COMP 322: Fundamentals of Parallel Programming

Lecture 5: Futures ---Tasks with Return Values

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

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

Clarification: IllegalStateException

Question: When I try to run my code I get the following exception, What does it mean?

```
java.lang.IllegalStateException: All 40 HJ worker threads are blocked!
[Queued task count=1121]

at
edu.rice.hj.runtime.forkjoin.ForkJoinThreadPool.threadBlockedNotification(
ForkJoinThreadPool.java:99)
```

Answer: See Piazza post

You can configure HJ to use more worker threads by setting the following property before calling initializeHabanero():

```
System.setProperty(HjSystemProperty.maxThreads.propertyKey(),
"100");
```

Extending Async Tasks with Return Values

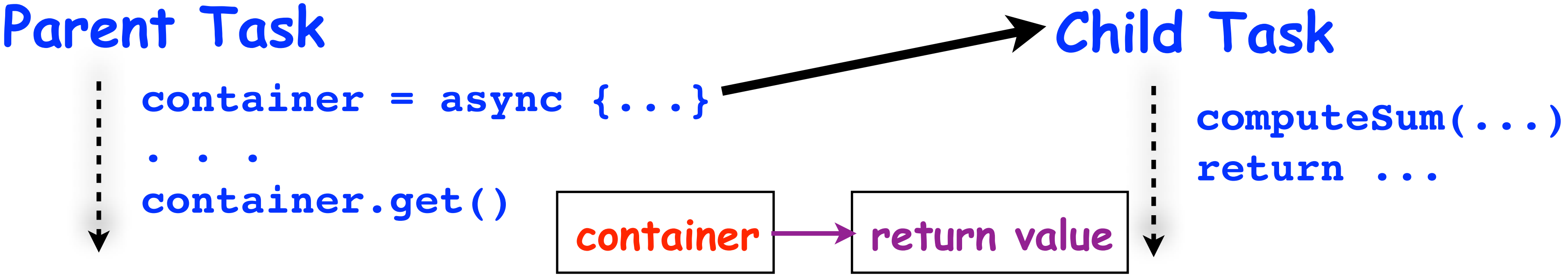
- Example Scenario in PseudoCode

```
// Parent task creates child async task
final future container =
                  async { return computeSum(X, low, mid); };
. . .
// Later, parent examines the return value
int sum = container.get();
```

- Two issues to be addressed:

  1) Distinction between container and value in container (box)

  2) Synchronization to avoid race condition in container accesses



Parent Task

```
container = async {...}
. . .
container.get()
```

Child Task

```
computeSum(...)
return ...
```

container → return value

HJ Futures: Tasks with Return Values

`async { Stmt-Block }`

- Creates a new child task that executes Stmt-Block, which must terminate with a return statement and return value
- Async expression returns a reference to a container of type future

`Expr.get()`

- Evaluates Expr, and blocks if Expr's value is unavailable
- Unlike finish which waits for *all* tasks in the finish scope, a get() operation only waits for the specified async expression

Example: Two-way Parallel Array Sum using Future Tasks (PseudoCode)

```
// Parent Task T1 (main program)
// Compute sum1 (lower half) and sum2 (upper half) in parallel
final future sum1 = async { // Future Task T2
  int sum = 0;
  for(int i=0 ; i < X.length/2 ; i++) sum += X[i];
  return sum;
};
final future sum2 = async { // Future Task T3
  int sum = 0;
  for(int i=X.length/2 ; i < X.length ; i++) sum += X[i];
  return sum;
};
//Task T1 waits for Tasks T2 and T3 to complete
int total = sum1.get() + sum2.get();
```

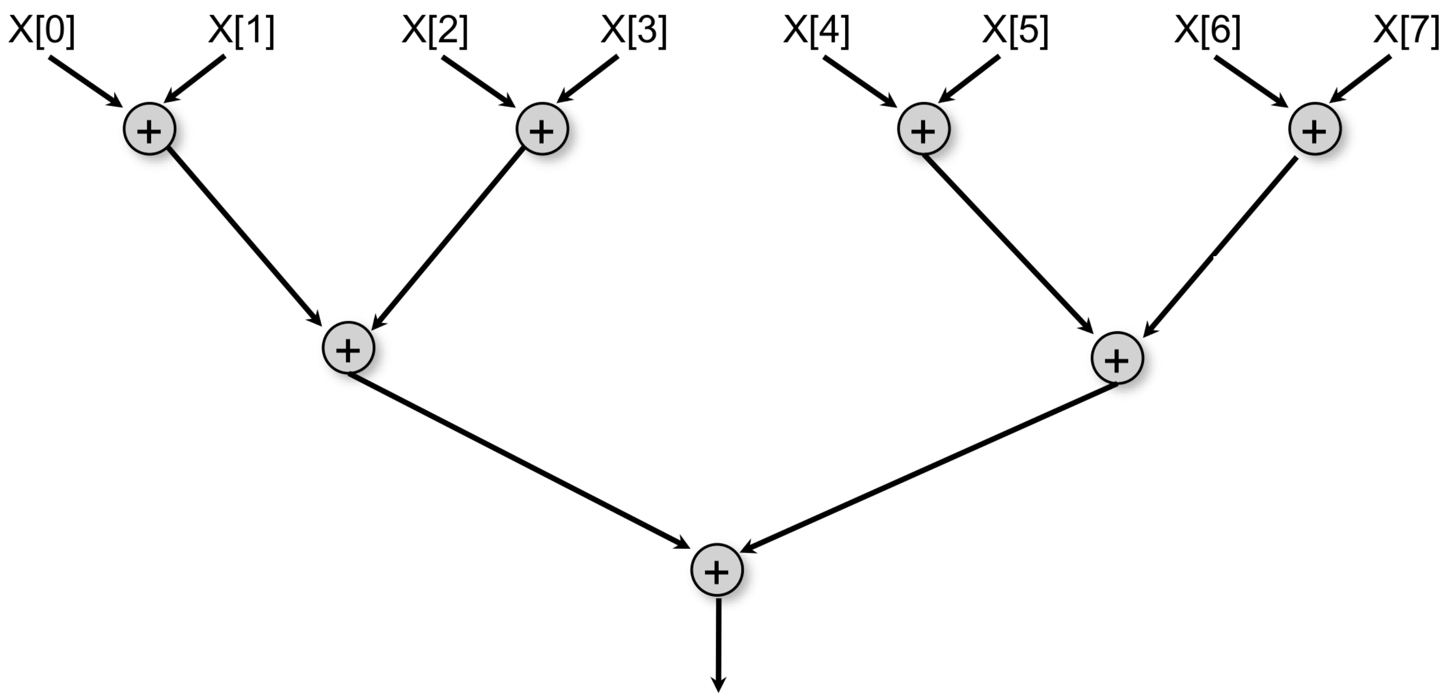
Future Task Declarations and Uses

- Variable of type future is a reference to a future object
  - Container for return value from future task
  - The reference to the container is also known as a “handle”
- Two operations that can be performed on variable V of type future:
  - Assignment: V1 can be assigned value of type future
  - Blocking read: V1.get() waits until the future task referred to by V1 has completed, and then propagates the return value

Comparison of Future Task and Regular Async Versions of Two-Way Array Sum

- Future task version initializes two references to future objects, sum1 and sum2, and both are declared as final
- No finish construct needed in this example
  - Instead parent task waits for child tasks by performing sum1.get() and sum2.get()
- Easier to guarantee absence of race conditions in Future Task version
  - No race on sum because it is a local variable in tasks T2 and T3
  - No race on future variables, sum1 and sum2, because of blocking-read semantics

Reduction Tree Schema for computing Array Sum in parallel



Question:

- How can we implement this schema using future tasks instead of async tasks?

Array Sum using Future Tasks (Seq version)

Recursive divide-and-conquer pattern

```
static int computeSum(int[] X, int lo, int hi) {
  if ( lo > hi ) return 0;
  else if ( lo == hi ) return X[lo];
  else {
    int mid = (lo+hi)/2;
    final sum1 = computeSum(X, lo, mid);
    final sum2 = computeSum(X, mid+1, hi);
   // Parent now waits for the container values
    return sum1 + sum2;
  }
 } // computeSum
int sum = computeSum(X, 0, X.length-1); // main program
```

Array Sum using Future Tasks (two futures per method call)

Recursive divide-and-conquer pattern

```
static int computeSum(int[] X, int lo, int hi) {
  if ( lo > hi ) return 0;
  else if ( lo == hi ) return X[lo];
  else {
    int mid = (lo+hi)/2;
    final future sum1 =
          async { return computeSum(X, lo, mid); };
    final future sum2 =
         async { return computeSum(X, mid+1, hi); };
   // Parent now waits for the container values
    return sum1.get() + sum2.get();
   }
 } // computeSum
int sum = computeSum(X, 0, X.length-1); // main program
```

Array Sum using Future Tasks (one future per method call)

Recursive divide-and-conquer pattern

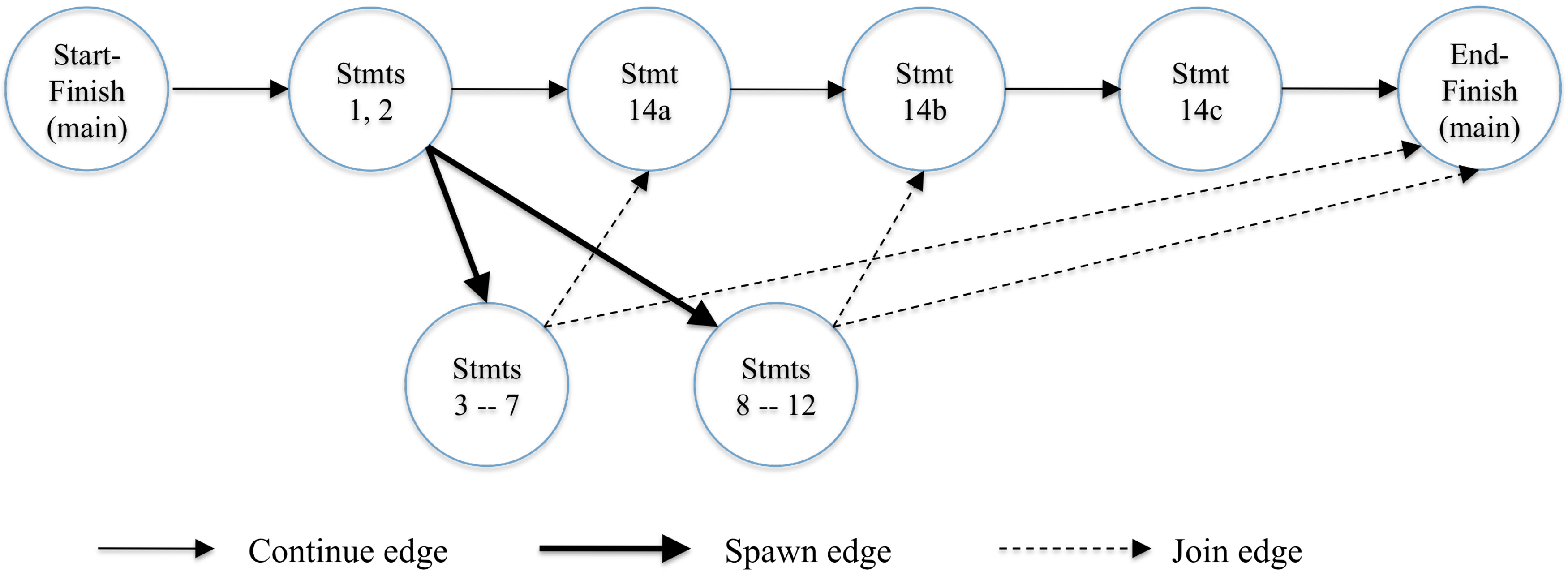
```
static int computeSum(int[] X, int lo, int hi) {
  if ( lo > hi ) return 0;
  else if ( lo == hi ) return X[lo];
  else {
    int mid = (lo+hi)/2;
    final future sum1 =
          async { return computeSum(X, lo, mid); };
    final sum2 = computeSum(X, mid+1, hi);
   // Parent now waits for the container values
   return sum1.get() + sum2;
   }
 } // computeSum
int sum = computeSum(X, 0, X.length-1); // main program
```

Computation Graph Extensions for Future Tasks

- Since a get() is a blocking operation, it must occur on boundaries of CG nodes/steps
  - May require splitting a statement into sub-statements e.g.,
    - 14: int sum = sum1.get() + sum2.get();

    can be split into three sub-statements
    - 14a int temp1 = sum1.get();
    - 14b int temp2 = sum2.get();
    - 14c int sum = temp1 + temp2;
- Spawn edge connects parent task to child future task, as before
- Join edge connects end of future task to Immediately Enclosing Finish (IEF), as before
- Additional join edges are inserted from end of future task to each get() operation on future object

Computation Graph for Two-way Parallel Array Sum using Future Tasks



NOTE: DrHJ's data race detection tool does not support futures as yet (it only supports finish, async, and isolated constructs)

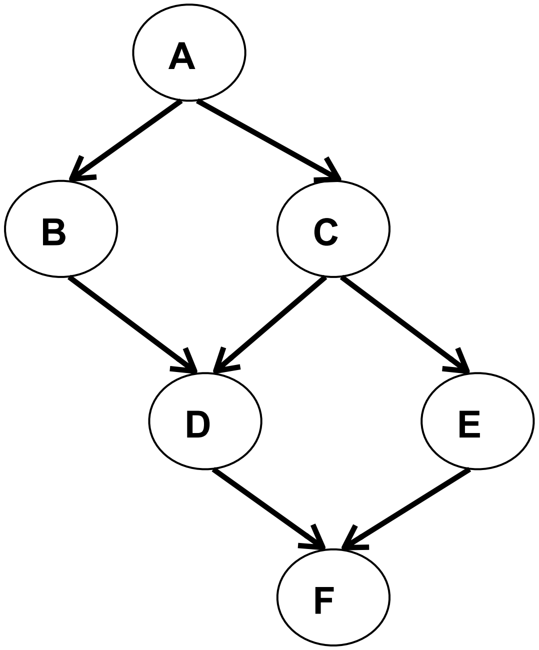
Course Announcements

- All Unit 1 lecture and demonstration quizzes are due by Jan 24th
  - Quizzes are still being uploaded into edX
- Homework 1 assigned today, and is due on Jan 31st
- 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
- Course grading rubric (see course wiki for details)
  - Six homeworks = 40% total (6.67% per homework)
  - Exam 1 = 20% (Take home, assigned Feb 26th, due by Feb 28th)
  - Exam 2 = 20% (Take home, assigned April 25th, due by May 2nd)
  - edX quizzes = 10% total
  - Class participation = 10% total (labs, worksheets, in-class Q&A, Piazza Q&A, bug reports, demonstration volunteers, ...)

Worksheet #5: Computation Graphs for Async-Finish and Future Constructs

Name: ______________________ Netid: ______________________

1) Can you write HJ 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.



2) Can you write HJ 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.

Use the space below for your answers