Sistemas Operacionais

Multicat integer sort

Mr. Robot

Victor Gomes Sampaio | 225133

14 de Novembro de 2018

Objetivo

Este projeto visa a criação de um programa que utilize múltiplas threads para ler números inteiros de vários ($ñ \geq 1$) arquivos com valores inteiros em diferentes quantidades, ordenar todos esses dados e concatená-los em um único arquivo resultante. realizando a análise do desempenho desse programa com 2, 4, 8 e 16 threads.

Desenvolvimento

O código foi desenvolvido de maneira a utilizar as threads para ordenar pequenas partes de um vetor. Cada thread, fica responsável por ordenar uma fração de um vetor de inteiros, após todas as threads terminarem seu serviço, um novo ciclo de ordenação é iniciado, agora com a metade do número de threads realizando a operação de ordenação. Isso acontece até sobrar uma única thread, que percorrerá o vetor do início ao fim ordenando tudo o que ainda não estiver ordenado. Dessa maneira, quando a última thread percorre o vetor inteiro ordenando-o, grande parte dele já foi ordenado por outras threads, melhorando o desempenho da ordenação.

Instruções

Para executar o programa, é necessário ter o gcc instalado no seu sistema. Para isso, abra o terminal linux e digite (para sistemas Debian/Ubuntu):

```
$ sudo apt install gcc
```

Depois disso, compile o programa com o comando:

```
$ make
```

Por fim, execute o programa:

```
$ ./multicat 16 arq1.in arq.out
```

onde **16** é a quantidade de threads a ser utilizada, **arq1.in** é o arquivo de entrada que será lido e ordenado (podem existir mais de 1), e finalmente, **arq.out** é o arquivo de saída, que receberá os valores já ordenados.

Algoritmo em alto nível:

- Leitura de arquivos

1) Aloca um vetor para o arquivo lido;
2) Aloca o vetor principal da aplicação;
3) Carrega o vetor do arquivo com os inteiros lidos dentro do respectivo arquivo;
4) Carrega o vetor principal com todos os elementos de todos os vetores de todos os arquivos;

- Ordenação

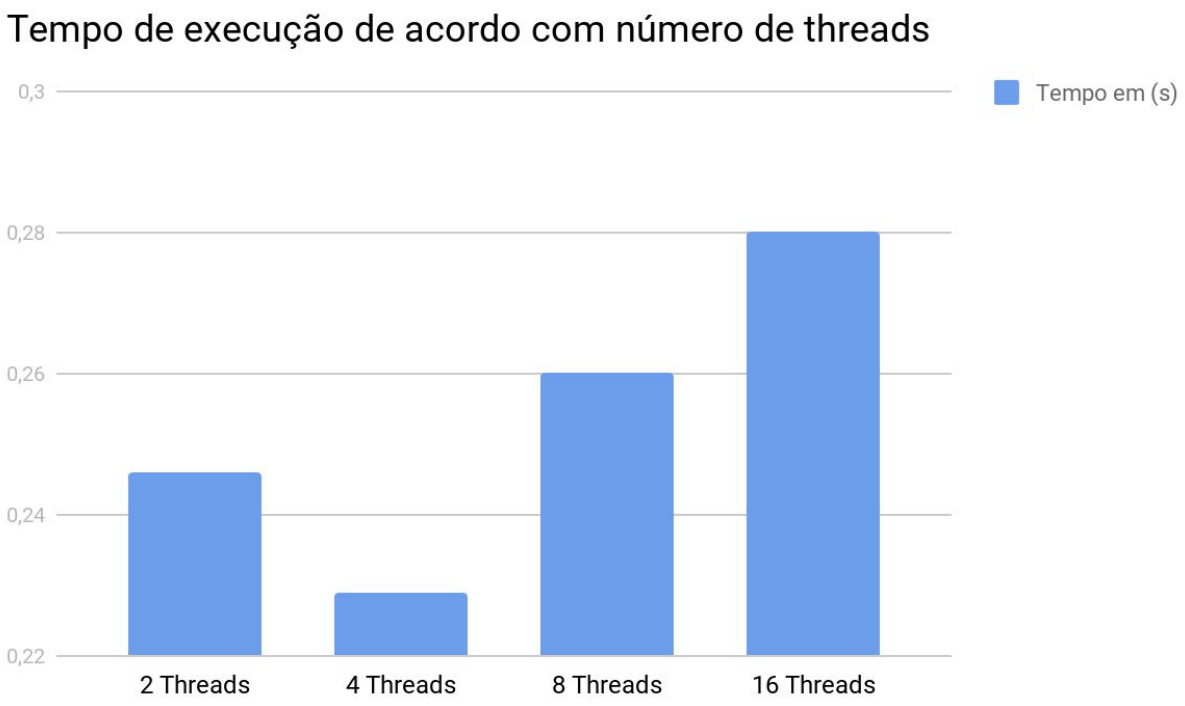
1) Declara a quantidade de threads recebida pela linha de comando;
2) Divide o tamanho máximo do vetor pela quantidade de threads, para obter o trecho que cada thread irá ordenar;
3) Após todos as threads terminarem, o “ciclo” de ordenação é finalizado;
4) Inicia-se um novo “ciclo” de ordenação com a metade do número de threads do “ciclo” anterior;
5) Repete o passo “4)” até restar somente uma thread;
6) Última thread percorre o vetor inteiro ordenando o que resta;

- Impressão

1) Imprime o vetor principal no arquivo de saída recebido por linha de comando.

Resultados

Os testes foram executados em um computador com processador *AMD^{tm} Ryzen 5 1400 3.2GHz 4 cores/8 Threads* rodando Linux Mint 19. O programa recebeu 10 arquivos com 100.000 (cem mil) inteiros cada, totalizando 1.000.000 (milhão) de inteiros. Os resultados estão demonstrados abaixo:



Tempos de execução (em média)

- 2 Threads: 0.246192s
- 4 Threads: 0.229061s
- 8 Threads: 0.267370s
- 16 Threads: 0.277476s

Discussão:

Fica explícito analisando os resultados que a diferença, nesse caso, do uso de threads foi bastante sútil. O algoritmo do programa funciona bem em teoria, já que permite que cada thread ordene porções pequenas do vetor, com a intenção de reduzir o tempo de ordenação. Na prática, mostrou-se que a diferença foi mínima, já que apesar da ordenação de pequenos vetores realmente acelerar o processo, a ordenação é feita várias vezes, fazendo com que a diferença de performance não seja tão significativa.

Links importantes

- Repositório GitHub:

  https://github.com/VictorGom3s/ProjetoSOVictorGomes