UNIVERSIDADE DE SÃO PAULO – USP
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO
DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO

RELATÓRIO
Projeto 3 - Fibonacci em $O \log_n$

Alunos: Adams Vietro Codignotto da Silva - 6791943
Guilherme da Silva Biondo - 8124267

São Carlos
2013

1 Introdução

O trabalho foi implementado utilizando as técnicas fornecidas na descrição do mesmo. Foi desenvolvido utilizando a linguagem C, no Linux Xubuntu 13.10 64bits. O arquivo contem um arquivo *Makefile* para compilação do programa, usando os comandos *make all2* para compilação inicial, *make all* para compilações futuras e *make run* para execução do mesmo.

2 Modelagem do problema

2.1 Matriz base de Fibonacci

Primeiramente foi necessário encontrar a matriz base para o cálculo da Sequência de Fibonacci. Vamos mostrar que a matriz $\left(\begin{smallmatrix}1 & 1\\ 1 & 0\end{smallmatrix}\right)$ é a matriz utilizada para o cálculo dos termos de Fibonacci. Temos inicialmente a sequência definida por $F_n = F_{n-1} + F_{n-2}$. Da combinação linear, podemos calcular F_2 como:

$$\begin{bmatrix}2\\1\end{bmatrix} = \begin{bmatrix}F_2\\F_1\end{bmatrix} = \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}F_1\\F_0\end{bmatrix} = \begin{bmatrix}1 & 1\\1 & 0\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix} = A\begin{bmatrix}1\\1\end{bmatrix}$$

Podemos calcular F_3 conforme a equação a seguir, denotando A = $\left(\begin{smallmatrix}1 & 1\\ 1 & 0\end{smallmatrix}\right)$:

$$\begin{bmatrix}3\\2\end{bmatrix} = \begin{bmatrix}F_3\\F_2\end{bmatrix} = A \times \begin{bmatrix}F_2\\F_1\end{bmatrix} = A \times A \times \begin{bmatrix}F_1\\F_0\end{bmatrix} = A^2\begin{bmatrix}1\\1\end{bmatrix}$$

Então generalizando, temos:

$$\begin{bmatrix}F_{n+1}\\F_n\end{bmatrix} = A^n \times \begin{bmatrix}1\\1\end{bmatrix}$$

2.2 Algoritmo de Strassen

Também utilizamos o Algoritmo de Strassen para efetuar a multiplicação de matrizes, uma vez que este também faz parte de métodos de Divisão e Conquista. Este algoritmo se resume em realizar a multiplicação de matrizes 2x2 em $O(n^2)$ ao invéz do usual $O(n^3)$. Dadas duas matrizes A=$\left(\begin{smallmatrix}a & b\\ c & d\end{smallmatrix}\right)$ e B=$\left(\begin{smallmatrix}e & f\\ g & h\end{smallmatrix}\right)$, temos que A×B pode ser resumido em:

$$\begin{array}{ll}
P_1 = a \times (f - h) & r = P_5 + P_4 - P_2 + P_6 \\
P_2 = (a + b) \times h & s = P_1 + P_2 \\
P_3 = (c + d) \times e & t = P_3 + P_4 \\
P_4 = d \times (g - e) & u = P_5 + P_1 - P_3 - P_7 \\
P_5 = (a + d) \times (e + h) & \\
P_6 = (b - d) \times (g + h) & \\
P_7 = (a - c) \times (e + f) &
\end{array}$$

$$A \times B = \begin{bmatrix}r & s\\t & u\end{bmatrix}$$

Este algoritmo utiliza 7 multiplicações e 18 adições/subtrações, resultando em $T(n) = 7T(n/2) + \Theta(n^2)$, uma vez que este divide as matrizes A e B em duas submatrizes $(n/2) \times (n/2)$.

2.3 Potenciação

Normalmente se calcula a^n em $O(n)$. Utilizando o algoritmo de divisão e conquista provido nas instruções do trabalho, temos que $T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\log_n)$. Note que, para facilitar a implementação, utilizamos a propriedade

$$(x^{n/2})^2 = x^{n/2} * x^{n/2} = (x^2)^{n/2} \tag{1}$$

3 Experimentos e Resultados

Durante a implementação do código, foi necessário o uso de *long long int*, uma vez que a entrada é de tamanho 10^{11}. Como utilizamos varias matrizes 2x2, o uso de memória tornou-se gigantesco, então todas as estruturas foram declaradas estaticamente. Deste modo evitamos o uso excessivo de memória ao alocarmos dinâmicamente as matrizes, pois com o uso de alocação dinâmica várias matrizes são criadas e usadas como parâmetro de retorno, assim não podendo ser desalocadas e gerando o uso excessivo de memória. A execução foi extremamente rápida, levando menos de 1 segundo para qualquer uma das entradas no intervalo definido.

4 Conclusões

A combinação de todos os algoritmos (Strassen ($O(n^2)$) e Potenciação em $O(Log_2 n)$) resultou em um programa de $\simeq O(Log_n)$. Fez-se a saída ser o resto da divisão de F_n por 10^6 ($F_n \text{mod} 10^6$) pelo fato de que ao realizarmos as multiplicações, iria ocorrer o overflow de variáveis nas matrizes. Então a cada iteração, realiza-se o mod 10^6 sobre cada multiplicação durante a execução do algoritmo de Strassen, uma vez que todos os casos sempre iniciam-se com $\left(\begin{smallmatrix} 1 & 1 \\ 1 & 0 \end{smallmatrix}\right)$.

O método utilizado no trabalho mostrou-se muito mais eficaz que sua versão iterativa ou recursiva, que possuem $O(n)$. Entretanto, este método só é mais eficiente pela forma com que calculamos a potênciação das matrizes, uma vez que se não utilizassemos a Potenciação em $O(Log_2 n)$ este método degeneraria para sua versão iterativa/recursiva (e ainda pior, pois utilizamos muito mais memória para os cálculos).