Definitions

Multioperation computer - a computer capable of performing more than one operation at a time.

$T_p(n)$ – Time to compute $n$ terms using $p$ processors

The speedup of performing some computation on a multioperation computer (with $p$ processors or $p$ function units) compared to a uniprocessor is given by

$$S_p = \frac{T_1}{T_p}$$

where $T_1$ is the time to perform the computation on the uniprocessor and $T_p$ is the time to perform the computation using $p$ processors. Ideally $p$ processors would be $p$ times faster than a single processor. i.e. $T_p = \frac{T_1}{p}$. This is the best possible case and in practice we would expect the speedup to be less than $p$.

Efficiency is the measurement of how close we come to achieving ideal speed up.

$$E_p = \frac{T_1}{T_p} \times \frac{S_p}{p} \leq 1$$

Order: $f(x) = O(g(x))$ if there is a constant $r > 0$ such that $\lim_{x \to \infty} \left( \frac{f(x)}{g(x)} \right) = r$.

Examples:

$$5n^2 + 99n - 999 = O(n^2) \text{ since } \lim_{n \to \infty} \left( \frac{5n^2 + 99n - 999}{n^2} \right) = 5 \geq 0$$

$$5n^2 + 99n - 999 \neq O(n^3) \text{ since } \lim_{n \to \infty} \left( \frac{5n^2 + 99n - 999}{n^3} \right) \to 0 \text{ i.e. } n^3 \text{ grows faster than } n^2.$$ 

$$5n^2 + 99n - 999 \neq O(n) \text{ since } \lim_{n \to \infty} \left( \frac{5n^2 + 99n - 999}{n} \right) \to 0 \infty$$

$$n / \log(n) \neq O(n) \text{ since } \lim_{n \to \infty} \left( \frac{n / \log(n)}{n} \right) = \lim_{n \to \infty} \left( \frac{1}{\log(n)} \right) = 0 \text{ i.e. } n \text{ grows faster than } n / \log(n).$$

2 - 8/27/2009
ASSUME: All operations take one unit of time. All instructions and data are available when needed. ie. We don't have to wait for memory or communication.

### CONVENTIONAL UNIPROCESSOR

<table>
<thead>
<tr>
<th></th>
<th>A1*B1</th>
<th>A2*B2</th>
<th>A3*B3</th>
<th>A4*B4</th>
<th>...</th>
<th>An*Bn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>/</td>
<td></td>
<td></td>
<td>/</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>*</td>
<td>/</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>*</td>
<td>/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td>*</td>
<td>/</td>
<td></td>
</tr>
</tbody>
</table>

For 4 terms $T_1 = 4$

In general for $n$ terms $T_i(n) = n$

$T_1$ is of order $n$.

### Multiprocessor MIMD (unlimited processors)

<table>
<thead>
<tr>
<th></th>
<th>A1*B1</th>
<th>A2*B2</th>
<th>A3*B3</th>
<th>A4*B4</th>
<th>...</th>
<th>An*Bn</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

For $n$ terms and at least $n$ processors

$T_n = 1$ of order 1.

Seed up of using $n$ processors verses a single processor is:

$n = \frac{T_i}{T_n} = \frac{n}{1} = n = O(n)$

$E = \frac{S}{n} = \frac{n}{n} = 1 = 100\%$

### Parallel processor SIMD (unlimited processors)

Same as above
Multifunction Computer (2 * )

<table>
<thead>
<tr>
<th>T</th>
<th>A1*B1</th>
<th>A2*B2</th>
<th>A3*B3</th>
<th>A4*B4</th>
<th>...</th>
<th>An*Bn</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>\ /</td>
<td>\ /</td>
<td>\ /</td>
<td>\ /</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>*</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For 4 terms \( T_2 = 2 \)
In general for \( n \) terms
\[
T_2 = \begin{cases} \frac{n}{2} & \text{if } n \text{ is even} \\ \frac{n+1}{2} & \text{if } n \text{ is odd} \end{cases}
\]

Better form
\( T_2 = \lceil \frac{n}{2} \rceil \)

Ceiling \( \frac{n}{2} \) ie. \( \lceil 5.5 \rceil = 6 \), \( \lceil 5.0 \rceil = 5 \).

\[
S = \frac{T_1}{T_2} = \left\lfloor \frac{n}{\frac{n}{2}} \right\rfloor = \begin{cases} \frac{n}{2} & \text{for } n \text{ even} \\ \frac{n}{n+1} & \text{for } n \text{ odd} \end{cases}
\]
\[ \rightarrow \text{2 for } n \text{ large}, \ S = O(1). \]

\[
E = \frac{S_2}{2} = \left\lfloor \frac{n}{2} \right\rfloor / n \approx 100\%
\]
Now consider $P = \prod_{i=1}^{n} A_i = A_1 A_2 \ldots A_n$

Using a uniprocessor

\[ \begin{array}{cccccc}
A_1 & A_2 & A_3 & A_4 & A_5 & A_6 \\
\star & & & & & \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array} \]

In general $T_i = n - 1$

Using an unlimited number of processors:

\[ \begin{array}{cccccc}
A_1 & A_2 & A_3 & A_4 & A_5 & A_6 \\
\star & & & & & \\
1 & 2 & 3 & 4 & 5 & 6 \\
\end{array} \]

In general $T_\infty = \log_2(n)$

Why

Time \quad n \text{- number of terms}
1 \quad 2 = 2^1
2 \quad 4 = 2^2
3 \quad 8 = 2^3
\ldots \quad \ldots
\begin{align*}
k & = n = 2^k \\
2^k & = n \\
k \log(2) & = \log(n) \\
k & = \frac{\log(n)}{\log(2)} = \log_2(n)
\end{align*}

But, when $n$ is not a power of 2 we must round up to the next highest integer. Therefore,

\[ k = \lceil \log_2(n) \rceil \]
Suppose we build a two operation (three operand) computer capable of performing $A \cdot B \cdot C$ in a single operation. (IBM has a workstation that uses $A \cdot B + C$ as its fundamental floating point operation.)

Q. How long would it take to perform $P = \prod_{i=1}^{n} A_i = A_1 A_3 \ldots A_n$?

Ans. $\log_3(n)$

Suppose we build a 32 bit adder using 4 input gate. Best time we can hope for. $\log_4(64)$
EXAMPLE

Algorithm can effect speedup. (a). Only one processor can be used at a time.
(b). Two processors can produce fastest result.

\[
A (B \ C \ D + E) = A B C D + A E
\]

\[
\begin{array}{ccc}
1 & \times & / \\
2 & \times & / \\
3 & + \\
4 & +
\end{array}
\]

\[
T_1 = T_\infty = 4 \\
T_2 = 3 \\
S = \frac{T_1}{T_2} = \frac{4}{3} = 1 \frac{1}{3} \\
E = \frac{T_1}{n} = \frac{4}{2} = \frac{2}{3} = 67\%
\]

SAME FOR SIMD
Consider the evaluation of the polynomial.

\[ F = \sum_{i=0}^{n} A_i X^i = A_0 + A_1 X + A_2 XX + A_3 XXX + \ldots \]

For a single operation computer the polynomial can be evaluated as shown below.

**METHOD 1.**

\[
F = A_0 + A_1 X + A_2 XX + A_3 XXX + A_4 XXXX + \ldots
\]

For \( n = 4 \), \( T_1 = 14 \)

In general

\[
T_1 = \text{time for } n \text{ adds} + \text{time for } (1+2+3+4+\ldots+n) \text{ multiplies}
\]

\[
= n + 1 + 2 + 3 + 4 + \ldots + n
\]

\[
= n + n(n+1)/2
\]

\[
= n(n+3)/2 = O(n^2)
\]
METHOD 2.
This can be done faster by not recomputing known terms. i.e. Using a better compiler.

\[ F = A_0 + A_1 X + A_2 X^2 + A_3 X^3 + A_4 X^4 + X^5 \]

For \( n = 4 \), \( T_2 = 11 \)
In general
\( T = n \) times for \( n \) adds + \( n \) times for multiplying \( A_i \) and \( X^n \)
\[ + (n - 1) \text{times for multiplying } X \text{ and } X^{i-1} \]
\[ = n + n + n - 1 \]
\[ = 3n - 1 \text{ of order } n. \]
METHOD 3.

A new algorithm makes the solution even faster.

\[
F = A0 + A1*x + A2*x*x + A3*x*x*x + A4*x*x*x*x
= A0 + x \times (A1 + x \times (A2 + x \times (A3 + x \times (A4 \ldots ))))
\]

For \( n = 4 \), \( T = 8 \).

In general, \( T = n \) adds + \( n \) multiplications = \( 2n \) of order \( n \).

This new algorithm produces a speed up over METHOD 1 of

\[
S = \frac{T_1}{T_3} = \frac{n(n+3)/2}{2n} = \frac{n}{4} + \frac{3}{4} \rightarrow \frac{n}{4} \text{ for large } n.
\]

The speed up of METHOD 3 over METHOD 2 is:

\[
S = \frac{T_2}{T_3} = \frac{(3n-1)}{2n} = \frac{3}{2} - \frac{1}{n} \rightarrow \frac{3}{2} \text{ for large } n.
\]
Assume we have a parallel processor that can perform an unlimited number of additions and multiplications simultaneously.

Using the previous algorithm:

\[ F = A_0 + A_1 \times A_2 + A_3 \times A_4 \times \cdots \]

For \( n = 4 \), \( T = 7 \).

In general, \( T = n \text{ adds} + n \text{ multiplications} = 2n \) of order \( n \).

No improvement over uniprocessor!

Returning to the original algorithm.

\[ F = A_0 + A_1 \times A_2 + A_3 \times A_4 \times \cdots \]

For \( n = 4 \), \( T = 5 \).

It is difficult to find a general solution for the time as a function of \( n \). I have obtained a solution, but have not proved that it is correct. However, it is easy to obtain a good least upper bounds on the time. This can be done by first doing all adds then doing all multiplies. Then \( T \leq \text{time to do all adds} \times \text{time to do all multiplies} \) or

\[ T(n) \leq \left\lfloor \log_2(n+1) \right\rfloor \times \left\lfloor \log_2(n+1) \right\rfloor = 2^{\left\lfloor \log_2(n+1) \right\rfloor} \]

For example when \( n=9 \)

\[ T(9) \leq \left\lfloor \log_2(10) \right\rfloor = 2 \times \left\lfloor 3.3219 \right\rfloor = 2 \times 4 = 8 \]

The exact answer is for \( n=9 \) is \( T = 7 \). Example: Consider a multiply add unit capable of computing \( a \times b + c \) in one unit of time.
Show how to compute: \( F(n) = \sum_{i=0}^{n} A_i X^i \) using the multiply add unit.

Consider the case when \( n = 4 \).

\[
F = A_0 + A_1 X + A_2 X^2 + A_3 X^3 + A_4 X^4
\]

\[
= A_0 + X \ast (A_1 + X \ast (A_2 + X \ast (A_3 + X \ast (A_4 \ldots ))))
\]

It appears that in general the time to compute \( F(n) = \sum_{i=0}^{n} A_i X^i \) is given by \( T(n) = n \).

Proof: Assume \( T(n) = n \) is the time to compute \( F(n) = \sum_{i=0}^{n} A_i X^i \), and show that it follows that \( T(n+1) = n+1 \).

\[
F(n + 1) = \sum_{i=0}^{n+1} A_i X^i = A_0 + X \sum_{i=0}^{n} A_{i+1} X^i
\]

Once \( \sum_{i=0}^{n} A_{i+1} X^i \), has been computed the remainder can be computed in one unit of time.

Therefore, \( T(n+1) = T(n) + 1 = n+1 \).

Since we can easily show that \( T(1) = 1 \), it follows that \( T(1+1) \) or \( T(2) = 2 \). Since \( T(2) = 2 \), \( T(3) = 2+1 \). etc. for all values of \( n \).