
COMP 322: Fundamentals of Parallel Programming

Lecture 2: Computation Graphs, Ideal Parallelism

Zoran Budimlić and Mack Joyner
{zoran, mjoyner}@rice.edu

<http://comp322.rice.edu>



Async and Finish Statements for Task Creation and Termination

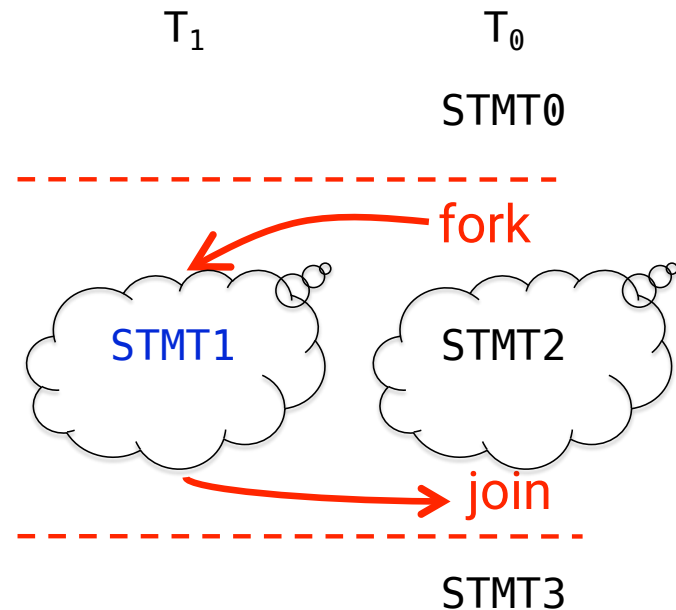
async S

- Creates a new child task that executes statement S

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

finish S

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



One possible solution to Problem #1 in Worksheet 1 (without statement reordering)

```
1. finish {
2.   async { Watch COMP 322 video for topic 1.2 by 1pm on Wednesday
3.           Watch COMP 322 video for topic 1.3 by 1pm on Wednesday
4.         }
5.   async Make your bed
6.   async { Clean out your fridge
7.           Buy food supplies and store them in fridge }
8.   finish { async Run load 1 in washer
9.            async Run load 2 in washer }
10.  async Run load 1 in dryer
11.  async Run load 2 in dryer
12.  async Call your family
13. }
14. Post on Facebook that you're done with all your tasks!
```



Another possible solution to Problem #1 in Worksheet 1 (with statement reordering)

1. `finish {`
2. `async Call your family`
3. `async Make your bed`
4. `async { Clean out your fridge`
5. `Buy food supplies and store them in fridge }`
6. `async { Run load 1 in washer`
7. `Run load 1 in dryer }`
8. `async { Run load 2 in washer`
9. `Run load 2 in dryer }`
10. `Watch COMP 322 video for topic 1.2 by 1pm on Wednesday`
11. `Watch COMP 322 video for topic 1.3 by 1pm on Wednesday`
12. `}`
13. `Post on Facebook that you're done with all your tasks!`



Is this a correct solution for Problem #2 in Worksheet 1?

```
1.finish {
2.  for (int i = 0 ; i < N ; i++)
3.    for (int j = 0 ; j < N ; j++)
4.      for (int k = 0 ; k < N ; k++)
5.        async {
6.          C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.        } // async
8.} // finish
```



Is this a correct solution for Problem #2 in Worksheet 1?

```
1.finish {
2.  for (int i = 0 ; i < N ; i++)
3.    for (int j = 0 ; j < N ; j++)
4.      for (int k = 0 ; k < N ; k++)
5.        async {
6.          C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.        } // async
8.} // finish
```

“Data race” bug! Reads and writes can occur in parallel on the same $C[i][j]$ location!



What order of reads/writes at location C[0][0] causes an incorrect result? Assume N is 2

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     for (int j = 0 ; j < N ; j++)
4.       for (int k = 0 ; k < N ; k++)
5.         async {
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.         } // async
8. } // finish
```

Run in Parallel

W1

R1

W2

R2

$C[0][0] = C[0][0] + A[0][0] * B[0][0];$

$C[0][0] = C[0][0] + A[0][1] * B[1][0];$



One Possible Solution to Problem #2 in Worksheet 1 (Parallel Matrix Multiplication)

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     for (int j = 0 ; j < N ; j++)
4.       async {
5.         for (int k = 0 ; k < N ; k++)
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.       } // async
8. } // finish
```



One Possible Solution to Problem #2 in Worksheet 1 (Parallel Matrix Multiplication)

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     for (int j = 0 ; j < N ; j++)
4.       async {
5.         for (int k = 0 ; k < N ; k++)
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.       } // async
8. } // finish
```

This program generates N^2 parallel async tasks, one to compute each $C[i][j]$ element of the output array. Additional parallelism can be exploited within the inner k loop, but that would require more changes than inserting `async` & `finish`.



Another Possible Solution to Problem #2 in Worksheet 1 (Parallel Matrix Multiplication)

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     async for (int j = 0 ; j < N ; j++)
4.       async {
5.         for (int k = 0 ; k < N ; k++)
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.       } // async
8. } // finish
```



Another Possible Solution to Problem #2 in Worksheet 1 (Parallel Matrix Multiplication)

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     async for (int j = 0 ; j < N ; j++)
4.       async {
5.         for (int k = 0 ; k < N ; k++)
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.       } // async
8. } // finish
```

This program generates $N + N^2$ parallel async tasks, but generates the same amount of parallelism among instances of statement S6 as before.

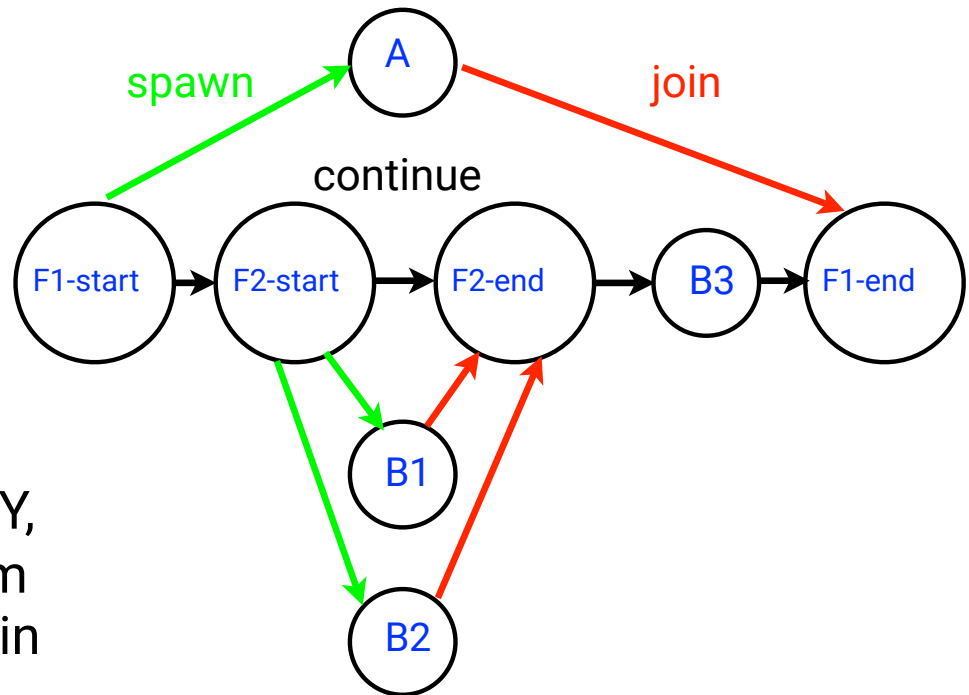


Which statements can potentially be executed in parallel with each other?

```
1. finish { // F1
2.   async A;
3.   finish { // F2
4.     async B1;
5.     async B2;
6.   } // F2
7.   B3;
8. } // F1
```

Key idea: If two statements, X and Y, have *no path of directed edges* from one to the other, then they can run in parallel with each other.

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

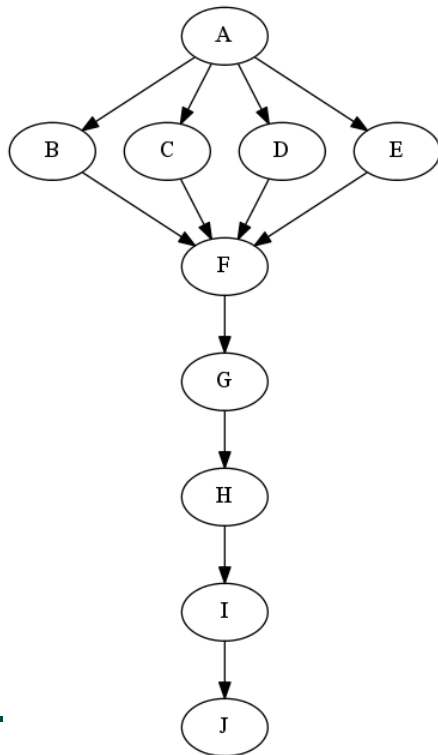
- $\text{TIME}(N)$ = execution time of node N
- $\text{WORK}(G)$ = sum of $\text{TIME}(N)$, for all nodes N in CG G
 - $\text{WORK}(G)$ is the total work to be performed in G
- $\text{CPL}(G)$ = length of a longest path in CG G , when adding up execution times of all nodes in the path
 - Such paths are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



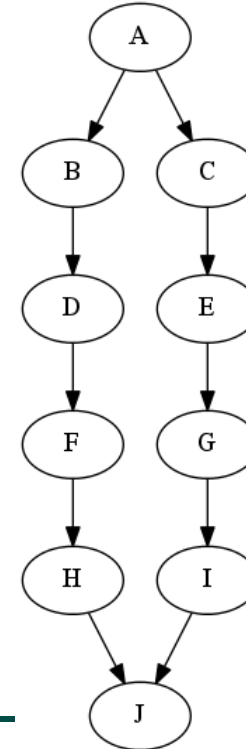
Which Computation Graph has more ideal parallelism?

Assume that all nodes have TIME = 1, so WORK = 10 for both graphs.

Computation Graph 1



Computation Graph 2



Data Races

A data race occurs on location L in a program execution with computation graph CG if there exist steps (nodes) $S1$ and $S2$ in CG such that:

1. $S1$ does not depend on $S2$ and $S2$ does not depend on $S1$, i.e., $S1$ and $S2$ can potentially execute in parallel, and
 2. Both $S1$ and $S2$ read or write L , and at least one of the accesses is a write.
- A data-race is usually considered an error. The result of a read operation in a data race is undefined. The result of a write operation is undefined if there are two or more writes to the same location.
 - Note that our definition of data race includes the case that both $S1$ and $S2$ write the same value in location L , even if that may not be considered an error.
 - Above definition includes all “potential” data races i.e., we consider it to be a data race even if $S1$ and $S2$ end up executing on the same processor.



Data Race Example: Buggy Matrix Multiply with N = 2

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     for (int j = 0 ; j < N ; j++)
4.       for (int k = 0 ; k < N ; k++)
5.         async {
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.         } // async
8.} // finish
```



Data Race Example: Buggy Matrix Multiply with $N = 2$

```
1. finish {
2.   for (int i = 0 ; i < N ; i++)
3.     for (int j = 0 ; j < N ; j++)
4.       for (int k = 0 ; k < N ; k++)
5.         async {
6.           C[i][j] = C[i][j] + A[i][k] * B[k][j];
7.         } // async
8.} // finish
```

No directed edge in computation graph between $S6(i=0,j=0,k=0)$ and $S6(i=0,j=0,k=1)$, but both read and write $C[0][0]$.



Announcements & Reminders

- IMPORTANT:
 - Bring your laptop to tomorrow's lab at 4pm on Thursday (Sewall 301)
 - Watch videos for topic 1.4 for next lecture on Friday
- HW1 will be assigned today and be due on Jan 24th
- Each quiz (to be taken online on Canvas) will be due on the Friday after the unit is covered in class. The first quiz for Unit 1 (topics 1.1 - 1.5) will be assigned on next Wednesday and is due by Jan 26.
- See course web site for syllabus, work assignments, due dates, ...
 - <http://comp322.rice.edu>
- Contact instructors with special registration form if need to convert your registration from ELEC 323 to COMP 322, or vice versa

