COMP 322: Fundamentals of Parallel Programming

Lecture 3: Computation Graphs, Ideal Parallelism

Instructors: Vivek Sarkar, Shams Iman
Department of Computer Science, Rice University
{vsarkar, shams}@rice.edu

http://comp322.rice.edu
One Possible Solution to Worksheet 2 (Reverse Engineering a Computation Graph)

Observations:
• Any node with out-degree > 1 must be an async (must have an outgoing spawn edge)
• Any node with in-degree > 1 must be an end-finish (must have an incoming join edge)
• Adding or removing transitive edges does not impact ordering constraints

```
1. A();
2. finish { // F1
3.   async D();
4.   B();
5.   {
6.     E();
7.     finish { // F2
8.       async H();
9.       F();
10.     } // F2
11.   }
12. } // F1
13. } // F1
14. C();
```
Ideal Parallelism (Recap)

- Define **ideal parallelism** of Computation G Graph as the ratio, WORK(G)/CPL(G)

- Ideal Parallelism only depends on the computation graph, and is the speedup that you can obtain with an unbounded number of processors

**Example:**

\[ \text{WORK}(G) = 26 \]
\[ \text{CPL}(G) = 11 \]

Ideal Parallelism = \( \frac{\text{WORK}(G)}{\text{CPL}(G)} = \frac{26}{11} \approx 2.36 \)
Computation Graphs are used in Project Scheduling as well

- Computation graphs are referred to as “Gantt charts” in project management
- Sample project for preparing a printed document
  
  —Source: http://www.gantt.com/creating-gantt-charts.htm
Scheduling of a Computation Graph on a fixed number of processors: Example

Node label = time(N), for all nodes N in the graph

NOTE: this schedule achieved a completion time of 11. Can we do better?
Scheduling of a Computation Graph on a fixed number of processors, P

- Assume that node N takes TIME(N) regardless of which processor it executes on, and that there is no overhead for creating parallel tasks.

- A schedule specifies the following for each node:
  - $\text{START}(N) =$ start time
  - $\text{PROC}(N) =$ index of processor in range 1...P

such that

- $\text{START}(i) + \text{TIME}(i) \leq \text{START}(j)$, for all CG edges from i to j (Precedence constraint)

- A node occupies consecutive time slots in a processor (Non-preemption constraint)

- All nodes assigned to the same processor occupy distinct time slots (Resource constraint)
Greedy Schedule

- A greedy schedule is one that never forces a processor to be idle when one or more nodes are ready for execution.
- A node is ready for execution if all its predecessors have been executed.
- Observations
  - $T_1 = \text{WORK}(G)$, for all greedy schedules
  - $T_\infty = \text{CPL}(G)$, for all greedy schedules
- where $T_p = \text{execution time of a schedule for computation graph } G \text{ on } P \text{ processors}$
Lower Bounds on Execution Time of Schedules

- Let $T_P$ = execution time of a schedule for computation graph $G$ on $P$ processors
  - Can be different for different schedules
- Lower bounds for all greedy schedules
  - Capacity bound: $T_P \geq \text{WORK}(G)/P$
  - Critical path bound: $T_P \geq \text{CPL}(G)$
- Putting them together
  - $T_P \geq \max(\text{WORK}(G)/P, \text{CPL}(G))$
Upper Bound on Execution Time of Greedy Schedules

Theorem [Graham '66]. Any greedy scheduler achieves
\[ T_P \leq \frac{\text{WORK}(G)}{P} + \text{CPL}(G) \]

Proof sketch:
Define a time step to be **complete** if \( \geq P \) nodes are ready at that time, or **incomplete** otherwise

\[ \# \text{ complete time steps} \leq \frac{\text{WORK}(G)}{P} \]

\[ \# \text{ incomplete time steps} \leq \text{CPL}(G) \]

<table>
<thead>
<tr>
<th>Start time</th>
<th>Proc 1</th>
<th>Proc 2</th>
<th>Proc 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>N</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>D</td>
<td>N</td>
<td>I</td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>N</td>
<td>J</td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>N</td>
<td>K</td>
</tr>
<tr>
<td>6</td>
<td>D</td>
<td>Q</td>
<td>L</td>
</tr>
<tr>
<td>7</td>
<td>E</td>
<td>R</td>
<td>M</td>
</tr>
<tr>
<td>8</td>
<td>F</td>
<td>R</td>
<td>O</td>
</tr>
<tr>
<td>9</td>
<td>G</td>
<td>R</td>
<td>P</td>
</tr>
<tr>
<td>10</td>
<td>H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Bounding the performance of Greedy Schedulers

Combine lower and upper bounds to get

$$\max(\text{WORK}(G)/P, \text{CPL}(G)) \leq T_P \leq \text{WORK}(G)/P + \text{CPL}(G)$$

**Corollary 1:** Any greedy scheduler achieves execution time $T_P$ that is within a factor of 2 of the optimal time (since $\max(a,b)$ and $(a+b)$ are within a factor of 2 of each other, for any $a \geq 0, b \geq 0$).

**Corollary 2:** Lower and upper bounds approach the same value whenever

- There’s lots of parallelism, $\text{WORK}(G)/\text{CPL}(G) \gg P$
- Or there’s little parallelism, $\text{WORK}(G)/\text{CPL}(G) \ll P$
Abstract Performance Metrics

- Basic Idea
  - Count operations of interest, as in big-O analysis
  - Abstraction ignores many overheads that occur on real systems

- Calls to doWork()
  - Programmer inserts calls of the form, `doWork(N)`, within a step to indicate abstraction execution of N application-specific abstract operation
    - e.g., in the Homework 1 programming assignment (Parallel Sort), we will include one call to `doWork(1)` in each call to `compareTo()`, and ignore the cost of everything else.

- Abstract metrics are enabled by calling
  - `HjSystemProperty.abstractMetrics.set(true);`

- If an HJ program is executed with this option, abstract metrics are printed at end of program execution with $\text{WORK}(G)$, $\text{CPL}(G)$, Ideal Parallelism = $\frac{\text{WORK}(G)}{\text{CPL}(G)}$
Reminders

• Send email to comp322-staff@rice.edu if you do not have access to Piazza site (otherwise use Piazza for class communications, as far as possible)

• Office hours today will be held during 2pm - 3pm in Duncan Hall 3092

• Watch videos and read handout for topic 1.5 for next lecture on Wednesday, Jan 20th

• Complete this week’s assigned quizzes on edX by 11:59pm today (all quizzes for topics 1.1, 1.2, 1.3, 1.4 including last quiz titled “Multiprocessor Scheduling”)

• HW1 will be assigned today, and is due by 12noon on Jan 29th

• See course web site for all work assignments and due dates
  • http://comp322.rice.edu