Fibonacci Research Paper

Darren Hill, Hieu Dang, Ethan Janicki, and Noah Trim

University of Central Florida

Abstract- **The Fibonacci Sequence is a sequence of numbers where the n$^{\text{th}}$ digit in the sequence is generated by adding the two previous digits in the sequence which can be denoted as $F(n) = (n-1) + (n-2)$ where $F(0) = 0$ and $F(1) = 1$. The generation of this sequence is often used in computer science to teach run-time analysis and showcase different optimization methods. These optimization methods often include using different data structures, avoiding recursion, or adding support for the use of multiple threads. In this project, we will be analyzing 3 different optimization methods (top-down dynamic programming, bottom-up dynamic programming, and the use of matrices) and analyze how the inclusion of multi-threading can improve or inhibit the run time of these algorithms on the generation of a Fibonacci Sequence.**

I Introduction

The Fibonacci Sequence is a sequence of Fibonacci numbers (F_n) in which each number is the sum of the two preceding numbers. The first two entries of the sequence are commonly considered to be 0 and 1 although some people including Fibonacci himself start from 1 and 1.

For the purposes of this paper we will define the Fibonacci numbers by the following recurrence relation for $n \in \mathbb{N}$:

$$F_n = \begin{cases} 0, & \text{if n = 0} \\ 1, & \text{if n = 1} \\ F_{n-1} + F_{n-2} & \text{if n > 1} \end{cases}$$

As time has gone on more and more optimized approaches to calculating Fibonacci numbers were developed where often time, the new methodology was built off the previous, similar in nature to the Fibonacci Sequence itself.

A. Problem Statement

Integrating multithreading with optimized Fibonacci algorithms will provided a significant run-time boost to finding the n$^{\text{th}}$ term of the Fibonacci Sequence. We will be using the naive recursive algorithm as our benchmark algorithm to which compare our multithreaded approaches. We seek to provide further insight into the effectiveness of various non-concurrent algorithms for calculating the Fibonacci Sequence when compared to their concurrent counterparts, and additionally. provide insight for individuals wondering if the trade-off of implementing the required overhead for various concurrent algorithms for calculating the Fibonacci Sequence is worth the change in run-time.

II Motivation

The Fibonacci numbers appear often enough in nature that they are impossible to ignore. This means that mathematicians, engineers, and other professions frequently find themselves using Fibonacci numbers. The only issue with this is the Fibonacci sequence , while not strictly exponential, exhibits exponential-like growth between numbers. As such the larger the n the exponentially more computation power it takes to calculate F_n. Our hope is that by integrating multi-threading with some of the more common optimized Fibonacci algorithms we can further improve the overall run-time for larger values in the sequence, and in doing so, provide a way for algorithms relying on the Fibonacci Sequence to be improved as well.

III Implementation

The following is the breakdown of tasks for this project:

- Write the Fibonacci algorithm
  - Using naive approach
  - Using top-down dynamic programming (memoization recurssive)
  - Using bottom-up dynamic programming (iterative)
  - Using matrices
- Rewrite each Fibonacci algorithm with multi-threading
- Create a testing environment to evaluate performance based on a variety of benchmarks

A. Plan to Implement

The following is the plan of implementation for each algorithm:

The Naive Approach

B. Challenges

We anticipated the following problems to arise: overflow from large Fibonacci numbers,

Pros/cons top-down vs bottom-up approach No DP mutlithread approach because max number of threads approach is linear in nature

IV Testing

The following is the breakdown of how testing will work Record the time of each algorithm to generate n Fibonacci numbers

- Each test will output data in the following form

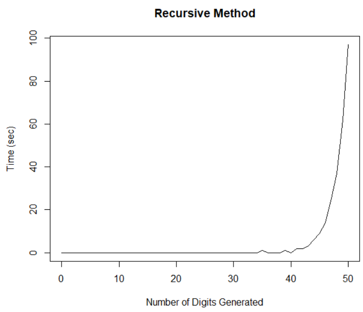
Number of Digits Generated	Time
n	x seconds

- Each test will be put through an R script to:
  - Generate a graph to visualize each individual test
  - Generate graphs that compare each method to the naive method (control)
  - Generate graphs that compare each method with their multi-threaded counterpart
  - Run a statistical analysis on whether implementing multi-threading had a notable impact on efficiency

Algorithm 1: Naive Fibonacci

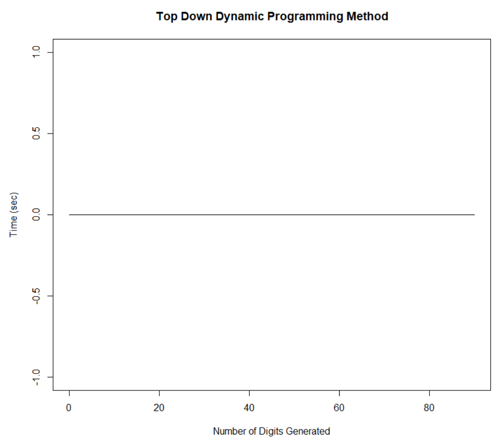
Data: $n \geq 0$
Result: F

V Evaluation



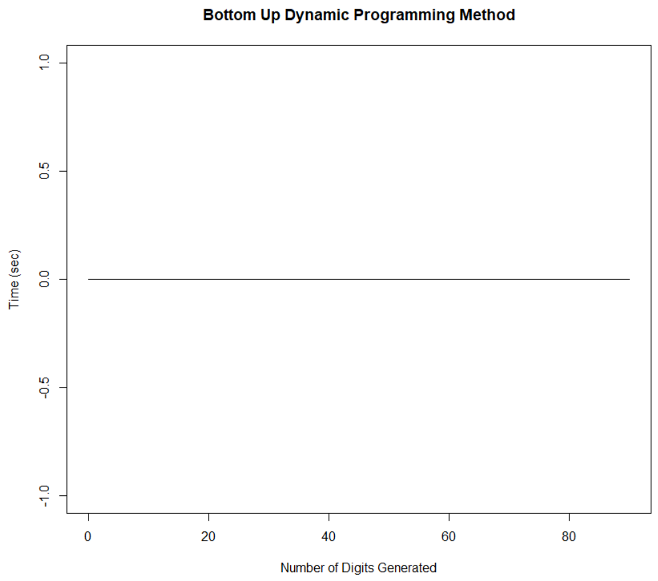
Algorithm 2: Top-Down Dynamic Programming Fibonacci

Data: $n \geq 0$
Result: F



Algorithm 3: Bottom-Up Dynamic Programming Fibonacci

Data: $n \geq 0$
Result: F



Algorithm 4: Fibonacci with Matrices

Data: $n \geq 0$
Result: F