Review

Chapter 3: How to design parallel algorithms/programs

Chapter 4: Communication costs

- Start-up cost \( t_s \)
  - We pay this cost with each message sent
  - The cost does not depend on message size or # hops

- Per hop cost \( t_h \)
  - Depends on whether we are using store-forward or cut-through routing
  - We pay this cost for each intermediate node

- Bandwidth cost \( t_w \)
  - This cost depends on the size of the message

\[ \text{Total Cost} = t_s + (t_w \cdot m + t_h) \cdot l \]

\( l \) is the number of hops or intermediate nodes to travel

In most systems \( t_h \) (overhead at intermediate nodes) is very small compared to \( t_w \cdot m \), so the total cost is simplified to

\[ \text{Total Cost} = t_s + t_w \cdot m \cdot l \]

Store and Forward vs Cut-Through (or Worm-Hole Routing)

Store and Forward: The message in its entirety is stored at each intermediate node

Total cost = \( t_s + (t_w \cdot m + t_h) \cdot l \)

In most systems \( t_h \) (overhead at intermediate nodes) is very small compared to \( t_w \cdot m \), so the total cost is simplified to

Total cost = \( t_s + t_w \cdot m \cdot l \)

Cut-Through or Worm-Hole Routing: At intermediate node, we do not store the entire message -- only enough of header information to know which route to use.

Total Cost = \( t_s + (t_w \cdot m) + (t_h \cdot l) \)

Again if we ignore the cost at intermediate nodes \( t_h \),

Total Cost = \( t_s + (t_w \cdot m) \) \( \rightarrow \) faster because cost does not depend on \( l \)
CSCE 5160 Parallel Processing

Most algorithms will involve one of the following forms of communication
one-to-one
one-to-small group
one-to-all broadcast
all-to-all broadcast
all-to-one receive
all-to-all receive

We need to analyze the cost of these forms of communication for different interconnection networks
2-D mesh
hypercube
tree
ring

We will use store-and-forward and use \( t_s + t_w \cdot m \cdot l \) as cost of sending one message of size \( m \)

Some Cost Equations

\textbf{one-to-one using}

\begin{align*}
\text{Ring:} & \quad t_s + t_w \cdot m \cdot (p-1) \\
2-D \text{ Mesh} & \quad t_s + t_w \cdot m \cdot 2^*(v/p/2) \\
\text{Hypercube} & \quad t_s + t_w \cdot m \cdot \log_2(p)
\end{align*}

\textbf{one-to-all broadcast}

\begin{align*}
\text{Ring} & \quad t_s + t_w \cdot m \cdot (p/2) \\
2-D \text{ Mesh} & \quad 2^* [t_s + t_w \cdot m \cdot (v/p/2)] \\
\text{Hypercube} & \quad (t_s + t_w \cdot m) \cdot (\log_2 p)
\end{align*}

Read Chapter 4 to get a better understanding of costs and algorithms

Chapter 5: Analyzing algorithms for performance and cost
I will use different notation (simplified)

Execution time: Need to worry about memory access times
CSCE 5160 Parallel Processing

Speed up = \frac{\text{Execution time on one processor}}{\text{Execution time on n processors}}

What is maximum speedup you can get?

Maximum speedup = \# processors

Efficiency = \frac{\text{Speed up}}{\# processors} = \frac{T_1}{p \cdot T_p}

What is maximum efficiency?

\textbf{Efficiency} = \frac{\text{Speed-Up}}{\# processors}

Ideal Efficiency = 100%

\textbf{Super Linear Speed Up?} Because of increased memory size

Because of parallel search

\begin{align*}
\text{Amdahl’s law.} \\
\text{Speed up is significantly affected by the serial fraction} \\
S = \frac{T_1}{T_p} \\
\text{Let f be the serial fraction of the total program.} \\
\text{So, } S = \frac{1}{\left( \frac{1-f}{p} \right) + f} = \frac{p}{\left( 1 + f^p(p-1) \right)}
\end{align*}

Note: The key is how to calculate execution time with p processors

\textbf{Cost and cost optimal}

Cost is normally related to the number of processors but may also include communication cost

The cost of a parallel algorithm can be viewed as \( p^* T_p \)

An algorithm is cost optimal if the efficiency is 1

<table>
<thead>
<tr>
<th>f</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>0.77</td>
<td>0.59</td>
<td>0.40</td>
<td>0.24</td>
<td>0.14</td>
</tr>
<tr>
<td>0.15</td>
<td>0.69</td>
<td>0.49</td>
<td>0.31</td>
<td>0.18</td>
<td>0.10</td>
</tr>
</tbody>
</table>
CSCE 5160 Parallel Processing

Efficiency also depends on input size (for a fixed number of processors) or serial fraction (and other overheads) depend on input size.

Consider an example of adding $n$ numbers using $p$ processors.

```cpp
for (i=0; i<n; i++) sum = sum + a[i];  Sequential time = $O(n)$
```

Each processor adds $n/p$ elements and then we use a tree like accumulation. For this example, we will assume that it takes 1 time unit to add two numbers and 1 unit to communicate over a link.

In the optimal algorithm, we view the accumulation of partial sums as a binary tree of adders.

- The total execution time using $p$ processors = \((n/p) + 2\times\log(p)\)
- The “2” comes from 1 for communication and 1 for computation.

CSCE 5160 Parallel Processing

Speed up = \(n\times p / [(n+2p\times\log(p))]\) → $n$ is the input size and $p$ is number of processors.

Efficiency = speedup/p \(= n/[n+2p\times\log(p)]\)

Consider plotting the efficiency as a function of $p$ and $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th>$p=1$</th>
<th>$p=4$</th>
<th>$p=8$</th>
<th>$p=16$</th>
<th>$p=32$</th>
</tr>
</thead>
<tbody>
<tr>
<td>64</td>
<td>1.00</td>
<td>0.85</td>
<td>0.57</td>
<td>0.33</td>
<td>0.17</td>
</tr>
<tr>
<td>192</td>
<td>1.00</td>
<td>0.92</td>
<td>0.80</td>
<td>0.60</td>
<td>0.38</td>
</tr>
<tr>
<td>320</td>
<td>1.00</td>
<td>0.95</td>
<td>0.87</td>
<td>0.71</td>
<td>0.50</td>
</tr>
<tr>
<td>512</td>
<td>1.00</td>
<td>0.97</td>
<td>0.91</td>
<td>0.80</td>
<td>0.62</td>
</tr>
</tbody>
</table>

As we increase $p$, Efficiency is dropping for the same $n$.
We can maintain same efficiency by increasing input size.

The rate at which the input size must grow with $p$ (# processors) is called Iso-efficiency function.
It is a measure of the scalability of your algorithm.
Gustavson’s Law. Here instead of comparing the execution times we will try to see how much more work a multiprocessing system can perform compared a single processor in the same amount of time.

Suppose in some fixed time, uniprocessor 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 it looks more positive than Amdahl’s law \( \Rightarrow \) p is in the numerator

We can do more work with more processors
What is the maximum work you can do?

<table>
<thead>
<tr>
<th>f</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
<th>100000</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.10</td>
<td>3.70</td>
<td>7.30</td>
<td>14.50</td>
<td>28.90</td>
<td>57.70</td>
<td>900000.10</td>
</tr>
<tr>
<td>0.15</td>
<td>3.55</td>
<td>6.95</td>
<td>13.75</td>
<td>27.35</td>
<td>54.55</td>
<td>850000.15</td>
</tr>
<tr>
<td>0.5</td>
<td>2.50</td>
<td>4.50</td>
<td>8.50</td>
<td>16.50</td>
<td>32.50</td>
<td>500000.50</td>
</tr>
</tbody>
</table>

In addition to sequential fraction, parallel programs incur overhead
creation of threads/tasks
coordinating activities
context switching between threads
sending/receiving messages

So, for some applications, depending on the overhead costs there is a maximum
Number of threads/tasks beyond which no speed up can be achieved.

In general we need to analyze and see if something can be done to improve the performance
use profiling tools
Iso-Efficiency

Defining problem size itself is not trivial -- should we use the input data size? Consider matrix multiplication -- can we use n as the problem size for n*n matrices?

The problem with this is, if we double n, the execution time increases by 8-fold since the complexity is n^3

So we can use the sequential complexity as uniprocessor work

Overhead function. The reason why we do not achieve ideal efficiency is due to overhead.

For example, the communication cost,

- creating new threads,
- delays due to waiting for mutual exclusion,
- Operating System overhead
- and so on.

Overhead is the extra work in a parallel implementation that we do not need in a serial implementation.

For adding n numbers on a single processor, 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

\[ T_o = f(W, p) \]  we already saw that input size and #processors impact efficiency

Now we can define the execution time on p processors as

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

\[
\text{Speed up} \quad S = \frac{W}{T_p} = \frac{(W*p)}{[W + T_o(W, p)]}
\]

\[
\text{Efficiency} \quad E = \frac{S}{p} = 1 / \left( 1 + T_o(W, p)/W \right) = \frac{W}{W+T_o(W, p)}
\]

As W increases, efficiency should increase for fixed p -- overhead, hopefully does not increase with W rapidly
As p increases, efficiency decreases for a fixed W, since overhead may 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 an *iso-efficiency function*.

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

$$W = (E/(1-E)) * T_o(W,p) = K * T_o(W,p) \rightarrow \text{note } E/(1-E) \text{ is a constant for a given efficiency}$$

For example if we want to maintain 80% efficiency $K = (0.8)/(0.2) = 4$

If we know the overhead function, we can find $W$ for a given efficiency $E$ and $p$.

Let us examine our previous example of adding $n$ numbers

- **serial complexity** $n = W$
- **Execution time with $p$ processors** $= (n/p) + 2^*\log(p) = W/p + 2^*\log(p)$
- $T_p = (W+2^*p^*\log(p))/p$ 
- What is the overhead part?
- $T_o(W,p) = 2^* p^*\log(p)$

Our iso-efficiency function will be $K * 2^* p^*\log(p)$

Let us plot this function for different efficiency levels

<table>
<thead>
<tr>
<th>E</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>64</td>
<td>192</td>
<td>512</td>
<td>1280</td>
</tr>
<tr>
<td>0.9</td>
<td>144</td>
<td>432</td>
<td>1152</td>
<td>2880</td>
</tr>
<tr>
<td>0.95</td>
<td>304</td>
<td>912</td>
<td>2432</td>
<td>6080</td>
</tr>
<tr>
<td>0.99</td>
<td>1584</td>
<td>4752</td>
<td>12672</td>
<td>31680</td>
</tr>
</tbody>
</table>

These numbers indicate how many times more the problem size should be

Consider another example: **matrix multiplication** on a 2-D mesh.

Suppose we set the number of processors $= p$ and we are looking at matrices of $n^2$.

How can we distribute the matrices among 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 $C$. 
To Compute the elements of C do we need to communicate?

Consider the C elements to be computed by processor P0

\[ C[0,0], C[0,1], \ldots \ldots C[0,(n/p-1)], \ldots \ldots C[0,n-1] \]

\[ C[1,0], C[1,1], \ldots \ldots C[1,(n/p-1)], \ldots \ldots C[1,n-1] \]

\[ \ldots \ldots \]

\[ C[(n/p-1),0], C[(n/p-1),1], \ldots \ldots C[(n/p-1),(n/p-1)], C[(n/p),(n/p)], \ldots \ldots C[(n/p-1),n-1] \]

Need the columns of B from other processors (we need all columns of B at all processors).

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

We can use all to all broadcast on 2-D mesh of \( p^{1/2} \times p^{1/2} \)

See the analysis on pages 164-165 about all to all broadcast on mesh

\[ T_{comm} = 2t_s(p^{1/2} -1) + t_w m^*(p-1) \]

\[ m \text{ is message size (??)} \]

We need \( (p^{1/2} -1) \) messages

But, we need to send \( n/p \) columns from each processor to all other processors

the size of the message \( m = (n/p)^*n \)

Let us set \( t_w = t_s = 1 \).

Communication cost is \( 2 \times (p^{1/2} -1) + (p -1)^*(n^3/p) \)

If we ignore the first term communication cost \( = (p -1)^*(n^3/p) = O(n^3) \)
Let us find the computation time.

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

How many computational steps are needed?

To compute one \(C\) element we need \(n\) multiplications and \(n\) additions.

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

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

Can we write this as \(T_p = [W + T_o(W,p)]/p\)

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

Note for matrix multiplication, \(W = n^3\)

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

Let us find the computation time.

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

How many computational steps are needed?

To compute one \(C\) element we need \(n\) multiplications and \(n\) additions.

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

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

Can we write this as \(T_p = [W + T_o(W,p)]/p\)

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

Note for matrix multiplication, \(W = n^3\)

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

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

remember \(K = (E/1-E)\)

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

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

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

So to maintain the same efficiency, we can use the above equation to find the work that must be increased as \(p\) increases (need to increase as a cube of # processors)

<table>
<thead>
<tr>
<th>E</th>
<th>Work Processes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
</tr>
<tr>
<td>0.6</td>
<td>5.04</td>
</tr>
<tr>
<td>0.75</td>
<td>13.82</td>
</tr>
<tr>
<td>0.8</td>
<td>18.96</td>
</tr>
<tr>
<td>0.9</td>
<td>35.05</td>
</tr>
</tbody>
</table>
CSCE 5160 Parallel Processing

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

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

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

\textbf{Is there a minimum bound for iso-efficiency?}

Suppose the problem size is $W$.
We can only use $p=W$ processors (the limit on $p$)

In the Iso-Efficiency function if $W$ grows \textit{slower than $p$} (that is iso-efficiency function is sub-linear with $p$) then eventually $p$ will be greater than $W$, and we have some processors doing no work and cannot be used effectively.

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

---

CSCE 5160 Parallel Processing

\textbf{Concurrency vs Iso-efficiency}

The maximum parallelism in an algorithm \rightarrow \textit{degree of concurrency} = the maximum number of processors we can use

Note the degree of parallelism, say $C(W)$ is the maximum number of operations that can be performed in parallel – \textit{can also be viewed as the number of processors needed to achieve minimum execution – as defined by the critical path}

Consider matrix multiplication.

If there is no cost for communication or synchronization cost, what is the 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 the concurrency

---
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 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?
Assuming you have as many processors as you need, even infinite

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
\[ \frac{dT_p}{dt} = 0 \]

Consider the example of accumulating the results of some parallel computations
Or say adding n numbers using p processors

\[ T_p = \frac{n}{p} + 2 \log p \]

note the "2" is because of communication cost + computation cost

What is the minimum value as we increase p?
\[ \frac{dT_p}{dt} = 0 \]

\[ T_{\text{parallel}} = 2 \log p. \]

This makes sense.

\[ T_p = \frac{n}{p} + 2 \log p \text{ or} \]

\[ T_{\text{parallel}} = 2 \log p. \]