Universidade Estadual de Campinas

Faculdade de Tecnologia - FT

Ordenação com Threads em C

Prof. André Leon S. Gradvohl

Realizado pelos discentes:

Izabella Julia dos Santos, RA 169048

Gabriel Villar Scalese, RA 174973

Sumário

Introdução

Neste relatório estão descritos os diversos experimentos realizados com o código de ordenação usando múltiplas threads. Com a implementação em C, a avaliação de eficiência no processamento foi feita, comparando o desempenho de diferentes configurações de threads (2, 4 e 8 threads).

A análise dos resultados apresentados mostra os efeitos da paralelização de tarefas e como isso pode melhorar o tempo de execução do algoritmo, destacando os benefícios e limitações do modelo multithreaded aplicado a este tipo de processo.

Descrição do problema

A proposta do projeto é implementar um código funcional em C que realiza a ordenação de números inteiros presentes em diversos arquivos de entrada, utilizando múltiplas threads de processamento. E, ao final de todo o processo, alocar todos esses números ordenados em um único arquivo de saída.

O desenvolvimento contou com diversos recursos. Primeiro, foi criado um repositório no GitHub, e todo o versionamento foi feito por meio do Git. Em seguida, para facilitar a compilação, foi desenvolvido um script Makefile, automatizando o processo. Em relação ao código, um ponto relevante é a utilização do algoritmo de ordenação quicksort, adicionado como método de ordenação principal do projeto.

Ao final, como avaliação de desempenho, foram realizados experimentos para analisar a eficiência com o uso do multithreading. Todos os resultados se encontram nas próximas seções deste relatório, apresentando a resolução desse problema de codificação.

Instruções para execução do projeto

Ao seguir a sequência de passos completa descrita abaixo, será possível baixar, compilar e executar o código. Ela garante o funcionamento correto do programa e quase todas as etapas são obrigatórias. A geração de arquivos é a única parte opcional e pode ser realizada separadamente, fora deste processo.

Instalação

```
# Clone o repositório
git  clone
https://github.com/GabrielVScalese/multithread-sort.git
# Navegue até o diretório do projeto
cd  multithread-sort
```

Gerar arquivos de entrada

Assim como comentado, a geração de arquivos pelo projeto é uma parte opcional. Entretanto, é possível utilizar o script *numbers_generator.py* para criar os arquivos, localizado na pasta *utils*. Com esse script, basta escolher o nome e a quantidade de números que serão gerados, criando um arquivo de texto (salvo na raiz do projeto).

Compilação

Compile o código utilizando o makefile. É necessário estar no mesmo diretório no qual esse arquivo se encontra, ou seja, na raiz do projeto.

```
# Comando de compilação
make
```

Execução

Após a compilação, execute o programa utilizando a seguinte sequência de argumentos:

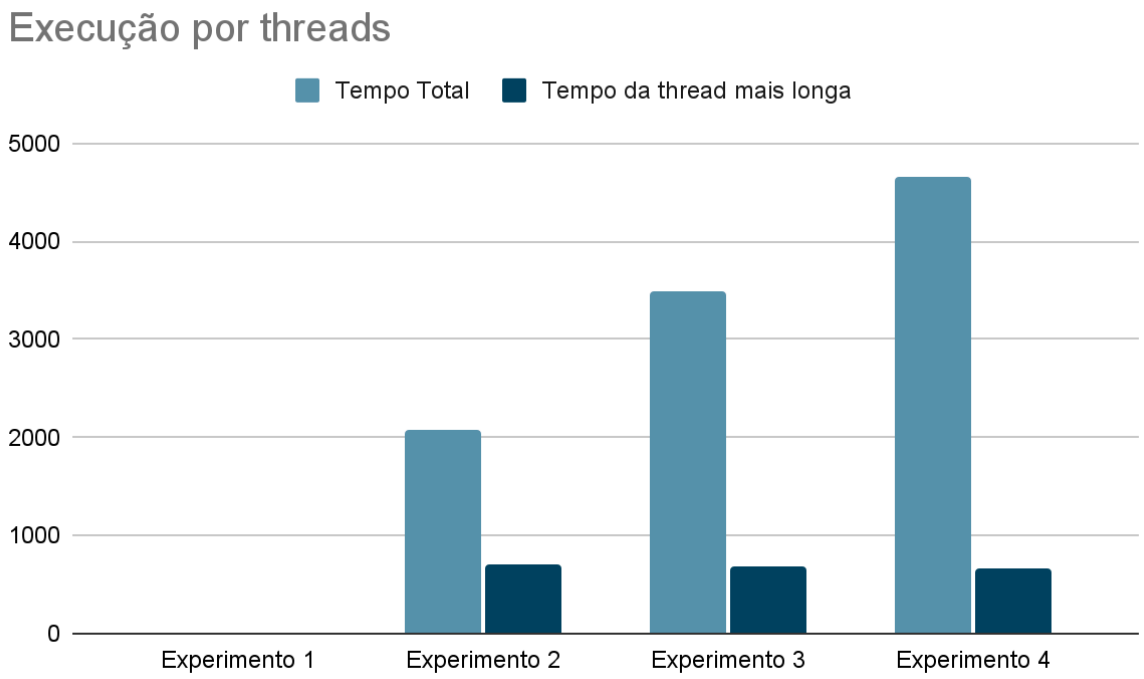
```
# Comando para a execução
./mergesort 2 arq1.txt -o saida.txt
```

Resultados

Para a realização dos experimentos, foram utilizados até 10 arquivos, cada um contendo mil números variando de 1 a 100. Os testes foram feitos para quatro cenários, descritos abaixo.

Gráficos

A seguir, estão tanto os gráficos como as saídas de terminal após a realização de cada experimento.



1. **Experimentos com 1 thread**

Em relação aos experimentos com 1 thread, na própria definição do trabalho está explícito que o programa deve funcionar para apenas 2, 4 e 8 threads. Por esse motivo, qualquer número diferente deles encerraria o programa, pois a capacidade de threads estaria incorreta.

```
izabella@Izabella:~/Desktop/multithread-sort$ ./mergesort 1 arq1.txt arq2.
txt arq3.txt arq4.txt arq5.txt arq6.txt arq7.txt arq8.txt arq9.txt arq10.t
xt -o saida.txt
Capacidade de threads incorreta!
```

Imagem 1: saída de erro do programa ao inserir 1 thread

O programa deve funcionar para T *threads*, onde T pode ser 2, 4 ou 8 *threads*. A quantidade de *threads* que o programa utilizará nunca deverá ultrapassar o valor de T. O programa deve usar a máxima quantidade de *threads* possível.

Imagem 2: descrição do trabalho em relação a threads

2. Experimentos com 2 threads

```
izabella@Izabella:~/Desktop/multithread-sort$ ./mergesort 2 arq1.txt arq2.txt arq3.txt a
rq4.txt arq5.txt arq6.txt arq7.txt
arq8.txt arq9.txt arq10.txt -o saida.txt
Tempo de execucao do Thread 0: 0.000709 segundos.
Tempo de execucao do Thread 1: 0.000590 segundos.
Tempo total de execucao: 0.002080 segundos.
```

3. Experimentos com 4 threads

```
izabella@Izabella:~/Desktop/multithread-sort$ ./mergesort 4 arq1.txt arq2.txt arq3.txt a
rq4.txt arq5.txt arq6.txt arq7.txt
arq8.txt arq9.txt arq10.txt -o saida.txt
Tempo de execucao do Thread 1: 0.000604 segundos.
Tempo de execucao do Thread 2: 0.000286 segundos.
Tempo de execucao do Thread 0: 0.000676 segundos.
Tempo de execucao do Thread 3: 0.000206 segundos.
Tempo total de execucao: 0.003485 segundos.
```

4. Experimentos com 8 threads

```
izabella@Izabella:~/Desktop/multithread-sort$ ./mergesort 8 arq1.txt arq2.txt arq3.txt a
rq4.txt arq5.txt arq6.txt arq7.txt arq8.txt arq9.txt arq10.txt -o saida.txt
Tempo de execucao do Thread 0: 0.000646 segundos.
Tempo de execucao do Thread 3: 0.000223 segundos.
Tempo de execucao do Thread 2: 0.000094 segundos.
Tempo de execucao do Thread 1: 0.000657 segundos.
Tempo de execucao do Thread 4: 0.000129 segundos.
Tempo de execucao do Thread 5: 0.000117 segundos.
Tempo de execucao do Thread 6: 0.000219 segundos.
Tempo de execucao do Thread 7: 0.000170 segundos.
Tempo total de execucao: 0.004662 segundos.
```

Análise dos resultados

É possível notar que o maior número de threads implica num maior tempo de processamento total. Isso ocorre devido ao fato de a criação de threads gerar um custo de tempo. No entanto, esse aumento de threads provoca uma redução no tempo de processamento individual das threads, pois cada uma delas trabalhará com menos dados.

Conclusão

Como conclusão, é possível ver, através dos resultados apresentados, que a utilização de threads pode melhorar o desempenho do código em determinadas situações, quando não há um excesso do uso delas.

Por fim, é importante ressaltar que esse código serviu para a aprendizagem de diversos conceitos, colocados em prática. Eles podem ser listados como as threads, compilação pelo makefile, alocação dinâmica, entre muitos outros.

Referências Bibliográficas

QUICK Sort. Quick Sort, [*s. l.*], 2024. Disponível em: https://www.geeksforgeeks.org/quick-sort-algorithm. Acesso em: 13 ago. 2024.