Análise Computacional de um Cluster Executando o Algoritmo de Cálculo de Números Primos Implementado com MPI

Murilo Cambruzzi Paiva, Willians Gomes Nunes, Wanderson Roger A. Dias, Danilo P. Escudero

Coordenadoria do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas (CCSTADS)
Laboratório de Arquiteturas Computacionais e Computação Paralela (LACCP)
Instituto Federal de Rondônia (IFRO)
Ji-Paraná – RO – Brasil

{murilolpaivacambruzzi, willians.gnunes01, wradias, danilopescudero}@gmail.com

Resumo. *A constante necessidade por desempenho computacional, tem se elevado cada vez mais, estimulado então a busca por novas alternativas para agregar desempenho às arquiteturas. Atualmente, computadores com alto poder de processamento têm sido desenvolvidos pela indústria utilizando estratégias que possibilitam as execuções paralelas das aplicações. Assim, este artigo apresenta os resultados do desempenho computacional de um Cluster executando paralelamente o algoritmo de cálculo de números primos usando a biblioteca MPI. Foi possível constatar que o Cluster escalado com 16 núcleos (arquitetura com 4 nós) teve um desempenho superior de ≈42% quando comparado com o Cluster com 8 núcleos (arquitetura com 2 nós) e o consumo médio energético foi de 1,03V para cada núcleo do Cluster.*

1. Introdução

É possível constatar que a indústria da computação tem como objetivo melhorar cada vez mais o desempenho computacional através do emprego de diversas formas de paralelismo. Para tanto, *hardwares* paralelos estão presentes nos principais equipamentos ou plataformas computacionais atuais. Estações de trabalho, servidores, supercomputadores, ou até mesmo sistemas embarcados fazem parte desta exploração de paralelismo para prover maior desempenho em suas respectivas plataformas (SOUTO *et al.*, 2007).

Esta abordagem de baixo nível de paralelismo acaba direcionando o desenvolvedor para mecanismos de programação mais complexos e com pouca portabilidade. Somente um pequeno número de programadores encara este desafio, a grande maioria não se sujeita a trabalhar neste paradigma. Até mesmo programadores com mais experiência tentam ignorar a programação paralela, pelos problemas que esta abordagem necessita tratar nos mais diversos ambientes de arquiteturas.

Então, os computadores em um contexto geral, já suportam *multithreading* e/ou *multi-core*. A decomposição de tarefas e de dados é uma forma de abstrair a complexidade na exploração de paralelismo destes processadores (Lima *et al.*, 2016). Sendo que o escalonamento dos threads ou processos é de responsabilidade do sistema operacional realizar a distribuição da carga entre os elementos de processamento, (SILBERSCHATZ *et al.*, 2001).

Ao mesmo tempo em que há um progresso no desenvolvimento de *hardware*, especialmente de arquiteturas paralelas, existe a necessidade de se oferecer recursos de programação compatíveis com os diferentes ambientes computacionais. Além disso, também é importante que haja mecanismos de programação capazes de integrar as diferentes arquiteturas paralelas existentes, simplificando assim o processo de programação. Desta forma, cria-se uma camada de abstração entre a aplicação em si e a sua plataforma de execução, podendo esta ser feita através de bibliotecas específicas para programação paralela, tais como: OpenMP e MPI (*Open Multi-Processing*, *MessagePassing Interface*, respectivamente) (OPENMP, 2022), (MPI, 2022).

Portanto, nesse artigo apresentamos a análise do desempenho computacional de um *Cluster*, executando o algoritmo de cálculo de números primos, implementado de forma paralela (usando a biblioteca MPI). O restante do artigo está organizado da seguinte forma: a Seção 2 apresenta uma breve contextualização, a fim de ambientar o estudo corrente; A Seção 3 apresenta as análises através dos resultados e discussões e a Seção 4 finaliza com as conclusões e ideias para trabalhos futuros.

2. Contextualização

2.1. Programação Paralela

Na Programação Paralela é realizada a divisão de uma determinada aplicação em partes, de maneira que essas partes possam ser executadas simultaneamente, por vários elementos de processamento. Os elementos de processamento devem cooperar entre si utilizando primitivas de comunicação e sincronização, realizando a quebra do paradigma de execução sequencial do fluxo de instruções (GEBALI, 2011). Então, a programação paralela permite tirar proveito dos recursos disponibilizados pelas arquiteturas paralelas, assim, é necessário que os algoritmos das aplicações estejam preparados para operar neste tipo de arquitetura, a fim de melhorar a sua velocidade de processamento.

Para a programação de ambientes de execução paralela, foram criadas as bibliotecas de comunicação paralela que possibilitam a implementação de programas paralelos em ambientes com memória compartilhada e distribuída. As bibliotecas como: OpenMP, PVM (*Parallel Virtual Machine*) e MPI possibilitam escrever programas paralelos usando as linguagens C, C++ e Fortran, para serem executados em arquiteturas paralelas (JIN *et al.*, 2011).

Já o modelo de programação paralela híbrida, na mesma implementação, é usado em conjunto o MPI e OpenMP, sendo que utiliza-se MPI para a comunicação entre os nós e OpenMP para paralelização dentro do nó. A ideia básica desse modelo é permitir que qualquer processo MPI possa criar um grupo de *threads*OpenMP (JIN *et al.*, 2011). Então, a principal motivação para utilizar programação paralela híbrida é tirar o máximo de proveito das melhores características de ambos os modelos de programação, mesclando a paralelização explícita de grandes tarefas com o MPI com a paralelização de tarefas simples com o OpenMP.

2.2. Algoritmo de Números Primos

Os números primos são fundamentais nos sistemas de criptografia modernos, pois possibilitam que tenhamos segurança em nossas transações financeiras, por exemplo. Devido às suas propriedades especiais para a fatoração, embora não seja tão difícil encontrar números primos relativamente grandes, é inevitavelmente difícil de levar um número grande de volta para os primos. Uma coisa é descobrir que 35 é (7x5), e outra bem diferente para descobrir que 2.244.354 é (2x3x7x53437). E é outra bem diferente para encontrar novamenteos fatores primos de um número cinquenta dígitos. Um supercomputador pode levar até alguns milhões anos ou até mais para resolver um problema de fatoração de 256 *bits* (AKEL, 2016). Sendo assim, neste artigo, implementamos a versão deste algoritmo paralelamente usando a biblioteca MPI, com o propósito de calcular a quantidade existente de números primos no intervalo de 1 a 500.000 e executamos na arquitetura de um *Cluster* computacional com 2, 3 e 4 nós.

2.3. Consumo Energético em Computação Paralela

O aumento da eficiência energética é importante em todo contexto computacional, independente do hardware adotado. Ou seja, para todas as arquiteturas e implementações físicas de máquinas paralelas, as quais podem apresentar diferentes consumos de potência, a eficiência energética é um fator crucial a ser analisado. Nesse sentido, a programação paralela tem papel fundamental na melhoria da eficiência, utilizando melhor os recursos computacionais e aumentando o desempenho das aplicações. Por outro lado, a arquitetura e organização do processador paralelo podem ajudar ainda mais na redução do consumo energético, com dispositivos mais eficientes e novas técnicas de aumento de desempenho. Além disso, existe a atuação sobre o consumo de potência, onde a arquitetura, seja através de componentes mais sofisticados, seja pelo desligamento de unidades que não estão sendo utilizadas em uma certa porção de tempo, poderá colaborar com a redução do consumo de potência.

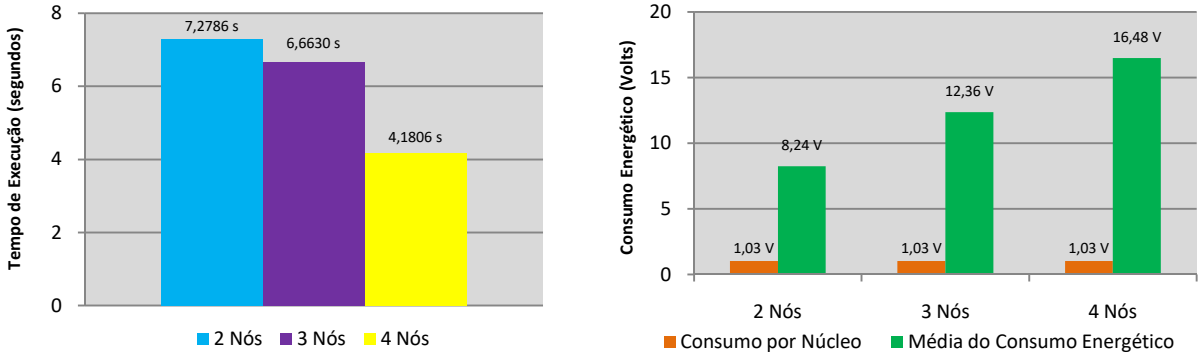
Assim, a diminuição no tempo de execução de um algoritmo tem um impacto significativo no consumo energético. Conforme (MÓR *et al.*, 2010), o objetivo de utilizar a programação paralela para a economia de recursos é obter um aumento de *Speedup*. Pois, a programação paralela tem como papel quanto à eficiência energética o uso dos recursos disponíveis de maneira eficiente, fazendo com que o mínimo de unidades de processamento ligadas fique ociosas, o que aumenta o *Speedup*.

3. Resultados e Discussão

Foi realizada comparação das métricas de “tempo de execução” e “consumo energético” do algoritmo de cálculo de números primos implementado com MPI e executando em uma arquitetura paralela (*Cluster*)

escalada com 4 nós, 3 nós e 2 nós. Estes cenários de testes foi composto por computadores de arquitetura homogênea com processador Intel Core i3 3240, com 2 núcleos físicos e 2 lógicos, com frequência de 3.4 GHz, 8 GB de memória RAM, sistema operacional Linux distribuição Mint e *Kernel* 5.4.0-9. Para os testes, utilizou-se o compilador GCC (versão *4:9.3.0-1ubunto2*) para compilar o código do algoritmo de cálculo de números primos (propositalmente não otimizado para gerar ainda mais estresse no processamento) implementado na linguagem C e usando a biblioteca MPI na versão *3.3.2-2build1*. Para a comunicação entre os nós do *Cluster*, foi usado o protocolo de comunicação SSH na versão *1:8.2p1-4ubunto0.5*. Também foi usado o software CPU-X (versão *3.2.4*) para mensurar a métrica do consumo de energia e a *flag* "*Time*" do compilador GCC para mensurar o tempo de execução do algoritmo no *Cluster*.

Na arquitetura paralela, à rede de interconexão usada para conectar os nós do *Cluster*, foi configurada com a topologia estrela e usou um *Switch* marca HP 1920-24G, modelo JG924A, além de cabos UTP da categoria 6. Para execução dos testes, foram executados 500 (quinhentos) processos na configuração do *Cluster* com 4 nós (16 núcleos), 3 nós com 12 núcleos e posteriormente com 2 nós e 8 núcleos, sendo a metade desses núcleos físicos e a outra metade lógicos, conforme já mencionado anteriormente. O consumo energético foi extraído no mesmo tempo do processamento do algoritmo, tendo assim, uma análise real em tempo de execução. A Figura 1 (a) apresenta o tempo de execução do algoritmo de cálculo de números primos executando em um *Cluster* com 4, 3 e 2 nós, respectivamente e na Figura 1 (b) é apresentado a média do consumo energético dos núcleos individuais e a média de consumo em cada nó do *Cluster*.



(a) Tempo de Execução **(b) Consumo Energético**

Figura 1. Desempenho Computacional do Cluster

Conforme observado na Figura 1 (a) e na Tabela 1, é possível destacarmos que o *Cluster* escalado com 4 nós (16 núcleos), executou o algoritmo, que realiza o cálculo de números primos, ≈37,2% mais rápido quando comparado com o *Cluster* escalado com apenas 3 nós, já quando a análise leva em consideração o *Cluster* escalado com 2 nós, ou seja, 8 núcleos de processamento, a diferença no tempo de execução foi ainda mais elevado, chegando em mais de ≈42% (ver Tabela 1). Em todas as execuções, foi possível constatar que o consumo de energia em cada núcleo foi de 1,03V quando usado 100% da CPU, impactando assim, diretamente no consumo médio em nós no *Cluster*.

Tabela 1. Tempo de Execução do Algoritmo do Cálculo de Números Primos

Execuções	Média do Tempo de Execução		
	2 Nós	3 Nós	4 Nós
Execução 1	8,1660 s	6,8940 s	4,1350 s
Execução 2	8,1430 s	6,4320 s	4,2930 s
Execução 3	7,9620 s	6,6340 s	4,3230 s
Execução 4	7,8460 s	6,5430 s	3,9330 s
Execução 5	7,9870 s	6,1790 s	4,2190 s
Média	**7,2786 s**	**6,6630 s**	**4,1806 s**

Portanto, analisando os valores da Tabela 1 (média), podemos destacar que conforme os nós foram escalados no *Cluster*, o desempenho computacional apresentou uma performance na métrica tempo de

execução, tal fato pode ser justificado pela uma melhor distribuição entre os 500 processos gerados pelo algoritmo e processados pela arquitetura de alto desempenho.

4. Conclusões e Trabalhos Futuros

Mediante aos experimentos realizados com o algoritmo de cálculo de números primos, utilizando o sistema operacional Linux distribuição Mint, com alocação de nós de uma arquitetura *Cluster* de alto desempenho, foi constatado e apresentado nesse artigo (ver Figura 1 (a) e Tabela 1) que a execução distribuída entre 4 nós (16 núcleos), atingiu uma melhor performance do tempo de execução, ou seja, mais de ≈37% e ≈42% quando comparado com o *Cluster* escalados com 3 nós (12 núcleos) e 2 nós (8 núcleos), respectivamente. No entanto, quando comparamos a média do consumo energético de cada arquitetura, observou-se que o *Cluster* escalado com 4 nós, consumiu 25% e 50% a mais do que o *Cluster* escalado com 3 nós e 2 nós, respectivamente. Essa diferença se dá pelo fato da quantidade de núcleos utilizados em cada uma das arquiteturas, pois, como já mencionado anteriormente, cada processador consome exatamente 1,03V quando a CPU atinge 100% do seu uso e isto implica diretamente no consumo de energia das alocações dos nós.

Como ideias para trabalhos futuros sugerem-se: (i) análise comparativa escalando mais nós no *Cluster*; (ii) análise comparativa executando um algoritmo de cálculo de números primos de forma serial e paralela; (iii) análise do desempenho computacional de um *Cluster* em diferentes topologias de rede de interconexão; (iv) análise computacional de *Cluster* executando o algoritmo de cálculo de números primos entre modelos de programação paralela apenas com MPI e a programação paralela híbrida, ou seja, MPI e OpenMP; (v) análise computacional com o algoritmo implementado de forma otimizado, dentre outras.

Agradecimentos

Os autores agradecem ao Instituto Federal de Rondônia (IFRO), pelo apoio financeiro concedido através dos Editais Nº 9/2022/REIT - PROPESP/IFRO e Nº 11/2022/REIT - PROPESP/IFRO, que proporcionou a execução dessa pesquisa.

Referências

Akel, A. (2022) "A Importância dos Números Primos – Unidades Imaginárias". Disponível em https://medium.com/unidades-imaginárias/a-importância-dos-números-primos-1249a54cc57e. Acessado em 10 de Maio de 2022.

De Rose, C. A. F.; Navaux, P. O. A. (2008) "Arquiteturas Paralelas". – Porto Alegre, RS, Brasil: Bookman, 2ª edição, 152p.

Gebali, F. (2011) "Algorithmsand Parallel Computing". – Wiley, 1st edition, 364p.

Jin, H.; Jespersen, D.; Mehrotra, P.; Biswas, R.; Huang, L.; Chapman, B. (2011) "High Performance Computing using MPI and OpenMP on Multi-core Parallel Systems". In *Parallel Computing*, 37(9):562-575.

Mór, S. D. K.; Alves, M. A. Z.; Lima, J. V. F.; Maillard, N. B.; Navaux, P. O. A. (2010) "Eficiência Energética em Computação de Alto Desempenho: Uma Abordagem em Arquitetura e Programação para Green Computing". In *XXX Seminário Integrado de Software e Hardware (SEMISH 2010)*, Belo Horizonte, MG, Brazil, pg 60-75.

Mpi. (2022) "A Message-Passing Interface Standard Version 2.1". Disponível em www.mpi-forum.org/docs/mpi21-report.pdf. Acessado em 25 de Junho de 2022.

Lima, F. A.; Moreno, E. D.; Dias, W. R. A. (2016) "Performance Analysis of a Low Cost Cluster with Parallel Applications and ARM Processors". In *IEEE Latin America Transactions*, 14(11):4591-4596.

Openmp. (2022) "The OpenMP API Specification for Parallel Programming". Disponível em http://openmp.org/. Acessado em 17 de Junho de 2022.

Silberschatz, A.; Peter, G.; Gagne, G. (2001) "Sistemas Operacionais - Conceitos e Aplicações". – Campus, 8ª edição, 618p.

Souto, R. P.; Ávila, R. B.; Navaux, P. A. O.; Py, M.; Maillard, N.; Diverio, T. A.; Velho, H. F. de C.; Stephany, S.; Preto, A.; Panetta, J.; Rodrigues, E.; Almeida, E. (2007) "Processing Mesoscale Climatology in a Grid Environment". In *Proceedings of the Seventh IEEE International Symposiumon Cluster Computing and the Grid (CCGRID'07)*, Rio de Janeiro, RJ, Brazil, pg 363-370.