COMP 322: Fundamentals of Parallel Programming

Lecture 15: Review of Module1 HJ-lib API

Shams Imam (guest lecturer)
Department of Computer Science, Rice University
shams@rice.edu

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

Outline of Today's Lecture

- HJ-lib configuration
  - Worker threads
  - Abstract metrics
- Parallel Constructs
  - Async-Finish
  - Loop-level parallelism
  - Finish Accumulator
  - Futures and Data-driven Futures
  - Phasers

Important Javadoc Links

- **Module 1**
  - —http://www.cs.rice.edu/~vs3/hjlib/doc/edu/rice/hj/Module1.html
- **The HJ API package**
  - —http://www.cs.rice.edu/~vs3/hjlib/doc/edu/rice/hj/api/package-frame.html
- **Finish Accumulator**
  - —http://www.cs.rice.edu/~vs3/hjlib/doc/edu/rice/hj/runtime/accumulator/FinishAccumulator.html

Initializing/Finalizing the Habanero-Java Runtime

- initializeHabanero()
  - Initialize the Habanero-Java execution environment and worker threads.
- finalizeHabanero()
  - Cleans up all Habanero-Java state.
- initialize/finalize pair can be called multiple times in a single program, if needed.
- Example:

```
public static void main(String[] args) {
  ...
  initializeHabanero();
  // parallel program
  ...
  finalizeHabanero();
}
```

HJ Configuration

- HJ-lib programmatically configured using System properties
- Need to be set before the call to initializeHabanero()
- Available configurations are listed in the HjSystemProperty enum
  - abstractMetrics
    - Enables abstract execution metrics.
  - executionGraph
    - Whether to display the execution graph.
  - speedUpGraph
    - Whether to display the speed-up graph.

HJ Configuration (contd.)

- Available configurations are listed in the HjSystemProperty enum
  - maxThreads
    - Maximum number of worker threads to use.
  - numWorkers
    - Number of workers to use.
  - showRuntimeStats
    - Show runtime stats.
  - trackDeadlocks
    - Whether to track deadlock situations when all workers become idle.

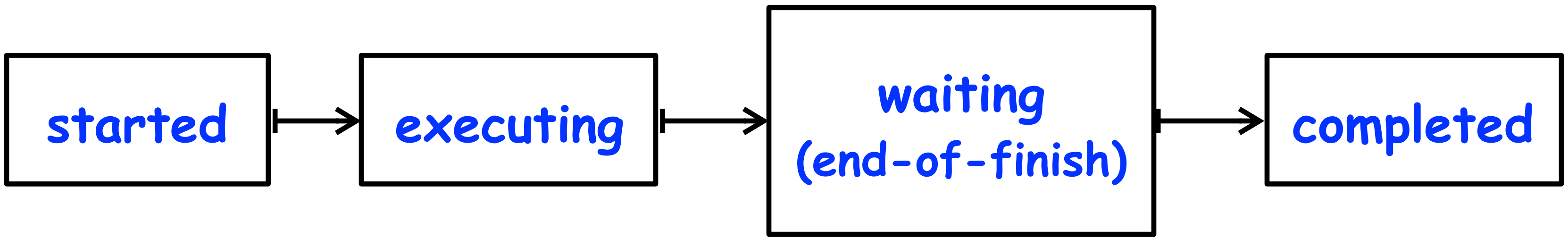
Async statement

- async(HjRunnable runnable)
  - Creates a new asynchronous task to execute the wrapped statements.
- Async Task Lifecycle:

- asyncSeq(boolean sequentialize, HjRunnable runnable)
  - boolean condition determines if the async should just be executed sequentially (when the condition is true) in the parent task.
- asyncSeq(boolean sequentialize, HjRunnable seqRunnable, HjRunnable parRunnable)
  - boolean condition determines if the async should just be executed sequentially as seqRunnable (when the condition is true) or in parallel as parRunnabe in the parent task.

Finish statement

- <u>finish(HjRunnable runnable)</u>
  - Creates a new finish scope to execute the wrapped statements.
  - Nested finish scopes are implemented using a Stack.
- Lifecycle:



forseq, forall and forasync

- forseq is a sequential loop
  - convenience method to ease porting to parallel loops
- All forall API's include an implicit finish.
  - forall(startInc, endInc, (k) -> S2(k))

    Semantics are as follows:

```
finish(() -> {
   for (int k = startInc; k <= endInc; k++) {
     final int kk = k;
     async(() -> { S2(kk); });
   }
 });
```

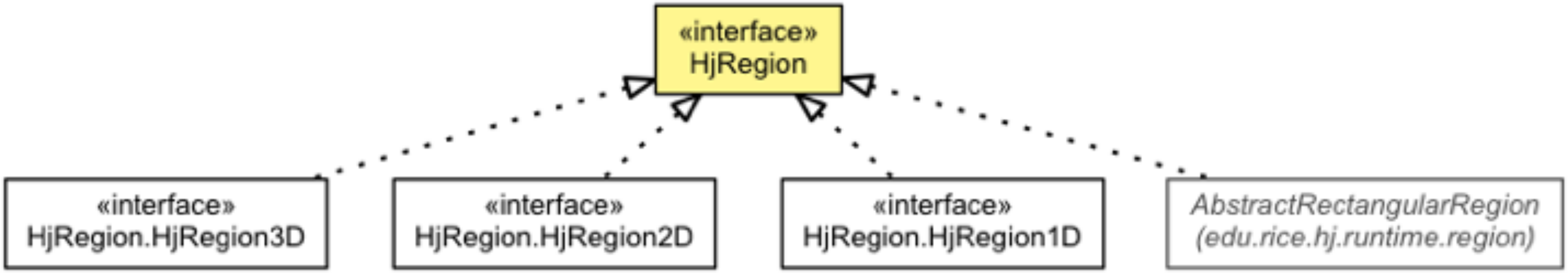
- forasync is like forall, but without the finish.

HjPoint

- HjPoint
  - A point represents an ordered tuplet of n terms, (a0,a1,a2,...,an-1) where n is the rank of the space in which the point is located.
  - The HjPoint interface provides an immutable view to a point.
  - Access to the individual terms is zero-indexed.
  - A point variable can hold values of different ranks over its lifetime.
  - Factory method: newPoint(int... values)

HjRegion

- **<u>HjRegion</u>**
  - A region represents a (sparse or dense) k-dimensional space of points.
  - A convex k-dimensional region is easy to represent, e.g. as a list of k (min, max) pairs.
  - Supports Iterable interface, returns List of HjPoint



forall variants

- forall(edu.rice.hj.api.HjRegion.HjRegion1D hjRegion, edu.rice.hj.api.HjProcedureInt1D body)
- forall(edu.rice.hj.api.HjRegion.HjRegion2D hjRegion, edu.rice.hj.api.HjProcedureInt2D body)
- forall(edu.rice.hj.api.HjRegion.HjRegion3D hjRegion, edu.rice.hj.api.HjProcedureInt3D body)
- forall(int s0, int e0, edu.rice.hj.api.HjProcedure<java.lang.Integer> body)
- forall(int s0, int e0, int s1, int e1, edu.rice.hj.api.HjProcedureInt2D body)
- forall(java.lang.Iterable<T> iterable, edu.rice.hj.api.HjProcedure<T> body)
- …
- forasync has similar variants
- chunked variants also available, see documentation

Finish Accumulators

- Creation
  - ```
    FinishAccumulator ac =
        newFinishAccumulator(Operator, type);
    ```
  - operator can be Operator.SUM, Operator.PROD, Operator.MIN, Operator.MAX
  - type can be Integer.class or Double.class
    ```

Finish Accumulators

- Registration
  - `finish(ac1, ac2, …, () -> { ... } );`
  - Accumulators ac1, ac2, ... are registered with the finish scope
- Accumulation
  - `ac.put(data);`
  - can be performed by any statement in finish scope that registers ac
- Retrieval
  - `ac.get();`
  - get() is nonblocking because finish provides the necessary synchronization
  - Either returns initial value before end-finish or final value after end-finish
  - result from get() will be deterministic if operator is associative and commutative

Futures

- Future represents an async with a return value
- future(HjCallable<V> callable)
  - Construct to create an asynchronous task that returns a result which will be available in the future.
  - Returns an HjFuture
- futureSeq(boolean sequentialize, HjCallable<V> callable)
- futureSeq(boolean sequentialize, HjCallable<V> seqCallable, HjCallable<V> parCallable)

Data-Driven Futures

`HjDataDrivenFuture<T1> ddfA = newDataDrivenFuture();`

- Allocate an instance of a data-driven-future object (container)
- Object in container must be of type T1

`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")

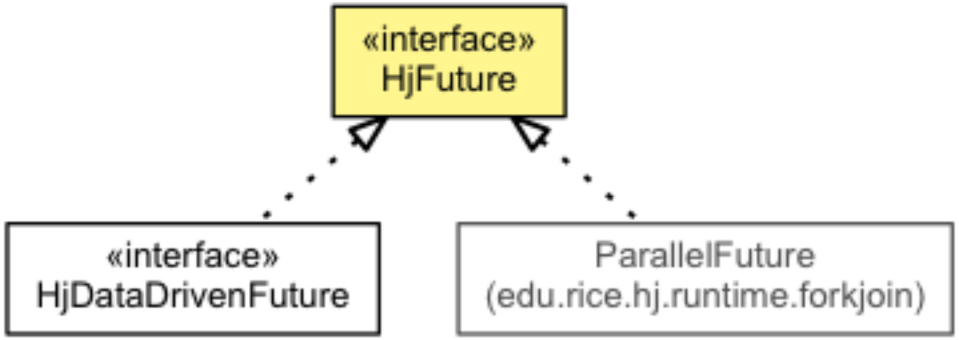
`ddfA.put(V);`

- Store object V (of type T1) in ddfA, thereby making ddfA available
- Single-assignment rule: at most one put is permitted on a given DDF

`ddfA.get();`

- Return value (of type T1) stored in ddfA
- Can only be safely performed by async's that contain ddfA in their await clause (hence no blocking is necessary for DDF gets)

HjFutures and HjDataDrivenFuture



- future.get()
  - Returns the value wrapped in the future.
- future.resolved()
  - Returns whether the future has been resolved, i.e. the value has been computed.
    - WARNING: use of resolved() can introduce nondeterminism