COMP 322: Fundamentals of Parallel Programming

Lecture 2: Computation Graphs, Ideal Parallelism

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

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

Async and Finish Statements for Task Creation and Termination (Recap)

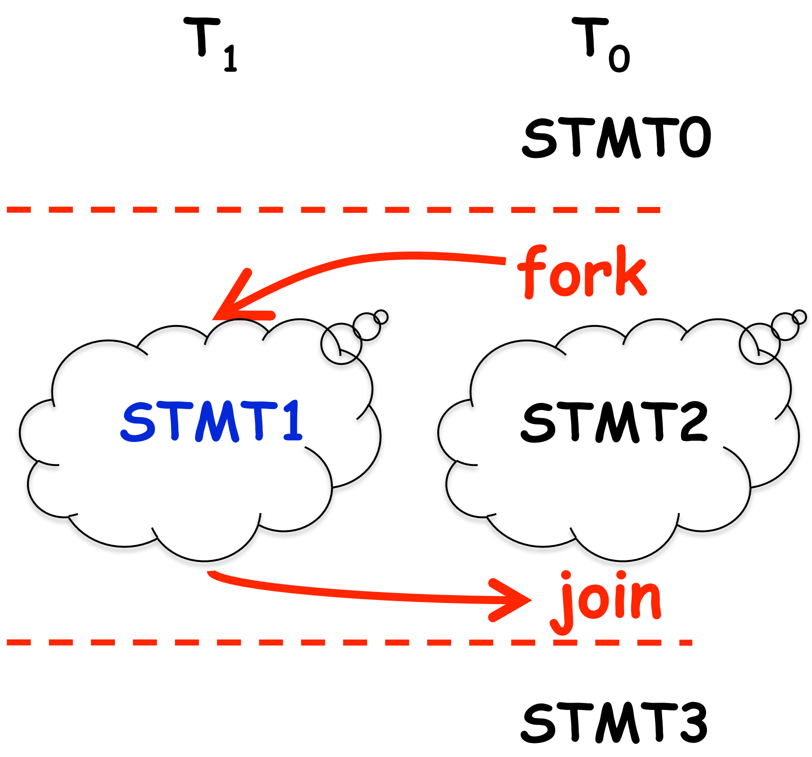
async S

- Creates a new child task that executes statement S

finish S

- Execute S, but wait until *all* asyncs in S's scope have terminated.

```
// T0(Parent task)
STMT0;
finish {     //Begin finish
  async {
    STMT1; //T1(Child task)
  }
  STMT2;     //Continue in T0
             //Wait for T1
}            //End finish
STMT3;       //Continue in T0
```



Solution to Worksheet 1 (Parallel Matrix Multiplication)

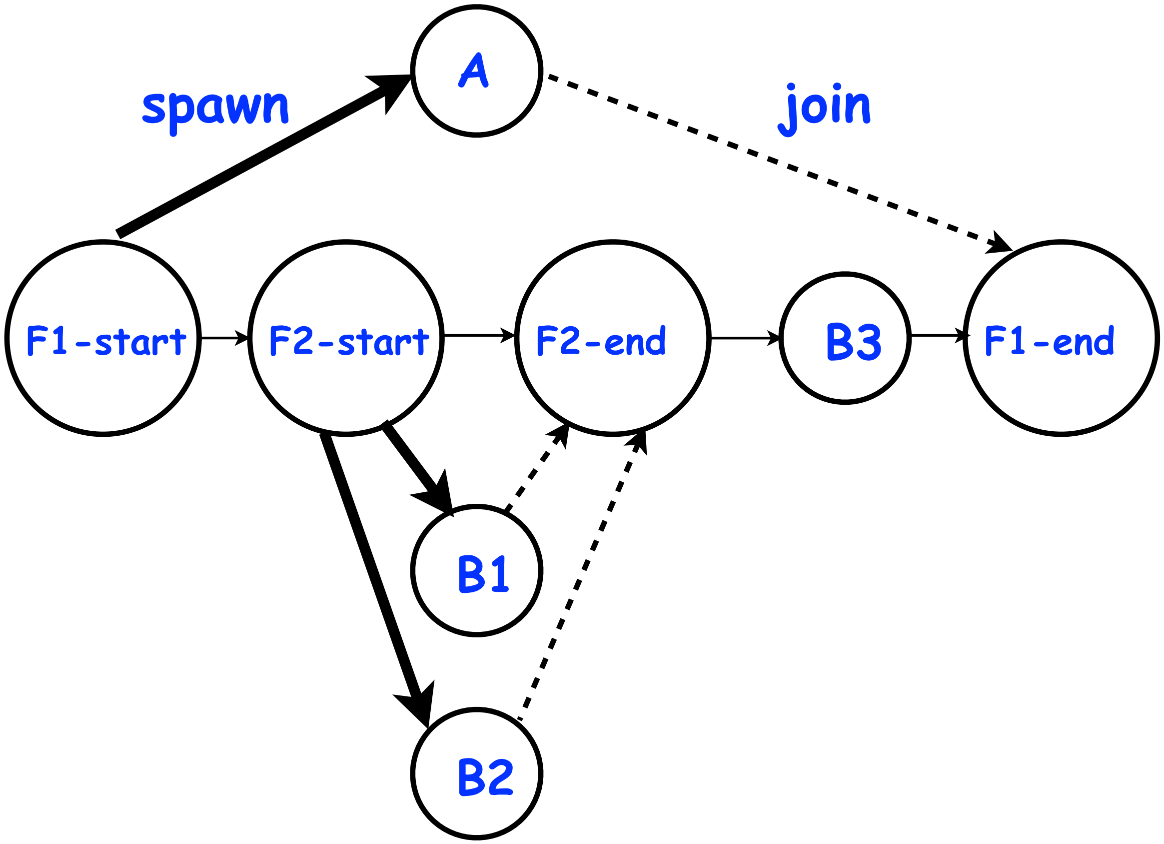
```
finish {
  for (int i = 0 ; i < N ; i++)
    for (int j = 0 ; j < N ; j++)
      async {
        for (int k = 0 ; k < N ; k++)
          C[i][j] += A[i][k] * B[k][j];
      } // async
} // finish
```

This program generates N^2 parallel async tasks, one to compute each C[i][j] element of the output array

Which statements can potentially be executed in parallel with each other?

```
finish { // F1
  async A;
  finish { // F2
    async B1;
    async B2;
  } // F2
  B3;
} // F1
```

Computation Graph



Computation Graphs

- A Computation Graph (CG) captures the dynamic execution of a parallel program, for a specific input
- CG nodes are "steps" in the program's execution
  - A step is a sequential subcomputation without any async, begin-finish and end-finish operations
- CG edges represent ordering constraints
  - "Continue" edges define sequencing of steps within a task
  - "Spawn" edges connect parent tasks to child async tasks
  - "Join" edges connect the end of each async task to its IEF's end-finish operations
- All computation graphs must be acyclic
  - It is not possible for a node to depend on itself
- Computation graphs are examples of "directed acyclic graphs" (dags)

Complexity Measures for Computation Graphs

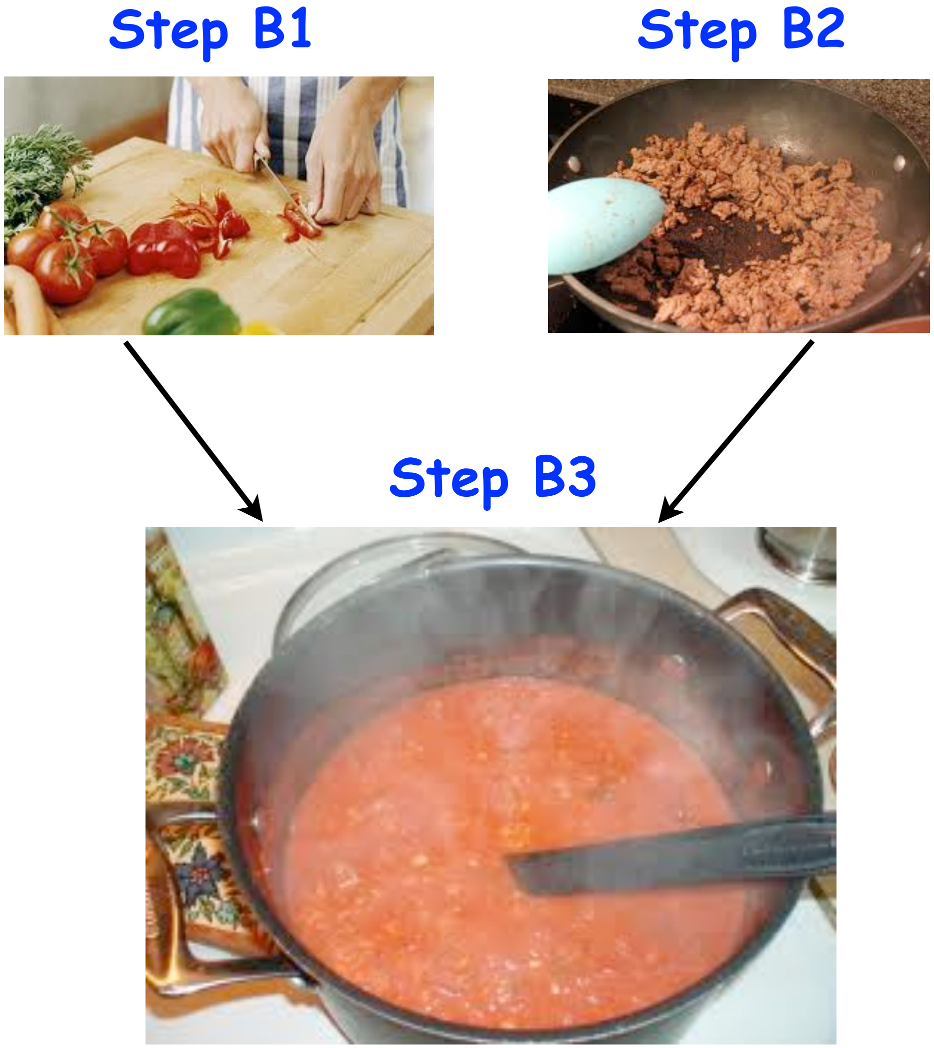
Define

- TIME(N) = execution time of node N
- WORK(G) = sum of TIME(N), for all nodes N in CG G
  - WORK(G) is the total work to be performed in G
- CPL(G) = length of a longest path in CG G, when adding up execution times of all nodes in the path
  - Such paths are called *critical paths*
  - CPL(G) is the length of these paths (critical path length)
  - CPL(G) is also the smallest possible execution time for the computation graph

What is the critical path length of this parallel computation?

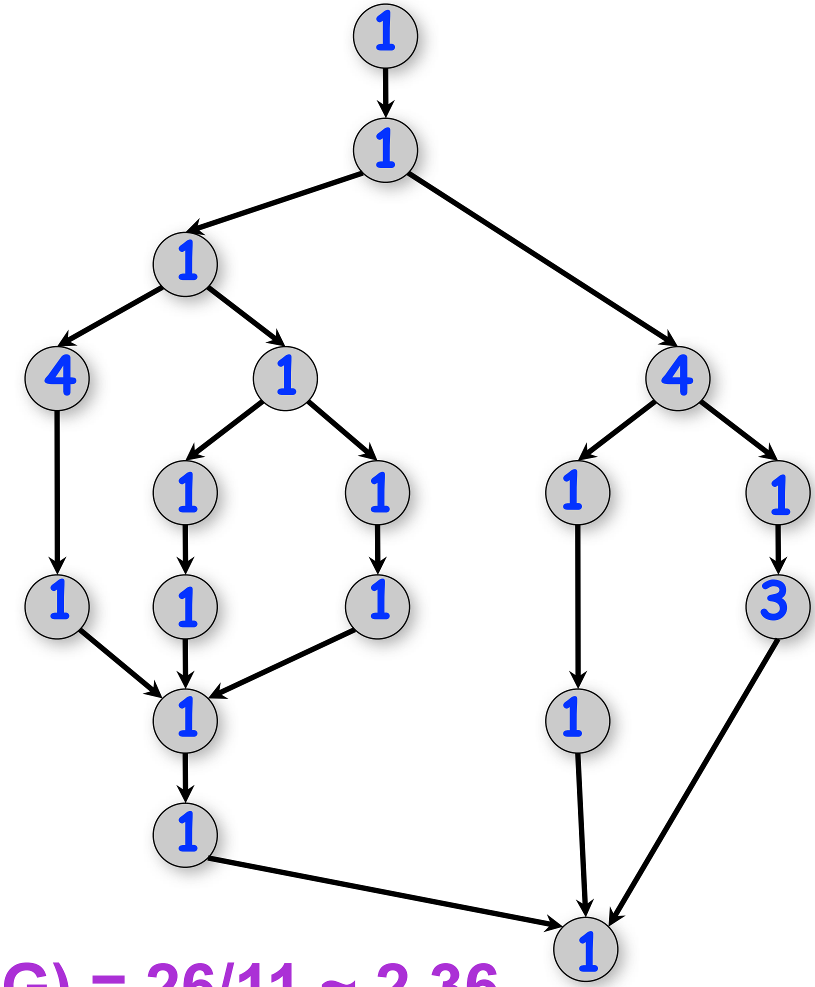
```
finish { // F1
  async A; // Boil pasta
  finish { // F2
    async B1; // Chop veggies
    async B2; // Brown meat
  } // F2
  B3; // Make pasta sauce
} // F1
```

Step A



Ideal Parallelism

- Define ideal parallelism of Computation G Graph as the ratio, WORK(G)/CPL(G)
- Ideal Parallelism is independent of the number of processors that the program executes on, and only depends on the computation graph



Example:

WORK(G) = 26

CPL(G) = 11

Ideal Parallelism = WORK(G)/CPL(G) = 26/11 ~ 2.36

String Search Problem

- **Inputs**
  - —text: a long string with N characters to search in
  - —pattern: a short string of M characters to search for
- **Output**
  - —Existence of an occurrence (boolean value)
- **Example**
  - —text: “abacadabrabracabracadababacadabrabracabracadabrabrabr”
  - —pattern: aca
  - —output: true (pattern found)
- **Applications**
  - —Word processing, virus scans, information retrieval, computational biology, web search engines, ...
- **Variations**
  - —Count of occurrences, index of any occurrence, indices of all occurrences

Brute Force Sequential Algorithm for String Search

```
public static boolean search(char[] pattern, char[] text) {
   int M = pattern.length; int N = text.length;
   boolean found = false;
   for (int i = 0; i <= N - M; i++) {
     int j; // search for pattern starting at text[i]
     for (j = 0; j < M; j++) {
       // Count each char comparison as 1 unit of work
       if (text[i+j] != pattern[j]) break;
     } // for (j = ... )
     if (j == M) found = true; // found at offset i
   }
   return found;
 }
```

What is the complexity (work) of this algorithm?

Parallel Algorithm for String Search

- Consider a parallel algorithm in which each i iteration is spawned as a separate async task
- For this above algorithm (assuming N >> M)
  - WORK ~ M*N,
  - CPL ~ M
  - Ideal Parallelism ~ N
- Big-O notation: We say that a cost function Cost(n) is "order f(n)", or simply "O(f (n))" (read "Big-O of f (n))") if
  - Cost(n) < factor * f (n), for sufficiently large n, for some constant factor
- If we consider M to be a constant in the String Search example then WORK = O(N), CPL = O(1), and Ideal Parallelism = O(N)

Course Announcements

- All Unit 1 lecture and demonstration quizzes are due by Jan 24th
  - Quizzes are still being uploaded into edX (see schedule on wiki)
- Homework 1 will be assigned on Jan 17th, and will be due on Jan 31st
- We will begin including programming exercises as in-class activities starting Jan 17th
  - Please bring laptops to class with HJlib set up for the exercises. Laptops can be shared within groups.
- Next week's schedule (Jan 20-24)
  - No lecture on Monday (MLK Jr Day)
  - No lab next week on Monday or Wednesday
  - We will have lectures on Wednesday & Friday as usual