13º Congresso de Inovação, Ciência e Tecnologia do IFSP - 2022

ESTUDO DE PROGRAMAÇÃO PARALELA UTILIZANDO AS BIBLIOTECAS OPENMP E OPEN MPI.

EDER L. S. FILHO[1], VINÍCIUS H. NASCIMENTO[2], RENATO C. BARROS[3]

[1] Graduando em Engenharia de Computação, Bolsista PIBIFSP, IFSP, Campus Birigui, eder.filho@ifsp.edu.br.
[2] Graduando em Engenharia de Computação, IFSP, Campus Birigui, vinicius.henrique@ifsp.edu.br.
[3] Doutor em Agronomia, IFSP, Campus Birigui, rbarros@ifsp.edu.br.
Área de conhecimento (Tabela CNPq): 1.03.01.01-1 Computabilidade e Modelos de Computação

RESUMO: Esta pesquisa tem como objetivo aprofundar os conceitos de programação paralela, permeando assuntos como bibliotecas, funções, métodos e algoritmos comumente utilizados, e posteriormente desenvolver um problema a ser solucionado paralelamente. A metodologia adotada busca, além da compreensão teórica da programação paralela, apresentar técnicas e desenvolver dois algoritmos como exemplo prático da mesma, sendo eles baseados na regra do trapézio e utilizando as bibliotecas OpenMP e Open MPI e comparando-as respectivamente. Os resultados obtidos não puderam ser conclusivos quanto ao desempenho das bibliotecas, porém apresentaram resultados coerentes para soluções em sistemas paralelos, ou seja, concluíram o objetivo de realizar os cálculos propostos em menor tempo possível. Um dos principais obstáculos para a comparação do desempenho entre as bibliotecas se deu ao ambiente virtualizado de testes, onde favoreceu parcialmente a OpenMP devido a sua estrutura ser baseada em um hardware único, diferente da Open MPI que necessita de mais de um hardware para sua execução. A proposta inicial tinha como objetivo específico desenvolver uma interface gráfica para manipulação das bibliotecas, porém devido ao curto tempo de desenvolvimento não foi possível realizar.

PALAVRAS-CHAVE: paralelismo; algoritmo; regra do trapézio; cluster beowulf.

PARALLEL PROGRAMMING STUDY USING THE OPENMP AND OPEN MPI LIBRARIES.

ABSTRACT: This research aims to delve into concepts of parallel programming permeating subjects such as libraries, functions, methods and commonly used algorithms, and later to develop a problem to be solved in parallel. The adopted methodology seeks, in addition to the theoretical understanding of parallel programming, to present techniques and develop two algorithms as a practical example of it, being them based on the trapeze rule and using the OpenMP and Open MPI libraries and comparing them respectively. The results obtained could not be conclusive regarding the performance of the libraries, but they presented coherent results for solutions in parallel systems, that is, they concluded the objective of performing the proposed calculations in the shortest possible time. One of the main obstacles for comparing the performance between the libraries was the virtualized test environment, which partially favored OpenMP due to its structure being based on a single hardware, unlike Open MPI, which requires more than one hardware for its execution. . The initial proposal had the specific objective of developing a graphical interface for handling the libraries, but due to the short development time it was not possible to perform.

KEYWORDS: parallelism; algorithm; trapeze rule; beowulf cluster.

INTRODUÇÃO

Segundo Pitanga (2002) com a crescente demanda de processamento dos computadores nas últimas décadas, se encadeou uma evolução constante de softwares e hardwares que pudessem processar grandes volumes de dados e mais dados em pouco tempo. Porém, com os gargalos da

inovação física dos computadores, os softwares começaram a ganhar mais espaço, de tal modo que a programação paralela tornou-se o front da maximização de processamento.

As bibliotecas OpenMP e Open MPI são a vanguarda do paralelismo, atuando com grande valor na resolução de problemas computacionais complexos.

A OpenMP (Interface de Multi-Processamento) funciona como uma API, atuando estritamente na distribuição de processamento entre threads, criando assim uma distribuição multithreads de um sistema, realizando o escalonamento de serviços e o gerenciamento de acessos e requisitos da memória do computador. Já a Open MPI (Interface de Passagem de Mensagens) é um protocolo de passagem de mensagem desenvolvido para ser padrão de comunicação entre processos em ambientes de memória distribuída, realizando a distribuição de processos através das conexões entre máquinas e seus núcleos integradas no sistema compartilhado, onde por sua vez realiza a execução das tarefas.

MATERIAL E MÉTODOS

Buscando ilustrar o funcionamento do paralelismo e o uso das bibliotecas OpenMP e Open MPI, foram desenvolvidos quatro algoritmos utilizando linguagem C, que realizam o cálculo da integral de uma função matemática com a técnica "regra do trapézio" (ou regra trapezoidal). A ideia principal da regra do trapézio é aproximar a função por um polinômio de primeiro grau, ou seja, dividir a área de um trapézio em vários "pedaços" buscando traçar uma reta sobre eles. A função f(x) utilizada nos testes aqui apresentados está indicada na equação 1.

$$f(x) = \sqrt{1200^2 - x^2} \quad (1)$$

Os materiais utilizados na pesquisa estão restritos a:

- 1 Computador contendo um processador AMD FX 8-Core Black Edition Processors 3.3GHz (8 núcleos | 8 threads);
- Sistema Operacional Windows 10;
- Software Oracle VirtualBox.

Para realização dos testes utilizando a biblioteca OpenMP, utilizou-se:

- 1 Máquina virtual utilizando 6 núcleos de processamento;
- Sistema Operacional Linux Lubuntu 20.04;
- 3 Algoritmos, cada um contendo respectivamente um dos seguintes balanços de carga:
    - Schedule Static;
    - Schedule Dynamic;
    - Schedule Guided.

Para realização dos testes utilizando a biblioteca Open MPI, utilizou-se três máquinas virtuais com a arquitetura de um cluster beowulf (cluster de desempenho escalável, baseados numa infraestrutura de hardware comum, rede privada e software 'open source' (Linux)), como visto a seguir:

- 3 Máquinas virtuais sendo:
    - 1 mestre (utilizando 2 núcleos de processamento) e
    - 2 nós (cada uma utilizando 2 núcleos de processamento);
- Sistema Operacional Linux Lubuntu 20.04.
- 1 Algoritmo utilizando rotina de comunicação coletiva, sendo ela:
    - MPI_Allreduce.

Utilizando os materiais citados, executou-se os algoritmos desenvolvidos, alternando respectivamente entre 1, 2, 4 e 6 threads, e posteriormente realizou-se uma análise de desempenho entre as bibliotecas.

RESULTADOS E DISCUSSÃO

O problema proposto (Equação 1) é comumente utilizado em estudos de programação paralela pois possui complexidade ajustável ao nível de processamento requerido, desse modo realizou-se a construção de dois algoritmos para resolver o problema citado.

Para a utilização da OpenMP foi utilizada uma máquina virtual através de 1 prompt de comando. O código foi desenvolvido utilizando 3 tipos de Schedules(funções de balanço de carga) presentes na biblioteca, sendo elas:

- Static - distribui iterações do tamanho do “chunk” (“pedaço”) para cada thread;
- Dynamic - cada thread pega um “chunk” (“pedaço”) de iterações da fila até que todas as iterações sejam executadas;
- Guided - as threads pegam blocos de iterações dinamicamente, iniciando de blocos grandes reduzindo até o tamanho do “chunk” ("pedaço") e a técnica de divisão em threads para melhorar seu tempo de execução.

A Open MPI foi utilizada com apenas uma função, allReduce, que tem o objetivo de reduzir os valores e distribuir os resultados para todos os processos. O diferencial desta biblioteca se deu por ser utilizada em um cluster beowulf, utilizando a técnica de divisão por máquina, contendo 3 máquinas virtuais onde cada núcleo corresponde a uma thread. O cluster desenvolvido possui uma máquina configurada como mestre, com 2 núcleos e as outras duas como nós, contendo 2 núcleos cada.

Para poder realizar a análise das bibliotecas, realizou-se 16 ensaios, sendo 12 para a OpenMP onde cada um ocorreu utilizando uma das schedules variando a quantidade de threads em 1, 2, 4 e 6 respectivamente. Já para a Open MPI foram realizados 4 ensaios devido a utilização de apenas uma função de redução, variando os núcleos em 1, 2, 4 e 6 respectivamente.

Em todos os ensaios utilizou-se a complexidade/discretização em 0,000001, tornando o problema o mais similar possível, alterando apenas parâmetros de execução como núcleos e biblioteca.

O resultado da integral está presente na Equação 2 e os tempos de execução (em segundos) adquiridos nos ensaios estão listados na Tabela 1.

$$\int_{0}^{100} \sqrt{1200^2 - x^2} = 119860,9660751 \quad (2)$$

TABELA 1. Tempo de execução utilizando 0,000001

Threads	1	2	4	6
OpenMP (Static)	5,326s	2,895s	1,600s	1,253s
OpenMP (Dynamic)	5,683s	3,642s	3,770s	4,889s
OpenMP (Guided)	5,489s	2,904s	1,614s	1,195s
Open MPI	5,715s	3,911s	2,496s	1,908s

Os métodos de balanceamento de carga e de redução aqui utilizados estão comumente presentes na resolução de problemas complexos, ou seja, problemas mais difíceis para serem processados por somente um núcleo. Sendo assim, o tempo de execução é menor utilizando mais núcleos, porém é necessário validar se o tempo de execução com um núcleo é maior que o tempo de execução com diversos núcleos.

Através dos resultados obtidos, pode-se compreender que utilizando mais núcleos/threads, se obtém um tempo de execução menor, portanto maior velocidade na resolução do problema. Com exceção da função dynamic utilizada na OpenMP, que, por sua vez, acaba por se tornar a menos eficiente na divisão de iterações por thread, utilizando uma fila para distribuir os “pedaços”, gerando um gargalo de requisições, onde cada nó busca na fila o que deve processar, realizando esta busca repetidas vezes, gerando assim o acréscimo no tempo por requerimentos.

Para analisar o ganho de desempenho entre os núcleos/threads é possível utilizar o fator de desempenho (speedup) (Equação 3), onde é feita uma divisão entre o tempo obtido em uma execução sequencial (1 núcleo) e o tempo de uma execução paralela (diversos núcleos), sendo assim, quanto maior o valor resultante, maior o desempenho obtido com a execução paralela e distribuída.

$$S(p) = \frac{tempo\ de\ execução\ sequencial}{tempo\ de\ execução\ paralela} = \frac{t_s}{t_p} \quad (3)$$

Na Tabela 2 está presente o cálculo de speedup aplicando nos tempos obtidos na Tabela 1.

TABELA 2. Speedup

Threads	2	4	6
OpenMP (Static)	1,840	3,329	4,251
OpenMP (Dynamic)	1,560	1,507	1,162
OpenMP (Guided)	1,890	3,401	4,593
Open MPI	1,461	2,290	3,000

A Figura 1 ilustra os valores obtidos na Tabela 2, onde se torna visualmente claro o ganho de velocidade pela divisão de processamento, onde a OpenMP possui um ganho maior quando comparado a Open MPI, o que pode ser explicado quando analisamos que todos os testes foram realizados em máquinas virtuais, limitando o uso e a real funcionalidade de um cluster com a Open MPI.

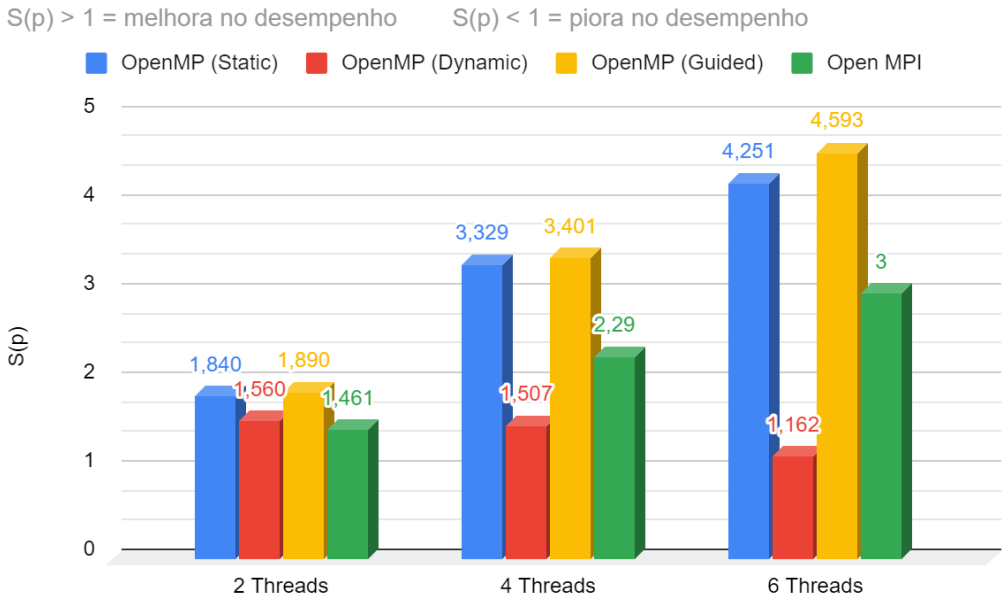

FIGURA 1. Speedup S(p) x Thread.

É possível identificar que conforme foi elevada a complexidade do algoritmo, ambas as bibliotecas obtiveram melhores resultados, levando em conta que na análise de speedup que se S(p) > 1 houve uma melhora no desempenho, ou seja, os valores obtidos indicam que obteve-se respectivamente uma melhora de 1 a 4 vezes da execução sequencial.

Devido a utilização de máquinas virtuais para execução dos testes, ocorreu uma limitação crucial na execução, mesmo com o emprego da programação paralela, através de funções e códigos específicos, dizendo para o sistema operacional (Linux) virtualizado a maneira que o paralelismo ocorra, ainda existe a limitação das máquinas sendo geridas por “cima” de outro sistema operacional (Windows), onde o mesmo realiza a própria gestão de processos por núcleo. Para compensar essa limitação procurou-se manter a máquina física com o mínimo de processos nativos e secundários presentes operando no momento dos testes, assim evitando grandes filas de processos e programas dividindo o poder de processamento.

CONCLUSÕES

Devido a complexidade do estudo e da construção do ambiente para execução dos testes necessários durante a pesquisa, o tempo estipulado para a realização das etapas se provou insuficiente, impossibilitando que se obtivesse êxito em todos objetivos estimados, ocasionando a falta de tempo para a construção da interface visual proposta, sendo o único ponto que não foi concluído.

Foi possível compreender os conceitos da programação paralela e aplicá-los nos algoritmos desenvolvidos, o que provou-se de enorme valor para a continuidade da pesquisa. Após a análise dos resultados não foi possível comparar, de forma palpável, os resultados das bibliotecas, levando em conta que utilizou-se um ambiente virtualizado para os testes, elevando assim a complexidade da análise e não possibilitando uma manipulação dos núcleos utilizados em cada máquina. Tal ambiente apenas foi utilizado por ausência de material e laboratório adequado para o desenvolvimento da pesquisa.

Ambas as bibliotecas possuem objetivos e aplicações diferentes, sendo a OpenMP mais utilizada para uma arquitetura de um computador simples, muitas vezes se limitando a apenas um, enquanto a Open MPI se utiliza de mais de um hardware para funcionar, elevando assim sua eficiência quando aplicada em um cluster de computadores.

Mesmo tendo ciência que não é possível realizar uma comparação entre as bibliotecas, e conforme já citado, a biblioteca Open MPI necessita de uma arquitetura distribuída (utilizada aqui como o cluster beowulf) para seu funcionamento, os resultados dos ensaios realizando o cálculo da integral através da regra do trapézio, comprova sua eficiência e o seu uso ser o mais adequado para projeto principal.

Tais resultados e algoritmos aqui desenvolvidos servirão como complemento do projeto "Configurando um Cluster de Processamento Paralelo utilizando Raspberry Pi", que tem como finalidade automatizar a configuração de um cluster de processamento paralelo em uma rede de computadores, apenas conectando um dispositivo via USB, justificando sua importância e eficiência.

Para projeto futuro, espera-se realizar os testes em um ambiente real com uma arquitetura de cluster beowulf.

AGRADECIMENTOS

Agradeço ao meu orientador e colegas de pesquisa, por se disponibilizarem a compartilhar seus conhecimentos sobre os assuntos contidos nesta pesquisa.

Agradeço ao Programa Institucional de Bolsas de Iniciação Científica e Tecnológica do Instituto Federal de Educação Ciência e Tecnologia de São Paulo (PIBIFSP), por proporcionar a oportunidade e apoio para realizar a pesquisa.

REFERÊNCIAS

INTEL MODERN CODE PARTNER - OPENMP – AULA 01. Grupo de Processamento Paralelo e Distribuído (GPPD). **UFRGS**. Disponível em: <https://www.inf.ufrgs.br/gppd/intel-modern-code/slides/workshop-1/MCP_Pt2_Pratica.pdf>.

KENDALL, Wes. MPI Reduce and Allreduce. **MPI Tutorial**. Disponível em: <https://mpitutorial.com/tutorials/mpi-reduce-and-allreduce/>.

OpenMP: The OpenMP API specification for parallel programming. **OpenMP**. Disponível em: <https://www.openmp.org/>.

Open MPI: Open Source High Performance Computing. The Open MPI Project. **Software in the Public Interest (SPI)**. Disponível em: <https://www.open-mpi.org/>.

PACHECO, Peter. An Introduction to Parallel Programming. Elsevier, 2011.

PITANGA, Marcos José. Construindo supercomputadores com linux. 1. ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2002. 183 p. ISBN 8574521043.