General

This document outlines a multithreaded implementation of the Fibonacci sequence calculation using the fast doubling algorithm in C programming language. The program employs POSIX threads for concurrency and optimization techniques to enhance performance.

Usage

By default, the program calculates Fibonacci numbers using `fast_fibonacci` function, leveraging the fast doubling algorithm for efficient computation. To compile and run the program, execute the following commands, simply:

```
make n=10
```

A sequence of Fibonacci numbers up to the specified limit `n`(in this example, 10) will be displayed, along with the elapsed time for computation.

Implementation

The program begins by including necessary header files such as `stdio.h`, `stdlib.h`, `pthread.h`, and `time.h`. It declares a global array `shm` to store computed Fibonacci values, initialized with zeros. The `fast_fibonacci` function implements the fast doubling algorithm to calculate Fibonacci numbers efficiently. It utilizes memoization by storing previously computed values in the `shm` array to avoid redundant calculations.

A `runner` function is defined to be executed by each thread. It receives a parameter `n`, which represents the upper limit for Fibonacci calculation. Within the `runner` function, each thread computes Fibonacci numbers up to the given limit and stores the results in the `shm` array.

In the `main` function, the program accepts a command-line argument `n`, indicating the desired Fibonacci sequence limit. It creates a single pthread and passes the value of `n` to the `runner` function. After joining the pthread, the program calculates and prints Fibonacci numbers up to `n`, along with the elapsed time for computation.

Analysis

- The program utilizes the fast doubling algorithm for efficient computation of Fibonacci numbers, reducing time complexity to O(log n).
- The program has the limitation of calculating Fibonacci numbers up to a maximum value of 93 due to the constraints of the `unsigned long long` data type.

- Single-threaded and multithreaded implementations are compared to demonstrate the performance benefits of concurrency in Fibonacci sequence calculation. Below results are obtained by `fast_fibonacci` function.

n	Single-threaded	Double-threaded	Triple-threaded
10	0.000070s	0.000181s	0.000397s
20	0.000077s	0.000242s	0.000239s
30	0.000041s	0.000172s	0.000263s
40	0.000041s	0.000214s	0.000344s
50	0.000043s	0.000149s	0.000257s
60	0.000045s	0.000122s	0.000234s
70	0.000056s	0.000163s	0.000159s
80	0.000043s	0.000142s	0.000166s
90	0.000061s	0.000186s	0.000298s

The table above shows the execution times for calculating Fibonacci numbers up to n using different numbers of threads. As the number of threads increases, the computation time increases due to thread creation overhead and synchronization costs. In terms of large Fibonacci numbers, the number has no effect on the computation time seemingly. It is likely that the overhead of creating and managing multiple threads outweighs the benefits of parallelism for small computations.

- Additional comparisons using `traditional_fibonacci` function are provided in the source code for reference. The `traditional_fibonacci` function uses a naive recursive approach to calculate Fibonacci numbers. As the number of threads increases, the computation time also increases due to the reasons mentioned above. However, for large Fibonacci numbers, the computation time increases significantly compared to the fast doubling algorithm, underscoring the inefficiency of the naive recursive approach.

n	Single-threaded	Double-threaded	Triple-threaded
10	0.000086s	0.000208s	0.000345s
20	0.000066s	0.000298s	0.000377s
30	0.002808s	0.012438s	0.002892s
40	1.119209s	1.107290s	1.109266s