As diferentes técnicas de implementação paralela de algoritmos recursivos em C

Tiago M. Rohde, Luciano Destefani, Edilaine R. Ferrari, Rogério Martins

Departamento de Ciências Exatas e Engenharias - Universidade Regional do Noroeste do Estado do Rio Grande do Sul (UNIJUI)
Rua do Comércio, 3000, Bairro Universitário CEP 98700-000 - Ijuí – RS - Brasil

{tiago.mri, lucianodestefani, edinharevers}@hotmail.com, rogerio@aezon.com

Resumo. *Tendo em vista o atual crescimento do número de núcleos na área de arquitetura de processadores, este artigo busca apresentar uma nova técnica de cálculo da fórmula de Fibonacci. Serão apresentadas diversas técnicas de paralelização em threads, cada qual com suas vantagens e desvantagens, sendo realizada uma análise do tempo de processamento de cada algoritmo.*

Introdução

A programação paralela baseia-se na execução de diferentes partes, simultaneamente, de um mesmo processo. Essa execução simultânea é utilizada para otimizar o programa, diminuindo assim, o tempo total de processamento e obtendo maior desempenho, conforme Machado; Maia (2002).

De acordo com Menezes, et al (2011), vários dos algoritmos foram desenvolvidos para serem executados em apenas um dos núcleos do processador, perdendo poder de processamento. Com a técnica de divisão de blocos de código em threads a execução do Algoritmo acontece de forma conjunta, ou seja, um bloco é executado em um núcleo e os demais nos outros, fazendo com que o programa execute mais rápido.

Um dos problemas encontrados, conhecido como Overhead, acontece quando se cria um número de threads além da capacidade de gerenciamento de threads do processador, ou até mesmo quando se cria threads para executar tarefas muito pequenas, perde-se muito tempo na criação destas, conforme Maia (1998).

1. A importância do Uso de Threads

Atualmente uma das soluções encontradas para resolver a parte ociosa do processador é utilizar threads, estes são executados de forma assíncrona. Machado; Maia explicam esses blocos de código de forma simplificada, "um thread pode ser definido como uma sub-rotina de um programa que pode ser executada de forma assíncrona (...)", (2002, p.86). Os mesmos autores ponderam que o uso de threads não é paralelizado pelo Sistema Operacional e sim pelo programador, pois ainda não há um compilador que paralelize eficientemente um código, sendo assim faz-se necessário a interferência manual do programador, o qual deve analisar o controle de fluxo (MACHADO; MAIA, 2002).

1.1. Benefícios

Conforme escrevem Silberschatz; et al (2001, p.83) a utilização desta tecnologia trás diversos benefícios, dentre eles estão a capacidade de resposta ao usuário, economia de memória e a utilização de arquitetura de multiprocessamento.

Para medir o aumento de desempenho do processamento utiliza-se o cálculo chamado de Speedup que determina a relação existente entre o código executado em threads e seqüencial. Para Blume Speedup é "a medida que tem por objetivo determinar a relação existente entre o tempo dispensado para executar um algoritmo em um único processador (T1) e o tempo gasto para executá-lo em 'p' processadores (Tp). Speedup = T1/Tp", (2002, p. 60).

2. Fibonacci

Para demonstrar que a paralelização de processos, se bem implementada, possui um melhor desempenho usou-se o cálculo da seqüência de Fibonacci, já que ele tem a possibilidade de divisão do processo em duas partes concorrentes.

De acordo com o site Oracle (2011), a maioria das flores tem pétalas com o número referente a este cálculo, como as margaridas, que tem um número de pétalas referente ao cálculo dos números 9, 10 e 11.

O cálculo de Fibonacci inicia-se com os elementos 0 e 1 e a partir do terceiro é o valor da soma dos dois anteriores, conforme mostra a fórmula da Figura 01. Para demonstrar como o processo será dividido, a Figura 02 representa a criação de dois novos processos, denominados threads.

$$F(n)= \begin{cases} \text{se } n<=1, n \\ \text{senão } F(n-1)+ f(n-2) \end{cases}$$

Figura 01 – Fórmula de Fibonacci

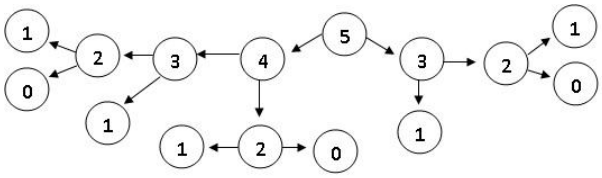

Figura 02 – Divisão do Processo

3. Formas de Implementação do Algoritmo

As formas de implementação que serão apresentadas são: recursivo, paralelo sem controle de criação e por fim, o paralelo que cria um determinado número de threads.

3.1 Fibonacci Seqüencial Recursivo:

No Algoritmo 01 a função demonstra o cálculo seqüencial recursivo, a seguir serão demonstrados alguns escopos de códigos.

```
unsigned int fib(unsigned int n)
{
return n < 2 ? n : fib(n-1)+fib(n-2);
}
```

Algoritmo 01 – Fibonacci Sequencial

```
typedef struct {
        unsigned int n;  // É o número que passa como parâmetro para o calculo.
        unsigned int r;  // É o número de retorno do calculo.
        unsigned int d;  // É o controlador da quantidade de threads na função do Fibonacci com um número
determinado de threads.
} param;
```

Algoritmo 02

Para haver comunicação entre os threads foi utilizada uma variável *struct*, conforme demonstrado no Algoritmo 02.

```
void *fib(void *arg)
 {
        if((*(param*)arg).n > 1)
        {
```

Algoritmo 03

```
        }
        else
        {
                (*(param*)arg).r = (*(param*)arg).n;
        }
        return (void*)0;
 }
```

Algoritmo 04

3.2 Fibonacci Paralelizado:

No algoritmo a seguir tem-se a função que paraleliza o código, ela cria dois threads, um para executar o n-1, e a outro para executar o n-2. Um dos problemas encontrados é o overhead, pois nessa implementação quanto maior o valor a ser calculado, mais threads precisarão ser criados podendo ocasionar o estouro da pilha do sistema.

```
Idem Algoritmo 03
                param p1 = {(*(param*)arg).n-1, 0}, p2 = {(*(param*)arg).n-2, 0};

                pthread_t t1, t2;
                pthread_create(&t1, NULL, fib, (void*)&p1);
                pthread_create(&t2, NULL, fib, (void*)&p2);

                pthread_join(t1, NULL);
                pthread_join(t2, NULL);
Idem Algoritmo 04
```

Algoritmo 05

3.3 Fibonacci Paralelizado com um número determinado de threads:

Quando é criado um novo thread tem-se a perda de tempo e também a possibilidade do overhead, por isso na implementação a seguir é utilizado "unsigned int d" para controlar o número de threads a ser criado.

```
Idem Algoritmo 03
                param p1 = {(*(param*)arg).n-1, 0, (*(param*)arg).d/2}, p2 = {(*(param*)arg).n-2, 0,
(*(param*)arg).d/2};
                if((*(param*)arg).d > 1)
                {
                        pthread_t t1, t2;
                        pthread_create(&t1, NULL, fib, (void*)&p1);
                        pthread_create(&t2, NULL, fib, (void*)&p2);
                        pthread_join(t1, NULL);
                        pthread_join(t2, NULL);
                }
                else
                {
                        p1.r = fib(p1.n);
                        p2.r = fib(p2.n);
                }
Idem Algoritmo 04
```

Algoritmo 06

4. CONCLUSÃO

A partir dos resultados encontrados percebe-se que a criação de threads, quando bem implementada, aumenta o poder de processamento. Ficou evidente que a quantidade ideal de threads deve ser proporcional ao número de núcleos do processador, caso contrário o overhead gerado na criação dos threads irá inviabilizar a criação destes.

A seguir encontra-se a tabela onde os primeiros resultados foram executados num processador Intel Core 2 Dou com dois núcleos, sem HT, modelo E4600, já os outros resultados foram obtidos num processador Intel Core i7 com 8 núcleos, utilizando tecnologia HT, modelo 2860KM.

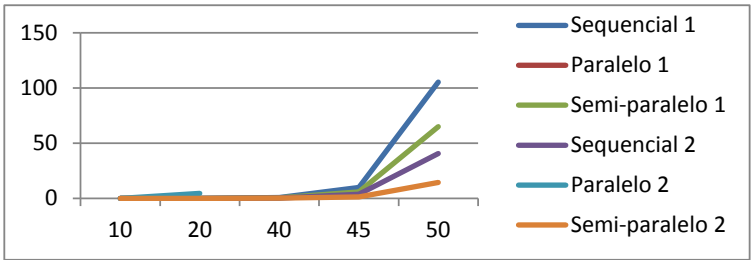

Figura 03 - Gráfico de Analise de Tempo de Execução

Tabela de Análise de Tempo de Execução					
	10	20	40	45	50
Seqüencial 1	0,002	0,002	0,853	9,731	105,113
Paralelo 1	0,008				
Semi-paralelo 1	0,002	0,002	0,522	5,914	64,92
Seqüencial 2	0,001	0,001	0,329	3,720	40,653
Paralelo 2	0,008	4,501			
Semi-paralelo 2	0,007	0,006	0,126	1,267	14,433

Figura 04 – Tabela de Analise de Tempo de Execução

5. REFERENCIAS BIBLIOGRÁFICAS

BLUME, Evandro. “Algoritmo de Organização Paralelo: Um modelo proposto e implementado”. Dissertação (Mestrado em Ciência da Computação) Universidade Federal de Santa Catarina, Florianópolis/SC, 2007.

Oracle - ThinkQuest Education Foundation. Disponível em http://library.thinkquest.org/27890/applications5.html. Acesso em 10/11/2011.

MACHADO, Francis Berger; MAIA, Luis Paulo. “Arquitetura de Sistemas Operacionais”. 3ª Edição, LCT, 2002.

MAIA, Luis Paulo. Monografia: “Multithread” - 18/05/1998. Instituto de Matemática da UFRJ/NCE.

MENEZES, Gustavo C.; LINS, André; CABRAL, Raquel da S.; NAKAMURA, Fabíola G. “Uma Abordagem Paralela para os Problemas de Cobertura e Conectividade em Redes de Sensores Sem Fio”. Disponível em http://homepages.dcc.ufmg.br/~alla/sbpo_05.pdf. Acesso em 14/11/ 2011.

SILBERSCHATZ, Abrahan; PETER, Galvin; GREG, Gagne. “Sistemas Operacionais: Conceitos e Aplicações”. Elsevier Editora Ltda, 2001.