ALGORITMO DE ORDENAÇÃO HEAPSORT UTILIZANDO A API OPENMP

Rodrigo Curvêllo[1], Mathias Artur Schulz[2]
[1] Professor do Instituto Federal Catarinense – Campus Rio do Sul
[2] Estudante da 8ª fase do curso de Ciência da Computação do Instituto Federal Catarinense – Campus Rio do Sul
rodrigo.curvello@ifc.edu.br, mathiasschulz34@gmail.com

Resumo. *Este artigo apresenta diversos testes realizados com o algoritmo de ordenação Heapsort desenvolvido na linguagem de programação C utilizando uma versão serial do algoritmo e outra versão paralela do algoritmo por meio da API OpenMP. Para os testes são apresentados os tempos de execução, gráficos dos resultados e os cálculos de speedup.*
Palavras-chave: *Heapsort; OpenMP; Paralelismo.*

1. Introdução

Segundo Gulo (2015) com o passar do tempo, cada vez mais o desenvolvimento científico necessita das simulações numéricas resultados cada vez mais confiáveis e próximos da realidade. Entretanto, para alcançar esses resultados é necessário um processamento de alto desempenho que, em geral, é atendido por equipamentos e sistemas computacionais de custo elevado (GULO, 2015).

Dessa forma, a popularização das GPUs (Graphics Processing Units) e a aplicação de técnicas consolidadas de programação paralela estão permitindo que diversas áreas, como a área de processamento e análise de imagem obtenham avanços e ótimos resultados sem a necessidade de grandes investimentos financeiros (GULO, 2015).

A partir das informações acima, este short paper possui como objetivo realizar testes e comparações da utilização do algoritmo Heapsort de forma serial e também de forma paralela com o auxílio da API OpenMP. Sendo assim, para os resultados serão apresentados os tempos de execução obtidos, gráficos e análise de desempenho a partir do cálculo de speedup.

2. Fundamentação Teórica

Neste capítulo, são abordados conceitos relacionados à área de programação paralela, no qual são fundamentais para a realização e clareza deste trabalho.

2.1 Heapsort

Segundo Carlsson (2021) o HeapSort é um algoritmo de ordenação apresentado por Williams no ano de 1964, no qual o tempo do pior caso possível é O (n log n). Depois da apresentação do algoritmo, ainda no mesmo ano, melhorias no número de comparações para classificar foram apresentadas por Floyd e depois Carlsson também apresentou melhorias no algoritmo.

A ideia do algoritmo Heapsort é considerar os elementos em um array como nós em uma árvore binária completa (CARLSSON, 2021).

De acordo com Farias (2014), o Heapsort utiliza uma estrutura de dados chamada heap binário (árvore binária mantida na forma de um vetor) para ordenar os elementos à medida que os insere na estrutura. Dessa forma, quando todos os elementos estiverem inseridos, eles podem ser sucessivamente removidos da raiz da heap, na ordem desejada.

Para uma ordenação crescente deve ser construído um heap máximo, no qual o maior elemento fica na raiz e uma ordenação decrescente, deve ser construído um heap mínimo, no qual o menor elemento fica na raiz (FARIAS, 2014).

2.2 OpenMP

Segundo Sena e Costa (2008), o OpenMP consiste em uma interface de programação (API) e um conjunto de diretivas que permite a criação de programas paralelos com compartilhamento de memória através da implementação automática e otimizada de um conjunto de threads.

O uso do OpenMP proporciona algumas vantagens, como: Na maioria dos casos, são feitas poucas alterações no código serial existente; Possui uma robusta estrutura para suporte à programação paralela; Fácil compreensão e uso das diretivas; Suporte a paralelismo aninhado; Possibilita o ajuste dinâmico do número de threads (SENA e COSTA, 2008).

2.3 Speedup

Ao resolvermos um problema num sistema paralelo, o principal interesse é saber o ganho que teremos sobre a implementação serial deste problema. Dessa forma, uma das principais métricas para avaliação de desempenho de aplicações paralelas é o Speedup.

O Speedup representa o ganho de velocidade de processamento de uma aplicação quando executada com ‘n’ processadores. Dessa forma, quanto maior o Speedup, mais rápido se encontra o código paralelo.

3. Desenvolvimento

Este projeto foi baseado em um projeto já existente disponível no GitHub a partir do link: https://github.com/felipetenfen/Heapsort.

Dessa forma, foram realizados alguns ajustes visuais no código e também foram realizados alguns ajustes para rodar no Linux.

Os tempos de execução já estavam implementados, entretanto no código disponível não havia nenhum cálculo de speedup ou de geração dos gráficos, dessa forma essas duas rotinas foram implementadas no código.

4. Resultados

Todos os testes e resultados apresentados abaixo foram realizados em um computador com os seguintes requisitos:

- Sistema Operacional: Linux Mint 20.1 Cinnamon;
- Versão Cinnamon: 4.8.6;
- Linux Kernel: 5.4.0-67-generic;
- Processador: AMD Ryzen 5 3500X 6-Core Processor;
- Memória RAM: Team Group T-Force Vulcan Pichau 16GB (1x16) DDR4 3600MHz;
- Placa Mãe: Prime B450M Gaming/BR (ASUSTeK Computer INC.);
- Placa de vídeo: Radeon RX 580 Series;
- Versão GCC: 9.3.0 (Ubuntu 9.3.0-17 ubuntu1~20.04).

Todos os dados que serão apresentados a seguir, foram obtidos a partir dos testes realizados através da implementação do algoritmo de ordenação Heapsort, na versão serial e paralela.

Utilizou-se dois tipos de vetores de números inteiros, vetor aleatório e vetor ordenado inverso, com tamanhos definidos em: 10.000, 100.000, 1.000.0000 e 10.000.000.

Para os dois tipos de vetores, foram realizados testes utilizando os seguintes números de threads 2, 4, 8, 16, 32, 64, 128 e 256.

Dessa forma, para o vetor de números inteiros com valores aleatórios o seguinte resultado foi obtido.

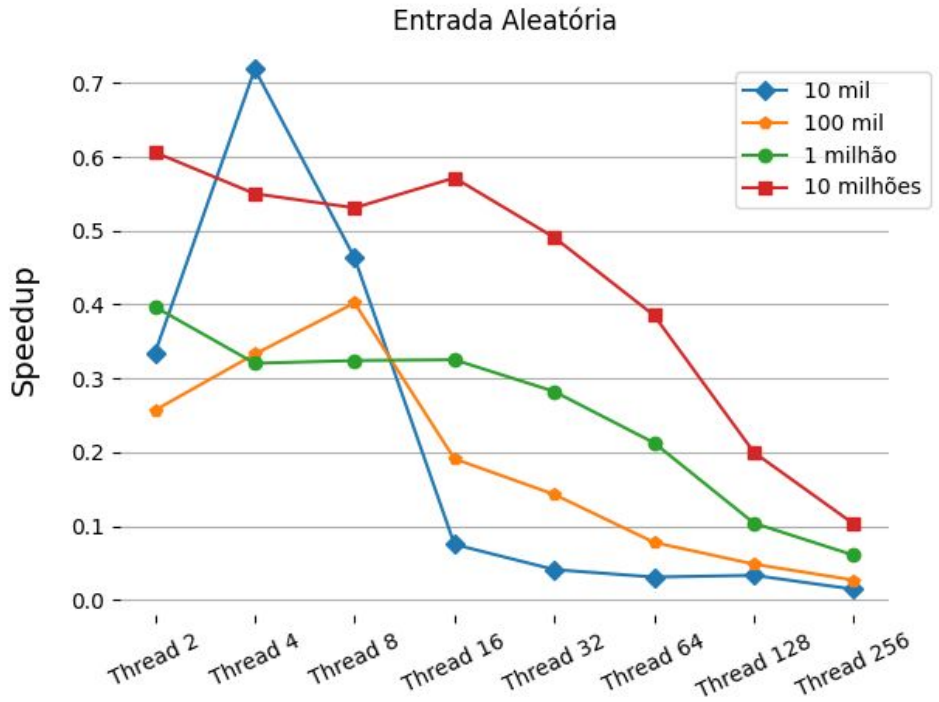

Figura 1: Entrada aleatória

Fonte: Elaborado pelo autor (2021)

Para o vetor de números inteiros com valores ordenados da forma inversa o seguinte resultado foi obtido.

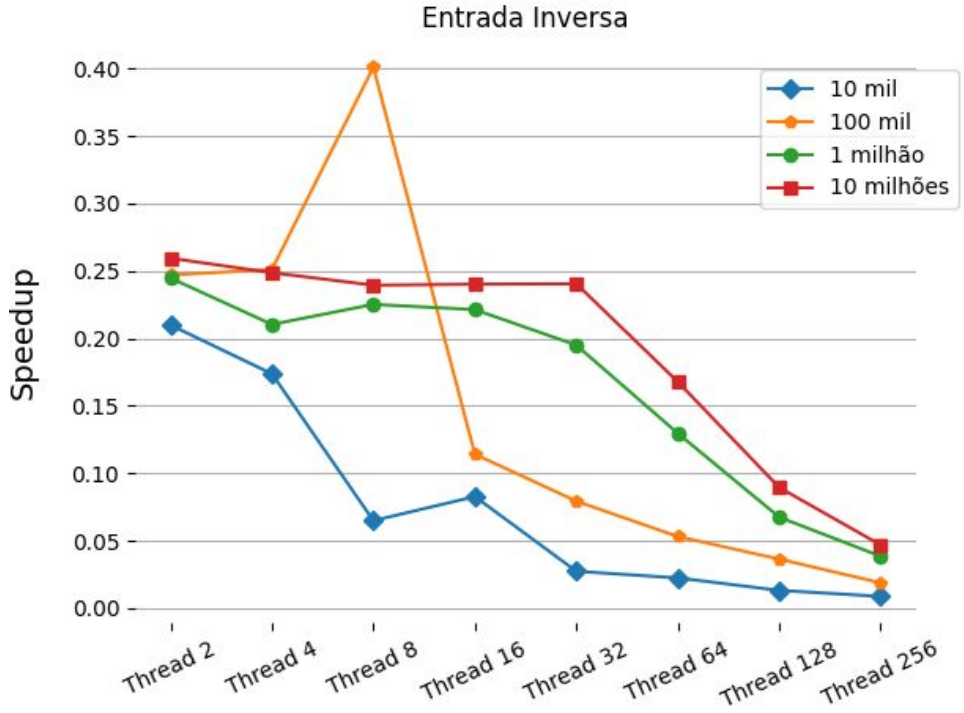

Figura 2: Entrada inversa

Fonte: Elaborado pelo autor (2021)

Como pode ser visualizado nas duas figuras acima, ambas possuem o '*speedup*' que tende a piorar de acordo com o aumento do número de threads. Dessa forma, independente da ordenação dos valores no array, o resultado tende a ser melhor com um número menor de threads.

5. Considerações finais

Através do estudo de caso realizado neste trabalho, pode-se verificar que é possível obter benefícios e vantagens na paralelização do algoritmo de ordenação Heapsort. Os cálculos de desempenho, mostraram que os melhores resultados se encontram em um menor número de threads, entre 2 e 32 threads. Dessa forma, aumentando o número de threads para mais de 32 o resultado tende a piorar.

Todo o código utilizado para construção deste projeto está disponível no github pelo link https://github.com/mathiasarturschulz/heapsort.

Referências

CARLSSON, Svante. **AVERAGE-CASE RESULTS ON HEAPSORT**. Disponível em: https://link.springer.com/content/pdf/10.1007/BF01937350.pdf. Acesso em: 14 mar. 2021.

FARIAS, Professor Ricardo. **Estrutura de Dados e Algoritmos**. 2014. Disponível em: https://www.cos.ufrj.br/~rfarias/cos121/aula_09.html. Acesso em: 14 mar. 2021.

GULO, Carlos Alex S. J.. Técnicas de Computação de Alto Desempenho para o Processamento e Análise Eficiente de Imagens Complexas. 2015. Disponível em: <https://web.fe.up.pt/~tavares/downloads/publications/relatorios/Projeto-Tese-17Carlos Gulo.pdf>. Acesso em: 08 abr. 2019.

SENA, M. C. R; COSTA, J. A. C. Tutorial OpenMP C/C++. Ed 01. Maceió: Sun Microsystems, 2008.