COMP 322: Fundamentals of Parallel Programming

Lecture 3: Computation Graphs, Ideal Parallelism

Vivek Sarkar, Eric Allen
Department of Computer Science, Rice University

Contact email: vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322
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

```plaintext
1. A();
2. finish { // F1
3.   async D();
4.   B();
5.   async {
6.     E();
7.     finish { // F2
8.       async H();
9.       F();
10.   } // F2
11.   G();
12. } // F2
13. } // F1
14. C();
```
Dynamic Finish-Async nesting structure and Immediately Enclosing Finish (IEF)

```c
finish { // F1
    // Part 1 of Task A0
    async {A1; async A2;}
    finish { // F2
        // Part 2 of Task A0
        async A3;
        async A4;
    }
    // Part 3 of Task A0
}
```

- IEF(A3) = IEF(A4) = F2
- IEF(A1) = IEF(A2) = F1
- Module 1 handout: Listing 4 & Figure 6 (Section 1.1)
Ideal Parallelism (Recap)

- Define **ideal parallelism** of Computation G Graph as the ratio, $\text{WORK}(G)/\text{CPL}(G)$

- Ideal Parallelism is independent of the number of processors that the program executes on, and only depends on the computation graph

**Example:**

$\text{WORK}(G) = 26$

$\text{CPL}(G) = 11$

**Ideal Parallelism** = $\text{WORK}(G)/\text{CPL}(G) = 26/11 \sim 2.36$
Scheduling of a Computation Graph on a fixed number of processors: Example

NOTE: this schedule achieved a completion time of 11, which is the same as the CPL. Can we do better?
Scheduling of a Computation Graph on a fixed number of processors, $P$

- Assume that node $N$ takes $\text{TIME}(N)$ 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:
  - $\text{START}(N) = \text{start time}$
  - $\text{PROC}(N) = \text{index of processor in range 1…}P$

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 $T_p$ = execution time of a schedule for computation graph $G$ on $P$ processors.
Lower Bounds on Execution Time of Schedules

• Let $T_P = \text{execution time of a schedule for computation graph } G \text{ on } P \text{ processors}$
  —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

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

\# incomplete time steps \( \leq \text{CPL}(G) \)
Bounding the performance of Greedy Schedulers

Combine lower and upper bounds to get

\[ \max(\frac{\text{WORK}(G)}{P}, \text{CPL}(G)) \leq T_P \leq \frac{\text{WORK}(G)}{P} + \text{CPL}(G) \]

**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, \( \frac{\text{WORK}(G)}{\text{CPL}(G)} \gg P \)
- Or there’s little parallelism, \( \frac{\text{WORK}(G)}{\text{CPL}(G)} \ll P \)
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., adds, compares, stencil ops, data structure ops
  - Multiple calls dynamically add to the execution time of current step in computation graph

- 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 = $\text{WORK(G)} / \text{CPL(G)}$
Reminders

• Send email to comp322-staff@mailman.rice.edu if you did NOT receive a welcome email from us, or if you don’t have svn access
• A Lab 1 help session will be held today, immediately after class
• Watch videos and read handout for topic 1.5 for next lecture on Wednesday, Jan 21st
• 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 on Jan 28th
• See course web site for all work assignments and due dates
  • https://wiki.rice.edu/confluence/display/PARPROG/COMP322