COMP 322: Fundamentals of Parallel Programming

Lecture 9: Memoization

Vivek Sarkar, Eric Allen
Department of Computer Science, Rice University

Contact email: vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322
Worksheet #8 solution: Analysis of Map Reduce Example

Analyze the total WORK and CPL for the Map reduce example in the previous slide, under the following assumptions:

- Assume that each Map step has WORK = number of input words, and CPL=1
- Assume that each Reduce step has WORK = number of input word-count pairs, and CPL = \log_2(# occurrences for input word with largest # pairs)

WORK/CPL for all Map steps:
- WORK = 15
- CPL = 1

WORK/CPL for Reduce 1 step:
- WORK = 5
- CPL = \log_2(4) = 2

WORK/CPL for Reduce 2 step:
- WORK = 4
- CPL = \log_2(3) = 2 (round up)

Total WORK and CPL
- WORK = 15+5+4 = 24
- CPL = 1 + 2 = 3
Background: Functional Programming

• Eliminate side-effects
  • emphasizes functions whose results that depend only on their inputs and not on any other 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: $C(n, k) = \binom{n}{k} = \frac{n!}{k!(n-k)!}$
- Recursive Formula
  $C(n, k) = C(n – 1, k – 1) + C(n – 1, k)$
  Base cases: $C(n, n) = C(n, 0) = C(0, k) = 1$
Example: Binomial Coefficient (Recursive Sequential version)

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 Recursive Pseudocode)

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 =
6.         future { return choose (N-1, K-1); }
7.     final future<int> right =
8.         future { return choose (N-1, K); }
9.     return left.get() + right.get();
10. }

- Use of futures supports incremental parallelization with low developer effort
What inefficiencies do you see in the recursive Binomial Coefficient algorithm?

```
<table>
<thead>
<tr>
<th></th>
<th>C(2, 0)</th>
<th>C(2, 1)</th>
<th>C(3, 1)</th>
<th>C(3, 2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>C(1, 0)</td>
<td>C(1, 1)</td>
<td>C(2, 1)</td>
<td>C(2, 2)</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>= 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Memoization

- Memoization is the idea of saving and reusing previously computed values of a function rather than recomputing them
  - A space-time tradeoff
- 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
Pascal’s Triangle is an example of Memoization

\[ C(n, k) = C(n - 1, k - 1) + C(n - 1, k) \]
Example: Binomial Coefficient
(sequential memoized version)

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. }


Example: Binomial Coefficient (parallel memoized version w/ futures)

1. final Map<Pair<Int, Int>, future<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).get();
6.     }
7.     future<Int> f = future {
8.         if (N == 0 || K == 0 || N == K) return 1;
9.         int left = future { return choose (N-1, K-1); }
10.        int right = future { return choose (N-1, K); }
11.        return left.get() + right.get();
12.     }
13.     cache.put(key, f);
14.     return f.get();
15. }
16. }
• Assumes availability of a “thread-safe” cache library, e.g., ConcurrentHashMap
References

• Topic 2.2 Lecture & Demonstration
• Recursion and Memoization: http://zoo.cs.yale.edu/classes/cs201/Spring_2014/topics/topic-memoization.pdf
• Memoization: http://en.wikipedia.org/wiki/Memoization
• Binomial coefficient: http://en.wikipedia.org/wiki/Binomial_coefficient