Use of Task Graph Model for Parallel Program Design

Detailed steps for parallel program design and implementation

1. Preparing Parallelism
   - Computational task partitioning. Aggregate tasks when needed.
   - Dependence analysis to derive a task graph

2. Mapping & Scheduling of parallelism
   - Map tasks $\rightarrow$ processors (cores)
   - Order execution

3. Parallel Programming
   - Coding
   - Debugging

4. Performance Evaluation
Example

1. Parallelism

\[ x = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 \]

2. Processor mapping and scheduling

Schedule

\[
\begin{array}{cccc}
1 & 2 & 3 & 4 \\
5 & & 6 & \\
7 & & & \\
\end{array}
\]
Task Graphs with Scheduling

A Simple Model for Parallel Computation

- A set of Tasks

- Data dependence among tasks

- Task Graph

\[ w = (a_1 + a_2 + a_3)(a_1 + a_2 + a_4) \]
Scheduling of task graph

Use a *gantt chart* to represent a schedule.

I) Assign tasks to processors.
II) Order execution within each processor. Each task
    1) Receives data from parents.
    2) Executes computation.
    3) Sends data to children.

The left schedule can be expressed as:

<table>
<thead>
<tr>
<th></th>
<th>$T_1$</th>
<th>$T_2$</th>
<th>$T_3$</th>
<th>$T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proc Assign.</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Start time</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

$$\tau = 1$$
$$c = 0$$

$$\tau = 1$$
$$c = 0.5$$
Performance Evaluation

- \( Seq \) — Sequential Time (\( \sum \) task weights)
- \( PT_p \) — Parallel Time (Length of the schedule)

\[
\text{Speedup} = \frac{Seq}{PT_p}
\]

\[
\text{Efficiency} = \frac{\text{Speedup}}{p}
\]

Ex.

\[
\begin{array}{c|c}
0 & 1 \\
\hline
T_1 & \\
\hline
T_2 & T_3 \\
\hline
T_4 & \\
\end{array}
\]

\( Seq = 4 \quad p = 2, \quad PT_p = 4 \)

\( Speedup = 1 \)

\( Efficiency = \frac{1}{2} = 50\% \)
Performance Limited by

- Parallelism availability
- Task granularity \( \left( \frac{\text{Computation Cost}}{\text{Communication Cost}} \right) \)

Revisit Amdahl’s Law: Given sequential time \( Seq \), define \( \alpha \) as fraction of computation that has to be done sequentially.

Parallel time is modeled as

\[
PT_p = \alpha \, Seq + \frac{(1 - \alpha) \, Seq}{p}
\]

\[
\text{Speedup} = \frac{Seq}{PT_p} = \frac{1}{\alpha + (1 - \alpha)/p}
\]

Example:

\( \alpha = 0, \quad \text{Speedup} = p \)

\( \alpha = 0.5, \quad \text{Speedup} = \frac{2}{1 + p^{-1}} < 2 \)
Performance bounds for task graph execution

Define

- **Critical path** is the longest path (including computation weights). The length of critical path is also called Span.
- **Degree of parallelism** be the maximum size of independent task sets in the graph.
- Seq = Sequential time (or called work load)

**Span Law**

\[ PT \geq \text{Length of the critical path}. \]

**Work Law**

\[ PT \geq \frac{Seq}{p} \]

Additionally

\[ \text{Speedup} \leq \text{Degree of parallelism} \]
Example.

No of processors $p = 2$. Task weight $\tau = 1$. Sequential time $Seq = 9$. Communication cost $c = 0$.

Maximum independent set $= \{x_3, y, z\}$.

Degree of parallelism $= 3$.

$CP = \text{critical path} = \{ x, x_2, x_3, x_4, x_5 \}$.

Length($CP$) $= 5$.

$$PT \geq \max(\text{Length}(CP), \frac{Seq}{p}) = \max(5, \frac{9}{2}) = 5.$$ 

$$\text{Speedup} \leq \frac{Seq}{5} = \frac{9}{5} = 1.8$$

Speedup $\leq 3$ Degree of parallelism
Pseudo Parallel Code

- **SPMD** - Single Program / Multiple Data
  - Data and program are distributed among processors, code is executed based on a predetermined schedule.
  - Each processor executes the same program but operates on different data based on processor identification.

- **Master/slaves**: One control process is called the master (or host). There are a number of slaves working for this master. These slaves can be coded using an SPMD style.
Pseudo Library Functions

- **mynode().**
  Return the processor ID. $p$ processors are numbered as $0, 1, 2, \ldots, p-1$.

- **numnodes().** Return the number of processors allocated.

- **send(data, dest).**
  Send data to a destination processor.

- **recv(data_buffer, source_id) or recv(data_buffer).**
  Executing `recv()` will get a message from a processor (or any processor) and store it in the space specified by `data_buffer`.

- **broadcast(data).**
  Broadcast a message to all processors.
Two examples of SPMD Code

- SPMD code:  
  ```
  Print
  "hello";
  ```

  Execute in 4 processors. The screen is:
  ```
  hello
  hello
  hello
  hello
  ```

- SPMD code:
  ```
  x=mynode();
  If x > 0, then Print "hello from " x.
  ```

  Screen:
  ```
  hello from 1
  hello from 2
  hello from 3
  ```
Example 3: Parallel Programming Steps

Sequential program:

\[ x = a_1 + a_2; \]
\[ y = x + a_3; \]
\[ z = x + a_4; \]
\[ w = y \times z; \]

Task Graph:

\[ w = y \times z \]
\[ x = a_1 + a_2 \]
\[ y = x + a_3 \]
\[ z = x + a_4 \]
\[ w = (a_1 + a_2 + a_3)(a_1 + a_2 + a_4) \]

Schedule:
**SPMD Code:**

```c
int i, x, y, z, w, a[5];
i = mynode();
if (i==0) then {
    x=a[1]+a[2];
    send(x, 1);
    y=x+a[3];
    receive(z);
    w=y*z;
}
else{
    receive(x);
    z=x+a[4];
    send(z,0);
}
```
Example 4: Parallel Programming Steps

Sequential program:

\[ x = 3 \]

For \( i = 0 \) to \( p - 1 \).

\[ y(i) = i \times x; \]

Endfor

Task Graph:

\[
\begin{array}{c}
0x \\
1x \\
2x \\
\vdots \\
(p-1)x
\end{array}
\]

Schedule:

\[
\begin{array}{cccccc}
\text{send} & x = 3 & & & & \\
0x & 1x & 2x & \cdots & (p-1)x & \text{receive}
\end{array}
\]
SPMD Code:

```c
int x,y,i;
i = mynode();
if (i==0) then { x=3;
                broadcast(x); }
else receive(x);
y = i*x;
```

Evaluation:

Assume that each task takes one unit $W$ and broadcasting takes $C$.

$$Seq = (p + 1)W, \quad PT = W + C + W.$$  

$$Speedup = \frac{(p + 1)W}{2W + C}.$$
me=mynode(); p=4;
sum = sum of local numbers at this processor;
if(?for some leaf node?) Send sum to node ?f(me)?;
for i= 1 to tree depth do{
    if(?I am still used in this depth?){
        x=receive partial sum from node ?f(me)?;
        sum = sum +x
        if (?I will not be used in next depth?)
            Send sum to node ?f(me)?;
    }
}