Implementação Sequencial e Distribuída do Crivo de Eratóstenes Utilizando MPI

Eros Henrique Lunardon Andrade[1], Igor de Souza Pinto[1], Victor Henrique Vechi[1]

[1]Campus Pinhais – Instituto Federal do Paraná (IFPR)
Pinhais - PR - Brasil

eroshla.13@gmail.com, igorpinto103@gmail.com, victorvechi@gmail.com

Abstract. *This work presents the sequential and distributed implementation of the Sieve of Eratosthenes for prime number computation. The sequential version was used as the performance baseline, while the distributed version was implemented using the MPI library, distributing contiguous segments of the numerical interval among the processes. Experiments were executed on a heterogeneous two-machine cluster, varying the problem sizes and number of processes. Five executions were performed for each configuration. Performance metrics such as speedup, efficiency, and theoretical limits based on Amdahl's Law were computed. Results show that small inputs do not benefit from MPI due to communication overhead, while larger inputs present meaningful speedups, although limited by the heterogeneity of the hardware and network constraints.*

Resumo. *Este trabalho apresenta a implementação sequencial e distribuída do Crivo de Eratóstenes para detecção de números primos. A versão sequencial foi utilizada como referência de desempenho, enquanto a versão distribuída foi desenvolvida com a biblioteca MPI, realizando a divisão do intervalo numérico em blocos independentes entre os processos. Os experimentos foram executados em um cluster heterogêneo composto por duas máquinas, variando-se os tamanhos de entrada e o número de processos envolvidos. Cada configuração foi executada cinco vezes. Foram calculadas métricas de desempenho como speedup, eficiência e o limite teórico baseado na Lei de Amdahl. Os resultados indicam que entradas pequenas não apresentam ganho com paralelização distribuída, enquanto entradas maiores obtêm speedups perceptíveis, ainda que limitados pela heterogeneidade do hardware e custos de comunicação.*

1. Introdução

O objetivo deste trabalho foi implementar o Crivo de Eratóstenes para encontrar números primos até um limite n, tanto em sua forma sequencial quanto em uma versão distribuída utilizando MPI. A versão sequencial serviu como referência de desempenho, enquanto a versão distribuída permitiu avaliar a viabilidade do paralelismo em um ambiente real de computação distribuída.

As atividades envolveram a execução dos algoritmos em diferentes tamanhos de entrada e configurações de processos, possibilitando o cálculo de métricas como speedup e eficiência. A partir disso, foi possível discutir a escalabilidade do algoritmo e entender os impactos do overhead de comunicação em um ambiente distribuído heterogêneo.

2. Fundamentação Teórica

O Crivo de Eratóstenes é um algoritmo clássico para encontrar todos os números primos menores ou iguais a n. Ele funciona marcando múltiplos sucessivos de cada primo encontrado, eliminando números compostos. Sua complexidade é:

O(n log log n)

Em termos de desempenho, algoritmos paralelos são analisados utilizando:

2.1 Speedup

$$S(p) = T_{seq} / T_p$$

Onde:
T_seq = tempo da versão sequencial
T_p = tempo da versão paralela com p processos

2.2 Eficiência

$$E(p) = S(p) / p$$

2.3 Lei de Amdahl

O limite máximo teórico de aceleração é dado por:

$$S_{max} = 1 / ((1 - f) + (f / p))$$

onde f é a fração paralelizável.

Neste trabalho foram usadas operações como **MPI_Bcast**, **MPI_Reduce** e **MPI_Gatherv**, essenciais para coordenar as etapas do algoritmo em múltiplos processos.

3. Metodologia

A execução foi dividida entre a versão sequencial do algoritmo e sua versão distribuída com MPI. Ambas as versões foram desenvolvidas em C++.

3.1 Versão Sequencial

A versão sequencial implementa o Crivo clássico, marcando múltiplos a partir de p^2 até n. O tempo da execução foi medido com a biblioteca <*chrono*>, considerando:

- Tempo total
- Tempo da etapa de eliminação de múltiplos
- Tempo pós-processamento (coleta dos primos)

Foram realizadas **5 execuções por tamanho de entrada**, calculando a média obtida para comparação com a versão distribuída.

3.2 Versão Distribuída com MPI

A versão distribuída divide o intervalo [2, n] entre os processos. O funcionamento geral é:

1. O processo 0 calcula os **primos base** até sqrt(n).
2. Esses primos são enviados aos demais via **MPI_Bcast**.
3. Cada processo recebe um bloco do intervalo e marca múltiplos localmente.
4. As quantidades de primos são agregadas com **MPI_Reduce**.
5. Para n pequeno, os primos encontrados são reunidos com **MPI_Gatherv**.

O tempo da versão paralela é dividido conceitualmente em:

$$T_p = T_{seq\_local} + T_{paral} + T_{com}$$

onde:

- T_{seq_local} = parte sequencial realizada por todos
- T_{paral} = parte paralelizável
- T_{com} = comunicação entre processos

3.3 Configuração Experimental

O ambiente distribuído utilizado foi composto por duas máquinas:

PC 1 (Manager)

- Intel Core i5-1340P (16 threads)

- 8 GB RAM

PC 2 (Worker)

- Intel Core i5-650 (4 threads)
- 4 GB RAM

3.4 Procedimento de Medição

Para cada tamanho de entrada (1M, 5M e 7M), foram realizadas 5 execuções utilizando:

- 2 processos
- 4 processos
- 8 processos
- 14 processos

Foram calculados:

- Tempo médio
- Speedup: S(p) = T_seq / T_p
- Eficiência: E(p) = S(p) / p

3.5 Referencial Máximo Teórico

A fração paralelizável medida experimentalmente foi aproximadamente:

f ≈ 0.88

Assim:

$$\mathbf{S_{max} = 1 / ((1 - 0.88) + (0.88 / 14)) \approx 5.96}$$

Esse valor é usado como referência na análise da escalabilidade.

4 Resultados e Discussões

4.1 Resultados Experimentais

Entrada: 1.000.000

- Sequencial: 0.00855 s
- MPI 14 processos: 0.01207 s
- Speedup: ~0.70 (sem ganho)

Com n pequeno, o overhead de comunicação domina o tempo total.

Entrada: 5.000.000

- Sequencial: 0.04139 s
- MPI 14 processos: 0.02305 s
- Speedup = 0.04139 / 0.02305 ≈ 1.79
- Eficiência = 1.79 / 14 ≈ 0.128

Resultado: ganho moderado.

Entrada: 7.000.000

- Sequencial: 0.06817 s
- MPI 14 processos: 0.02753 s
- Speedup = 0.06817 / 0.02753 ≈ 2.47
- Eficiência = 2.47 / 14 ≈ 0.176

Melhor desempenho registrado.

4.2 Resumo Comparativo

Entrada	Seq (s)	MPI (14p)	Speedup	Eficiência	Teórico
1M	0.00855	0.01207	0.70	0.05	~5.96
5M	0.04139	0.02305	1.79	0.128	~5.96
7M	0.06817	0.02753	2.47	0.176	~5.96

Análise:

- Entradas pequenas não escalam devido ao custo de comunicação.
- Entradas médias e grandes apresentam acelerações reais.
- A heterogeneidade do cluster reduz o speedup (um PC é muito mais fraco).
- O melhor equilíbrio entre speedup e eficiência ocorreu com n = 7M.

5 Conclusão

Este trabalho apresentou a implementação sequencial e distribuída do Crivo de Eratóstenes utilizando MPI. A versão sequencial serviu de base para comparação e a versão distribuída explorou paralelismo através da divisão de blocos.

Os resultados mostraram que a paralelização não traz ganhos para entradas pequenas devido ao custo de comunicação. Para entradas maiores, os testes demonstraram speedup consistente, embora inferior ao limite teórico devido à heterogeneidade das máquinas e à forte presença de overhead.

A eficiência caiu conforme o número de processos aumentou, em alinhamento com a Lei de Amdahl. O maior ganho absoluto ocorreu na entrada de 7 milhões, onde o paralelismo foi melhor aproveitado.

Como trabalhos futuros, sugere-se testar o algoritmo em ambiente homogêneo, explorar paralelização híbrida MPI + OpenMP e avaliar o uso de técnicas de balanceamento dinâmico de carga.

6 Referências

Amdahl, G. M. (1967). *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.* AFIPS Conference Proceedings, pp. 483–485.

Pacheco, P. S. (1996). *Parallel Programming with MPI.* Morgan Kaufmann.

Message Passing Interface Forum. (2015). *MPI: A Message-Passing Interface Standard, Version 3.1.* Disponível em: https://www.mpi-forum.org

Knuth, D. E. (1998). *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms.* Addison-Wesley.