit rate is 70%
So, we have $0.7 * 10\text{ns} + 0.3 * 100\text{ns} = 37\text{ns}$

(note in a multiprocessor system we have effectively smaller data per cache and thus higher hit rate).

Speed up (ignore computational costs) = $37/58 = 0.638$
No speedup?

Note that what we computed are access times. But if assume that we can complete m operations with each memory access and if we have p processors
we can complete $p * m$ operations in 58ns or $58/p * m$ ns per operation
we can only complete m operations with one processor in 37ns
or $37/m$ ns per operation

Speedup = $58/(37 * p)$