COMP 322: Fundamentals of Parallel Programming

Lecture 7: Finish Accumulators

Instructors: Vivek Sarkar, Mack Joyner
Department of Computer Science, Rice University
{vsarkar, mjoyner}@rice.edu

http://comp322.rice.edu

Worksheet #6 solution: Parallelizing Pascal's Triangle with Futures and Memoization

There are four variants of the Binomial Cooefficients program provided in four different HJlib methods in the next page:

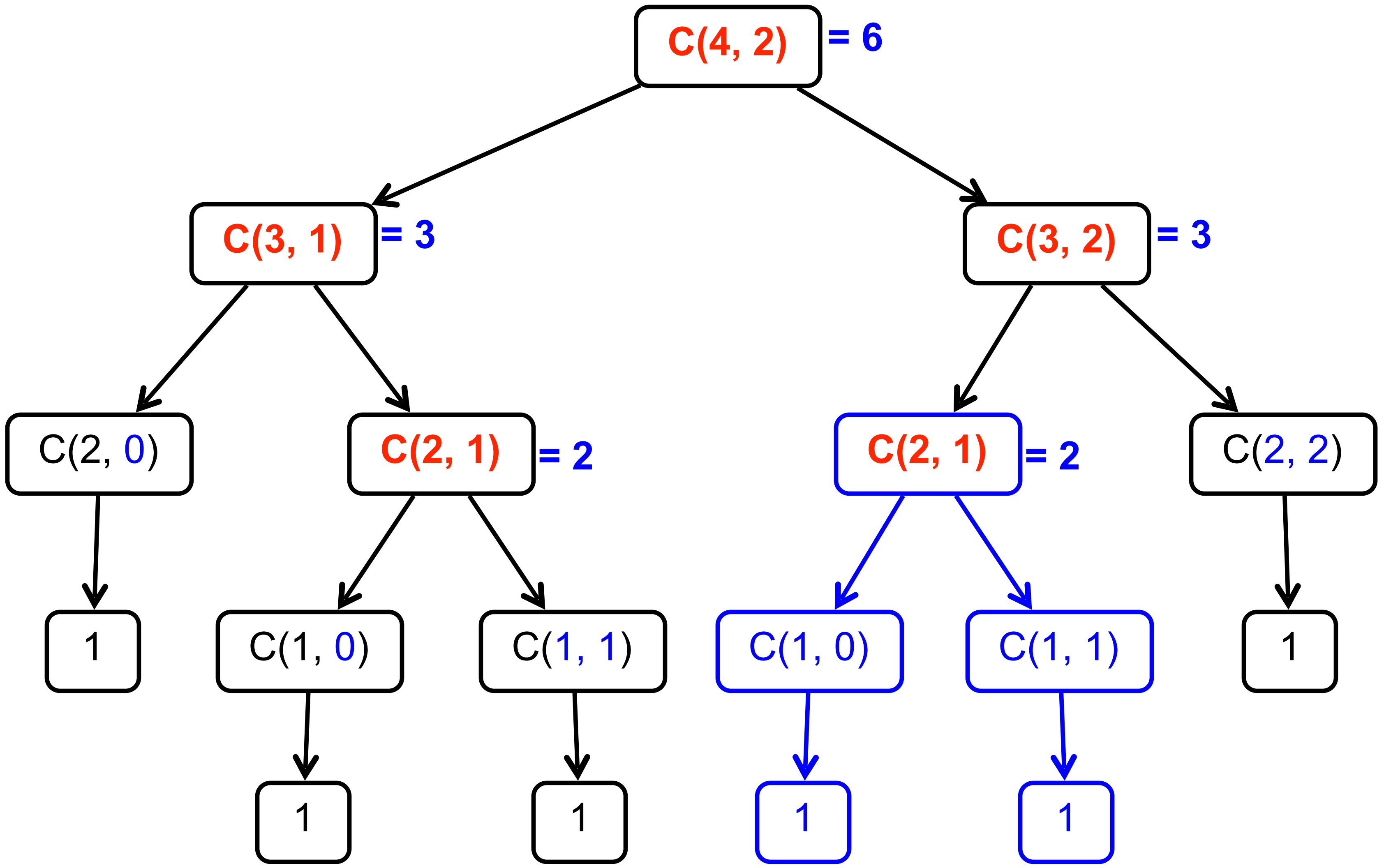
a. Sequential Recursive without Memoization (chooseRecursiveSeq())
b. Parallel Recursive without Memoization (chooseRecursivePar())
c. Sequential Recursive with Memoization (chooseMemoizedSeq())
d. Parallel Recursive with Memoization (chooseMemoizedPar())

Your task is to analyze the WORI
four versions, for the input N = 4
ComputeSum() has COST = 1, a
Complete all entries in the table:

Do you agree with the following statement: "Parallelization of inefficient algorithms often leads to more ideal parallelism than parallelization of efficient algorithms" in the context of this worksheet?

Variant	Work	CPL	Ideal Parallelism
chooseRecursiveSeq	5	5	1
chooseRecursivePar	5	3	5/3 = 1.67
chooseMemoizedSeq	4	4	1
chooseMemoizedPar	4	3	4/3 = 1.33

REMINDER: computation structure of C(4,2) Nodes with calls to ComputeSum() are in red



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); }
```

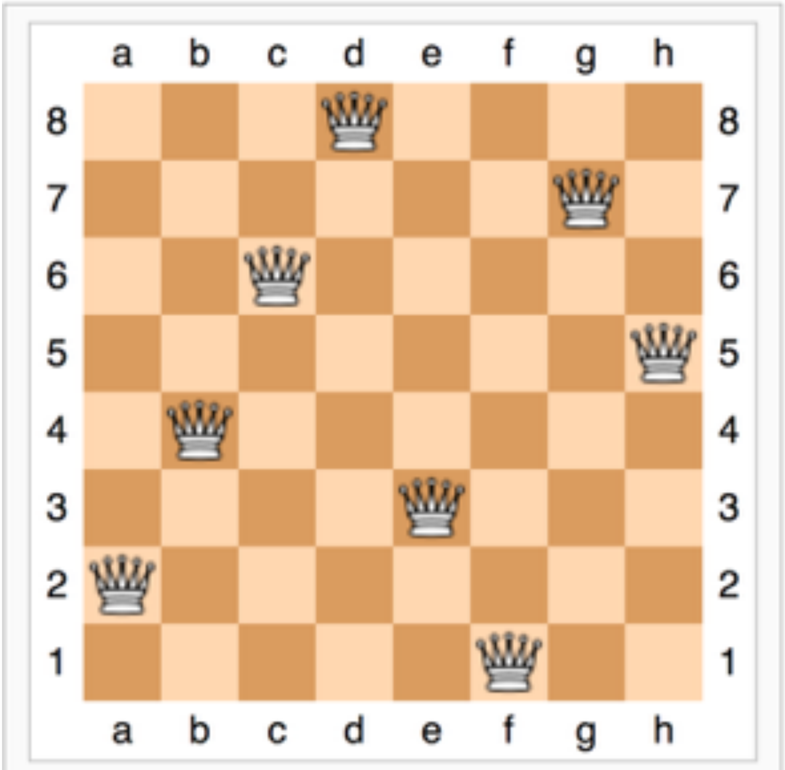
The N-Queens Problem

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

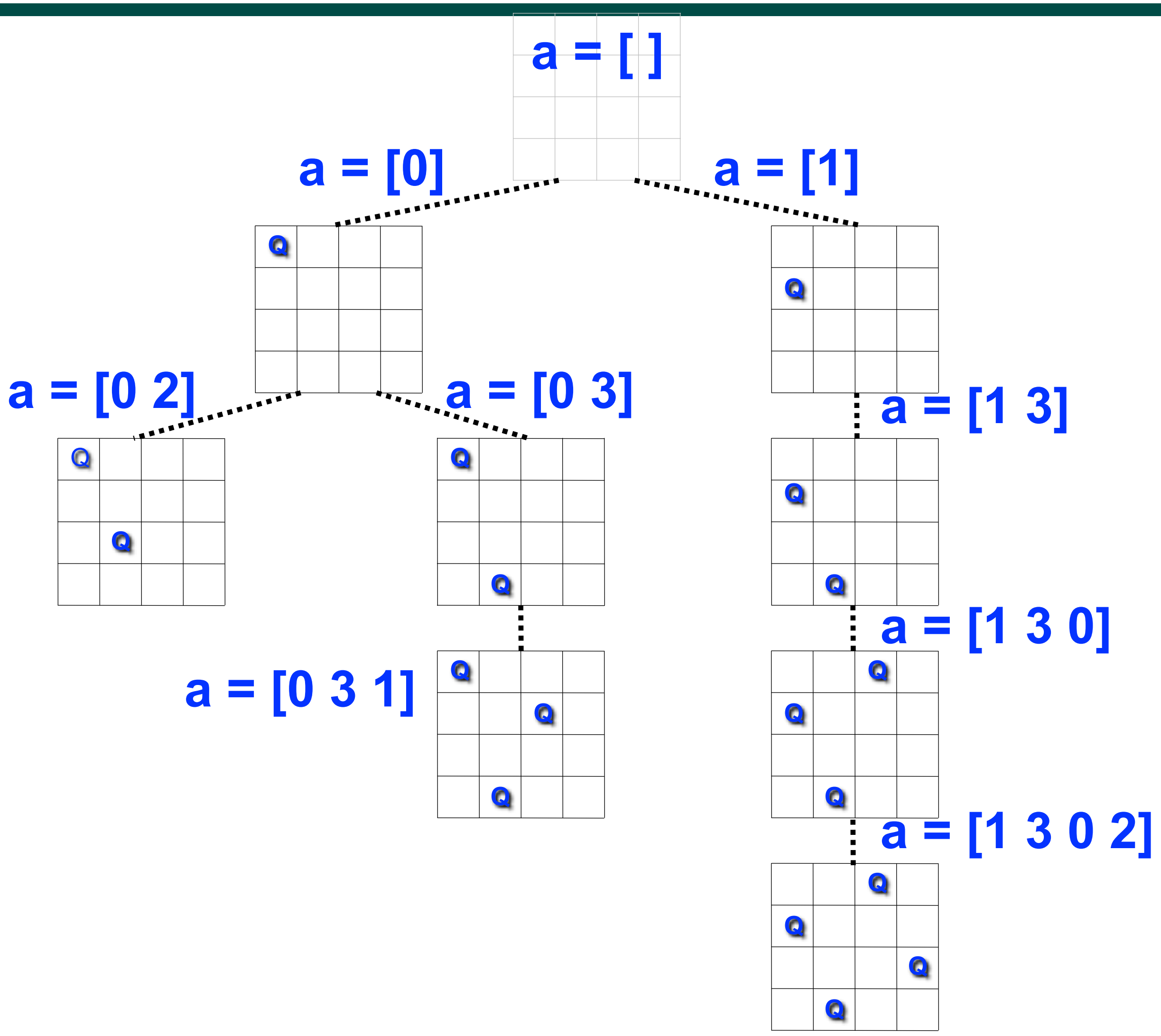
empty board

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(new int[0], 0);
System.out.println("No. of solutions = " + count);
. . .
void nqueens_kernel(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+1,b)) // check if placement is okay
          nqueens_kernel(b, depth+1);
    } // for
} // nqueens_kernel()
```

How to extend sequential solution to obtain a parallel solution?

```
count = 0;
size = 8; finish nqueens_kernel(new int[0], 0);
System.out.println("No. of solutions = " + count);
. . .
void nqueens_kernel(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+1,b)) // check if placement is okay
        nqueens_kernel(b, depth+1);
   } // for
} // nqueens_kernel()
```

But there's a data race on count?

How to extend sequential solution to obtain a parallel solution?

```
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 (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+1,b)) // check if placement is okay
         nqueens_kernel(b, depth+1);
    } // for-async
} // nqueens_kernel()
```

Announcements & Reminders

- IMPORTANT:
  - —Watch video & read handout for topic 2.4 for next lecture on Friday, Jan 27th
- HW1 is due by 11:59pm TODAY
- Quiz for Unit 1 (topics 1.1 - 1.5) is due by Friday (Jan 27th) on Canvas
- See course web site for all work assignments and due dates
- Use Piazza (public or private posts, as appropriate) for all communications re. COMP 322
- See Office Hours link on course web site for latest office hours schedule. Group office hours are now scheduled during 3pm - 4pm on MWF in DH 3092 by default, but WILL BE HELD IN DH 3076 TODAY

Worksheet #7: Associativity and Commutativity

Name: ________________________ Netid: ________________________

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?

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

b) g(x,y) = (x+y)/2, for integers x, y

c) h(s1,s2) = concat(s1, s2) for strings s1, s2, e.g., h("ab","cd") = "abcd"