COMP 322: Fundamentals of Parallel Programming

Lecture 35: Eureka-style Speculative Task Parallelism

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

http://comp322.rice.edu

What is "Eureka Style" Computation?

- Many optimization and search problems attempts to find a result with a certain property or cost

- Announce when a result has been found

  - An "aha!" moment – **Eureka** event

  - Can make rest of the computation unnecessary

==> Opportunities for "speculative parallelism", e.g., Parallel Search, Branch and Bound Optimization, Soft Real-Time Deadlines, Convergence Iterations, . . .

Image source: http://www.netstate.com/states/mottoes/images/ca_eureka.jpg

Simple Example: Search in a 2-D Matrix

```
1.    class AsyncFinishSearch {
2.      AtomicReference atomicRefFactory() {
3.        // [x, y] is pseudocode syntax for specifying an integer pair
4.        return new AtomicReference([-1, -1])
5.      }
6.      int[] doWork(matrix, goal) {
7.        val token = atomicRefFactory()
8.        finish (() -> {
9.          // How to break from a forasync loop?
10.          forasyncChunked (0, matrix.length - 1, (r) -> {
11.            procRow(matrix(r), r, goal, token)
12.          });
13.        });
14.        // return either [-1, -1] or valid index [i, j] matching goal
15.        return token.get()
16.      }
17.      void procRow(array, r, goal, token) {
18.        for (int c = 0; c < array.length(); c++)
19.          if goal.match(array(c)) // eureka!!!
20.            token.set([r, c])
21.            return
22.    } }
```

Challenges in Parallelizing a Eureka-Style Computation

- Detecting eureka events

  —need to pass token around as extra argument

- Terminating executing tasks after eureka

  —manual termination via cancellation tokens can be a burden

  —throwing an exception does not terminate other parallel tasks

  —"killing" a parallel task can lead to unpredictable results (depending on when the task was terminated)

7

Example of Manual termination via Cancellation Tokens

- Manual periodic checks with returns

- User controls responsiveness

```
1.      class AsyncFinishManualSearch {
2.        int[] doWork(matrix, goal) {
3.          val token = atomicRefFactory()
4.          finish (() -> {
5.            forasyncChunked (0, matrix.length - 1, (r) -> {
6.              if (token.get() != [-1, -1])
7.                return
8.              procRow(matrix(r), r, goal, token)
9.            });
10.         });
11.         // [-1, -1] or valid index [i, j] matching goal
12.         return token.get()
13.       }
14.       void procRow(array, r, goal, token) {
15.         for (int c = 0; c < array.length(); c++)
16.           if (token.get() != [-1, -1])
17.             return
18.           if goal.match(array(c)) // eureka!!!
19.             token.set([r, c])
20.             return
```

Repeated checks which are written manually

- Cumbersome to write

- Impossible to support inaccessible functions

HJlib solution: the Eureka construct

1. eureka = eurekaFactory() // create Eureka object

2. finish (eureka) S1 // register eureka w/ finish

   - Multiple finish'es can register on same Eureka

   - Wait for all tasks to finish as before

     - Except that some tasks may terminate early when eureka is resolved

3. async // task candidate for early termination

   - Inherits eureka registrations from immediately-enclosing finish

4. offer()

   - Triggers eureka event on registered eureka

5. check() // Like a "break" statement for a task

   - Causes task to terminate if eureka resolved

2D Matrix Search using Eureka construct (Pseudocode)

```
1.    class AsyncFinishEurekaSearch {
2.      HjEureka eurekaFactory() {
3.        return ...
4.      }
5.      int[] doWork(matrix, goal) {
6.        val eu = eurekaFactory()
7.        finish (eu, () -> { // eureka registration
8.          forasyncChunked (0, matrix.length - 1, (r) -> {
9.            procRow(matrix(r), r, goal)
10.         });
11.       });
12.       // return either [-1, -1] or valid index [i, j] matching goal
13.       return eu.get()
14.     }
15.     void procRow(array, r, goal) {
16.       for (int c = 0; c < array.length(); c++)
17.         check() // cooperative termination check
18.         if goal.match(array(c)) // eureka!!!
19.           offer([r, c]) // trigger eureka event
20.     } }
```

Eureka Variants (Pseudocode)

```
def eurekaFactory() {
    val initValue = [-1, -1]
    return new SearchEureka(initValue)
}
```

```
def eurekaFactory() {
    val K = 4
    return new CountEureka(K)
}
```

```
def eurekaFactory() {
    // comparator to compare indices
    val comparator = (a, b) -> {
        ((a.x - b.x) == 0) ? (a.y - b.y) : (a.x - b.x)
    }
    val initValue = [INFINITY, INFINITY]
    return new MinimaEureka(initValue, comparator)
}
```

```
def eurekaFactory() {
    val time = 4.seconds
    return new TimerEureka(time)
}
```

```
def eurekaFactory() {
    val units = 400
    return new EngineEureka(units)
}
```

COMP 322, Spring 2022 (M.Joyner, Zoran Budimlić)

Binary Tree Search Example

```
HjSearchEureka<Integer> eureka = newSearchEureka(null);
finish(eureka, () -> {
    async(() -> {
        searchBody(eureka, rootNode, elemToSearch);
    });
});


private static void searchBody(
  HjSearchEureka<Integer> eureka, Node rootNode,
  int elemToSearch) throws SuspendableException {
    eureka.check();
    if (rootNode.value == elemToSearch) {
        eureka.offer(rootNode.id);
    }
    if (rootNode.left != null) {
        async(() -> {
            searchBody(eureka, rootNode.left, elemToSearch);
        });
    }
    if (rootNode.right != null) {
        async(() -> {
            searchBody(eureka, rootNode.right, elemToSearch);
        });
    }
}
```

Inputs:
- binary tree, T
- Unique id for each node in T (for example, in breadth-first order: root.id = 0, root.left.id = 1, root.right.id = 2, …)
- value for each node in T that is the search target

Outputs:
- calls to offer() resolve eureka
- calls to check() can lead to early termination
- final value of eureka contains id of a node with value == elemToSearch

Tree Min Index Search Example

```
HjExtremaEureka<Integer> eureka = newExtremaEureka(
  Integer.MAX_VALUE, (Integer i, Integer j) -> i.compareTo(j));
finish(eureka, () -> {
    async(() -> {
        minIndexSearchBody(eureka, rootNode, elemToSearch);
    });
});

private static void minIndexSearchBody(
  HjExtremaEureka<Integer> eureka, Node rootNode,
  int elemToSearch) throws SuspendableException {
    eureka.check(rootNode.id);
    if (rootNode.value == elemToSearch) {
        eureka.offer(rootNode.id);
    }
    if (rootNode.left != null) {
        async(() -> {
            minIndexSearchBody(eureka, rootNode.left, elemToSearch);
        });
    }
    if (rootNode.right != null) {
        async(() -> {
            minIndexSearchBody(eureka, rootNode.right, elemToSearch);
        });
    }
}
```

Inputs:
- binary tree, T
- id for each node in T, in breadth-first order e.g., root.id = 0, root.left.id = 1, root.right.id = 2, …
- value for each node in T that is the search target

Outputs:
- calls to offer() update eureka with minimum id found so far (among those that match)
- calls to check() can lead to early termination if the argument is >= than current minimum in eureka
- final value of eureka contains minimum id of node with value == elemToSearch

AND-composition of Eurekas

```
1.    class AsyncFinishEurekaDoubleSearch {
2.      int[] doWork(matrix, goal1, goal2) {
3.        val eu1 = eurekaFactory()
4.        val eu2 = eurekaFactory()
5.        val eu = eurekaComposition(AND, eu1, eu2)
6.        finish (eu, () ->{ // eureka registration
7.          forasyncChunked (0, matrix.length - 1, (r) ->{
8.            procRow(matrix(r), r, goal1, goal2)
9.          });
10.        });
11.        // return either [-1, -1] or valid index [i, j] matching goal
12.        return eu.get()
13.      }
14.     void procRow(array, r, g1, g2) {
15.      for (int c = 0; c < array.length(); c++)
16.        val checkArg = [[r, c], [r, c]]
17.        check(checkArg) // cooperative termination check
18.        val loopElem = array(c)
19.        val res1 = g1.match(loopElem) ? [r, c] : null
20.        val res2 = g2.match(loopElem) ? [r, c] : null
21.        val foundIdx = [res1, res2] // pair of values for eu1 and eu2
22.        offer(foundIdx) // possible eureka event
23.    } }
```

Announcements & Reminders

- No lab this week

- Hw #5 is due Friday, April 22nd at 11:59pm