COMP 322: Fundamentals of Parallel Programming

Lecture 13: Finish Accumulators

Mack Joyner
mjoyner@rice.edu

http://comp322.rice.edu

Comparing Async-Finish with Future-Get

- Similarities:
  - Finish and Get can be used to synchronize and avoid data races
  - Finish waits for both async and future tasks
- Differences:
  - Futures have return values
  - Future gets can model a larger set of computation graphs than async-finish
  - Finish can wait for an unbounded set of tasks (determined at runtime)

Two-way Parallel Array Sum using async & finish constructs

Algorithm 2: Two-way Parallel ArraySum

Input: Array of numbers, X.
Output: sum = sum of elements in array X.

```
// Start of Task T1 (main program)
sum1 ← 0; sum2 ← 0;
// Compute sum1 (lower half) and sum2 (upper half) in parallel.
finish{
    async{
        // Task T2
        for i ← 0 to X.length/2 − 1 do
            sum1 ← sum1 + X[i];
    };
    async{
        // Task T3
        for i ← X.length/2 to X.length − 1 do
            sum2 ← sum2 + X[i];
    };
};
// Task T1 waits for Tasks T2 and T3 to complete
// Continuation of Task T1
sum ← sum1 + sum2;
return sum;
```

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)*

Example: count occurrences of pattern in text (sequential version)

```
// Count all occurrences
int count = 0;
{
  for (int ii = 0; ii <= N - M; ii++) {
    int i = ii;
    // search for match at position i
    for (j = 0; j < M; j++)
      if (text[i+j] != pattern[j]) break;
    if (j == M) count++; // Increment count
  } // for-ii
}
}
print count; // Output
```

Example: count occurrences of pattern in text (parallel version using finish accumulator)

```
// Count all occurrences
a = new Accumulator(SUM, int)
finish(a) {
  for (int ii = 0; ii <= N - M; ii++) {
    int i = ii;
    async { // search for match at position i
      for (j = 0; j < M; j++)
        if (text[i+j] != pattern[j]) break;
      if (j == M) a.put(1); // Increment count
    } // async
  }
} // finish
print a.get(); // Output
```

Error Conditions with Finish Accumulators

1. Non-owner task cannot access accumulator outside registered finish

```
// T1 allocates accumulator a
accumulator a = newFinishAccumulator(…);
a.put(1); // T1 can access a
async { // T2 cannot access a
  a.put(1); Number v1 = a.get();
}
```

2. Non-owner task cannot register accumulator with a finish

```
// T1 allocates accumulator a
accumulator a = newFinishAccumulator(...);
async {
  // T2 cannot register a with finish
  finish (a) { async a.put(1);  }
```

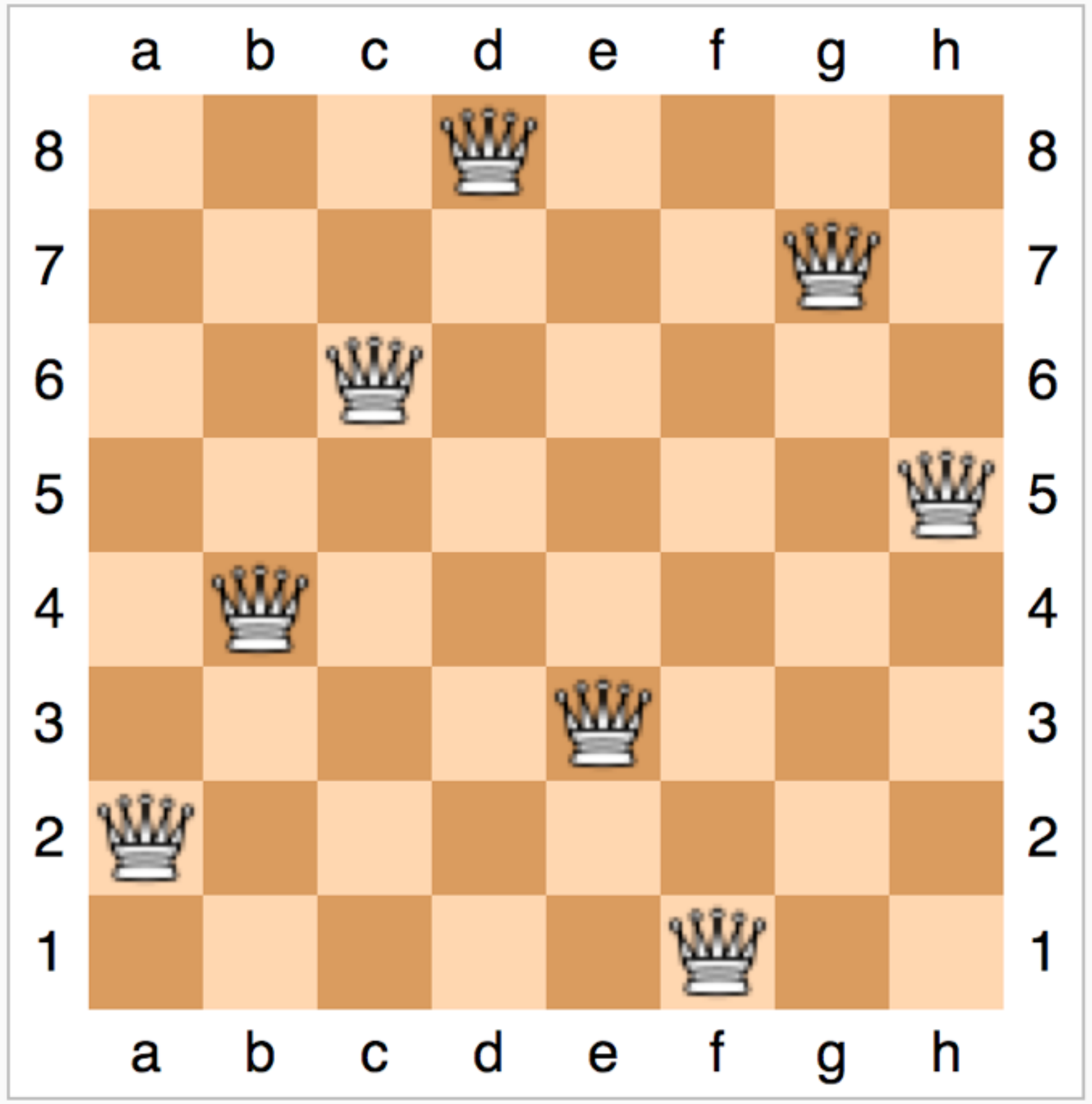
The N-Queens Problem

How can we place n queens on an n×n chessboard so that no two queens can capture each other?

A queen can move any number of squares horizontally, vertically, and diagonally.

Here, the possible target squares of the queen Q are marked with an x.

x			x			x	
	x		x		x		
		x	x	x			
x	x	x	Q	x	x	x	x
		x	x	x			
	x		x		x		
x			x			x	
			x				x



One solution to the eight queens puzzle

Backtracking Solution

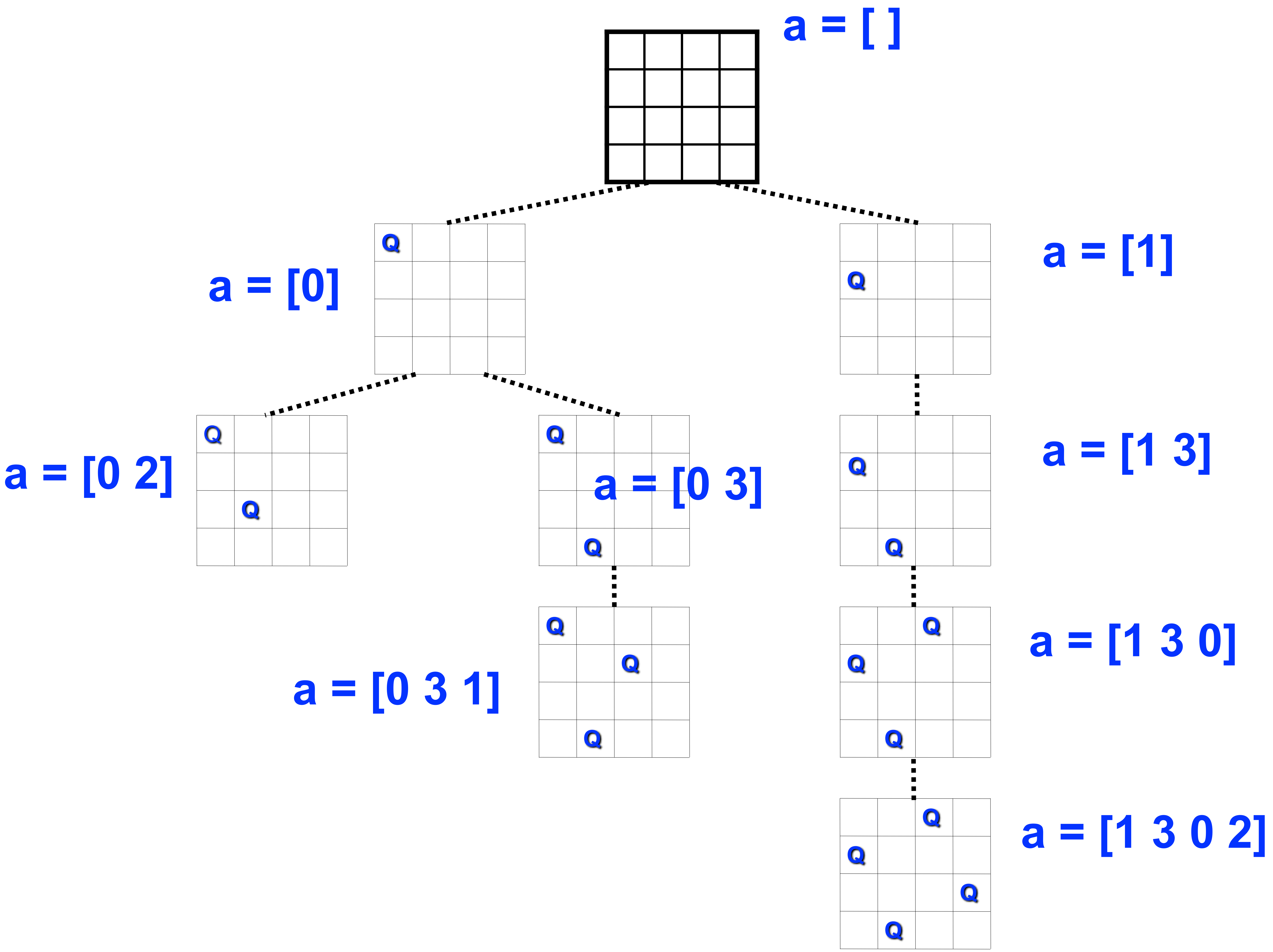
empty board

place 1st queen

place 2nd queen

place 3rd queen

place 4th queen



Sequential solution for NQueens (counting all solutions)

```
count = 0;
size = 8; nqueens_kernel_seq(new int[0], 0);
System.out.println("No. of solutions = " + count);
. . .
void nqueens_kernel_seq(int [] a, int depth) {
  if (size == depth) count++;
  else
    /* try each possible position for queen at depth */
    for (int i =  0; i < size; i++) {
      /* allocate a temporary array and copy array a into it */
      int [] b = new int [depth+1];
      System.arraycopy(a, 0, b, 0, depth);
      b[depth] = i; // Try to place queen in row i of column depth
      if (ok(depth,b)) // check if placement is okay
        nqueens_kernel_seq(b, depth+1);
    } // for
} // nqueens_kernel_seq()
```

How to extend sequential solution to obtain a parallel solution?

```
count = 0;
size = 8; finish { nqueens_kernel_par(new int[0], 0); }
System.out.println("No. of solutions = " + count);
. . .
void nqueens_kernel_par(int [] a, int depth) {
  if (size == depth) count++;
  else
    /* try each possible position for queen at depth */
    for (int i =  0; i < size; i++) async {
      /* allocate a temporary array and copy array a into it */
      int [] b = new int [depth+1];
      System.arraycopy(a, 0, b, 0, depth);
      b[depth] = i; // Try to place queen in row i of column depth
      if (ok(depth,b)) // check if placement is okay
        nqueens_kernel_par(b, depth+1);
    } // for
} // nqueens_kernel_par()
```

How to extend sequential solution to obtain a parallel solution?

```
count = 0;
size = 8; finish { nqueens_kernel_par(new int[0], 0); }
System.out.println("No. of solutions = " + count);
. . .
void nqueens_kernel_par(int [] a, int depth) {
  if (size == depth) count++;
  else
    /* try each possible position for queen at depth */
    for (int i =  0; i < size; i++) async {
      /* allocate a temporary array and copy array a into it */
     int [] b = new int [depth+1];
      System.arraycopy(a, 0, b, 0, depth);
      b[depth] = i; // Try to place queen in row i of column depth
      if (ok(depth,b)) // check if placement is okay
         nqueens_kernel_par(b, depth+1);
   } // for
} // nqueens_kernel_par()
```

DATA RACE!

How to extend sequential solution to obtain a parallel solution?

```
FinishAccumulator ac = newFinishAccumulator(Operator.SUM, int.class);
size = 8; finish(ac) { nqueens_kernel_par(new int[0], 0); }
System.out.println("No. of solutions = " + ac.get().intValue());
. . .
void nqueens_kernel_par(int [] a, int depth) {
  if (size == depth) ac.put(1);
  else
    /* try each possible position for queen at depth */
    for (int i =  0; i < size; i++) async {
      /* allocate a temporary array and copy array a into it */
      int [] b = new int [depth+1];
      System.arraycopy(a, 0, b, 0, depth);
      b[depth] = i; // Try to place queen in row i of column depth
      if (ok(depth,b)) // check if placement is okay
        nqueens_kernel_par(b, depth+1);
    } // for-async
} // nqueens_kernel_par()
```

Efficient Parallelism

```
FinishAccumulator ac = newFinishAccumulator(Operator.SUM, int.class);
size = 8; finish(ac) { nqueens_kernel_par(new int[0], 0); }
System.out.println("No. of solutions = " + ac.get().intValue());
. . .
void nqueens_kernel_par(int [] a, int depth) {
  if (size == depth) ac.put(1);
  else
    /* try each possible position for queen at depth */
    for (int i =  0; i < size; i++) async {
      /* allocate a temporary array and copy array a into it */
      int [] b = new int [depth+1];
      System.arraycopy(a, 0, b, 0, depth);
      b[depth] = i; // Try to place queen in row i of column depth
      if (ok(depth,b)) // check if placement is okay
        nqueens_kernel_par(b, depth+1);
    } // for-async
} // nqueens_kernel_par()
```

When depth is close to size, the async tasks get too small

Efficient Parallelism

```
FinishAccumulator ac = newFinishAccumulator(Operator.SUM, int.class);
size = 8; finish(ac) { nqueens_kernel(new int[0], 0); }
System.out.println("No. of solutions = " + ac.get().intValue());
. . .
void nqueens_kernel(int [] a, int depth) {
  if (depth > size - threshold) {
     nqueens_kernel_seq(a, depth)
  } else {
     nqueens_kernel_par(a, depth)
  }
} // nqueens_kernel()
```

Announcements & Reminders

- Homework #2 is due Wednesday, Feb. 14th at 11:59pm
- Midterm exam is Thursday, Feb. 22nd from 7pm - 10pm (Canvas)