Universidade Estadual de Campinas – UNICAMP
Faculdade de Tecnologia – FT

Grupo: Deepin

André Sacilotto, RA: 231396
Renan Douglas Sousa Holanda, RA: 243782
Wellington Abilio Alves Silverio, RA:178667

Projeto Multithreads
Build Matrix

Sumário:

1. Introdução:

Na aula de Sistemas Operacionais nos foi apresentado o problema “Build Matrix” (Projeto 2) o qual iremos apresentar nossa solução para o problema. Utilizando a biblioteca POSIX Threads.
Fizemos um software para organizar arquivos de números e gravar a matriz gerada por estes, passando por argumento a quantidade de threads a serem executadas e os arquivos para ordenar e adicionar na matriz de saída.
Você encontrará soluções para entender sobre multi threads e qual comportamento está teve nos testes feitos.

2. Links

Link do GitHub:
https://github.com/wellingtonsilverio/so-projeto_2_multthreads (master)

Link do Vídeo:
https://drive.google.com/file/d/1htcQGiMLePJap5qUcMRtk8Of2OGjF4aw/view?usp=sharing

3. Descrição do problema

O programa deve ler os valores inteiros de vários arquivos e, de forma ordenada, armazená-los em um arquivo de saída no formato de matriz com I linhas (I sendo quantidade de arquivos) e J colunas (J sendo a maior quantidade de valores dentro dos arquivos). Com o excesso sendo preenchidas com 0.
O desempenho do programa 2, 4, 8 e 16 threads, com arquivos com diferentes quantidades de números inteiros.

4. Solução do problema

A solução:

O algoritmo usado para a solução desse problema visa usar as threads para realizar a ordenação dos valores dos arquivos utilizando o algoritmo “Bubble Sort” e também utilizar as threads para popular a matriz que será usada para escrever o arquivo. O algoritmo necessita que os argumentos sejam dispostos corretamente para ser executado com sucesso.

Algoritmo de Alto Nível:

1. Descubra o número de threads e arquivos através dos argumentos.
2. Adquire os nomes dos arquivos de entrada e saída que é passado nos argumentos.
3. Leia os valores inteiros de cada arquivo de entrada e enquanto faz isso descubra qual arquivo tem maior quantidade de itens.
4. Crie uma matriz final usando o número de arquivos como linha e a maior quantidade de valores entre os N arquivos adquiridos anteriormente.
5. Popule a matriz com valor 0(zeros).
6. Conte o tempo inicial.

7. Ordene em ordem crescente as matrizes adquiridas ao ler os arquivos de entrada e utilizando múltiplas threads.
7. Popule a matriz final utilizando as matrizes ordenadas e utilizando múltiplas threads.
8. Conte o tempo final.
9. Imprima o arquivo de saída usando a matriz final.
10. Mostra no console tempo entre o início e fim, e a quantidade de threads.

5. Instruções para Compilar

Compilar o projeto

$ gcc main.c -o main -lpthread

Executar o projeto

$ ./main 2 arq1.dat arq2.dat arq3.dat arq4.dat arq5.dat arq6.dat arq7.dat -o saida.dat
$ ./main 4 arq1.dat arq2.dat arq3.dat arq4.dat arq5.dat arq6.dat arq7.dat -o saida.dat
$ ./main 8 arq1.dat arq2.dat arq3.dat arq4.dat arq5.dat arq6.dat arq7.dat -o saida.dat
$ ./main 16 arq1.dat arq2.dat arq3.dat arq4.dat arq5.dat arq6.dat arq7.dat -o saida.dat

6. Gráficos de Compilação

Número de Threads	2	4	8	16	32
Resultado 1	37 ms	17 ms	28 ms	25 ms	42 ms
Resultado 2	21 ms	58 ms	30 ms	21 ms	36 ms
Resultado 3	52 ms	13 ms	16 ms	21 ms	18 ms
Resultado 4	29 m	15 ms	24 ms	15 ms	31 ms
Resultado 5	43 ms	18 ms	23 ms	23 ms	34 ms
Média	36,4 ms	24,2 ms	24,2 ms	21 ms	32,2 ms
Mediana	37 ms	17 ms	24 ms	21 ms	34 ms

Tabela 1: Número de threads e resultados obtidos nos 5 testes com suas média e mediana.

Gráfico de todos os testes usando respectivamente (2/4/8/16/32) threads, mostrando a média e mediana das execuções do softwares no testes, percebemos que com mais threads a quantidade de tempo diminui até atingir a quantidade máxima de threads do computador, onde começa a demorar mais o tempo de execução

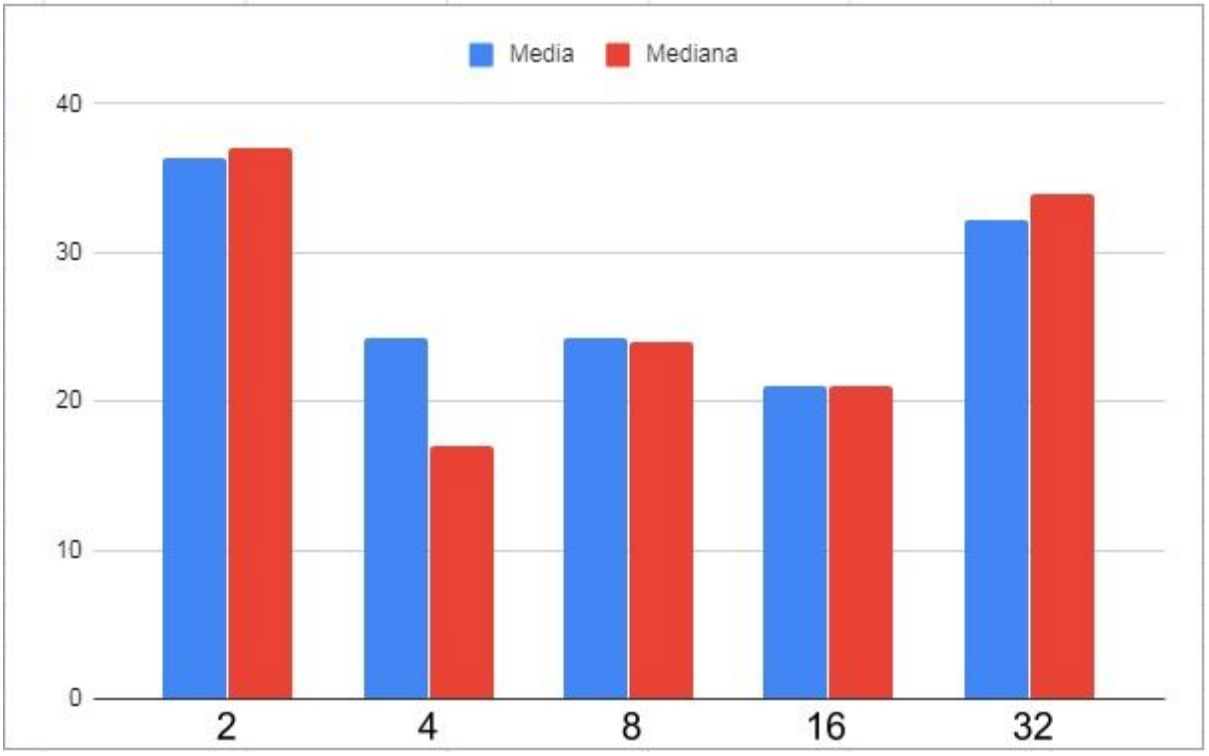

Gráfico 1: Tempo médio em ms e tempo mediano em ms da execução do software (eixo y) pela quantidade de threads (eixo x)

Gráfico dos 5 testes feitos em cada quantidade de thread (2/4/8/16/32), assim verificando a oscilação presente no tempo de execução.

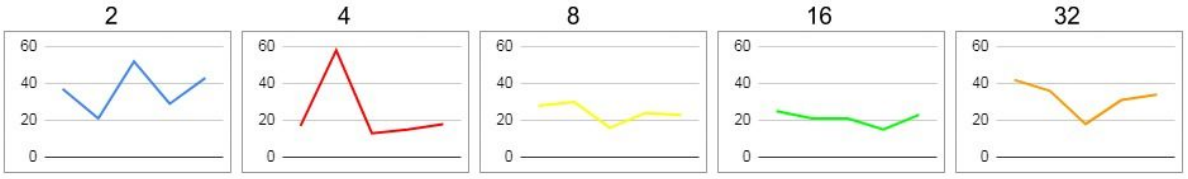

Gráfico 2: Tempo de execução em ms (eixo y) nos 5 testes do software

7. Conclusões

Com a execução de poucas threads vemos uma oscilação no tempo de execução maior e uma demora maior, como por exemplo com 2 threads a variação vai de 21 ms a 52 ms, já com 16 threads (quantidade de threads virtual do hardware que estava utilizando) teve pouco oscilação, fica entre 21 ms e 25 ms. Já quando utilizamos uma quantidade maior do que a quantidade de threads do nosso equipamento, a uma maior oscilação e um tempo maior para executar a tarefa (com 32 threads fica entre 18 ms a 42 ms).

E assim podemos, concluir que o uso de uma quantidade de threads compatível com o hardware que está em uso dará uma melhor performance para o software, caso não conhecemos a quantidade de threads que o computador suporta, devemos então colocar uma quantidade de threads média do mercado (de 4 a 16 threads) e caso isso não seja viável, devemos colocar uma quantidade pequena de threads, 1 ou 2 threads para que o programa não perca performance em computadores com 1 ou 2 threads.