COMP 322: Fundamentals of Parallel Programming

Lecture 9: Memoization

Shams Imam (Guest Lecturer)
Department of Computer Science
Rice University
shams@rice.edu

Background: Functional Programming

- Eliminate side-effects
  - emphasizes functions that produce results that depend only on their inputs and not on the program state
  - calling a function, f(x), twice with the same value for the argument x will produce the same result both times

Example: Binomial Coefficient

- The coefficient of the x^k term in the polynomial expansion of the binomial power $(1 + x)^n$
- Number of sets of k items that can be chosen from n items
- Indexed by *n* and *k*
  - written as C(n, k)
  - read as "n choose k"
- Factorial Formula: C(n, k) = $\left(\frac{n!}{k!(n-k)!}\right)$
- Recursive Formula

  C(n, k) = C(n – 1, k – 1) + C(n – 1, k)

  Base case: C(n, n) = C(n, 0) = C(0, k) = 1

Example: Binomial Coefficient (Recursive)

```
1.  int choose(final int N, final int K) {

2.      if (N == 0 || K == 0 || N == K) {
3.          return 1;
4.      }
5.      final int left  = choose (N - 1, K - 1);
6.      final int right = choose (N - 1, K);

7.      return left + right;
8.  }
```


Example: Binomial Coefficient (Parallel)

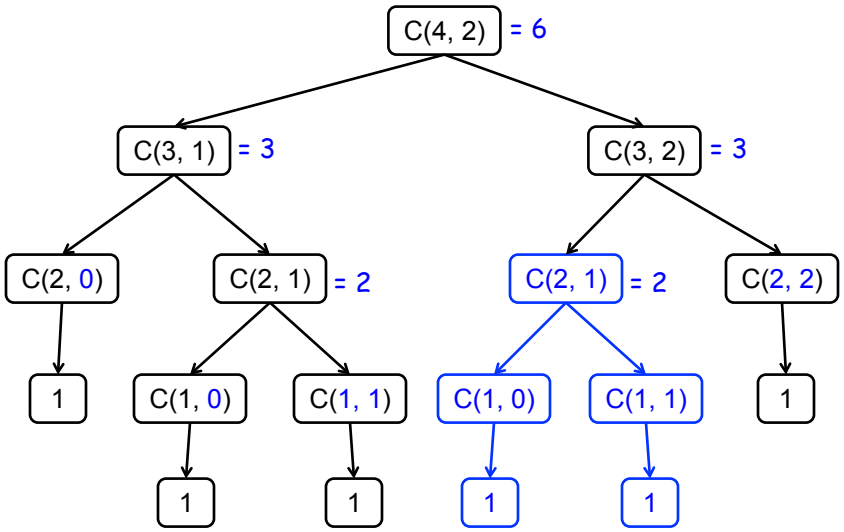
```
1.  int choose(final int N, final int K) {

2.      if (N == 0 || K == 0 || N == K) {
3.          return 1;
4.      }
5.      final future<int> left  = future { choose (N - 1, K - 1); }
6.      final future<int> right = future { choose (N - 1, K); }

7.      return left.get() + right.get();
8.  }
```

- Using futures supports incremental parallelization
- Low developer effort to achieve parallelism

Example: Binomial Coefficient



Memoization

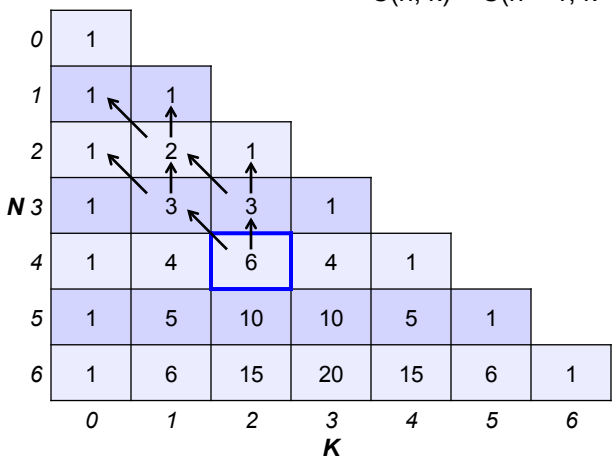
- Memoization is the idea of saving and reusing previously computed values of a function rather than recomputing them.
- A function can only be memoized if it is referentially transparent, i.e. functional
- Related to caching
  - memoized function "remembers" the results corresponding to some set of specific inputs
  - memoized function populates its cache of results transparently on the fly, as needed, rather than in advance

Memoization

- Lowers function's time-cost in exchange for space cost
  - memoization is a run-time rather than compile-time optimization
- Space-Time tradeoff:
  - If the results are reused, time is saved
  - If the results are not reused, space is wasted
- When do we know that results will be reused?
  - Structure of the problem
    - If the problem can be specified in terms of the sub-problems then we can identify whether sub-problems overlap

Binomial Coefficient with Pascal's Triangle

$$C(n, k) = C(n - 1, k - 1) + C(n - 1, k)$$



Example: Binomial Coefficient (Memoized)

```
1.  final Map<Pair<Int, Int>, Int> cache = new ...;
2.  int choose(final int N, final int K) {

3.      final Pair<Int, Int> key = Pair.factory(N, K);
4.      if (cache.contains(key)) {
5.          return cache.get(key);
6.      }

7.      if (N == 0 || K == 0 || N == K) {
8.          return 1;
9.      }
10.     final int left  = choose (N - 1, K - 1);
11.     final int right = choose (N - 1, K);

12.     final int result = left + right;
13.     cache.put(key, result);
14.     return result;
15. }
```


References

- Topic 2.2 Lecture: https://edge.edx.org/courses/RiceX/COMP322/1T2014R/courseware/a900dd0655384de3b5ef01e508ea09d7/9eacfea8754549a4bc42918149130a74/3
- Topic 2.2 Demonstration: https://edge.edx.org/courses/RiceX/COMP322/1T2014R/courseware/a900dd0655384de3b5ef01e508ea09d7/9eacfea8754549a4bc42918149130a74/4
- Recursion and Memoization: http://zoo.cs.yale.edu/classes/cs201/Spring_2014/topics/topic-memoization.pdf
- Memoization: http://en.wikipedia.org/wiki/Memoization
- Functional Programming: http://en.wikipedia.org/wiki/Functional_programming
- Binomial coefficient: http://en.wikipedia.org/wiki/Binomial_coefficient