Universidade do Vale do Itajaí
Ciências da Computação
Sistemas Operacionais
Prof. Felipe Viel

Bruno Pereira Freitas
Matheus da Costa Raimundo
Marco Antonio Corazza Jacinto

RELATÓRIO THREADS EM C/C++

ITAJAÍ
2022

Introdução

Este relatório se refere ao estudo do conteúdo, a implantação e testes de desempenho em multi-thread nas bibliotecas OpenMP e Pthreads. Comparando-as em alguns exemplos contra o Single Thread.

EXEMPLO 1 – MULTIPLICAÇÃO DE MATRIZES

Por início foi desenvolvido um código para realizar a multiplicação de matrizes feitas previamente, optamos por esta opção pois assim é possível comparar com mais precisão os resultados dentre as formas de distribuição de threads.

```
int mat_a[3][3] = {{1,3,2},{4,1,2},{9,2,3}};
int mat_b[3][3] = {{7,2,1},{8,9,1},{8,4,3}};
int mat_c[3][3] = {0,0,0,0,0,0,0,0,0};
```

Figura 1 Declaração das matrizes pré-definidas

```
// multiplicação de matrizes
for(i = 0; i < 3; i ++){
    for(j = 0; j < 3; j ++){
        for(k = 0; k < 3; k++){
            mat_c[i][j] = mat_c[i][j] + mat_a[i][k] * mat_b [k][j];
        }
    }
}
```

Figura 2 Execução da multiplicação

1.1 – **SINGLE THREAD**

Tivemos os resultados:

tentativa 1 = 0.000122 -max

tentativa 2 = 0.000096

tentativa 3 = 0.000083 -min

Neste método foi alcançada a média de 0.000100 segundos, utilizaremos o exemplo do Single Thread como base de comparação para os métodos de multi thread.

1.2 – MULTI THREAD PELO OPENMP

Utilizando o OpenMP é colocado:

```
#pragma omp parallel
```

E em seguida dentro de chaves o código que deseja ser feito o multi-thread, com isso a parte de criação e manutenção de threads é feita pelo compilador abstraindo essa parte do desenvolvedor.

Com a utilização da biblioteca <omp.h> foi possível conseguir os tempos:

tentativa 1 = 0.000080 -min

tentativa 2 = 0.000093

tentativa 3 = 0.000097 -max

Conseguindo a média de 0.000090 segundos que é 10% mais rápido que a versão single thread.

1.3 – MULTI THREAD PELO PTHREADS

Utilizando o Pthreads foram encontradas algumas dificuldades na hora de ajustar o código para conseguir utilizar da biblioteca, mas após refatorar grande parte do código e pesquisar sobre o funcionamento geral das funções foi possível realizar, embora sem ter um melhor resultado inferior até mesmo a versão em single thread.

Maioria do código precisou ser passado para funções para conseguir se encaixar no processo de criar as threads, a princípio foi feita a tentativa de paralelizar tudo, porém após alguns testes o foi visto que o código compilava, mas ao executar aparecia faltando dados ou fora de ordem, e foi aqui que percebemos que não daria para paralelizar tudo no código, pois enquanto uma thread executava uma parte necessária na “frente” outra ao mesmo tempo executava essa “frente” por isso foi necessário refatorar algumas vezes o código para esse exemplo do Pthread.

```
pthread_t t1, t2, t3;
int a1 = 1, a2 = 2, a3 = 3;


pthread_create(&t1, NULL, print_mat_a, (void *)(&a1));; //cria thread 1
pthread_create(&t2, NULL, print_mat_b, (void *)(&a2)); //cria thread 2
pthread_create(&t3, NULL, multiplica_matriz, (void *)(&a3)); //cria thread 3
```

```
pthread_join(t1,NULL);
pthread_join(t2,NULL);
pthread_join(t3,NULL);


print_mat_c();
```

Perceba que a impressão da matriz C não foi criada um thread. Utilizando essa biblioteca conseguimos o resultado:

tentativa 1 = 0.000197 -min

tentativa 2 = 0.000237 -max

tentativa 3 = 0.000233

Média = 0.000222

120% mais lento que a versão single thread

EXEMPLO 2 – VETORES

Neste segundo exemplo foi solicitado para criar um vetor de X números de casas e preenche-lo de forma sequencial ex. [1, 2, 3, 4,..]. Para utilizar como exemplo foi feito um vetor de 150.000 itens, obviamente preenchidos por um laço de repetição que será onde os threads serão feitas:

```
#define TAM 150000
```

```
void load (){
    for(int i = 0; i< TAM; i++){
        array[i] = i;
    }
}
```

Em single thread foram obtidos os resultados:

tentativa 1 = 0.000452 -min

tentativa 2 = 0.000581 -max

tentativa 3 = 0.000494

Alcançando a média de 0.000509 segundos.

2.1 – OpenMP

Utilizando o OpenMP foi simples, bastou colocar a sentença em volta da chamada da função:

```
#pragma omp parallel
{
    load();
}
```

Com este método alcançamos:

tentativa 1 = 0.000474 -max

tentativa 2 = 0.000434 -min

tentativa 3 = 0.000467

Conseguimos a média de 0.000458 segundos, ligeiramente mais rápido que em single thread.

2.2 – Pthreads

Embora mais complicado que o OpenMP neste exemplo foi mais simple afinal a parte difícil de coleta e teste já havia sido realizada em matrizes, de todo modo utilizamos a função de criar a thread, de chamar e de matar como da última vez:

```
int main(){

    time1 = (double) clock();
        time1 = time1 / CLOCKS_PER_SEC;

    pthread_t t1;
    int a1 = 1;

    pthread_create(&t1, NULL, load, (void *)(&a1)); //cria thread 1
    pthread_join(t1,NULL);
    pthread_exit(NULL);
```

Com isso obtemos os seguintes resultados:

tentativa 1 = 0.000578 -max

tentativa 2 = 0.000495

tentativa 3 = 0.000461 -min

Conseguindo assim a média de 0.000511 segundos, mais uma vez os mais lento dos 3.

CONCLUSÃO

Foi possível notar que em ambos os exemplos o OpenMP se saiu melhor por sua facilidade de implementação e desempenho, como nos exemplos os threads eram mais simples e em número menor acabou pendendo mais para o lado da biblioteca mais simplista com uma média de desempenho superior ao single thread de mais ou menos 11% se destacou diante dos 3 métodos. O procedimento de soma (exemplo 2) foi ligeiramente mais rápido na biblioteca OpenMP que executou de melhor forma, acredito que há casos em que o a configuração mais minuciosa e precisa possa ser necessária onde o Pthreads entraria de forma mais necessária para a dada operação.

Todos os códigos estão disponíveis dentro do arquivo ZIP