COMP 322: Fundamentals of Parallel Programming

Lecture 17: Midterm Review

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

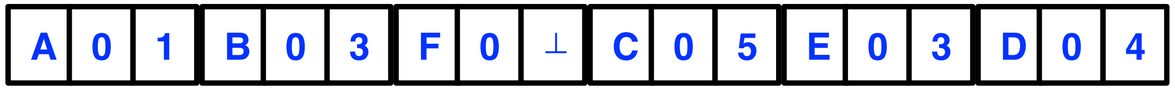
Announcements & Reminders

- Midterm exam is Thursday, Feb. 23rd from 7pm - 10pm (Canvas)
  - Take the exam wherever you want
  - Reserved: Brockman 101 (South colleges), DH McMurtry Aud. (North colleges)
  - Open notes exam. You may use course slides, module 1 handout, canvas videos, scratch paper
  - You may not copy/paste code into any kind of Java compiler/IDE (including IntelliJ)
- Quiz #3 is due today by 11:59pm
- Quiz #4 is due Monday, Feb. 27th by 11:59pm (good practice for midterm)
- No lab this week
- Homework #3 is out now (updated handout - expected performance)
  - Checkpoint 1 is due Monday, March 6th at 11:59pm
  - Checkpoint 2 is due Wednesday, March 22nd at 11:59pm
  - Final due date is Wednesday, March 29th at 11:59pm (includes written part)

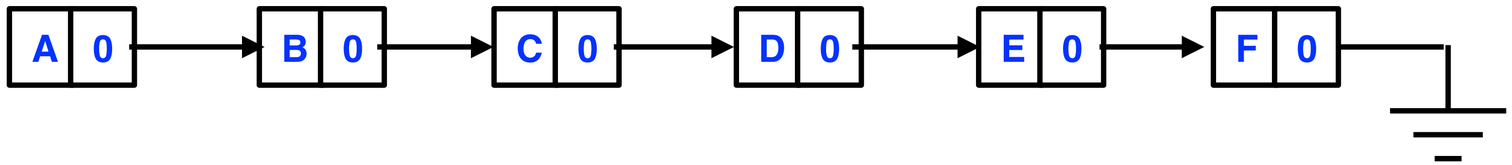
Worksheet: Pointer Skipping

You are given a linked list, and you need to compute the ***rank*** of each element of the list, i.e. the distance of that element from the end of the list.

Give a high-level idea of how would you solve this problem in parallel using pointer skipping. You can assume that the list is stored in a contiguous array, with a pointer to the next element in the list being a simple index of that element. For example, the following array:



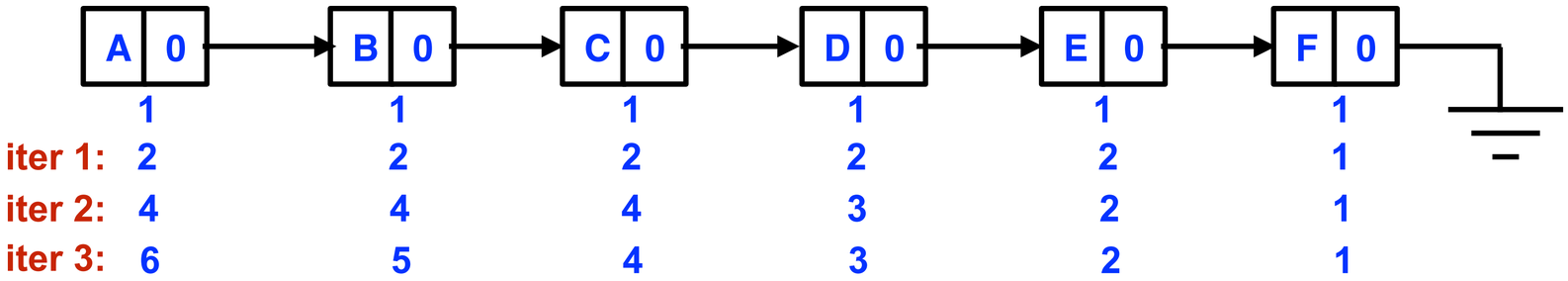
Represents the following list:



What is the total WORK that your solution would perform (integer addition counts as WORK(1), everything else is ignored)?

Worksheet: Pointer Skipping

Assume d[i] = 1 for all nodes



Algorithm:

1. Repeat log N times:
    1. Finish
        1. Create an async task for each list node
        2. In each task i:
            1. set d[i] += d[succ[i]]
            2. set succ[i] = succ[succ[i]]

What is the big-O for total WORK and CPL that your solution would perform (integer addition counts as WORK(1), everything else is ignored)?

WORK = O(N log N), CPL = O(log N)

Worksheet: Concurrency vs. Parallelism

Honor Code Policy for Worksheets: You are free to discuss all aspects of in-class worksheets with your other classmates, the teaching assistants and the professor during the class. You can work in a group and write down the solution that you obtained as a group. If you work on the worksheet outside of class (e.g., due to an absence), then it must be entirely your individual effort, without discussion with any other students. If you use any material from external sources, you must provide proper attribution. You should submit the worksheet in Canvas.

Parallelism vs. Concurrency
Next to each one of the following activity scenarios, write whether that scenario exhibits *Parallelism*, *Concurrency, Both* or *Neither.*

Scenario 1:
Task: Throwing a ball in the air and catching it.
Activity: A circus performer is juggling 5 balls at the same time.

Concurrency

Scenario 2:
Task: Throwing a knife at a target.
Activity: A circus performer is throwing knives at a target.

Neither

Scenario 3:
Task: Riding a monocycle.
Activity: Two circus performers are riding monocycles around the ring.

Parallelism

Scenario 4:
Task: Throwing the ball in the air and catching it.
Activity: Two circus performers are juggling 5 balls between each other.

Both

CPU Frequency and Power
Assume you have a CPU with 16 cores that consumes 160 Watts of power and runs at 2GHz. At what frequency would a single-core CPU with the same processing power have to run, and how much power would it have to consume?

32GHz, 40.96 KW (~32 average households)

Worksheet: Functional Programming and Recursion

Functional Programming in Java

Write a function *evens*, that uses purely functional approach (no mutation, probably want to use recursion), that takes a *GList<Integer>* and outputs a list that contains only and all the even numbers from the original list, in the same order. In which class(es) or interface(s) will you put this function?

"Functional" solution:

```
public interface GList<T> {
  static GList<Integer> evens(GList<Integer> input){
    if (input.isEmpty()) return input;
    return input.head() % 2 == 0
        ? evens(input.tail()).prepend(input.head())
        : evens(input.tail());
  }
}
```

Worksheet: Higher-order functions

Write a function *sumOddStringLengths*, that uses higher-order functions we studied today. This function should take as an argument GList<String>, find all the strings in that list that have an odd length, and compute the sum of the lengths of those odd-length strings.

```
public Integer sumOddStringLengths(GList<String> input) {
  return input.map(String::length).filter(x -> x % 2 == 1).foldRight(0,(a, e) -> a + e);
}
```


Worksheet: Lazy Computation

You are given the following supplier for a lazy list of integers:

```
public static LazyList<Integer> from(int i, int step) {
  System.out.println("Hi");
  return cons(i,()->from(i+step, step));
}
```

In the following sequence of statements, write how many times will “Hi” be printed out as a side effect of executing that statement:

```
var nats = from(0, 1);      // 1
var evens = from(0, 2);     // 1
var alsoEvens = nats.filter(x -> x % 2 == 0);      // 0
var yetAnotherEvens = nats.map(x -> x * 2);        // 0
var thirdEven = evens.tail().tail().head();        // 2
var thirdEvenAgain = evens.tail().tail().head();   // 0
var fiveEvens = evens.take(5);                     // 0
var sumFiveEvens = fiveEvens.foldRight(0 , (x, y) -> x + y);      // 3
```

Why is fold evaluating an “extra” element?

```
public LazyList<T> take(int n) {
  if (n < 1) {
    return empty();
  } else {
    return cons(headVal, ()-> tail().take(n - 1));
  }
}
```

```
public LazyList<T> take(int n) {
  if (n < 1) {
    return empty();
  } else if (n == 1) {
    return cons(headVal, ()-> empty());
  } else {
    return cons(headVal, ()-> tail().take(n - 1));
  }
}
```


Worksheet: Java Streams

What will the following piece of code print?

```
List<String> list = Arrays.asList("Rice", "Owls", "are", "the", "best");
String answer =
    list.stream()
        .skip(1)
        .map(e -> {
          System.out.println("Map was called on " + e);
          return e.substring(0, 3);
        })
        .filter(e -> {
          System.out.println("Filter was called on " + e);
          return e.charAt(2) == 'e';
        })
        .findFirst()
        .get();

System.out.println(answer);
```

Map was called on Owls
Filter was called on Owl
Map was called on are
Filter was called on are
are

Will your answer change (and how), if you replace list.stream() with list.stream.parallel()?

Could be anything that has
both a Map and a Filter on “are”.
For example:

Map was called on are
Filter was called on are
Map was called on best
Filter was called on bes
Map was called on Owls
Filter was called on Owl
are

Worksheet: Map/Reduce

You are given the following parallel Map/Reduce "framework" for processing a collection of Strings using Java Streams:

```
List<String> list = Arrays.asList("Rice", "Owls", "are", "the", "best");
var value =
    list.stream().parallel()
        .filter(_____A_____)
        .map(_____B_____)
        .reduce(_____C_____);
```

Using this framework, solve the following problems by filling in the blanks A, B and C (note that C can be 1, 2 or 3 arguments, depending on which variant of reduce you choose):

1. Find all the strings that contain the letter "s", convert them all to upper case, and concatenate them
2. Find the total length of all the strings that start with a lowercase letter

Worksheet: Map/Reduce

1. Find all the strings that contain the letter “s”, convert them all to upper case, and concatenate them

```
List<String> list = Arrays.asList("Rice", "Owls", "are", "the", "best");
var value =
    list.stream().parallel()
        .filter(e -> e.contains("s"))
        .map(e -> e.toUpperCase())
        .reduce("", String::concat, String::concat);
```

“OWLSBEST”

Worksheet: Map/Reduce

2. Find the total length of all the strings that start with a lowercase letter

```
List<String> list = Arrays.asList("Rice", "Owls", "are", "the", "best");
var value =
    list.stream().parallel()
        .filter(e -> e.charAt(0) >= 'a' && e.charAt(0) <= 'z')
        .map(String::length)
        .reduce(Integer::sum);
```

Optional[10]

Worksheet: Futures

Here is a recursive, sequential divide-and-conquer function for finding a maximum value in an array:

```
static int findMax(int[] X, int lo, int hi) {
    if ( lo > hi ) return 0;
    else if ( lo == hi ) return X[lo];
    else {
        int mid = (lo+hi)/2;
        var max1 =
                findMax(X, lo, mid);
        var max2 =
                findMax(X, mid+1, hi);

        return (max1 > max2)? max1 : max2;
    }
} // findMax
```

Indicate in the code the changes you need to make to this function in order to create a parallel, recursive divide-and-conquer function for finding a maximum value in an array.

Worksheet: Futures

Here is a recursive, sequential divide-and-conquer function for finding a maximum value in an array:

```
static int findMax(int[] X, int lo, int hi) throws SuspendableException {
    if ( lo > hi ) return 0;
    else if ( lo == hi ) return X[lo];
    else {
        int mid = (lo+hi)/2;
        var max1 = future(() ->
                findMax(X, lo, mid));
        var max2 = future(() ->
                findMax(X, mid+1, hi));
        // Parent now waits for the future values
        return (max1.get() > max2.get())? max1.get() : max2.get();
    }
} // findMax
```

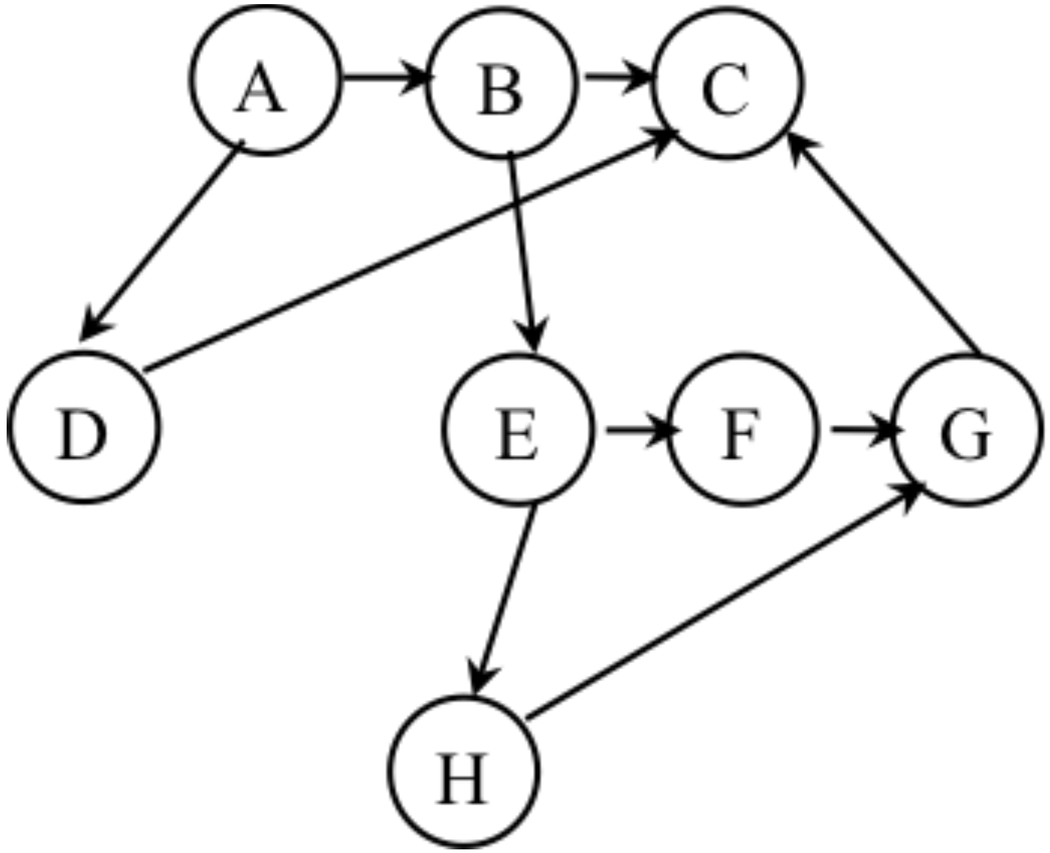
Indicate in the code the changes you need to make to this function in order to create a parallel, recursive divide-and-conquer function for finding a maximum value in an array.

Computation Graphs

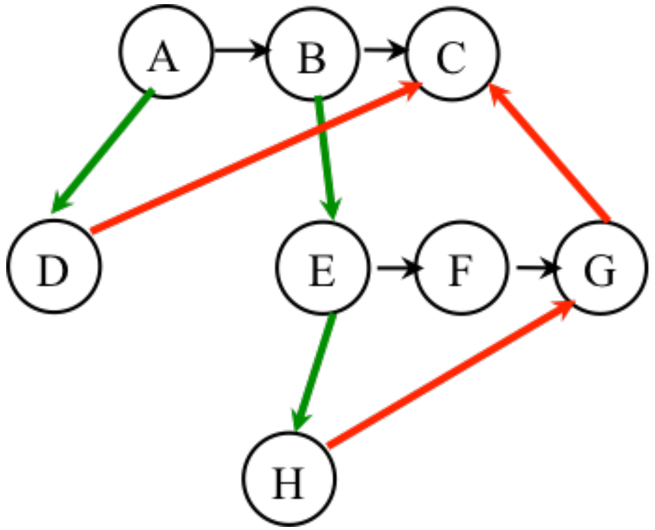
- A Computation Graph (CG) captures the dynamic execution of a parallel program, for a specific input
- CG nodes are "steps" in the program's execution
  - A step is a sequential subcomputation without any spawned, begin-finish or end-finish operations
- CG edges represent ordering constraints
  - "Continue" edges define sequencing of steps within a task
  - "Spawn" edges connect parent tasks to child spawned tasks
  - "Join" edges connect the end of each spawned task to its IEF's end-must 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)

Worksheet: Reverse Engineering a Parallel Program from a Computation Graph (CG)

Write a parallel program that generates exactly the same ordering constraints as the computation graph shown. The program should be written in pseudocode using must finish and spawn annotations. The CG nodes should be clearly identified as statements in the program e.g., as method calls A(), B(), etc. Since the CG edges are not labeled as spawn, continue, or join, you can make whatever assumptions you choose about the edges when writing your program. The only requirement is that the ordering constraints in your program coincide with those in the graph. Submit solution in Canvas.



One Possible Solution to Worksheet (Reverse Engineering a Computation Graph)



Observations:

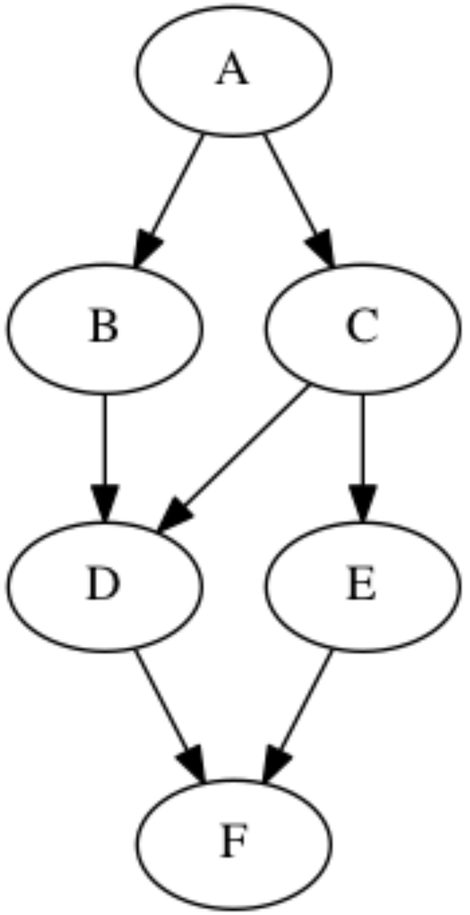
- Any node with out-degree > 1 must be an async (must have an outgoing spawn edge)
- Any node with in-degree > 1 must be an end-finish (must have an incoming join edge
- Adding or removing transitive edges does not impact ordering constraints

```
1.A();
2.must finish { // F1
3.   spawn D();
4.   B();
5.   E();
6.   must finish { // F2
7.      spawn H();
8.      F();
9.   } // F2
10. G();
11.} // F1
12.C();
```

Worksheet: Computation Graphs for Async-Finish and Future Constructs

1) Can you write pseudocode with async & finish constructs that generates a Computation Graph with the same ordering constraints as the graph on the right? If so, provide a sketch of the program. If not, why not?

2) Can you write pseudocode with future & get constructs that generates a Computation Graph with the same ordering constraints as the graph on the right? If so, provide a sketch of the program. If not, why not?



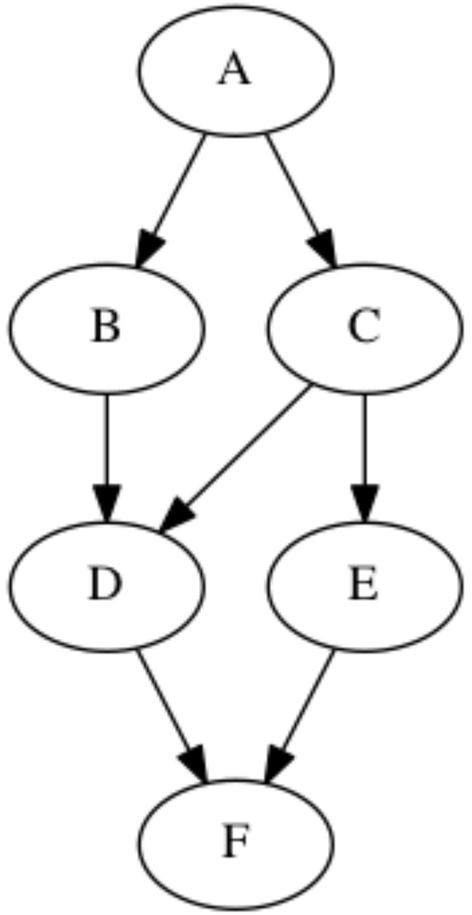
Worksheet solution

1) Can you write pseudocode with async-finish constructs that generates a Computation Graph with the same ordering constraints as the graph on the right? If so, provide a sketch of the program.

No. Finish cannot be used to ensure that D only waits for B and C, while E waits only for C.

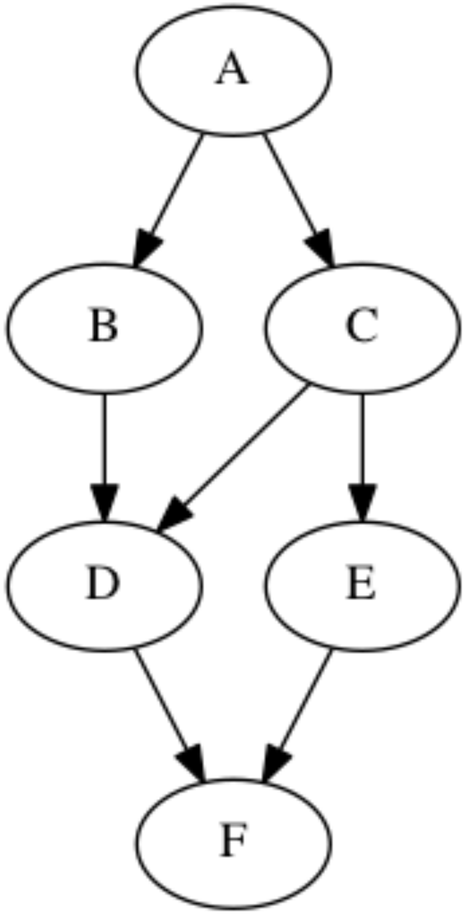
2) Can you write pseudocode with future async-get constructs that generates a Computation Graph with the same ordering constraints as the graph on the right? If so, provide a sketch of the program.

Yes, see program sketch with dummy return values.



Worksheet solution (contd.)

```
1.  var A = future(() -> {
2.      return "A"; });
3.  var B = future(() -> {
4.      A.get(); return "B"; });
5.  var C = future(() -> {
6.      A.get(); return "C"; });
7.  var D = future(() -> {
8.      // Order of B.get() & C.get() doesn't matter
9.      B.get(); C.get(); return "D"; });
10. var E = future(() -> {
11.     C.get(); return "E"; });
12. var F = future(() -> {
13.     D.get(); E.get(); return "F"; });
14. F.get();
```



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 the data race is benign.
- 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.

Exercise

What is the smallest (best-case) Big-O value of WORK and CPL in terms of N?

- P1 = number of primes in the range 2..N
- P2 = number of primes in the range 2..square-root(N)

```
1. // Initially, assume all integers are prime.
2. final boolean[] isPrime = new boolean[N + 1];
3. for (int i = 2; i <= N; i++)
4.    isPrime[i] = true;
5. // mark non-primes <= N using Sieve of Eratosthenes
6. finish(() -> {
7.  for (int i = 2; i*i <= N; i++) {
8.    // If i is prime, then mark multiples of i as nonprime
9.    // It suffices to consider multiples i, i+1, ..., N/i
10.   if (isPrime[i])
11.      async(() -> {
12.       for (int j=i; i*j <= N; j++) {
13.         isPrime[i*j] = false;
14.         doWork(1);
15.       } // for
16.      }); // async
17. } // for
18.}); // finish
19.// Integer i is prime if and only if isPrime[i]=true
20.
```


Extending HJ Futures for Macro-Dataflow: Data-Driven Tasks (DDTs)

asyncAwait(ddfA, ddfB, …, () -> Stmt);

- Create a new data-driven-task to start executing Stmt after all of ddfA, ddfB, … become available (i.e., after task becomes "enabled")
- Alternatively, you can pass a list to asyncAwait
- Await clause can be used to implement "nodes" and "edges" in a computation graph

ddfA.get()

- Return value (of type T1) stored in ddfA
- Throws an exception if put() has not been performed

ddfA.safeGet()

- Doesn't throw an exception
  - Should be performed by async's that contain ddfA in their await clause, or if there's some other synchronization to guarantee that the put() was performed

Worksheet: Data Driven Tasks

For the example below, will reordering the five async statements change the meaning of the program (assuming that the semantics of the reader/writer methods depends only on their parameters) ? If so, show two orderings that exhibit different behaviors. If not, explain why not.

```
1. var left = newDataDrivenFuture();
2. var right = newDataDrivenFuture();
3. finish {
4.   asyncAwait(left) leftReader(left); // Task3
5.   asyncAwait(right) rightReader(right); // Task5
6.   asyncAwait(left,right)
7.         bothReader(left,right); // Task4
8.   async left.put(leftWriter()); // Task1
9.   async right.put(rightWriter());// Task2
10. }
```

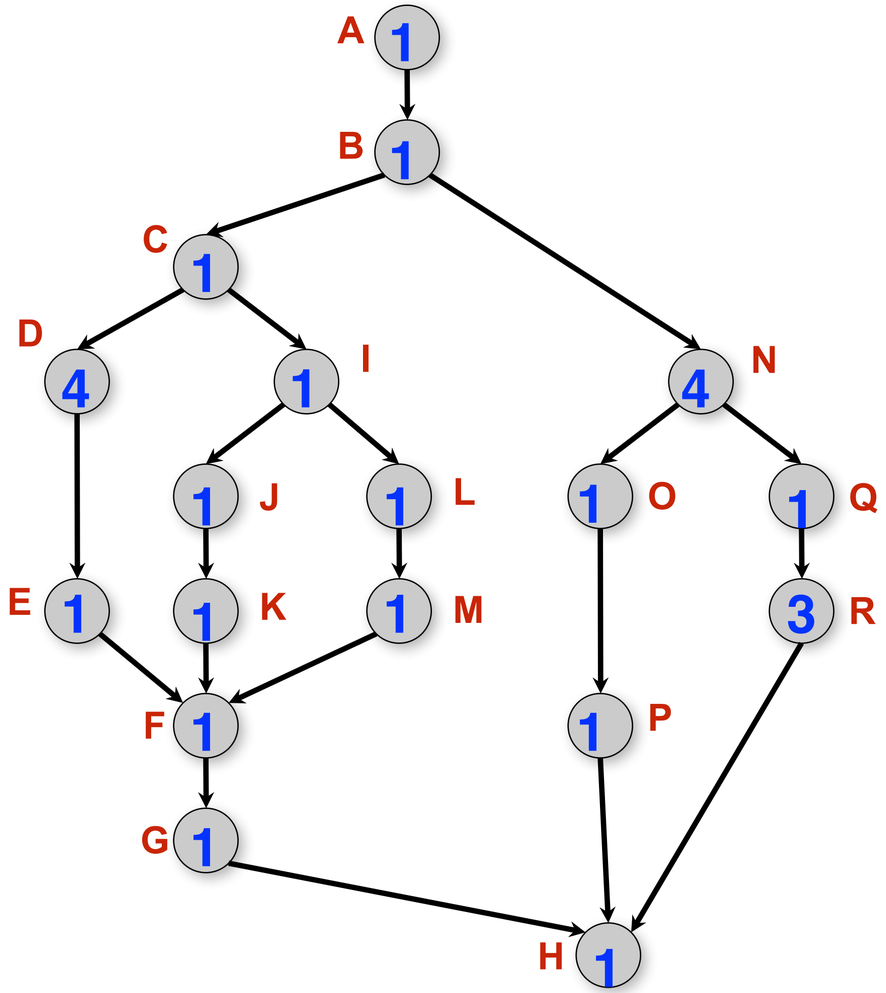

Worksheet solution

For the example below, will reordering the five async statements change the meaning of the program (assuming that the semantics of the reader/writer methods depends only on their parameters) ? If so, show two orderings that exhibit different behaviors. If not, explain why not.

No, reordering the asyncs doesn't change the meaning of the program. Regardless of the order, Task 3 will always wait on Task 1. Task 5 will always wait on Task 2. Task 4 will always wait on both Task 1 and 2.

```
1.var left = newDataDrivenFuture();
2. var right = newDataDrivenFuture();
3. finish {
4.   asyncAwait(left) leftReader(left); // Task3
5.   asyncAwait(right) rightReader(right); // Task5
6.   asyncAwait(left,right)
7.         bothReader(left,right); // Task4
8.   async left.put(leftWriter()); // Task1
9.   async right.put(rightWriter());// Task2
10. }
```


Worksheet: Scheduling



- As before, WORK = 26 and CPL = 11 for this graph
- T_2 = 15, for the 2-processor schedule on the right
- We can also see that max(CPL,WORK/2) <= T_2 < CPL + WORK/2
- There are 4 idle slots in this schedule — can we do better than T_2 = 15 ?

Start time	Proc 1	Proc 2
0	A	
1	B	
2	C	N
3	D	N
4	D	N
5	D	N
6	D	O
7	I	Q
8	J	R
9	L	R
10	K	R
11	M	E
12	F	P
13	G	
14	H	
15		

Parallel Speedup

- Define Speedup(P) = T_1 / T_P
  - —Factor by which P processors speeds up execution time relative to 1 processor, for fixed input size
  - —For ideal executions without overhead, 1 <= Speedup(P) <= P
    - —You see this with abstract metrics, but bounds may not hold when measuring real execution times with real overheads
  - —Linear speedup
    - When Speedup(P) = k*P, for some constant k, 0 < k < 1

- Ideal Parallelism = WORK / CPL = T_1 / T_∞

  = Parallel Speedup on an unbounded (infinite) number of processors

Worksheet: Speedup

Array Sum Speedup

- Assume T(S,P) = WORK(G,S)/P + CPL(G,S) = (S-1)/P + log2(S) for the parallel array sum computation shown in slide 4 (using the upper bound)

- Assume S = 1024 ==> log2(S) = 10

- Compute for 10, 100, 1000 processors (round to 1 decimal place)
  T(S,P) = (S-1)/P + log2(S) = 1023/P + 10
  Speedup(10) = T(1)/T(10) =
  Speedup(100) = T(1)/T(100) =
  Speedup(1000) = T(1)/T(1000) =

- Why does the speedup not increase linearly in proportion to the number of processors?

Worksheet solution

- Estimate T(S,P) ~ WORK(G,S)/P + CPL(G,S) = (S-1)/P + log2(S) for the parallel array sum computation shown in slide 4.
- Assume S = 1024 ==> log2(S) = 10
- Compute for 10, 100, 1000 processors
  - T(P) = 1023/P + 10, when P > 1
  - Speedup(10) = T(1)/T(10) = 1023/112.3 ~ 9.1
  - Speedup(100) = T(1)/T(100) = 1023/20.2 ~ 50.6
  - Speedup(1000) = T(1)/T(1000) = 1023/11.0 ~ 93.7
- Why does the speedup not increase linearly in proportion to the number of processors?
  - Because of the critical path length, log2(S), is a bottleneck

Extending Finish Construct with "Finish Accumulators" (Pseudocode)

- Creation

  accumulator ac = newFinishAccumulator(*operator, type*);

- *Operator must be associative and commutative (creating task "owns" accumulator)*
- Registration

  finish (ac1, ac2, ...) { ... }

- *Accumulators ac1, ac2, ... are registered with the finish scope*
- Accumulation

  ac.put(data);

- *Can be performed in parallel by any statement in finish scope that registers ac. Note that a put contributes to the accumulator, but does not overwrite it.*
- Retrieval

  ac.get();

- *Returns initial value if called before end-finish, or final value after end-finish*
- get() *is nonblocking because no synchronization is needed (finish provides the necessary synchronization)*

Worksheet solution: Associativity and Commutativity

Recap:
A binary function f is *associative* if f(f(x,y),z) = f(x,f(y,z)).
A binary function f is *commutative* if f(x,y) = f(y,x).

Worksheet problems:
1) Claim: a Finish Accumulator (FA) can only be used with operators that are *associative and commutative.*
Why? What can go wrong with accumulators if the operator is non-associative or non-commutative?
You may get different answers in different executions if the operator is non-associative or non-commutative
e.g., an accumulator can be implemented using one "partial accumulator" per processor core.

2) For each of the following functions, indicate if it is associative and/or commutative.

a) f(x,y) = x+y, for integers x, y, is associative and commutative
b) g(x,y) = (x+y)/2, for integers x, y, is commutative but not associative
c) h(s1,s2) = concat(s1, s2) for strings s1, s2, e.g., h("ab","cd") = "abcd", is associative but not commutative

Worksheet solution: Classifying different versions of parallel search algorithms

Enter “YES” or “NO”, as appropriate, in each box below

Example: String Search variation	Data Race Free?	Functionally Deterministic?	Structurally Deterministic?
V1: Count of all occurrences	YES	YES	YES
V2: Existence of an occurrence	NO	YES	YES
V3: Index of any occurrence	NO	NO	YES
V4: Optimized existence of an occurrence: do not create more async tasks after occurrence is found	NO	YES	NO
V5: Optimized index of any occurrence: do not create more async tasks after occurrence is found	NO	NO	NO