UNIVERSIDADE ESTADUAL PAULISTA
“Júlio de Mesquita Filho”

Pós-Graduação em Ciência da Computação

Rafael Henrique Moretti

Análise do Efeito de Entropia em Computação Quântica: Simulações em ambiente paralelo

UNESP

2015

Moretti, Rafael Henrique.

Análise do efeito de entropia em computação quântica : simulações em ambiente paralelo / Rafael Henrique Moretti. – São José do Rio Preto, 2015

86 f. : il.

Orientador: Geraldo Francisco Donegá Zafalon

Coorientador: Manoel Ferreira Borges Neto

Dissertação (mestrado) - Universidade Estadual Paulista "Júlio de Mesquita Filho", Instituto de Biociências, Letras e Ciências Exatas

1. Computação quântica. 2. Teoria quântica. 3. Entropia quântica. 4. Programação paralela (Computação) I. Zafalon, Geraldo Francisco Donegá. II. Borges Neto, Manoel Ferreira. III. Universidade Estadual Paulista "Júlio de Mesquita Filho". Instituto de Biociências, Letras e Ciências Exatas. IV. Título.

CDU - 530.145

Ficha catalográfica elaborada pela Biblioteca do IBILCE
UNESP - Câmpus de São José do Rio Preto

Rafael Henrique Moretti

Análise do Efeito de Entropia em Computação Quântica: Simulações em ambiente paralelo

Orientador: Prof. Dr. Geraldo Francisco Donegá Zafalon

Coorientador: Prof. Dr. Manoel Ferreira Borges Neto

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração - Computação Científica, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

UNESP

2015

Rafael Henrique Moretti

Análise do Efeito de Entropia em Computação Quântica: Simulações em ambiente paralelo

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração - Computação Científica, do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

BANCA EXAMINADORA

Prof. Dr. Geraldo Francisco Donegá Zafalon
UNESP - São José do Rio Preto
Orientador

Prof. Dr. Mário Luiz Tronco
USP - São Carlos

Prof. Dr. Wladimir Seixas
UFSCAR - Sorocaba

São José do Rio Preto, 27 de fevereiro de 2015.

A mente que se abre à uma
nova idéia jamais voltará ao
seu tamanho original.
Albert Einstein

A Deus, aos meus pais e

aos meus mestres.

Dedico

AGRADECIMENTOS

Primeiramente gostaria de me desculpar àqueles que acabei por esquecer de citar aqui, mas que tiveram enorme contribuição para a minha vida.

À Deus pelo dom da vida e sabedoria.

Agradeço aos meus pais Salvador e Rita, por serem meu porto seguro, pelo apoio incondicional e por tudo que fizeram e fazem por mim onde, sem eles, não estaria aqui hoje.

Aos meus irmãos Taysa Moretti, Fernando Cunha Biagini e Bianca Cunha Biagini por serem meus eternos companheiros.

Aos meus familiares, amigos da pós e da prefeitura de Olímpia, além de todos os que contribuíram para o meu desenvolvimento durante a pós-graduação e a elaboração deste trabalho, em especial Camila Brandão, Adriana F. Roberto Ártico, Edevar Bastreghi Filho e Lucas Machado Silveira.

Aos meus mestres Geraldo Zafalon, Manoel Borges, Carlos Valêncio e José Márcio Machado por todo apoio, paciência, orientação e amizade, além da saudade que teremos de ti Zé, que neste momento nos olha ao lado de Deus.

À Capes pela bolsa no período do mestrado, que foi de grande ajuda para me manter financeiramente.

Ao Núcleo de Computação Científica (NCC/GridUNESP) da Universidade Estadual Paulista (UNESP) por tornar possível a pesquisa graças aos recursos computacionais disponibilizados.

Ao Departamento de Computação e Estatística, professores e funcionários pelo suporte nesses anos de IBILCE-UNESP.

Sumário

1 Introdução **1**

1.1 Considerações Iniciais 1

1.2 Motivação 2

1.3 Objetivos 2

1.4 Organização dos Capítulos 2

2 Conceitos **4**

2.1 Princípios da Mecânica Quântica 4

2.1.1 Dualidade Onda-partícula 4

2.1.2 Princípio da Incerteza de Heisenberg 5

2.1.3 Superposição de Estados 6

2.2 Computação e Informação Quântica 6

2.2.1 O bit quântico 7

2.2.2 Circuitos Quânticos 8

2.2.3 Algoritmos Quânticos 10

2.3 Computador Quântico 11

2.4 Emaranhamento e Entropia 14

2.4.1 Entropia 15

2.4.2 Entropia de Von Neumann 15

2.4.3 Entropia de Tsallis 16

2.4.4 Emaranhamento no divisor de feixe 17

2.5 Computação Paralela 19

2.5.1 Processadores multicore 19

2.5.2 Clusters Computacionais 20

2.5.3 MPI - Message Passing Interface 20

3 Desenvolvimento **22**

3.1 Bibliotecas de precisão múltipla 22

3.1.1 GMP 22

3.1.2 MPFR 23

3.2 Implementação em um ambiente de um único processador 23

3.3 Implementação em um ambiente paralelo 25

3.4 Implementação em um ambiente de linguagem interpretada 29

4 Testes e Resultados **30**

4.1 Ambientes de Teste 30

4.2 Metodologia dos Testes 31

4.3 Entrada Única 32

4.3.1 Comparativos de tempo 36

4.4 Entradas iguais 39

4.4.1 Comparativos de tempo 43

4.5 Entradas diferentes 47

4.5.1 Comparativos de tempo 50

4.6 Speedup 54

4.6.1 Mathematica x Paralelismo com MPI x Grid 54

4.6.2 C x MPI x Grid 59

5 Conclusões **65**

5.1 Considerações Finais 65

5.2 Contribuições 65

5.3 Trabalhos Futuros 66

Lista de Figuras

2.1 *Esfera de Bloch.* 8
2.2 *Processador quântico proposto (adaptado) [1].* 12
2.3 *Processador quântico projetado (adaptado) [2].* 13
2.4 *Processador quântico projetado (adaptado) [3].* 13
2.5 *Divisor de feixe.* 17
2.6 *Organização do cluster* 20

3.1 Fluxograma da implementação em paralelo. 26
3.2 Fluxograma da cálculo da divisão das tarefas em blocos. 27
3.3 Junção das strings locais no processo mestre. 28
3.4 Implementação das entropias no *software* Mathematica®. 29

4.1 *Gráfico dos tempos médios de execução dos testes de entrada única.* 32
4.2 *Gráfico comparativo de tempo médio entre as implementações (50x0).* . . . 34
4.3 *Gráfico comparativo de tempo médio entre as implementações (100x0).* . . 35
4.4 *Gráfico comparativo de tempo médio entre as implementações (200x0).* . . 35
4.5 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (50x0).* . 37
4.6 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (100x0).* . 37
4.7 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (200x0).* . 38
4.8 *Gráfico comparativo de tempo médio C x MPI x Grid (50x0).* 38
4.9 *Gráfico comparativo de tempo médio C x MPI x Grid (100x0).* 39
4.10 *Gráfico comparativo de tempo médio C x MPI x Grid (200x0).* 39
4.11 *Gráfico dos tempos médios de execução dos testes de entradas iguais.* . . . 40
4.12 *Gráfico comparativo de tempo médio entre as implementações (5x5).* 42
4.13 *Gráfico comparativo de tempo médio entre as implementações (10x10).* . . 42
4.14 *Gráfico comparativo de tempo médio entre as implementações (50x50).* . . 43

4.15 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (5x5).* . . 44
4.16 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (10x10).* . 44
4.17 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (50x50).* . 45
4.18 *Gráfico comparativo de tempo médio C x MPI x Grid (5x5).* 45
4.19 *Gráfico comparativo de tempo médio C x MPI x Grid (10x10).* 46
4.20 *Gráfico comparativo de tempo médio C x MPI x Grid (50x50).* 46
4.21 *Gráfico dos tempos médios de execução dos testes de entradas diferentes.* . 47
4.22 *Gráfico comparativo de tempo médio entre as implementações (3x7).* 49
4.23 *Gráfico comparativo de tempo médio entre as implementações (4x16).* . . . 49
4.24 *Gráfico comparativo de tempo médio entre as implementações (30x70).* . . 50
4.25 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (3x7).* . . 51
4.26 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (4x16).* . 51
4.27 *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (30x70).* . 52
4.28 *Gráfico comparativo de tempo médio C x MPI x Grid (3x7).* 52
4.29 *Gráfico comparativo de tempo médio C x MPI x Grid (4x16).* 53
4.30 *Gráfico comparativo de tempo médio C x MPI x Grid (30x70).* 53
4.31 *Speedup Mathematica x MPI x Grid (50x0).* 54
4.32 *Speedup Mathematica x MPI x Grid (100x0).* 55
4.33 *Speedup Mathematica x MPI x Grid (200x0).* 55
4.34 *Speedup Mathematica x MPI x Grid (5x5).* 56
4.35 *Speedup Mathematica x MPI x Grid (10x10).* 56
4.36 *Speedup Mathematica x MPI x Grid (50x50).* 57
4.37 *Speedup Mathematica x MPI x Grid (3x7).* 57
4.38 *Speedup Mathematica x MPI x Grid (4x16).* 58
4.39 *Speedup Mathematica x MPI x Grid (30x70).* 58
4.40 *Speedup C x MPI x Grid (50x0).* 59
4.41 *Speedup C x MPI x Grid (100x0).* 60
4.42 *Speedup C x MPI x Grid (200x0).* 60
4.43 *Speedup C x MPI x Grid (5x5).* 61
4.44 *Speedup C x MPI x Grid (10x10).* 61
4.45 *Speedup C x MPI x Grid (50x50).* 62
4.46 *Speedup C x MPI x Grid (3x7).* 62

4.47 *Speedup C x MPI x Grid (4x16).* 63
4.48 *Speedup C x MPI x Grid (30x70).* 63

Lista de Tabelas

4.1 Resultados dos tempos dos testes de entrada a = 50 e b = 0. 33

4.2 Resultados dos tempos dos testes de entrada a = 100 e b = 0. 33

4.3 Resultados dos tempos dos testes de entrada a = 200 e b = 0. 34

4.4 Resultados dos tempos dos testes de entradas a = 5 e b = 5. 40

4.5 Resultados dos tempos dos testes de entradas a = 10 e b = 10. 41

4.6 Resultados dos tempos dos testes de entradas a = 50 e b = 50. 41

4.7 Resultado dos tempos dos testes de entradas $a = 3$ e $b = 7$. 47

4.8 Resultado dos tempos dos testes de entradas $a = 4$ e $b = 16$. 48

4.9 Resultado dos tempos dos testes de entradas $a = 30$ e $b = 70$. 48

Resumo

O crescente desenvolvimento tecnológico tem trazido à humanidade grandes benefícios, nas mais diversas áreas. De modo a dar continuidade a esse desenvolvimento, novas frentes de pesquisas vêm surgindo, em busca do domínio dessas tecnologias emergentes. Os limites físicos da computação clássica, baseada nos fenômenos eletromagnéticos, estão sendo alcançados e a computação quântica surge como uma possível solução para esses limites, bem como para apresentar um novo panorama para a computação, devido ao seu grande potencial. A fim de buscar um maior entendimento dos fenômenos que envolvem a computação quântica em uma transmissão de dados, em específico o fenômeno do emaranhamento, no presente trabalho apresenta-se um levantamento teórico sobre mecânica quântica, informação, computação e entropias quânticas, bem como computação paralela e MPI, propondo-se uma simulação com implementação em ambiente paralelo sobre o efeito da entropia de emaranhamento dos fótons em uma transmissão de dados. Além disso, realiza-se a comparação com a implementação em um ambiente de um único processador.

Palavras-chave: Mecânica Quântica, Computação Quântica, Informação Quântica, Entropia de Tsallis, Entropia de Von Neumann, Computação Paralela

Abstract

The increasing technological development has brought great benefits to humanity, in several areas. In order to continue this development, new research areas are emerging to reach new technologies. The physical limits of classical computing, based on electromagnetic phenomena are being achieved and quantum computing emerges as a possible solution to these limits, as well as to introduce a new scenario for computing, due to its great potential. In order to get a better understanding of phenomena involving quantum computing in a data transmission, in particular the phenomenon of entanglement, this work presents a theoretical quantum mechanics, information, computing and quantum entropies, as well as parallel computing and MPI, proposing a simulation with implementation in parallel environment on the effect of the entropy of entanglement of photons in data transmission and comparison with implementation in a single processor environment.

Keywords: *Quantum mechanics, Quantum Computation, Quantum Information, Tsallis Entropy, Von Neumann Entropy, Parallel Computing*

Capítulo 1

Introdução

1.1 Considerações Iniciais

O crescente desenvolvimento tecnológico tem trazido à humanidade grandes benefícios nas mais diversas áreas. De modo a dar continuidade a esse desenvolvimento, despontam-se novas frentes de pesquisas, com o intuito do domínio dessas tecnologias emergentes. Os limites físicos da computação clássica, baseada nos fenômenos eletromagnéticos, estão sendo alcançados e a computação quântica surge como uma possível solução para esses limites, bem como para apresentar um novo panorama para a computação, devido ao seu grande potencial.

Baseada nos princípios da física quântica, a computação quântica traz consigo novos desafios a serem solucionados e também soluções para problemas que a computação clássica não consegue lidar, como por exemplo a área de segurança da informação. Os métodos clássicos que regem a segurança da informação em meios eletrônicos, hoje são baseados em fatorações de grandes números primos e têm como principal algoritmo o RSA (Ronald **R**ivest, Adi **S**hamir e Leonard **A**dleman) [4]. Com o advento da computação quântica, a eficiência desses métodos se torna questionável pois o poder computacional é muitas vezes maior que o clássico, fazendo com que novos métodos tenham de ser desenvolvidos.

1.2 Motivação

Alguns trabalhos têm sido realizados nesta área de quântica, no entanto ainda é uma área incipiente, o que remete a um vasto campo de pesquisa. Além disso, em trabalhos anteriores, apresentou-se o formalismo matemático por trás da quantificação do emaranhamento no divisor de feixes via entropias de Von Neumann e Tsallis, com algumas poucas simulações para efeito de análise.

Posteriormente, Brandão [5] estendeu as simulações em um ambiente de linguagem interpretada porém, devido ao grande custo computacional, o número de fótons nas entradas do divisor de feixes foi limitado. Os resultados de Brandão [5] mostraram que as simulações são custosas computacionalmente e, com isso, despertou-se o interesse por estudos mais aprofundados no âmbito da simulação computacional, motivando o desenvolvimento do presente trabalho.

1.3 Objetivos

De modo a contribuir para um maior conhecimento dos fenômenos quânticos que ocorrem, uma vez que esta é uma área que ainda não foi exaustivamente estudada, este trabalho visa ampliar os testes de simulações que analisam o emaranhamento (fenômeno ligado à inter-relação dos estados físicos em um ambiente quântico) dos fótons em uma transmissão de dados [6, 5, 7], utilizando as entropias de Von Neumann [8] e Tsallis [9] através de um ambiente paralelo. A inclinação da curva na entropia de Von Neumann, observada nas simulações anteriores, se confirmada, pode significar uma tendência de diminuição de entropia e de auto-organização dos fótons.

Por meio de um ambiente paralelo este trabalho busca um menor tempo para a obtenção dos resultados em comparação com testes em um ambiente de um único processador e também expandir o número de fótons nas entradas do divisor de feixes para análise do comportamento das entropias.

1.4 Organização dos Capítulos

Este trabalho é dividido em outros quatro capítulos, além deste capítulo 1, sendo eles organizados da seguinte maneira:

- Capítulo 2: apresentou-se a fundamentação teórica do trabalho, bem como assuntos relacionados ao mesmo.
- Capítulo 3: apresentou-se o desenvolvimento do trabalho.
- Capítulo 4: apresentou-se os testes e resultados do desenvolvimento do trabalho.
- Capítulo 5: são realizadas considerações finais sobre o presente trabalho.

Capítulo 2

Conceitos

Este capítulo aborda a fundamentação teórica do trabalho a ser desenvolvido. É apresentada uma conceituação sobre a Mecânica Quântica, a computação e informação quânticas, o computador quântico, entropia e emaranhamento, bem como computação paralela.

2.1 Princípios da Mecânica Quântica

Através de Max Planck, em 1900 surgiu a física quântica [10]. Com base em estudos probabilísticos, diferentemente do determinismo clássico, na teoria quântica o próprio ato de observação do objeto de estudo causa alteração do estado do mesmo. Os princípios da mecânica quântica [11], fundamentação deste trabalho, são apresentados nas seções seguintes.

2.1.1 Dualidade Onda-partícula

A luz, durante o século XIX, era tida como uma onda eletromagnética, pois experimentos realizados apontavam um comportamento de onda, por conta da difração e interferência. Experimentos realizados no decorrer dos anos porém, acabaram por revelar também um comportamento de partícula da luz. Albert Einstein, dando o nome de fóton a partícula indivisível de luz, comprovou a dualidade partícula-onda [12]. Planck e Einstein demonstraram que a energia é transmitida quantizada através de pacotes de onda. A energia de um fóton é dada pela expressão (2.1), em que ν é a frequência da onda incidente e h a constante de Planck igual a 10^{-34} . Sua massa de repouso é zero,

de acordo com a Teoria da Relatividade, pois teria massa infinita ao alcançar velocidades próximas à da luz se tivesse massa de repouso. [13]

$$E = h * \nu \tag{2.1}$$

As relações de conexão entre as propriedades de partícula da luz com as propriedades de onda, são denominadas Relações de Broglie [14], dadas pelas equações:

$$\nu = \frac{E}{h} \tag{2.2}$$

e

$$\lambda = \frac{h}{P} \tag{2.3}$$

em que λ é o comprimento de onda, P o momentum e h a constante de Planck.

2.1.2 Princípio da Incerteza de Heisenberg

Com base nos princípios da Mecânica Clássica (Newtoniana), no mundo macroscópico pode-se determinar componentes de um sistema em um instante de tempo qualquer, determinando-se com precisão seus valores. Como exemplo dessa situação, considere uma pessoa se deslocando de um ponto ao outro: pode-se determinar sua posição, velocidade e estado em qualquer intervalo de tempo de maneira precisa.

A princípio, esse pensamento de que as regras do mundo macroscópico se aplicassem ao mundo microscópico e por consequência ao mundo quântico era tido como verdadeiro. Porém, Werner Heisenberg e Niels Böhr contestaram esse pensamento e Heisenberg, em 1927, formulou o Princípio da Incerteza [10] (expressões 2.4 e 2.5), em que não se pode fazer medidas precisas de maneira independente, de variáveis dependentes entre si, no qual as medidas de sua posição e velocidade não podem ser precisadas de maneira simultânea, pois a medida de uma influencia na medida da outra.

$$\Delta P_i \Delta q_i = -\Delta q_i \Delta P_i \Rightarrow \Delta P_i \Delta q_i = \frac{\alpha}{2}, se\ \alpha\ \sim 10^{-27} \tag{2.4}$$

$$Com\ \alpha = \frac{ih}{\pi}\ obtemos: \quad \Delta p \Delta q = \frac{ih}{2\pi} = \hbar \tag{2.5}$$

2.1.3 Superposição de Estados

A superposição de estados, segundo a teoria quântica, é o estado de um objeto em observação, considerando que este objeto pode se apresentar em dois estados distintos, o objeto se encontra simultaneamente em seus dois estados possíveis, ou seja, superpostos. O simples ato de observação acaba por desclassificar essa superposição, fazendo com que o objeto passe a um de seus estados possíveis.

Erwin Schrödinger, Nobel de Física em 1933 juntamente com Paul Adrien Maurice Dirac pela descoberta de novas formas produtivas da teoria atômica, explicou esse conceito através da parábola *O gato de Schrödinger*, descrevendo-se a situação de um gato em uma caixa. Em uma primeira observação, o gato se encontra vivo e então é depositado um frasco com veneno dentro da caixa e a mesma é fechada. Assim supomos que o gato está vivo se não abriu o frasco ou morto se o abriu. Porém, segundo a teoria quântica, o gato se encontra em uma superposição desses dois estados, estando vivo e morto ao mesmo tempo, em que se abrir a caixa para uma segunda observação, o gato tende a tomar um desses estados, desfazendo assim a superposição. Esse fenômeno traz à computação ganhos exponenciais se for aplicado ao processamento de informações, possuindo aplicações diversas.

2.2 Computação e Informação Quântica

De modo a aproveitar as possibilidades que os sistemas quânticos oferecem em relação aos sistemas clássicos, como o emaranhamento e a superposição de amplitudes de probabilidades, numa unificação entre a Teoria da Informação e a Teoria Quântica se formou, para uso na área da Informação [15, 16].

Baseados na arquitetura de Von Neumann, os sistemas clássicos estão chegando a seus limites físicos, uma vez que a velocidade de processamento e a capacidade de armazenamento estão cada vez maiores, superando até as previsões da Lei de Moore, que previa a duplicação da velocidade a cada 18 meses [17].

Com componentes cada vez menores, mais rápidos e complexos começam-se a projetar estruturas de ordem microscópica, nas quais as leis da mecânica clássica não conseguem explicar seus comportamentos, cabendo para a mecânica quântica tal tarefa.

Deste modo é necessário que se faça uma redefinição dos princípios básicos do tra-

tamento da informação tais como a definição da unidade básica de informação, portas lógicas, algoritmos e protocolos, não mais clássicos mas sim quânticos [18].

Em 1982, Feynman [19] deu início aos estudos sobre a computação quântica, apontando as falhas dos sistemas clássicos ao modelar sistemas quânticos, sugerindo que o processamento das informações fosse feito por computadores baseados nas leis da mecânica quântica. Assim poderiam ser aproveitadas as capacidades quânticas para resolver problemas inviáveis na computação clássica e, com isso, acelerar o desenvolvimento de algoritmos quânticos para a solução desses problemas.

Uma das implementações destes sistemas quânticos é baseada em RMN - Ressonância Magnética Nuclear [20], trabalhando com estados de *spins* nucleares, em que algumas simulações quânticas estão sendo realizadas [21, 22]. Porém ainda não existe, ou pelo menos ainda não foi divulgado, um *hardware* quântico totalmente funcional.

2.2.1 O bit quântico

Em âmbito quântico, a unidade básica de informação é denominada *qubit*, ou *q-bit* ou ainda *bit* quântico. Diferentemente do *bit* clássico, que pode assumir dois estados mutuamente excludentes (0 para ausência de carga elétrica e 1 para presença de carga elétrica), o *qubit*, por conta do fenômeno de superposição, pode assumir estados de superposição dos estados lógicos clássicos, tendo a representação de um estado dada por:

$$|\Psi >= \alpha|0 > +\beta|1 > \tag{2.6}$$

onde:

- $|0>$ e $|1>$ são vetores e formam uma base ortonormal do espaço vetorial $\mathbb{C}^2$.
- $|0>= \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ e $|1>= \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ formam a base computacional.
- $|\Psi>$ é chamado de superposição dos vetores $|0>$ e $|1>$.
- O *qubit* é um vetor de norma 1 de $\mathbb{C}^2$: $|\alpha|^2 + |\beta|^2 = 1$.

Em geral, os *qubits* são representados por fótons, ou *spins* de núcleos atômicos. Um *qubit* pode ser representado através de uma partícula de $\frac{1}{2}$ *spin* [18].

Através da parametrização dos estados puros de um *qubit*, pelos ângulos $\theta \in [0, \pi]$ e $\phi \in [0, 2\pi]$ e fazendo uso de parâmetros de forma a obter uma representação polar no $\Re^3$, obtêm-se uma esfera, chamada de Esfera de Bloch [23] (figura 2.1) que representa esses estados.

$$|\Psi >= \cos(\frac{\theta}{2})|0 > +e^{i\phi} \sin(\frac{\theta}{2})|1 > \tag{2.7}$$

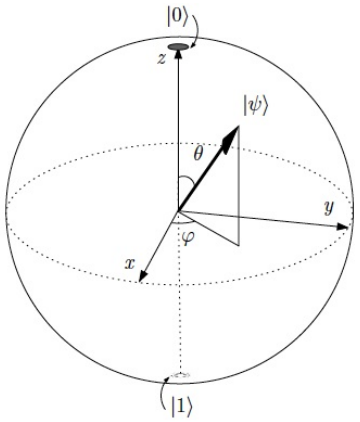

Figura 2.1: *Esfera de Bloch.*

Em um sistema computacional, a representação da forma matricial dos estados é a que segue:

$$|0 >= \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad \text{e} \quad |1 >= \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}$$

Fazendo uso da propriedade da superposição, a capacidade exponencialmente maior dos computadores quânticos se dá pelo fato de um computador quântico de n *qubits* conseguir processar 2^n estados lógicos, o que não acontece na computação clássica. A metodologia utilizada no processamento quântico é baseada na utilização dos *qubits* em estados de superposição, de modo que os estados que representam os resultados procurados tenham aumento na probabilidade de ocorrência, sofrendo, para isso, interferência construtiva. Contrariamente os outros estados devem sofrer interferência destrutiva.

2.2.2 Circuitos Quânticos

Da mesma forma que os sistemas clássicos trabalham por meio de circuitos que aplicam operações lógicas sobre os bits, os sistemas quânticos operam de forma similar, porém

trata-se os circuitos quânticos de forma temporal, pois são estados e não sistemas físicos determinísticos reais.

As operações lógicas são realizadas tanto na computação clássica quanto na quântica através de portas lógicas, sendo na computação clássica OR,XOR,AND e NOT. Em âmbito quântico, como exemplo de conjunto universal, tem-se as portas Hadamard, Porta T, CNOT e Porta de Fase. Porém, para a construção das portas é necessária apenas uma porta chamada CNOT de dois qubits. O processo de construção de portas lógicas em computação quântica se dá através do produto tensorial entre estados, conforme expressões 2.8 e 2.9:

$$|01>=|0>\otimes|1>=\begin{bmatrix}1\\0\end{bmatrix}\otimes\begin{bmatrix}0\\1\end{bmatrix}=\begin{bmatrix}0\\1\\0\\0\end{bmatrix} \tag{2.8}$$

e

$$|10>=|1>\otimes|0>=\begin{bmatrix}0\\1\end{bmatrix}\otimes\begin{bmatrix}1\\0\end{bmatrix}=\begin{bmatrix}0\\0\\1\\0\end{bmatrix} \tag{2.9}$$

Desta maneira, a porta quântica CNOT é definida como:

$$\begin{aligned} &QXor|0>\otimes|\psi>=|0>\otimes|\psi> \\ &QXor|1>\otimes|\psi>=|1>\otimes Xor|\psi> \end{aligned} \tag{2.10}$$

A porta Hadamard, bastante usada em algoritmos quânticos possui a seguinte forma:

$$H|0>=\tfrac{1}{\sqrt{2}}(|0>+|1>)=|+> \quad \text{e} \quad H|1>=\tfrac{1}{\sqrt{2}}(|0>-|1>)=|-> \tag{2.11}$$

Matricialmente, dada pela expressão 2.12:

$$H\equiv\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix} \tag{2.12}$$

Os estados de Bell, vetores do tipo não-decomponíveis ou emaranhados possuem a seguinte forma:

$$|\phi_{\pm}>=\frac{1}{\sqrt{2}}(|00>\pm|11>) \quad \text{e} \quad |\psi_{\pm}>=\frac{1}{\sqrt{2}}(|01>\pm|10>) \tag{2.13}$$

e

$$|\phi_{00}>=\frac{(|00>+|11>)}{\sqrt{2}} \quad , \quad |\phi_{01}>=\frac{(|01>+|10>)}{\sqrt{2}},$$
$$|\phi_{10}>=\frac{(|00>-|11>)}{\sqrt{2}} \quad \text{e} \quad |\phi_{11}>=\frac{(|01>-|10>)}{\sqrt{2}}, \tag{2.14}$$

A construção das portas lógicas quânticas exige uma alta precisão, dado que a influência nos qubits altera seus estados e destrói a superposição deles, inviabilizando o processo computacional. A taxa de erros, em grande parte, é devido ao meio em que o processo ocorre, fazendo com que técnicas sejam desenvolvidas e aprimoradas para que se tenha um ambiente favorável, tais como a ressonância magnética nuclear, eletrodinâmica quântica de cavidade e armadilha de íons.

2.2.3 Algoritmos Quânticos

De modo a apresentar soluções mais eficientes para os problemas que não podem ser resolvidos pela computação clássica ou ainda de grande custo computacional de forma mais eficiente, os algoritmos quânticos acabam por ser divididos em três classes, sendo os de maior notoriedade os algoritmos de Shor [18], Grover [24] e Deutsch [25].

Os algoritmos são classificados em uma das três classes de acordo com o ganho de tempo de solução em relação aos algoritmos clássicos, nos quais se enquadram a primeira e a segunda classe, ou ainda por simular sistemas quânticos, se enquadrando na terceira classe.

Baseados na Transformada de Fourier Quântica, em notação vetorial com ação sobre superposição dada pela expressão 2.15, a primeira classe apresenta algoritmos com soluções exponencialmente mais rápidos que os mais efetivos algoritmos clássicos. Como principal exemplo dessa categoria tem-se o algoritmo de fatoração de Shor, oferecendo ferramentas para a quebra de sistemas criptográficos clássicos que, até então, acreditava-se que fossem praticamente inquebráveis. Uma implementação dele, por Vandersypen et al. [26] em 2001, foi realizada utilizando RMN - Ressonância Magnética Nuclear, para fatoração do número 15.

$$\sum_{j=0}^{2^n-1} x_j|j> \rightarrow \frac{1}{\sqrt{2^n}} \sum_{k=0}^{2^n-1} \left[\sum_{j=0}^{2^n-1} e^{2\pi ijk/2^n} x_j \right] |k>=\sum_{k=0}^{2^n-1} y_k|k> \tag{2.15}$$

A segunda classe apresenta algoritmos com ganhos polinomiais em relação aos algoritmos clássicos, em problemas NP-completos. Como principal exemplo dessa categoria tem-se o algoritmo de Grover [25], o qual otimiza a velocidade na busca de um determinado elemento em uma lista não ordenada de n elementos. Experimentalmente, esse algoritmo foi implementado através de RMN [27], íons atômicos [28] e fótons [29].

A terceira classe é composta de algoritmos que simulam sistemas quânticos [19]. Os computadores clássicos possuem muitas limitações que inviabilizam as simulações de sistemas quânticos principalmente pelo fato de que para representar n *qubits* são necessários X^n *bits*, em que X varia de acordo com o sistema em simulação. Deste modo existe um ganho exponencial sobre os computadores clássicos. As implementações desta classe se deram através de RMN [30].

2.3 Computador Quântico

A computação quântica sempre foi vista com certo receio, como uma ciência muito teórica e abstrata, uma vez que não se tem ao alcance um computador efetivamente quântico. Porém em pesquisas recentes mudou-se este panorama, com diversos estudos destacando aspectos mais concretos. Em Mariantoni et al. [1] destaca-se como foi realizada a implementação de um processador quântico baseado na arquitetura de Von Neumann, mostrando uma unidade de processamento central quântica que realiza troca de dados com uma memória quântica de acesso aleatório integrada em um *chip*, com instruções armazenadas em um computador clássico.

O processador consiste de dois *qubits* supercondutores acoplados através de um barramento quântico, dois registradores zeradores e duas memórias quânticas. Foram executados dois algoritmos para computação quântica: a transformada quântica de Fourier e uma porta lógica Toffoli *OR* de fase de três *qubits*.

Usualmente são utilizados processadores quânticos baseados em ressonância magnética nuclear, íons aprisionados e dispositivos semicondutores. Em Mariantoni et al. [1] utilizou-se um circuito integrado supercondutor que combina um processador com memória e um registrador zerador em apenas um dispositivo, caracterizando assim a arquitetura de Von Neumann. Através dessa arquitetura, a *CPU* quântica desempenha portas de 1, 2 e 3 *qubits* que processam a informação quântica a ser escrita, lida e zerada. A estrutura do

processador é ilustrada pela figura 2.2.

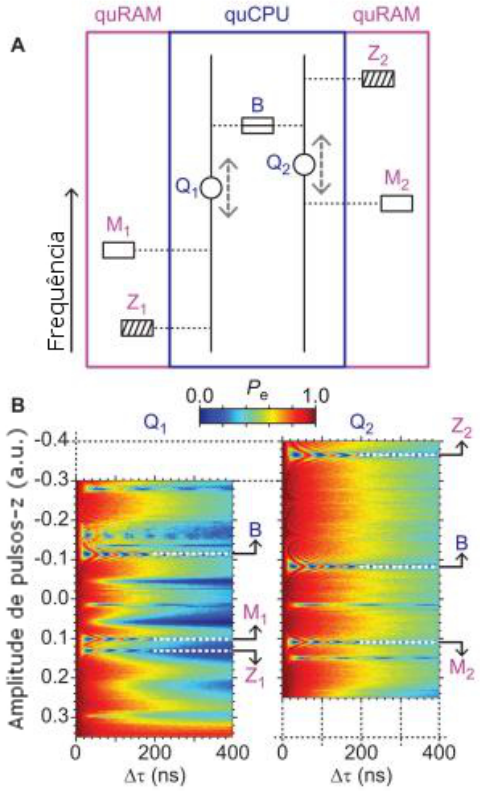

Figura 2.2: *Processador quântico proposto (adaptado) [1].*

A *CPU* quântica (*quCPU*) é composta dois *qubits Q1* e *Q2* e um barramento ressoador*B*. Já a memória quântica (*quRAM*) é composta por duas memórias *M1* e *M2* e dois registradores zeradores *Z1* e *Z2*. A direção vertical representa a frequência, sendo que *M1*, *M2*, *Z1* e *Z2* são fixas, enquanto que as frequências de transição dos *qubits* podem ser definidas através de pulsos *z* (linhas pontilhadas em cinza).

Os testes realizados com a transformada quântica de Fourier obtiveram 66% de fidelidade de fase, enquanto que o teste com a porta Toffoli obteve 98%. A partir desse modelo de processador pode-se citar outros trabalhos relacionados como em [3] e [2]. Em Dicarlo et. al [3] foram executados dois algoritmos quânticos que são busca de Groover e DeutschJozsa e em Lucero et al. [2] executou-se o algoritmo de Shor para fatorar o número 15. Nas figuras 2.3 e 2.4 são ilustrados os processadores projetados para a realização dos testes.

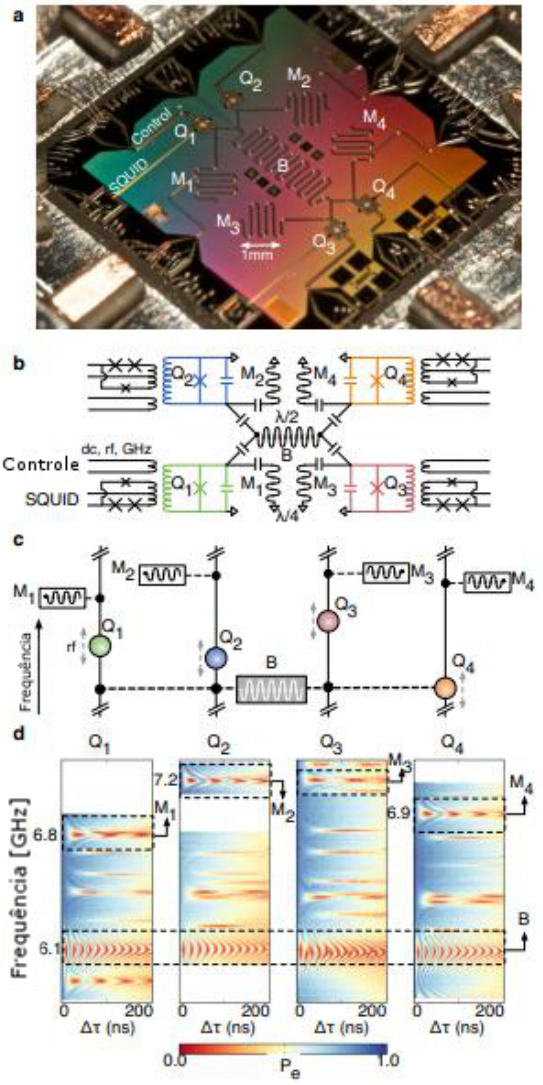

Figura 2.3: *Processador quântico projetado (adaptado) [2].*

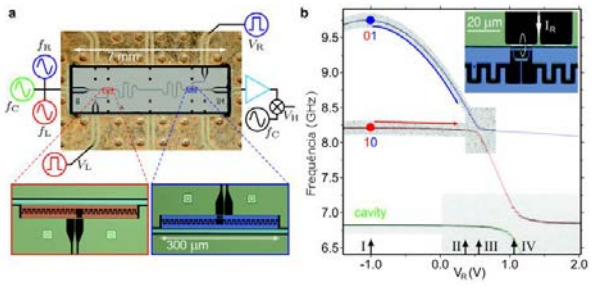

Figura 2.4: *Processador quântico projetado (adaptado) [3].*

2.4 Emaranhamento e Entropia

Fenômeno de difícil explicação mas de grande importância no âmbito quântico, o emaranhamento [31, 32, 33, 34] é experimentalmente ilustrado pelo ensaio de fendas duplas de Young[35], comprovando que há uma correlação não-local entre o fóton e um detector. Esta correlação é criada quando estes elementos estão juntos e se mantêm mesmo se forem separados a grandes distâncias. Deste modo não se pode lidar com as partes separadas, mas com um sistema único.

Historicamente, o emaranhamento quântico foi muito discutido através do paradoxo de EPR de 1935 [36], que questionava a capacidade da mecânica quântica em descrever completamente a realidade dos eventos físicos. As discussões continuaram, e trabalhos correlacionados foram desenvolvidos, com destaque para o trabalho denominado desigualdades de Bell [37, 38]. Em 1982, um experimento [39] comprovou que a mecânica quântica pode sim descrever de maneira completa a realidade, realizando os testes das desigualdades de Bell e admitindo correlações não-locais.

O emaranhamento possui diversas aplicações, sendo que hoje a teleportação quântica de estados, protocolos de criptografia quântica e a codificação superdensa são os de maior evidência [40, 41, 42]. Sucintamente o teleporte quântico de estados é a transmissão de um estado quântico entre dois locais sem o efeito da distância que os separa. Os protocolos de criptografia quântico utilizam-se do emaranhamento para garantir a segurança das comunicações, e a codificação superdensa é a transmissão, através de um *qubit*, de 2 *bits* clássicos de informação, utilizando um estado emaranhado e um canal quântico para se comunicar. Dessa forma, reforça-se o ganho no poder computacional oferecido pela computação quântica quando comparado à computação convencional.

Formalmente, o conceito de emaranhamento é definido como uma qualidade de todo estado físico que não pode ser representado como um produto tensorial simples dos elementos dos espaços de Hilbert multiplicados [31]. Caso a matriz de densidade de um subsistema for diferente da matriz densidade de um estado puro, diz-se que este subsistema é emaranhado, sendo a definição de emaranhamento dada por negação, ou seja:

$$\psi_{ab} \neq |\psi_a > \otimes |\psi_b > \tag{2.16}$$

$$\rho_{ab} \neq |\psi_{ab} >< \psi_{ab}| \tag{2.17}$$

De modo a quantificar o emaranhamento, utiliza-se a entropia como medida para tal, explicada a seguir.

2.4.1 Entropia

Entropia [43] é o termo dado ao grau de caoticidade de um sistema, muito aplicado em termodinâmica. Sua representação é dada pela letra S, sendo uma função de estado do sistema. Sua definição e aplicabilidade passa por áreas desde a Termodinâmica até as Telecomunicações, com Claude E. Shannon [44] e sua medida de entropia expressa em 2.18 para auxiliar na economia de transmissão e armazenamento de informação. Com o passar do tempo houve um ganho de importância da entropia em sistemas dinâmicos, surgindo a entropia não-extensiva de Constantino Tsallis [6, 45, 46].

Utilizou-se o conceito de entropia para quantificar o emaranhamento, a partir as entropias de Von Neumann e Tsallis para tal.

$$S(X) \equiv S(p_1, p_2, ..., p_n) \equiv -1 \sum_{i}^{W} p_i \, log_2 \, p_i \tag{2.18}$$

onde W é um conjunto de eventos equiprováveis.

2.4.2 Entropia de Von Neumann

Voltada à mecânica quântica, a entropia de Von Neumann possui conceito análogo à da entropia de Shannon, que em uma distribuição de probabilidade faz a medição da incerteza associada. Sendo ρ o operador densidade, tem-se a entropia de Von Neumann do estado associado dada por:

$$S(\rho) \equiv -Tr(\rho \, log_2 \, \rho) \tag{2.19}$$

Propriedades básicas da entropia de Von Neumann [18]:

- A entropia de Von Neumann é não-negativa. A entropia é zero se, e somente se, o estado é puro.

- Em um espaço d-dimensional de Hilbert a entropia é no máximo log d. Este valor é obtido se, e somente se, o sistema é um estado de mistura máxima, $\frac{I}{d}$, sabendo-se que $0 \leqslant S(\rho \| I/d) = -S(\rho) + log_2 d)$

- Se um sistema composto AB estiver em um estado puro, resulta que $S(A) = S(B)$.

- Suponha que p_i são probabilidades e que os operadores densidade ρ_i têm suporte em subespaços ortogonais. Então,

$$S\left(\sum_i p_i \rho_i\right) = H(p_i) + \sum_i p_i \, S(\rho_i). \tag{2.20}$$

- Sejam p_i probabilidades, $|i>$ estados ortogonais de um sistema A, e ρ_i qualquer conjunto de operadores densidade de um sistema B. Resulta que

$$S\left(\sum_i p_i \, |i><i| \otimes \rho_i\right) = H(p_i) + \sum_i p_i \, S(\rho_i), \tag{2.21}$$

  conhecida também como teorema da entropia conjunta.

2.4.3 Entropia de Tsallis

A entropia de Tsallis apresenta uma generalização da entropia de Boltzmann-Gibbs, sendo condizente à segunda lei da termodinâmica e com forte adaptação a diversos sistemas físicos. Frequentemente utilizada como medida adequada para a quantificação em sistemas dinâmicos de informação que possuem características não-extensivas. É dada por:

$$S_q = k \, \frac{1 - \sum_i^W p_i^q}{q - 1} \tag{2.22}$$

Propriedades básicas da entropia de Tsallis [47]:

- S_q é contínua em p_i, para $0 < p_i < 1$.

- Para um conjunto W de eventos equiprováveis, ou seja, $p_i = \dfrac{1}{W}$, então S_q é uma função monotônica crescente.

- Para dois subsistemas estatisticamente independentes A e B a entropia generalizada S_q do sistema composto $A + B$ satisfaz a relação de pseudo-aditividade

$$S_q(A + B) = S_q(A) + S_q(B) + (1 - q) \, S_q(A) \, S_q(B) \tag{2.23}$$

2.4.4 Emaranhamento no divisor de feixe

Os experimentos simulados neste trabalho têm como objeto de observação o que ocorre no divisor de feixes[1] por onde os fótons passam. Ao passarem, suas respectivas ondas são divididas sendo que uma parte é transmitida e a outra refletida, além dessas ondas exercerem interferência entre si(superposição), conforme ilustrado pela figura 2.5. Nestas simulações são tomados estados de Fock dos campos de entrada [7] , estados esses com um número definido de fótons.

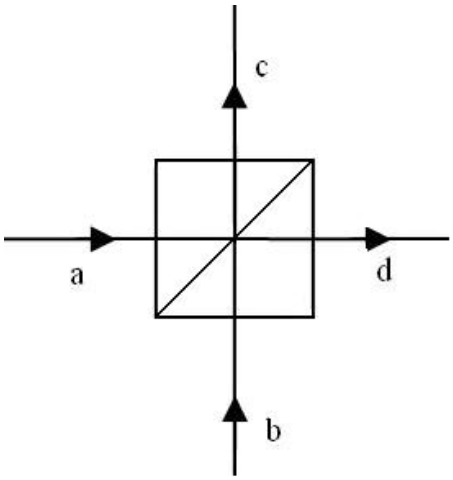

Figura 2.5: *Divisor de feixe.*

Os operadores de entrada e saída são os pares a e c para a porta 1 e os pares b e d para a porta 2 e pertencem ao espaço de Hilbert. Dessa forma tem-se os termos resultantes através dos coeficientes T e R referentes à transmissão e reflexão, respectivamente, com norma igual a 1. A diferença de fase entre transmissão e reflexão é dada por ϕ e tem-se que $T = \cos\dfrac{\theta}{2}$, $R = \sin\dfrac{\theta}{2}$. Assim a matriz geral do divisor de feixe é dada por:

$$B = e^{i\phi_0}\begin{pmatrix} \cos\theta e^{i\phi_T} & \sin\theta e^{i\phi_R} \\ -\sin\theta e^{-i\phi_R} & \cos\theta e^{-i\phi_T} \end{pmatrix} \tag{2.24}$$

Os operadores do campo de saída são dados por:

$$c = BaB^{\dagger} \quad \text{e} \quad d = BbB^{\dagger} \tag{2.25}$$

[1]Divisor de feixe: espelho de envoltório reflexivo delgado, ocasionando certa refração da luz por conta de sua estrutura, que funciona como um anteparo de cristal, no qual tem-se uma entrada de feixes de luz (a e b) e saída de estados emaranhados (c e d).

em que $B^\dagger$ denota o operador adjunto, ou hermitiano conjugado, do operador linear B.

Considerando estados de Fock de entrada independentes, o estado de saída $|\psi>$ será dado por:

$$|\psi>= B|n_1n_2>=\sum_{N_1N_2}<N_1N_2|B|n_1n_2>|N_1N_2>=\sum_{N_1N_2}B_{n_1n_2}^{N_1N_2}|N_1N_2> \tag{2.26}$$

no qual

$$B_{n_1n_2}^{N_1N_2}=e^{-i\theta(n_1-N_1)}\sum_{k=0}^{n_1}\sum_{l=0}^{n_2}(-1)^{n_1-k}R^{n_1+n_2-k+l}T^{k+l}\frac{\sqrt{n_1!n_2!N_1!N_2!}}{k!(n_1-k)!l!(n_2-l)!}\times$$
$$\times\delta_{N_1,n_2+k-l}\delta_{N_2,n_1-k+l} \tag{2.27}$$

Esse estado é dado pela superposição dos estados de entrada, em que δ representa a função de Kronecker, expressa da seguinte forma:

$$\delta_{ij}=\begin{cases}0, & \text{se} \quad i\neq j;\\ 1, & \text{se} \quad i=j.\end{cases} \quad \text{(Delta de Kronecker)}$$

O estado emaranhado de saída possui dimensão n_1+n_2+1, onde n_1+n_2 é a soma do número de fótons da entrada. Deste modo, utilizando o operador densidade reduzido $\rho_c = Tr_d\, B|n_1n_2><n_1n_2|\, B^\dagger$, as entropias de Von Neumann ($S(\rho_c)$) e Tsallis ($S_q(\rho_c)$) são dadas por:

$$S(\rho_c)=-\sum_{N_1N_2}|B_{n_1n_2}^{N_1N_2}|^2\, ln\, |B_{n_1n_2}^{N_1N_2}|^2 \tag{2.28}$$

$$S_q(\rho_c)=\frac{1}{q-1}\left(1-\sum_{N_1N_2}|B_{n_1n_2}^{N_1N_2}|^{2q}\right) \tag{2.29}$$

Desenvolvendo as fórmulas 2.28 e 2.29 para implementação, tem-se:

$$S=-\sum_{c=0}^{a+b}\sum_{d=0}^{a+b}(|\sum_{k=0}^{a}\sum_{l=0}^{b}((-1)^{a-k}\, r^{a+b-k-l}\, t^{k+l}\frac{\sqrt{a!b!c!d!}}{k!(a-k)!l!(b-l)!}\, \delta_{c,b+k-l}\ \ \delta_{d,a-k+l})|^2)*$$

$$*\, ln\, [|\sum_{k=0}^{a}\sum_{l=0}^{b}((-1)^{a-k}\ \ r^{a+b-k-l}\, t^{k+l}\frac{\sqrt{a!b!c!d!}}{k!(a-k)!l!(b-l)!}\, \delta_{c,b+k-l}\, \delta_{d,a-k+l})|^2+1\times10^{-29}] \tag{2.30}$$

$$S_q=\frac{1}{q-1}(1-\sum_{c=0}^{a+b}\sum_{d=0}^{a+b}(|\sum_{k=0}^{a}\sum_{l=0}^{b}((-1)^{a-k}\, r^{a+b-k-l}\, t^{k+l}\frac{\sqrt{a!b!c!d!}}{k!(a-k)!l!(b-l)!}\, \delta_{c,b+k-l}\, \delta_{d,a-k+l})|^{2q})) \tag{2.31}$$

2.5 Computação Paralela

Problemas que envolvem grandes quantidades de cálculos e dados acabam por demandar elevado poder de processamento. Como exemplo, pode-se citar estudos meteorológicos, astronômicos, prospecção de petróleo, bioinformática, dentre outros. De modo a solucionar esses problemas, a computação paralela vem sendo utilizada em larga escala, dado que economicamente é uma solução viável mesmo que não se utilize de supercomputadores comerciais.

Segundo a taxonomia de Flynn [48], os computadores paralelos são classificados de acordo com a quantidade de instruções e dados processados em um certo momento:

- SISD (*Single Instruction Single Data*): computadores sequenciais, em que um único fluxo de instruções trabalha sobre um único fluxo de dados (modelo clássico de Von Neumann). Como exemplo tem-se os computadores convencionais.
- SIMD (*Single Instruction Multiple Data*): computadores vetoriais e matriciais, em que há um único fluxo de intruções e múltiplos fluxos de dados. Como exemplo tem-se as máquinas IBM 9000, Cray X-MP e Thinking Machine CM-2.
- MISD (*Multiple Instruction Single Data*): arquitetura teórica em que múltiplos fluxos de instruções trabalham sobre um único fluxo de dados. Não foi implementada.
- MIMD (*Multiple Instruction Multiple Data*): arquitetura com computadores de múltiplos processadores ou sistemas com múltiplos computadores, em que diferentes processadores executam diferentes instruções em diferentes fluxos de dados. Como exemplo tem-se o IBM pSeries e os *clusters* Beowulf.

O foco deste trabalho é comparar o resultado de execuções de estratégias sequenciais com estratégias em paralelo. Os sistemas computacionais paralelos a serem utilizados se baseiam em duas configurações: uma com processador *multicore* e outra em um ambiente de *cluster* computacional.

2.5.1 Processadores multicore

Processadores *multicore* são aqueles que possuem dois ou mais núcleos de processamento encapsulados em um único *chip*. Deste modo podem desempenhar mais tarefas

simultaneamente, uma vez que o sistema operacional trata estes núcleos como processadores distintos, dividindo as tarefas. O surgimento se deu por conta do crescente desenvolvimento tecnológico na fabricação dos *microchips*, que hoje trabalham em escala nanométrica (10^{-9}m), e também pelos problemas de dissipação de calor em processadores com *clocks* cada vez maiores.

2.5.2 Clusters Computacionais

O conceito do *cluster* envolve uma infraestrutura convencional de *hardware* e rede, e *software open-source*. Sua organização (ilustrada pela figura (2.6)) é composta de um mestre (ou *Front-End*) interligado através de um *switch* que realiza a comutação de pacotes com os nós escravos. O mestre funciona como um gerente das atividades a serem computadas pelos nós escravos, delegando-as da maneira que lhe for conveniente, de modo a otimizar a computação, principalmente se os nós escravos possuírem diferentes configurações. O mestre pode também atuar no processamento das atividades.

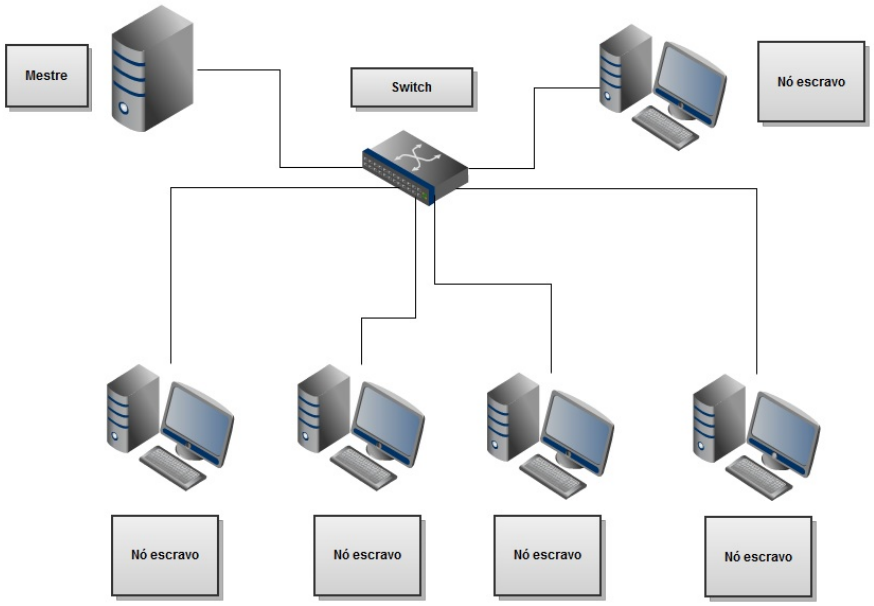

Figura 2.6: *Organização do cluster*

2.5.3 MPI - Message Passing Interface

A comunicação tanto entre núcleos de processamento, quanto entre elementos do *cluster*, em geral dá-se através da passagem de mensagens, cujo objetivo é viabilizar o gerenciamento dos processos entre os elementos de modo que todos estejam sincronizados. Para

a implementação do presente trabalho, o MPI -*Message Passing Interface* foi escolhido. Por ser um padrão desse paradigma de passagem de mensagens, as implementações utilizando MPI podem ser escritas em C, C++ ou ainda Fortran. Seu conjunto de funções e regras estão em constante desenvolvimento visando a portabilidade, desempenho e escalabilidade. Como exemplo do uso do MPI, serão apresentadas as funções básicas para a utilização, através de um código *"Hello World"* em linguagem C.

Primeiramente, o cabeçalho *mpi.h* deve estar incluído nos arquivos-fonte. A construção do código a princípio segue o padrão serial, definindo variáveis e inserindo instruções fora do ambiente paralelo. Para a inicialização do ambiente MPI, bem como para a finalização do mesmo são necessárias chamadas de funções para tal, que habilitam toda a gama de recursos que o MPI oferece e posteriormente encerram as atividades em paralelo. No algoritmo 2.1 verifica-se um exemplo de trecho de código em MPI.

```
#include <mpi.h>;
int main(int argc, char** argv) {
// Inicializando o ambiente MPI
  MPI_Init(NULL, NULL);
  // Obtendo o número de processos
  int world_size; MPI_Comm_size(MPI_COMM_WORLD, &world_size);
  // Obtendo a identificacao do processo
  int world_rank; MPI_Comm_rank(MPI_COMM_WORLD, &world_rank);
  // Obtendo o nome do processador
  char processor_name[MPI_MAX_PROCESSOR_NAME];
  int name_len;
  MPI_Get_processor_name(processor_name, &name_len);
  // Imprimindo um Hello world
  printf("Hello world do processador %s,
            identificacao %d de %d processos.\n",
            processor_name, world_rank, world_size);
  // Finalizando o ambiente MPI
  MPI_Finalize();
}
```

Algoritmo 2.1: Hello world utilizando MPI em linguagem C

Capítulo 3

Desenvolvimento

Neste capítulo são apresentados os processos detalhados sobre o desenvolvimento das implementações do presente trabalho. Primeiramente são apresentadas as bibliotecas de precisão múltipla utilizadas, seguida das implementações em ambiente de um único processador, em ambiente paralelo e em ambiente de linguagem interpretada. Com exceção do ambiente de linguagem interpretada, na qual o *software* foi desenvolvido para a plataforma *Windows*, as implementações foram realizadas utilizando o sistema operacional Linux e o compilador GCC.

3.1 Bibliotecas de precisão múltipla

Uma vez que o trabalho demanda o uso de números maiores que os limites das variáveis padrões da linguagem C, o uso de bibliotecasa de precisão múltipla se fez necessário, sendo as bibliotecas GMP [49] e MPFR [50] as que mais se adequaram às implementações.

3.1.1 GMP

A GMP (*GNU Multiple Precision Arithmetic Library*) [49] é uma biblioteca de precisão aritmética arbitrária, que opera em inteiros com sinal, números com ponto flutuante e também números racionais. O limite da precisão da biblioteca é o imposto pela memória disponível para uso na máquina que esta executando a computação.

Seu uso principal se dá principalmente em aplicações e pesquisa em criptografia, segurança, sistemas algébricos e simulações computacionais, este último no qual o presente trabalho se enquadra. Possui diversas funções que implementam algoritmos rápidos para

diversas soluções, como fatoriais e raízes com números grandes. Sua distribuição é gratuita sob a licença LGPL.

3.1.2 MPFR

A biblioteca MPFR (*Multiple Precision Floating-Point Reliable*) [50] serve de complemento para as bibliotecas padrões da linguagem C, sendo para uso de cálculos com precisão múltipla com arredondamento correto em variáveis de ponto flutuante. Tem sido apoiada continuamente no seu desenvolvimento, principalmente pelo INRIA, sendo baseada na biblioteca de precisão múltipla GMP. É distribuída de maneira gratuita, sob licença GNU LGPL versão 3 ou posterior.

O uso desta biblioteca se tornou necessário, uma vez que o presente trabalho manipula números maiores que o limite da maior variável da linguagem C, o *long double* ($1.7E\pm308$). Ela possui também implementações de melhor complexidade e maior eficiência que a biblioteca *math.h*, padrão da linguagem. Como exemplo, tem-se a função mpfr_fac_ui, que realiza operações de fatoriais em um tempo muito menor, se comparada às implementações padrões para tal em que acima de um n igual a 120, o algoritmo trivial se torna inviável.

A grande vantagem desta biblioteca é que pode-se escolher o modo como é feita a aproximação, sendo possível escolher entre: aproximação ao mais próximo, aproximação para zero, aproximação para mais infinito, arredondamento para menos infinito, ou ainda arredondamento longe de zero. Possui também grande compatibilidade com as variáveis da biblioteca GMP, uma vez que é baseada na mesma.

As pequenas peculiaridades da biblioteca estão no modo de uso das variáveis, em que se deve instanciar as mesmas e para utilizá-las é necessário primeiramente inicializar cada uma delas através de uma função da biblioteca (processo de alocação de memória para cada variável) e depois de utilizá-las, é também necessário a liberação do espaço de memória ocupado por elas.

3.2 Implementação em um ambiente de um único processador

O processo de implementação em um ambiente de um único processador consistiu na programação das entropias de Von Neumann ($S(\rho_c)$) e de Tsallis ($S_q(\rho)$), onde são

desenvolvidas as fórmulas:

$$S(\rho_c) = -\sum_{N_1 N_2} |B_{n_1 n_2}^{N_1 N_2}|^2 \; ln \; |B_{n_1 n_2}^{N_1 N_2}|^2 \tag{3.1}$$

$$S_q(\rho_c) = \frac{1}{q-1}\left(1 - \sum_{N_1 N_2} |B_{n_1 n_2}^{N_1 N_2}|^{2q}\right) \tag{3.2}$$

Assim, obtêm-se, para a implementação:

$$S = -\sum_{c=0}^{a+b}\sum_{d=0}^{a+b}(|\sum_{k=0}^{a}\sum_{l=0}^{b}((-1)^{a-k}\; r^{a+b-k-l}\; t^{k+l}\frac{\sqrt{a!b!c!d!}}{k!(a-k)!l!(b-l)!}\;\delta_{c,b+k-l}\;\;\delta_{d,a-k+l})|^2)*$$

$$*\; ln\;[|\sum_{k=0}^{a}\sum_{l=0}^{b}((-1)^{a-k}\;\; r^{a+b-k-l}\; t^{k+l}\frac{\sqrt{a!b!c!d!}}{k!(a-k)!l!(b-l)!}\;\delta_{c,b+k-l}\;\delta_{d,a-k+l})|^2 + 1\times 10^{-29}] \tag{3.3}$$

$$S_q = \frac{1}{q-1}(1 - \sum_{c=0}^{a+b}\sum_{d=0}^{a+b}(|\sum_{k=0}^{a}\sum_{l=0}^{b}((-1)^{a-k}\; r^{a+b-k-l}\; t^{k+l}\frac{\sqrt{a!b!c!d!}}{k!(a-k)!l!(b-l)!}\;\delta_{c,b+k-l}\;\delta_{d,a-k+l})|^{2q})) \tag{3.4}$$

A metodologia aplicada na execução da computação dos cálculos foi a de divisão e conquista, organizando em funções as partes componentes das fórmulas de entropia.

Toda manipulação numérica foi realizada através das variáveis da biblioteca MPFR, com exceção das variáveis auxiliares de controle de laços, para que a precisão fosse mantida, não havendo arredondamento ou truncamento padrões da linguagem C.

Para a geração dos resultados, o programa cria dois aquivos chamados *vetor-von.txt* e *vetor-tsallis.txt*, contendo um vetor linha com os valores das entropias calculados, todos em formato científico, como exemplo $1.72e320$. O número de valores existentes corresponde à precisão definida sendo, no projeto, igual a como 1000 pontos.

Para a otimização do código, a fim de se obter um menor tempo de execução do programa e que o mesmo seja moldado para utilizar as capacidades do processador onde foi compilado, as seguintes *CFLAGS* de compilação foram utilizadas [51] [52]:

CFLAG -O2: habilita as otimizações que são utilizadas por padrão com a *CFLAG* -O e todas as otimizações que não alterem o tamanho do arquivo binário final nem no *debug*. Foi escolhida essa *CFLAG* em detrimento da *CFLAG* -O3 por ela ser considerada mais segura, uma vez que o binário gerado com a *CFLAG* -O3 tem tamanho maior e fatalmente

ocuparia mais memória RAM, o que não é de interesse, pois em testes arbitrários os executáveis compilados com essa *CFLAG* apresentaram tempo de execução maior.

CFLAG -march=native: essa *CFLAG* instrui o GCC a compilar o programa para uma arquitetura específica. No caso foi usado *native* pelo fato da versão do GCC ser superior à 4.2, em que o compilador detecta automaticamente as características de arquitetura do processador em uso. O uso dessa *CFLAG* torna o programa incompatível com outras arquiteturas que sejam diferentes do local ele foi compilado.

3.3 Implementação em um ambiente paralelo

De posse da implementação realizada em ambiente de um único processador, a implementação em ambiente paralelo foi realizada de modo a acelerar a computação através do uso de mais processadores, ou processos na execução, uma vez que os cálculos das fórmulas de entropia demandam grande custo computacional.

Para alcançar o objetivo proposto, a abordagem foi também a divisão das tarefas, definida utilizando-se grão grosso, de modo que é distribuído a cada nó seu bloco correspondente ao vetor das entropias que serão calculadas. Uma vez definido esse bloco, a comunicação entre os nós escravos e o mestre se dará em apenas um outro momento, para envio dos resultados obtidos na computação dos cálculos. Deste modo buscou-se evitar com que houvesse congestionamento na rede de comunicação, pois a transmissão dos blocos é apenas uma *string* de cada nó escravo ao mestre no final da computação.

Fazendo uso da biblioteca de paralelização MPI e das bibliotecas de precisão múltipla GMP e MPFR a implementação em paralelo é ilustrada pelo fluxograma da figura 3.1. Nesta implementação o processo mestre não realiza apenas a coordenação da computação, mas também participa dos cálculos. Este processo é realizado tanto para a entropia de Von Neumann quanto para a entropia de Tsallis.

As partes componentes da implementação em paralelo são as seguintes:

- Inicialização do MPI: inicialização das variáveis de controle do MPI e definição dos *ID's* de cada nó participante da computação.

- Definição dos números de fótons de entrada: entrada de dados por parte do usuário para definir o número de fótons na entrada A e B do divisor de feixes.

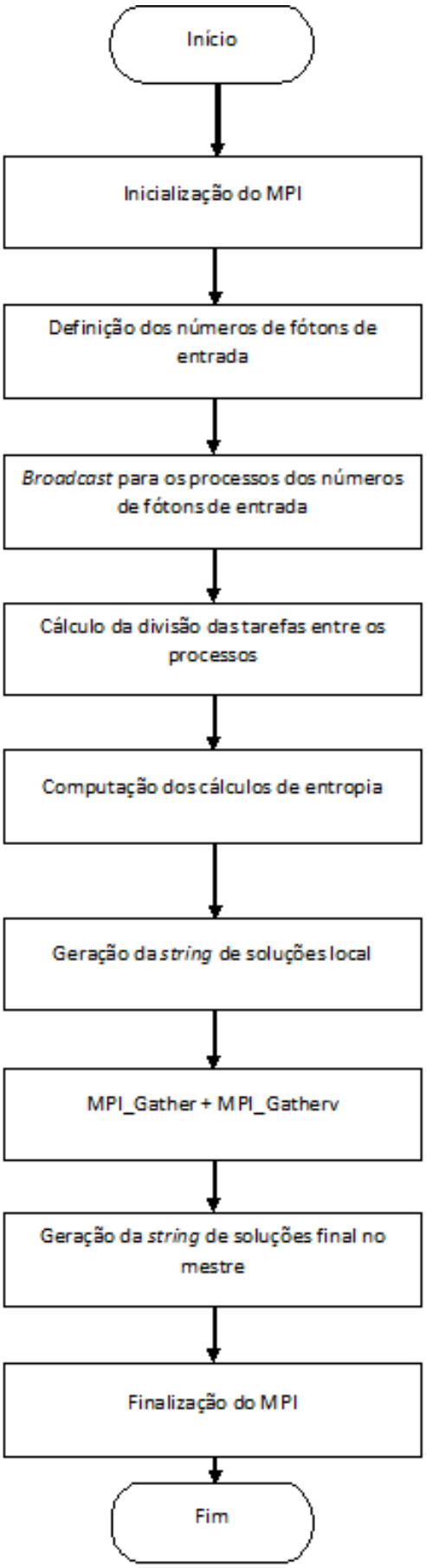

Figura 3.1: Fluxograma da implementação em paralelo.

- *Broadcast* para os processos dos números de fótons de entrada: uma vez definido o número de fótons nas entradas do divisor de feixe, é realizada a propagação desses valores para todos os processos envolvidos na computação.

- Cálculo da divisão das tarefas entre os processos: neste ponto é realizada a divisão das tarefas de computação entre os processos de modo que se o resto da divisão entre 1000 (número total de pontos do vetor de soluções) e o número de processos é igual a zero, então a divisão será igual entre cada processo envolvido. Senão, o mestre recebe adicionalmente o resto desta divisão e então blocos iguais são distribuídos entre os processos escravos. Este processo de decisão é ilustrado pelo fluxograma da figura 3.2.

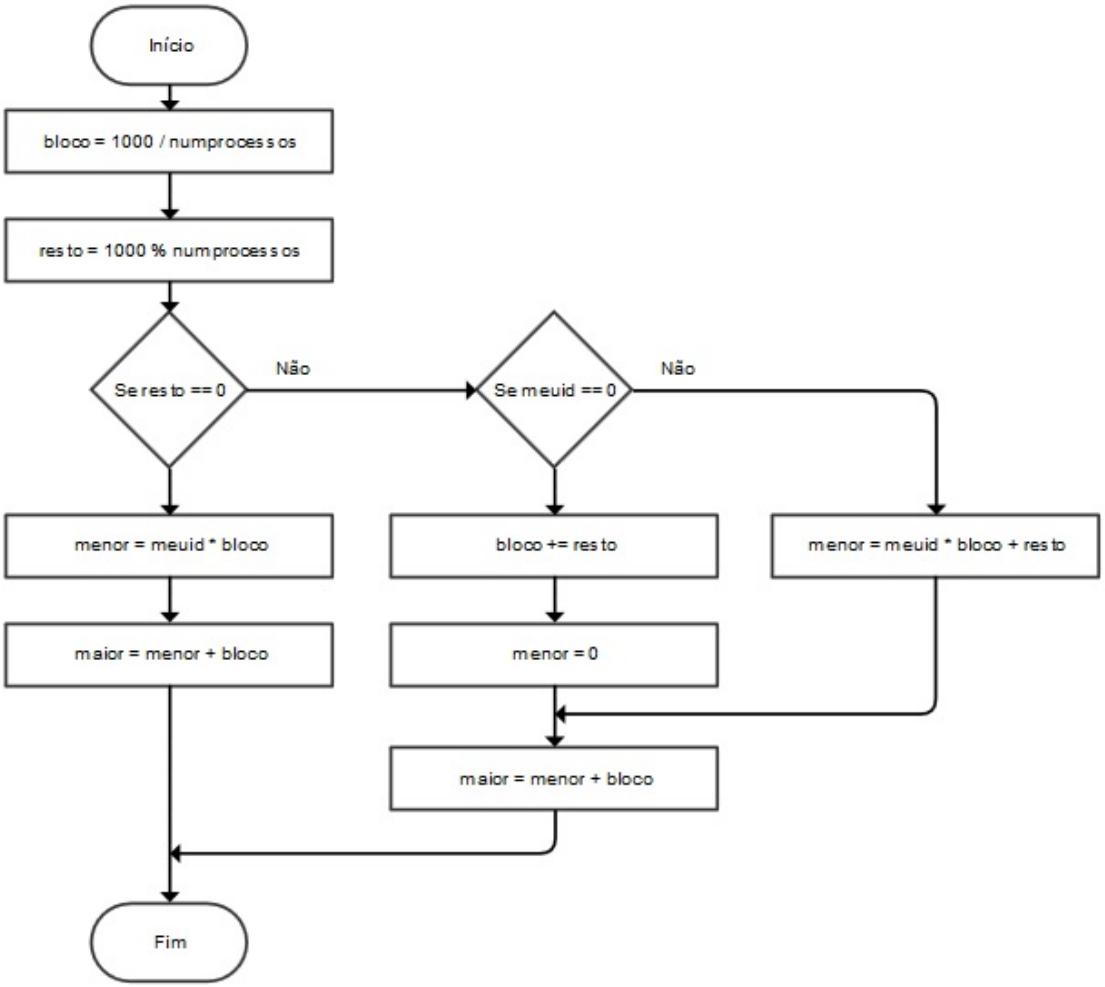

Figura 3.2: Fluxograma da cálculo da divisão das tarefas em blocos.

- Computação dos cálculos de entropia: de posse de seu respectivo bloco, os processos realizam a computação dos cálculos.

- Geração da *string* de soluções local: uma vez realizados os cálculos, cada processo gera uma *string* de soluções de seu respectivo bloco. Como os resultados obtidos em cada processo não serão utilizados em um cálculo final, a *string* que é do tipo *char*, economiza memória e trabalho computacional ao propagar para o processo mestre variáveis do tipo *mpfr_t*.

- MPI_Gather + MPI_Gatherv: neste ponto os processos geraram suas strings de solução locais e as funções da biblioteca MPI *MPI_Gather* e *MPI_Gatherv* realizam a junção dessas informações para o processo mestre, em que através da função *MPI_Gather* comunica-se o tamanho de cada *string* dos processos ao processo mestre. Além disso, a função *MPI_Gatherv* realiza de fato a junção das *strings* em uma *string* global com as soluções de cada processo, no processo mestre, conforme ilustrado pela figura 3.3.

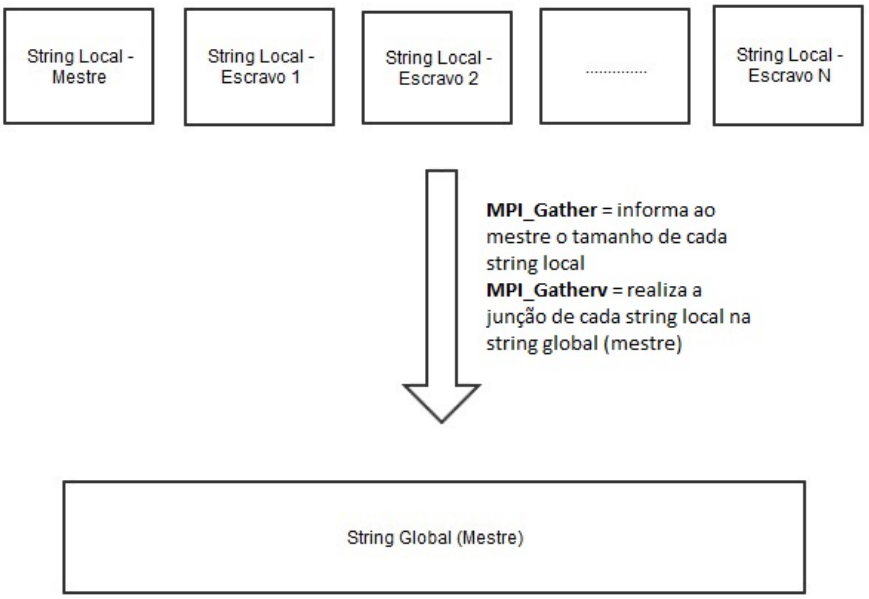

Figura 3.3: Junção das strings locais no processo mestre.

- Geração da *string* de soluções final no mestre: após a geração da *string* global de soluções é criado um arquivo no processo mestre contendo a *string* global, com os resultados da computação da entropia.

- Finalização do MPI: com o fim da computação dos cálculos e geração dos resultados, é finalizado também o MPI e o programa principal.

Para otimização de código foram utilizadas as mesmas *CFLAGS* da implementação em ambiente único: -O2 e -march=native.

3.4 Implementação em um ambiente de linguagem interpretada

Para a implementação em ambiente de linguagem interpretada, foi escolhido o *software* Mathematica®[1] por ser uma poderosa ferramenta de uso científico na resolução de problemas matemáticos. Por conta de sua versatilidade, suas aplicações são as mais diversas, tais como resolução de problemas que envolvem eletromagnetismo [53], hidrodinâmicos [54], dentre outros.

A implementação segue os padrões apresentados por Brandão [5]. Pelo fato de o *software* ser um ambiente de computação simbólica e voltado à resolução de problemas matemáticos, a implementação das fórmulas de entropia de Von Neumann e Tsallis é quase da forma de como são expressas matematicamente.

Na figura 3.4 é ilustrada a implementação das entropias realizada no *software*.

```
x = TimeUsed[]
Clear[N1, N2, n1, n2, j1, j2, r, a, b, c, d, q1, t, k, l]
Needs["PlotLegends`"]
{a = , b =, r = Sqrt[R], t = Sqrt[1 - r^2], q1 = .5}

A[c_, d_] =
 Sum[Sum[(-1)^(a-k) * r^(a+b-l-k) * t^(k+l) * Sqrt[a! * b! * c! * d!] / (k! * (a - k)! * l! * (b - l)!) * KroneckerDelta[c, b + k - l] * KroneckerDelta[d, a - k + l], {l, 0, b}], {k, 0, a}]

EntropiaTsallis = 1/(q1 - 1) * (1 - Sum[Sum[((Abs[A[c, d]])^(2 * q1)), {d, 0, a+b}], {c, 0, a+b}])

EntropiaVon = -Sum[Sum[((Abs[A[c, d]])^(2) * Log[(Abs[A[c, d]])^(2) + .00000000000000000000000000001]), {d, 0, a+b}], {c, 0, a+b}]

Plot[{EntropiaVon, EntropiaTsallis}, {R, 0, 1},
 PlotStyle → {{RGBColor[1, 0, 0]}, {RGBColor[0, 1, 0]}}, PlotLegend → {"VonNeumann", "Tsallis"},
 PlotPoints → 10001, LegendPosition → {1, -0.3}]

y = TimeUsed[]
z = (y - x)
tempo = z / 60
tempo1 = tempo / 60
tempo2 = tempo1 / 24
```

Figura 3.4: Implementação das entropias no *software* Mathematica®.

[1] www.wolfram.com/mathematica

Capítulo 4

Testes e Resultados

São apresentados neste capítulo os resultados obtidos através dos testes, realizados a partir das implementações das entropias de Von Neumann e Tsallis em ambiente de linguagem interpretada, de um único processador e em ambiente paralelo. O índice entrópico q adotado foi de $q = 0.5$ para a entropia de Tsallis e $q = 1.0$ para a entropia de Von Neumann, de modo realizar comparações com trabalhos anteriores [5, 7].

A implementação em ambiente de um único processador foi realizada utilizando-se a linguagem C juntamente com a biblioteca MPFR [50]. Em ambiente paralelo utilizou-se um ambiente simulado de *cluster* Beowulf utilizando MPI como ferramenta para a paralelização do código e execução dos programas e o GridUnesp[1], fazendo uso também da linguagem C aliada à biblioteca MPFR. Para fins de comparação com um ambiente de linguagem interpretada, será aplicada a implementação desenvolvida por Brandão[5] utilizando-se o *software* Mathematica®.

4.1 Ambientes de Teste

A configuração da máquina em que os testes foram realizados é a seguinte: Processador Intel Core i7 3610QM com 8 GB de RAM, rodando Windows 7 64 bits. O ambiente de virtualização utilizado foi o VMware Workstation, onde as máquinas virtuais possuem 768 MB de RAM e 1 núcleo do processador. Foram alocados 8 máquinas virtuais para compor o *cluster* Beowulf com rede virtual privada entre os nós.

O sistema operacional utilizado na implementação de um único processador e também

[1] www.ncc.unesp.br

da implementação em ambiente paralelo escolhido foi a versão 12.04 do Ubuntu Linux, o sistema operacional presente nos nós do *cluster* virtual é o CentOS 6.4, para demonstrar o funcionamento das implementações em diferentes ambientes. A versão utilizada do Mathematica foi a 7 *for Students*, em ambiente Windows 7.

Além dos ambientes de testes citados, utilizou-se o GridUnesp. O GridUnesp é uma estrutura em Grid de processamento distribuído, que interliga diversos recursos computacionais, atendendo áreas de pesquisa que precisam de grande poder de processamento, análise e armazenamento de dados tais como previsão do tempo, prospecção de petróleo, pesquisas genéticas, dentre outras. Possui um *cluster* central e 7 outros *clusters* secundários espalhados pelo interior do estado de São Paulo em campi da Unesp. O *cluster* central em termos de processamento possui 256 servidores, 2048 *cores*, 4096 GB de RAM e interconexão de 20 Gbps. Para armazenamento, dispõe de 36 TB via DAS de fibra óptica e mais 96 TB em 4 servidores. Os *clusters* secundários possuem 16 servidores de processamento, 128 *cores* e 12 TB para armazenamento.

4.2 Metodologia dos Testes

A execução da implementação em ambiente de um único processador foi realizada em uma das máquinas virtuais com as características citadas anteriormente. Os testes foram divididos em três partes, de acordo com as possibilidades de variação das entradas, sendo elas de entrada única, entradas iguais e entradas diferentes.

De modo a comparar a execução de cada teste, foi aferido o tempo médio de todos os testes realizados e em cada ambiente, ao se realizar 10 execuções dos mesmos, totalizando 990 execuções. Para medir o ganho que a computação paralela pode oferecer, foram realizados os mesmos testes de entrada com diferentes números de processadores e aferidos seus tempos médios de execução.

Para demonstração da execução da implementação em ambiente paralelo em sistemas de grande porte foram realizados testes com 8 processos no GridUnesp[2].

[2]http://www.ncc.unesp.br

4.3 Entrada Única

Nestes testes foram injetados fótons em apenas uma das entradas, deixando a outra apenas com vácuo. Considerando a a primeira entrada de feixes de luz do divisor e b a segunda, os testes de entrada foram $a = 50$ e $b = 0$; $a = 100$ e $b = 0$; e, por fim, $a = 200$ e $b = 0$. Os resultados dos testes de tempo de execução são representados no gráfico da figura 4.1 nas tabelas 4.1, 4.2 e 4.3.

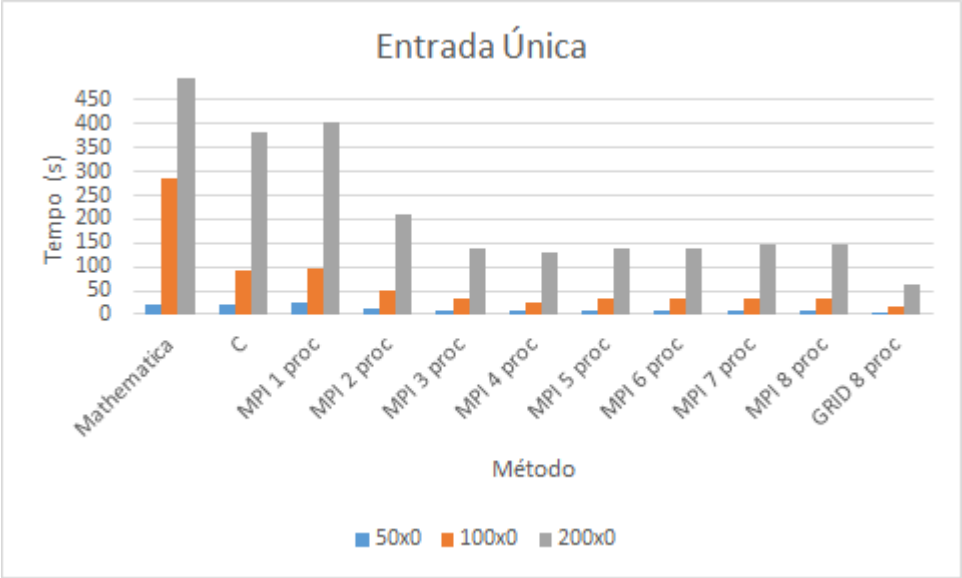

Figura 4.1: *Gráfico dos tempos médios de execução dos testes de entrada única.*

Tabela 4.1: Resultados dos tempos dos testes de entrada a = 50 e b = 0.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
	Mathematica	22.121	22.495	22.2782	0.122191
	C	21.53	22.54	22.035	0.330395
	MPI 1 proc	23.75799	25.51244	24.52898	0.506131
	MPI 2 procs	12.66668	12.92216	12.77766	0.104285
	MPI 3 procs	8.565164	8.727928	8.616078	0.047897
50x0	MPI 4 procs	6.518887	6.729202	6.605086	0.076096
	MPI 5 procs	6.814352	7.935361	7.475042	0.381974
	MPI 6 procs	7.045956	7.899927	7.358571	0.271298
	MPI 7 procs	6.929017	7.311013	7.078625	0.119343
	MPI 8 procs	6.798572	6.926121	6.863599	0.039710
	GRID 8 procs	3.814215	3.921296	3.849235	0.034597

Tabela 4.2: Resultados dos tempos dos testes de entrada a = 100 e b = 0.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
	Mathematica	282.378	285.389	284.2716	1.074244
	C	89.37	94.64	92.835	1.570762
	MPI 1 proc	94.37166	100.5929	97.79794	1.751744
	MPI 2 procs	49.84929	50.6702	50.27036	0.253935
	MPI 3 procs	33.51779	33.85355	33.665	0.141376
100x0	MPI 4 procs	25.34951	26.06585	25.5854	0.220409
	MPI 5 procs	27.23672	35.79004	32.60348	2.383496
	MPI 6 procs	30.96303	36.50023	32.87637	1.785179
	MPI 7 procs	31.80271	36.72182	33.76191	1.461049
	MPI 8 procs	32.24638	35.40561	33.90395	0.908912
	GRID 8 procs	15.078787	15.413593	15.135632	0.100597

Tabela 4.3: Resultados dos tempos dos testes de entrada a = 200 e b = 0.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
200x0	Mathematica	4152.23	4787.45	4288.289	198.110100
	C	377.93	386.63	382.75	2.978329
	MPI 1 proc	397.684	412.6994	404.5434	4.955585
	MPI 2 procs	207.1156	218.1923	209.1351	3.259151
	MPI 3 procs	137.6378	140.4448	139.2108	0.913487
	MPI 4 procs	128.022	138.5526	132.2241	3.515122
	MPI 5 procs	135.3741	144.9653	139.2318	3.133224
	MPI 6 procs	137.721	143.0915	140.2066	1.589672
	MPI 7 procs	143.5763	148.3288	145.7957	1.378022
	MPI 8 procs	141.7451	150.1475	146.1996	2.544109
	GRID 8 procs	63.010436	64.593241	63.448046	0.464636

Percebe-se, pelos resultados, um ganho considerável dos tempos de execução em ambientes paralelos

Foram compilados gráficos para ilustrar os tempos de execução de cada teste em particular, representados nas figuras 4.2, 4.3 e 4.4.

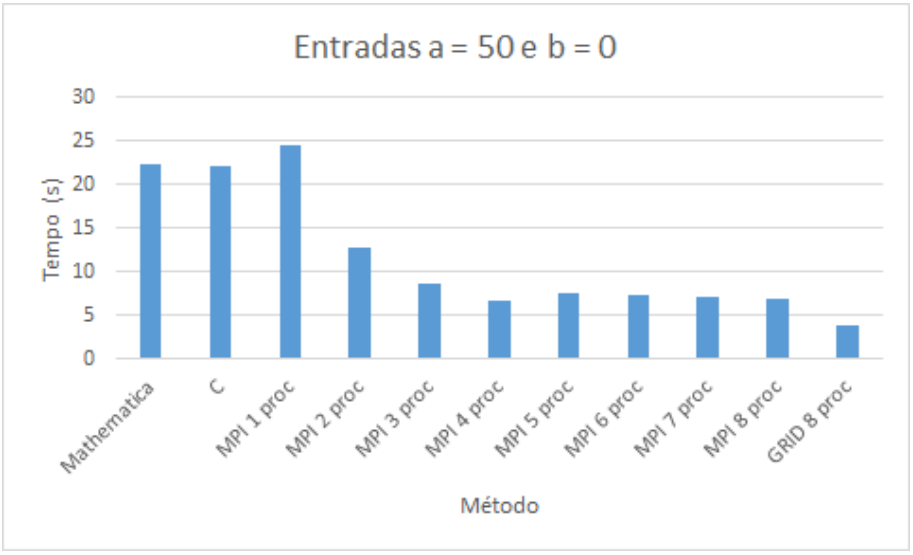

Figura 4.2: *Gráfico comparativo de tempo médio entre as implementações (50x0).*

O desvio padrão obtido através dos testes de entrada $a = 50$ e $b = 0$ no Mathematica

foi de 0.122191, para o ambiente de um único processador (C) de 0.330395 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 0.506131. O desvio padrão para o Grid nessas configurações foi de 0.034597.

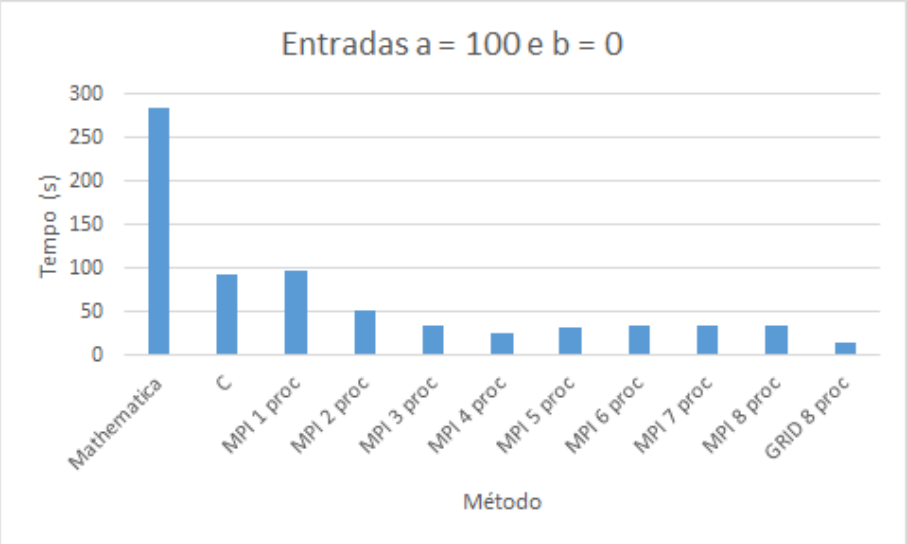

Figura 4.3: *Gráfico comparativo de tempo médio entre as implementações (100x0).*

O desvio padrão obtido através dos testes de entrada $a = 100$ e $b = 0$ no Mathematica foi de 1.074244, para o ambiente de um único processador (C) de 1.570762 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 1.751744. O desvio padrão para o Grid nessas configurações foi de 0.100597.

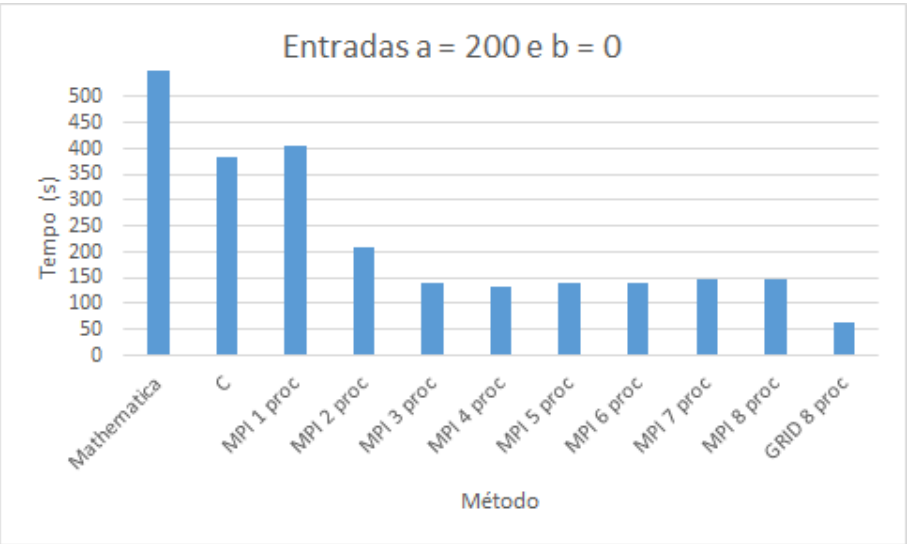

Figura 4.4: *Gráfico comparativo de tempo médio entre as implementações (200x0).*

O desvio padrão obtido através dos testes de entrada $a = 200$ e $b = 0$ no Mathematica

foi de 198.1101, para o ambiente de um único processador (C) de 2.978329 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 4.955585. O desvio padrão para o Grid nessas configurações foi de 0.464636.

Verifica-se que o desvio padrão obtido, nos ambientes compilados são menores que no ambiente interpretado. Isto indica uma maior estabilidade dos resultados. Além disso, verifica-se que conforme o tempo de execução aumenta, o desvio padrão também aumenta, o que é esperado, pois torna-se mais complexo manter tempos próximos em ambientes compartilhados. Por fim, verifica-se que as simulações adequam-se aos índices de escalabilidade, reduzindo o tempo de processamento de maneira significativa quando executadas em ambientes de grande porte, como é o caso do GridUnesp.

4.3.1 Comparativos de tempo

De posse dos dados de tempo de execução, foram compilados gráficos de comparação dos tempos de execução de cada teste entre a implementação em ambiente de linguagem interpretada (Mathematica) e a implementação em ambiente paralelo (MPI) e entre a implementação em ambiente de um único processador (C) e a implementação em ambiente paralelo (MPI).

Primeiramente foram comparados os tempos de execução de cada teste entre a implementação em ambiente de linguagem interpretada (Mathematica) e a implementação em ambiente paralelo (MPI), ilustrados pelas figuras 4.5, 4.6 e 4.7. Nota-se que a diferença de tempo entre a execução do teste em ambiente de linguagem interpretada e o teste utilizando apenas um processador em ambiente paralelo é consideravelmente menor, com exceção do primeiro teste de entradas $a = 50$ e $b = 0$.

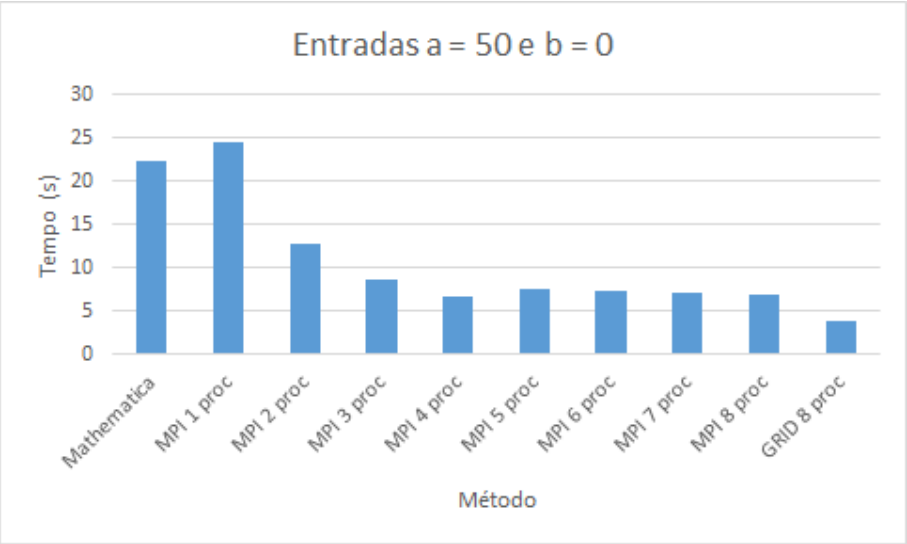

Figura 4.5: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (50x0).*

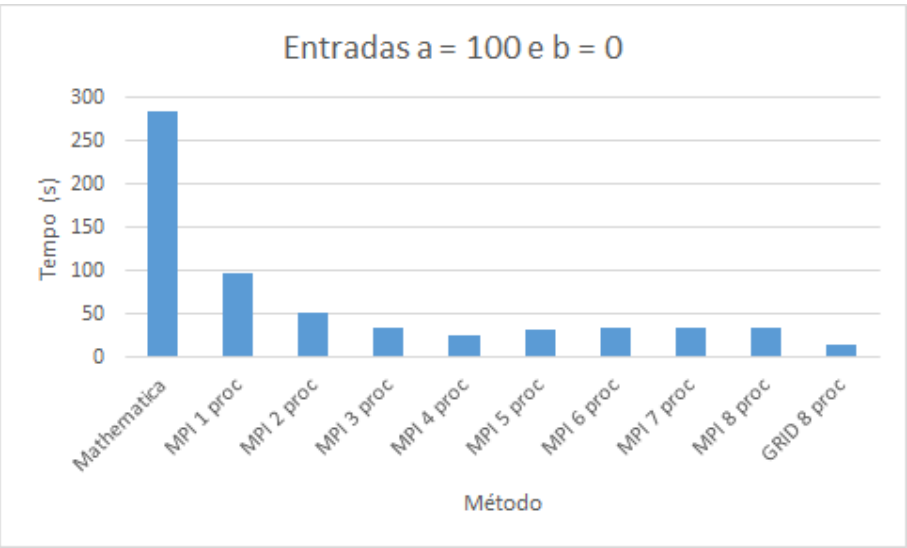

Figura 4.6: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (100x0).*

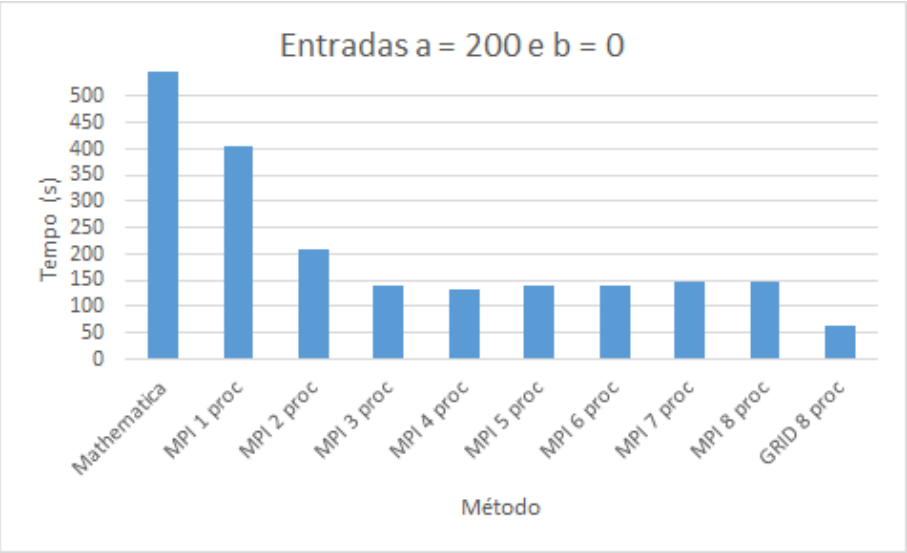

Figura 4.7: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (200x0).*

Posteriormente foram comparados os tempos de execução de cada teste entre a implementação em ambiente de um único processador (C) e a implementação em ambiente paralelo, ilustrados pelas figuras 4.8, 4.9 e 4.10. Nota-se que os resultados dos testes da implementação em ambiente paralelo utilizando apenas um processo mostram que a comunicação entre os nós gerada pelo MPI acaba por tornar o teste mais lento que a implementação em ambiente de um único processador, sendo seu ganho aparente apenas com 2 ou mais processadores, momento em que acontece o paralelismo de fato.

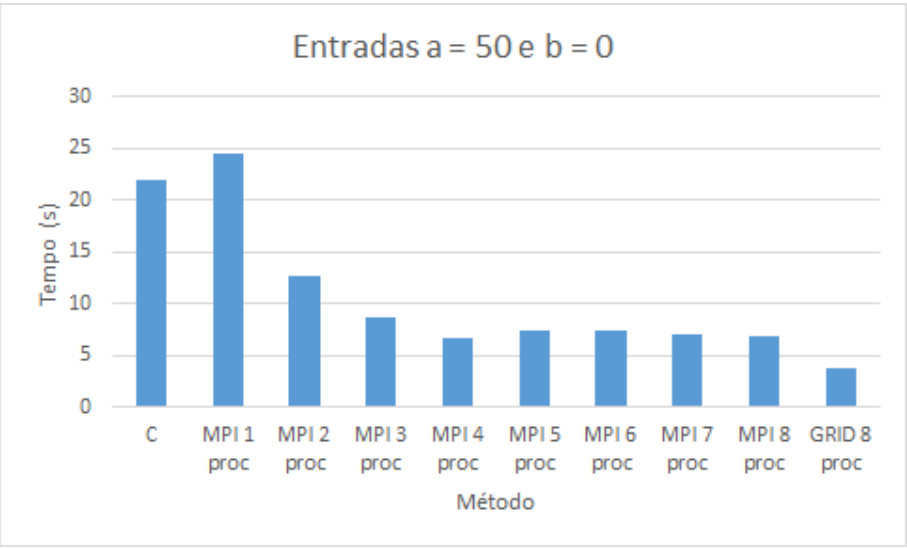

Figura 4.8: *Gráfico comparativo de tempo médio C x MPI x Grid (50x0).*

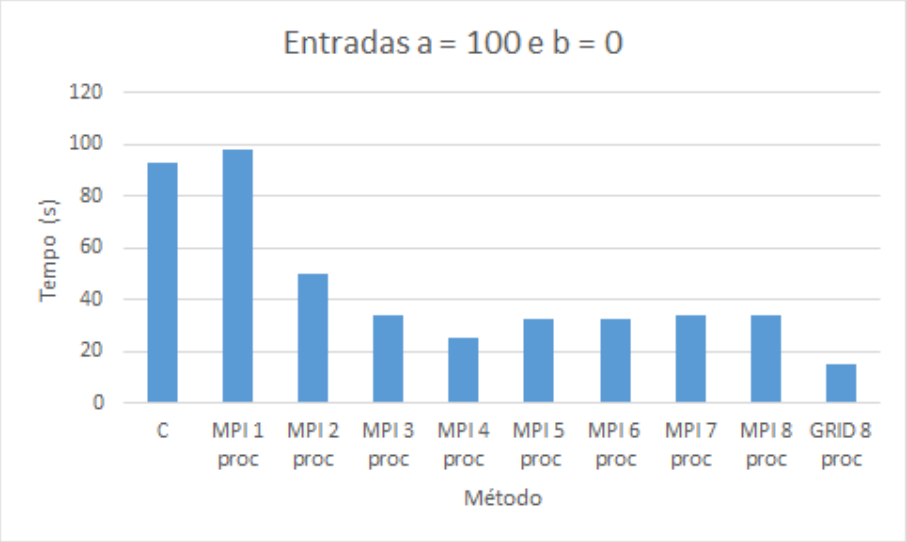

Figura 4.9: *Gráfico comparativo de tempo médio C x MPI x Grid (100x0).*

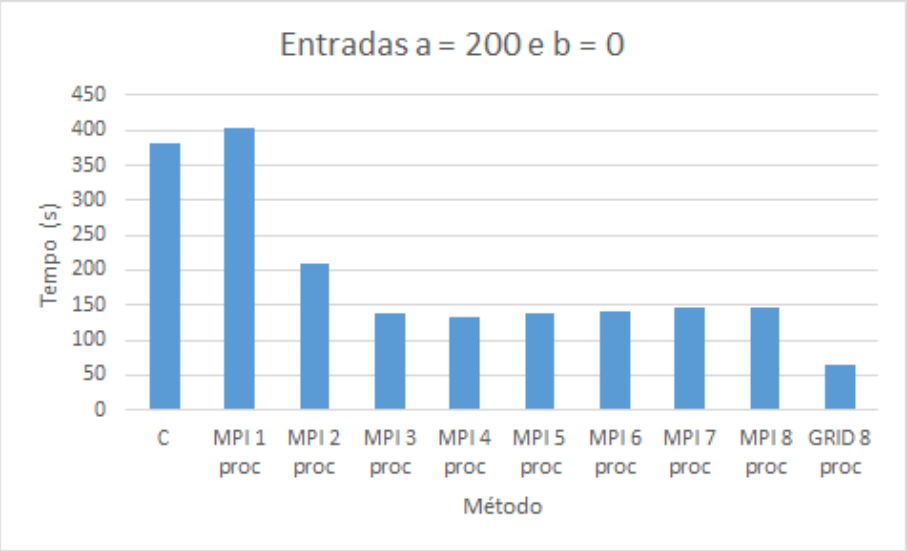

Figura 4.10: *Gráfico comparativo de tempo médio C x MPI x Grid (200x0).*

Novamente, verifica-se que no ambiente de grande porte do GridUnesp a redução é mais efetiva o que corrobora a importância do paralelismo para as simulações.

4.4 Entradas iguais

Nestes testes foram injetados fótons em número iguais em ambas as entradas. Os testes de entrada foram $a = b = 5$; $a = b = 10$; e, por fim, $a = b = 50$. Os resultados dos testes são representados no gráfico da figura 4.11 e nas tabelas 4.4, 4.5 e 4.6.

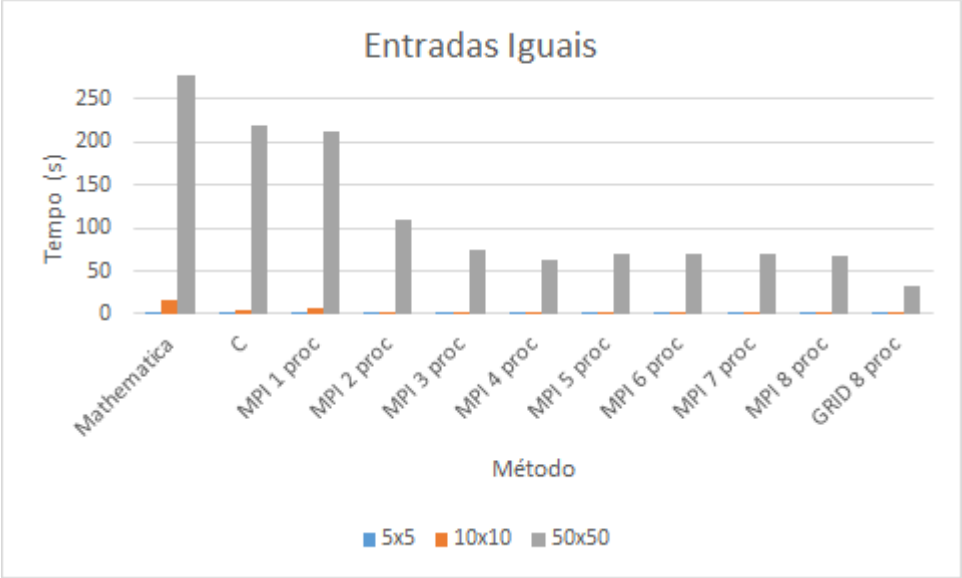

Figura 4.11: *Gráfico dos tempos médios de execução dos testes de entradas iguais.*

Tabela 4.4: Resultados dos tempos dos testes de entradas a = 5 e b = 5.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
5x5	Mathematica	2.917	3.104	3.0031	0.056573
	C	1.21	1.79	1.277	0.180373
	MPI 1 proc	1.69491	1.723708	1.707975	0.01004
	MPI 2 procs	0.922238	0.979831	0.948257	0.018742
	MPI 3 procs	0.65909	0.721097	0.676184	0.017649
	MPI 4 procs	0.517065	0.560093	0.540214	0.012486
	MPI 5 procs	0.603139	0.777012	0.695986	0.058088
	MPI 6 procs	0.605483	0.656598	0.627219	0.013493
	MPI 7 procs	0.557028	0.604608	0.576647	0.013258
	MPI 8 procs	0.531915	0.546119	0.539495	0.004849
	GRID 8 procs	0.282745	0.3032	0.289979	0.007775

Tabela 4.5: Resultados dos tempos dos testes de entradas a = 10 e b = 10.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
10x10	Mathematica	15.772	16.521	16.0402	0.231077
	C	4.67	4.79	4.716	0.042479
	MPI 1 proc	6.064007	6.264947	6.12635	0.064417
	MPI 2 procs	3.171024	3.257245	3.212049	0.031736
	MPI 3 procs	2.224513	2.32595	2.252964	0.031025
	MPI 4 procs	1.727054	1.890827	1.779885	0.045806
	MPI 5 procs	1.862814	2.199547	2.005396	0.124703
	MPI 6 procs	1.860444	2.042996	1.96026	0.052902
	MPI 7 procs	1.789922	1.91429	1.846586	0.040394
	MPI 8 procs	1.733032	1.786571	1.753718	0.015484
	GRID 8 procs	0.999232	1.027202	1.009124	0.009938

Tabela 4.6: Resultados dos tempos dos testes de entradas a = 50 e b = 50.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
50x50	Mathematica	5259.11	8788.64	7939.933	1411.262
	C	214.66	223.21	218.658	2.552758
	MPI 1 proc	209.4138	216.5459	213.2277	2.541348
	MPI 2 procs	109.3125	111.9508	110.1675	0.744634
	MPI 3 procs	73.3741	74.52104	73.79565	0.336083
	MPI 4 procs	58.5798	64.62433	62.88106	1.845962
	MPI 5 procs	67.15819	71.51021	69.03555	1.605007
	MPI 6 procs	68.15883	74.32248	71.05551	1.942128
	MPI 7 procs	66.92143	71.75623	69.19742	1.391925
	MPI 8 procs	66.67991	69.79805	68.13928	1.203162
	GRID 8 procs	32.598828	32.802472	32.656217	0.058644

Da mesma forma que os testes com entrada única, foram compilados os gráficos para ilustrar os tempos de execução de cada teste em particular, representados nas figuras 4.12, 4.13 e 4.14.

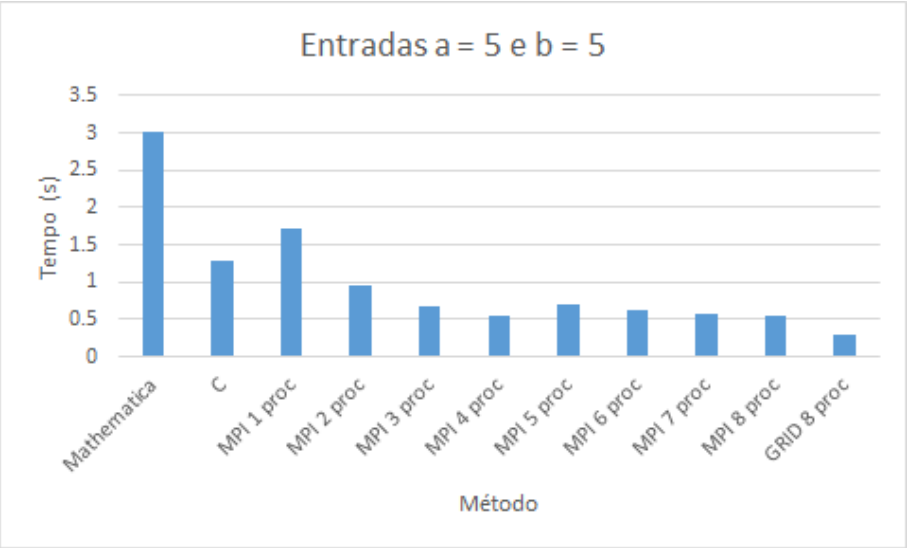

Figura 4.12: *Gráfico comparativo de tempo médio entre as implementações (5x5).*

O desvio padrão obtido através dos testes de entrada $a = 5$ e $b = 5$ no Mathematica foi de 0.056573, para o ambiente de um único processador (C) de 0.180373 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 0.058088. O desvio padrão para o Grid nessas configurações foi de 0.007775.

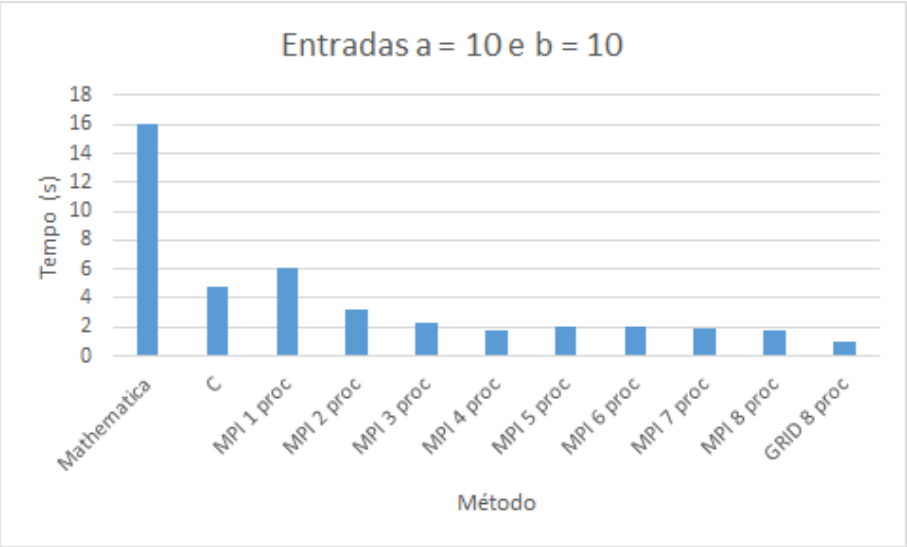

Figura 4.13: *Gráfico comparativo de tempo médio entre as implementações (10x10).*

O desvio padrão obtido através dos testes de entrada $a = 10$ e $b = 10$ no Mathematica foi de 0.231077, para o ambiente de um único processador (C) de 0.042479 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 0.124703. O desvio padrão para o Grid nessas configurações foi de 0.009938.

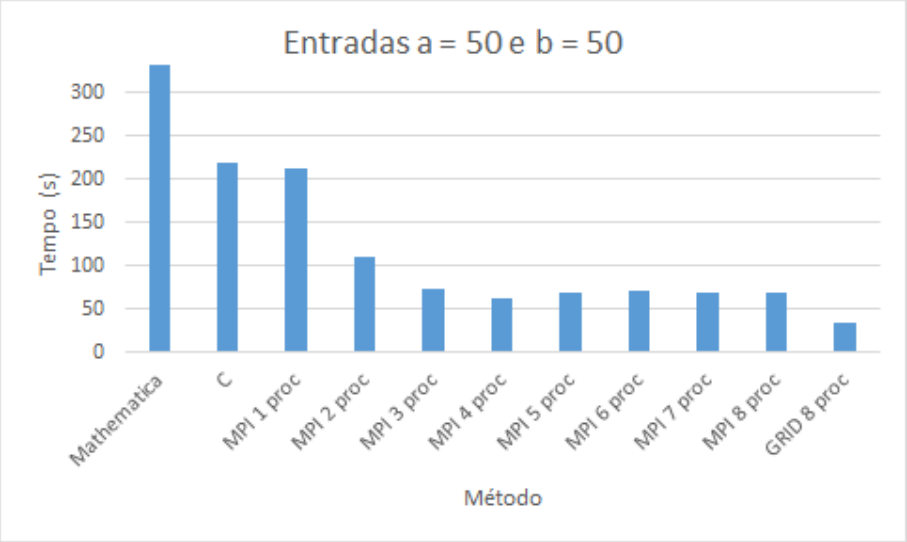

Figura 4.14: *Gráfico comparativo de tempo médio entre as implementações (50x50).*

O desvio padrão obtido através dos testes de entrada $a = 50$ e $b = 50$ no Mathematica foi de 1411.262, para o ambiente de um único processador (C) de 2.552758 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 2.541348. O desvio padrão para o Grid nessas configurações foi de 0.058644.

Novamente, verifica-se que as simulações adequam-se aos índices de escalabilidade, reduzindo o tempo de processamento de maneira significativa quando executadas em ambientes de grande porte, como é o caso do GridUnesp.

4.4.1 Comparativos de tempo

De posse dos dados de tempo foram compilados gráficos de comparação dos tempos de execução de cada teste entre a implementação em ambiente de linguagem interpretada (Mathematica) e a implementação em ambiente paralelo (MPI) e entre a implementação em ambiente de um único processador (C) e a implementação em ambiente paralelo (MPI).

Foram comparados os tempos de execução de cada teste entre a implementação em ambiente de linguagem interpretada (Mathematica) e a implementação em ambiente paralelo (MPI), ilustrados nas figuras 4.15, 4.16 e 4.17.

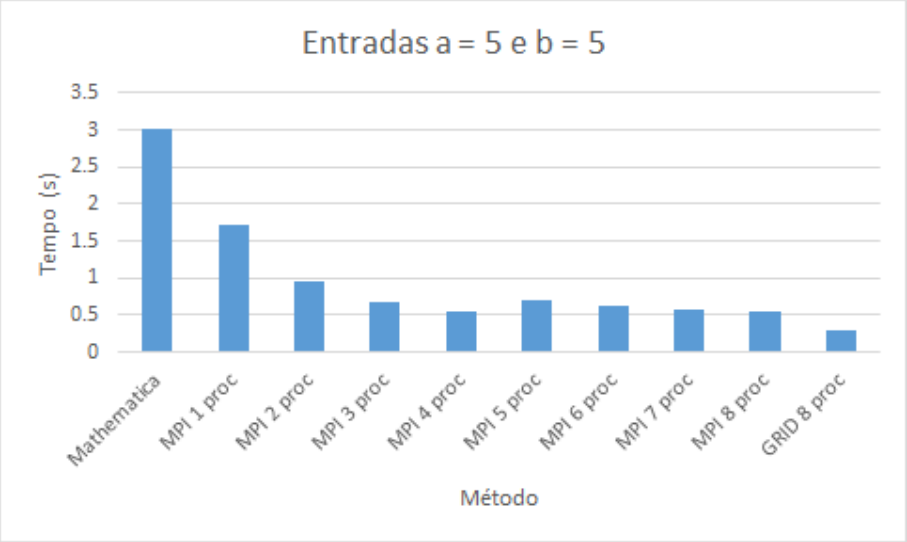

Figura 4.15: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (5x5).*

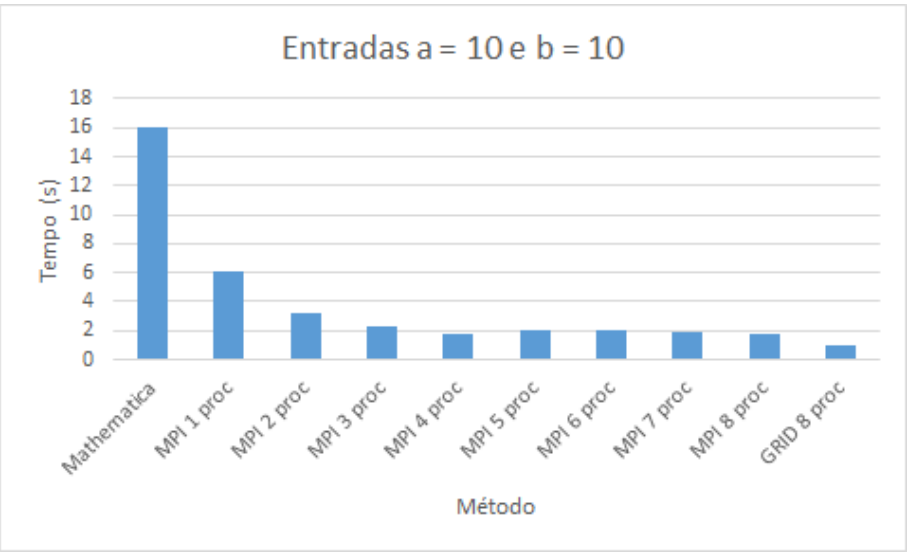

Figura 4.16: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (10x10).*

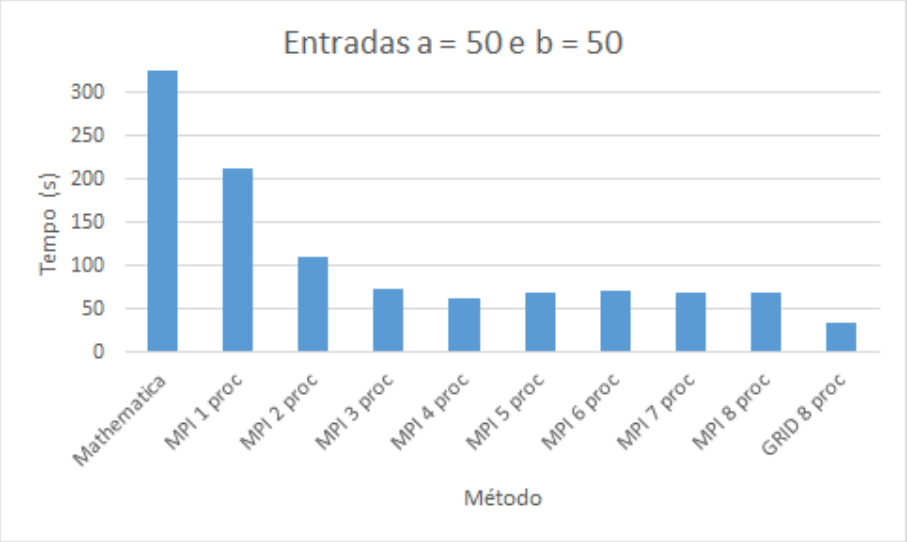

Figura 4.17: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (50x50).*

Assim como nos testes de entrada única, a diferença tempos de execução dos testes em ambiente paralelo é considerável, sendo nos testes de entradas iguais a diferença é ainda maior.

Posteriormente, foram comparados os tempos de execução de cada teste entre a implementação em ambiente de um único processador (C) e a implementação em ambiente paralelo, ilustrados nas figuras 4.18, 4.19 e 4.20.

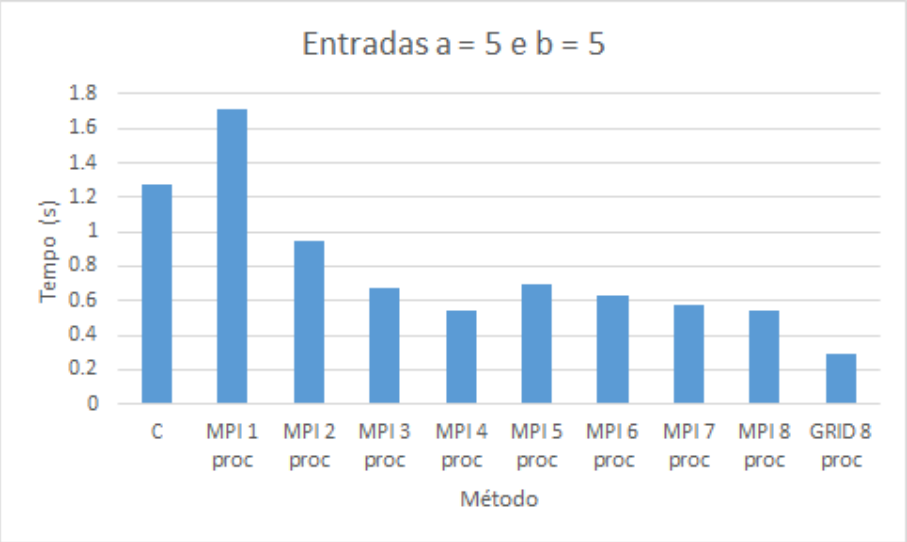

Figura 4.18: *Gráfico comparativo de tempo médio C x MPI x Grid (5x5).*

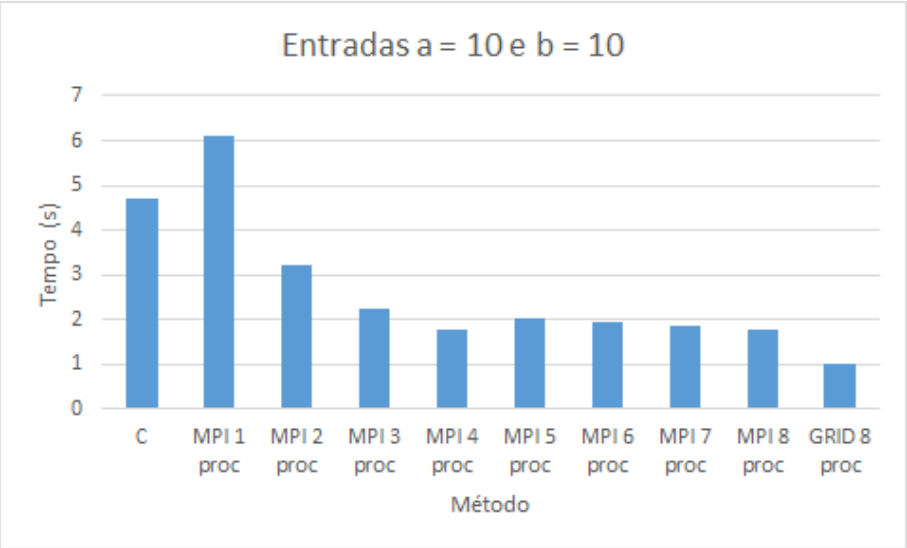

Figura 4.19: *Gráfico comparativo de tempo médio C x MPI x Grid (10x10).*

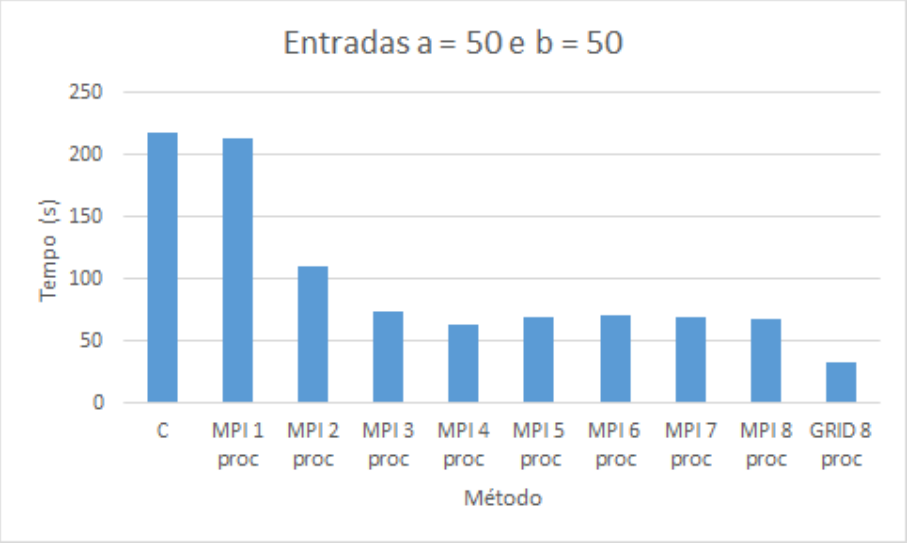

Figura 4.20: *Gráfico comparativo de tempo médio C x MPI x Grid (50x50).*

Nota-se mais uma vez que a comunicação gerada pelo MPI na implementação em ambiente paralelo acaba por deixar o teste com apenas um processador mais lento que a implementação em ambiente de um único processador. Nestes testes, também verifica-se que no ambiente de grande porte do GridUnesp a redução é mais efetiva, reforçando o papel do paralelismo para as simulações.

4.5 Entradas diferentes

Nestes testes foram injetados fótons com números diferentes nas entradas do divisor de feixe. Os testes de entrada foram $a = 3$ e $b = 7$; $a = 4$ e $b = 16$; e, por fim, $a = 30$ e $b = 70$. Os resultados dos testes são representados no gráfico da figura 4.21 e nas tabelas 4.7, 4.8 e 4.9.

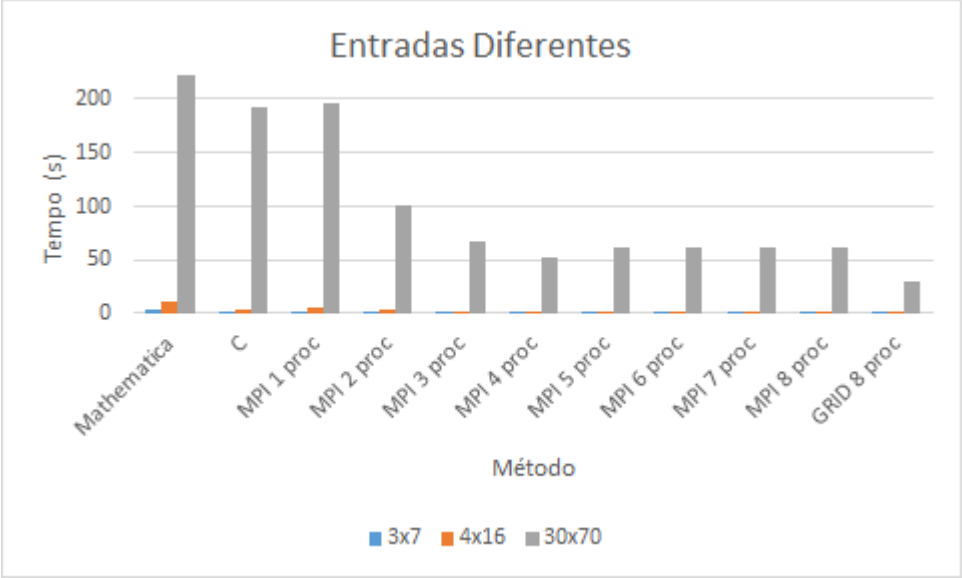

Figura 4.21: *Gráfico dos tempos médios de execução dos testes de entradas diferentes.*

Tabela 4.7: Resultado dos tempos dos testes de entradas $a = 3$ e $b = 7$.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
	Mathematica	2.933	3.073	3.0044	0.046027
	C	1.17	1.2	1.185	0.011785
	MPI 1 proc	1.618638	1.641781	1.62995	0.006761
	MPI 2 procs	0.889876	0.922427	0.902168	0.011677
	MPI 3 procs	0.631378	0.668745	0.651592	0.013085
3x7	MPI 4 procs	0.489222	0.581807	0.527364	0.029762
	MPI 5 procs	0.558719	0.703877	0.630151	0.05343
	MPI 6 procs	0.566441	0.645703	0.606498	0.021946
	MPI 7 procs	0.533195	0.572684	0.550879	0.012884
	MPI 8 procs	0.50672	0.587828	0.524161	0.022893
	GRID 8 procs	0.270856	0.284832	0.276719	0.004808

Tabela 4.8: Resultado dos tempos dos testes de entradas $a = 4$ e $b = 16$.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
	Mathematica	11.513	11.919	11.7267	0.128851
	C	4.25	4.45	4.322	0.065286
	MPI 1 proc	5.366068	5.496012	5.420703	0.044744
	MPI 2 procs	2.822336	2.918722	2.857296	0.037959
	MPI 3 procs	1.976626	2.030716	2.000766	0.019287
4x16	MPI 4 procs	1.519061	1.633113	1.567603	0.032646
	MPI 5 procs	1.646359	2.227276	1.96595	0.195503
	MPI 6 procs	1.7137	1.830334	1.771319	0.045199
	MPI 7 procs	1.602246	1.727257	1.64526	0.034117
	MPI 8 procs	1.552789	1.58993	1.569549	0.010227
	GRID 8 procs	0.879943	0.917273	0.892138	0.014299

Tabela 4.9: Resultado dos tempos dos testes de entradas $a = 30$ e $b = 70$.

Entrada	Método	Menor(s)	Maior(s)	Média(s)	Desvio Padrão
	Mathematica	6006.69	3123.04	6038.825	36.4701
	C	190.51	195.65	193.229	1.580298
	MPI 1 proc	193.8151	198.8308	196.3671	1.741283
	MPI 2 procs	99.82037	101.1274	100.5761	0.447448
	MPI 3 procs	67.14486	68.9188	67.8536	0.676196
30x70	MPI 4 procs	50.7155	55.16124	52.87432	1.386685
	MPI 5 procs	60.22487	64.04141	62.46058	1.141942
	MPI 6 procs	59.05384	65.51522	62.40736	2.198806
	MPI 7 procs	60.29727	63.74425	62.45311	1.110845
	MPI 8 procs	60.25423	62.29055	61.32264	0.691321
	GRID 8 procs	30.519916	31.148046	30.725815	0.1939854

Da mesma forma que os testes com entradas iguais, foram compilados os gráficos para ilustrar os tempos de execução de cada teste em particular, representados nas figuras 4.22, 4.23 e 4.24.

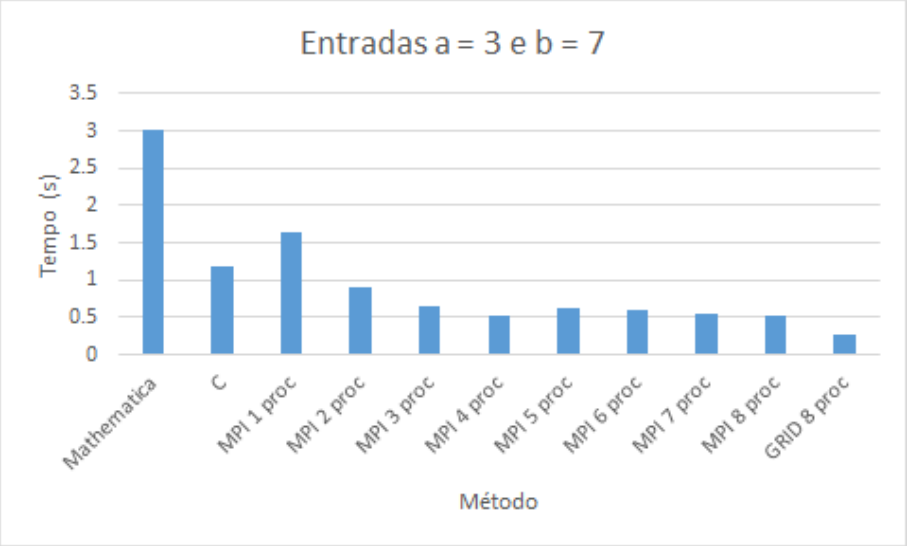

Figura 4.22: *Gráfico comparativo de tempo médio entre as implementações (3x7).*

O desvio padrão obtido através dos testes de entrada $a = 3$ e $b = 7$ no Mathematica foi de 0.046027, para o ambiente de um único processador (C) de 0.011785 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 0.05343. O desvio padrão para o Grid nessas configurações foi de 0.004808.

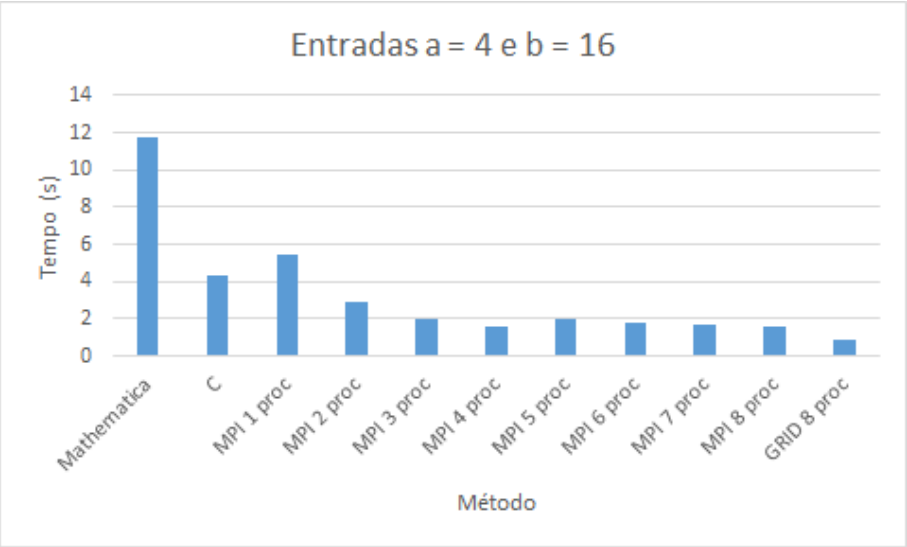

Figura 4.23: *Gráfico comparativo de tempo médio entre as implementações (4x16).*

O desvio padrão obtido através dos testes de entrada $a = 4$ e $b = 16$ no Mathematica foi de 0.128851, para o ambiente de um único processador (C) de 0.065286 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 0.195503. O desvio padrão para o Grid nessas configurações foi de 0.014299.

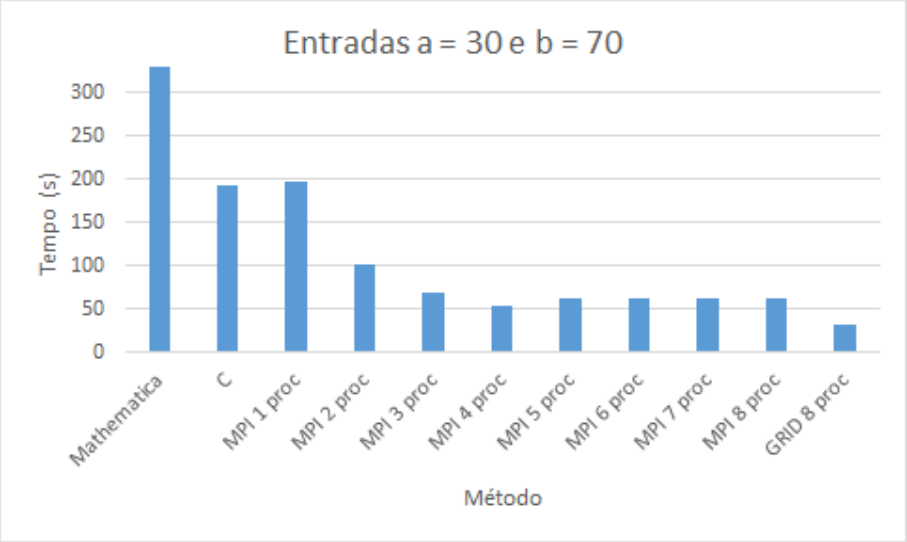

Figura 4.24: *Gráfico comparativo de tempo médio entre as implementações (30x70).*

O desvio padrão obtido através dos testes de entrada $a = 30$ e $b = 70$ no Mathematica foi de 36.4701, para o ambiente de um único processador (C) de 1.580298 e o desvio máximo apresentado pela implementação em ambiente paralelo foi de 2.198806. O desvio padrão para o Grid nessas configurações foi de 0.1939854.

Neste último bloco de testes, da mesma forma, verifica-se que a escalabilidade das simulações no GridUnesp.

4.5.1 Comparativos de tempo

De posse dos dados de tempo foram compilados gráficos de comparação dos tempos de execução de cada teste entre a implementação em ambiente de linguagem interpretada (Mathematica) e a implementação em ambiente paralelo (MPI) e entre a implementação em ambiente de um único processador (C) e a implementação em ambiente paralelo (MPI).

Foram comparados os tempos de execução de cada teste entre a implementação em ambiente de linguagem interpretada (Mathematica) e a implementação em ambiente paralelo (MPI), ilustrados nas figuras 4.25, 4.26 e 4.27.

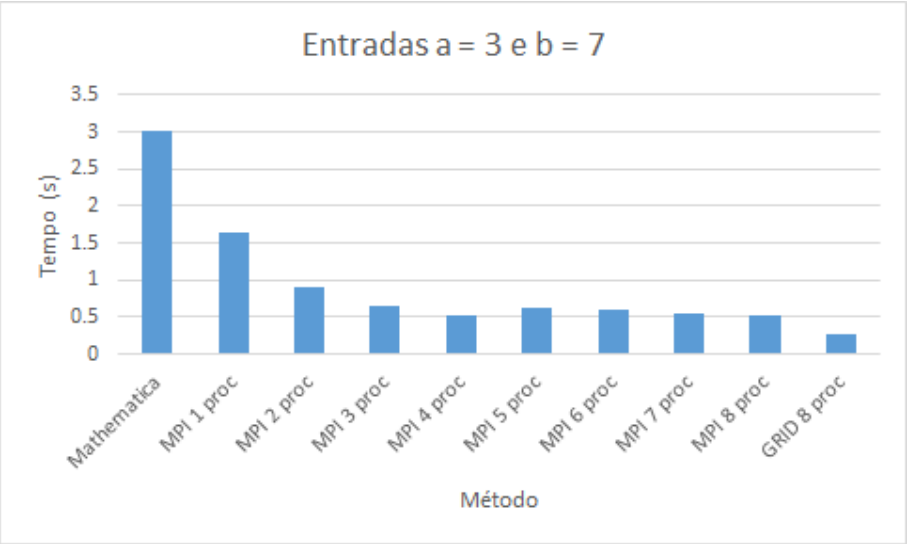

Figura 4.25: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (3x7).*

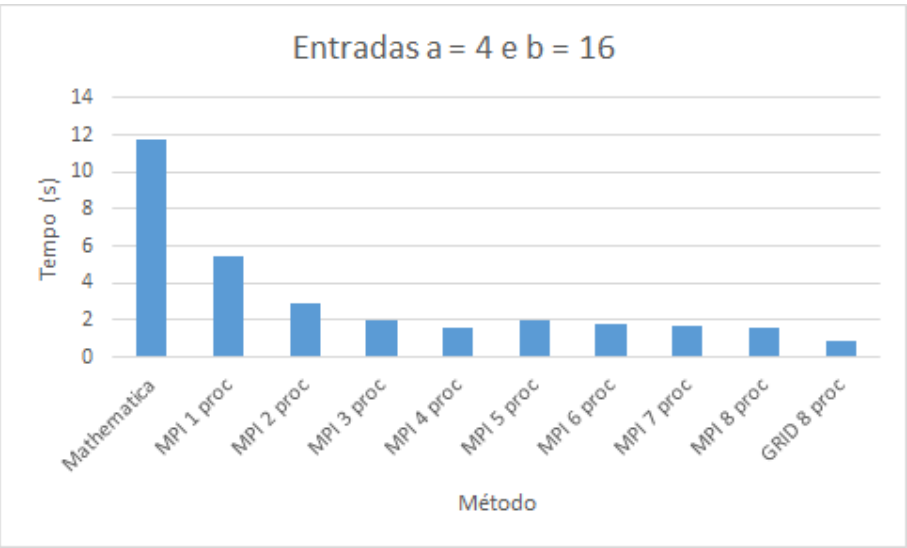

Figura 4.26: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (4x16).*

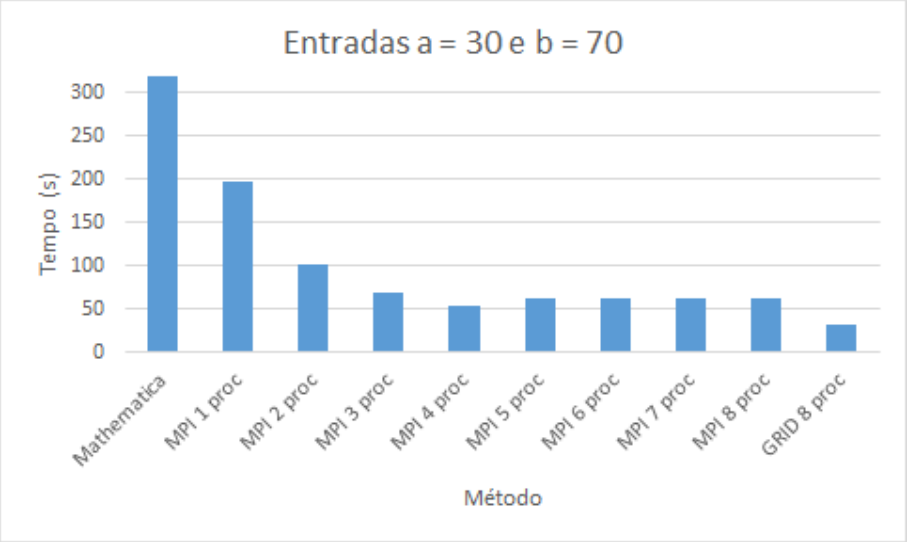

Figura 4.27: *Gráfico comparativo de tempo médio Mathematica x MPI x Grid (30x70).*

Através destes resultados confirma-se a tendência de que os testes em ambiente paralelo oferecem uma execução de tempo significativamente menor que os testes em ambiente de linguagem interpretada, a partir do aumento do número de fótons nas entradas.

Por fim, foram comparados os tempos de execução de cada teste entre a implementação em ambiente de um único processador (C) e a implementação em ambiente paralelo, ilustrados nas figuras 4.28, 4.29 e 4.30.

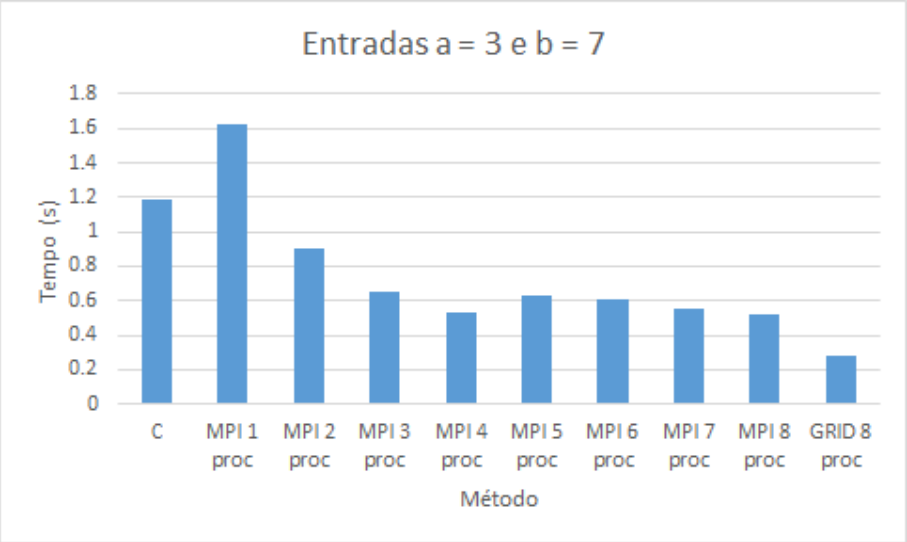

Figura 4.28: *Gráfico comparativo de tempo médio C x MPI x Grid (3x7).*

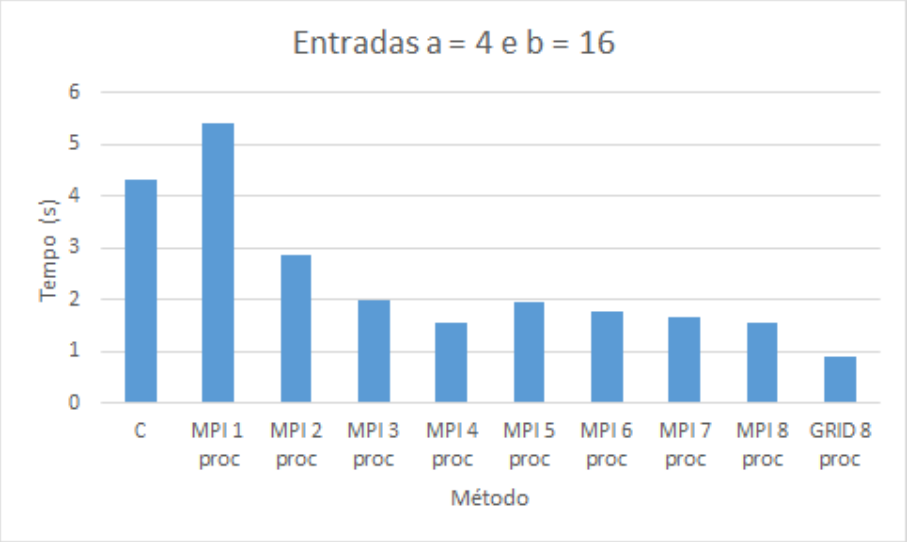

Figura 4.29: *Gráfico comparativo de tempo médio C x MPI x Grid (4x16).*

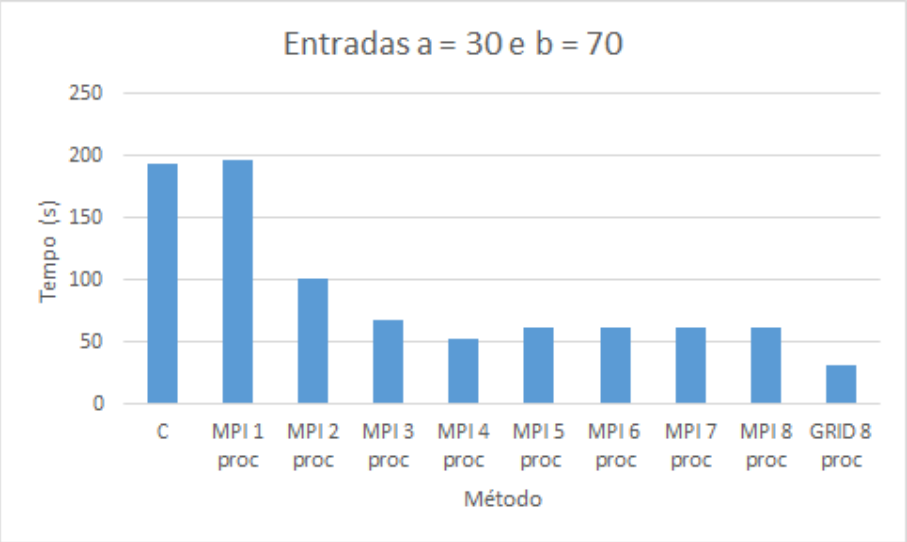

Figura 4.30: *Gráfico comparativo de tempo médio C x MPI x Grid (30x70).*

A partir dos resultados destes últimos testes confirma-se a tendência de que a implementação em ambiente de um único processador possui tempo de execução menor que a implementação em ambiente paralelo sendo executada em apenas um processador, devido à carga adicional de comunicação. Os tempos de execução em ambiente paralelo tendem a ser menores de acordo com o aumento do número de processadores. Nos testes nota-se que o menor tempo sempre foi obtido com o número de processadores igual à 4, devido à quantidade de núcleos do processador utilizado.

Neste último bloco de testes, a efetividade de redução de tempo de execução do am-

biente de grande porte do GridUnesp, também é mantida.

4.6 Speedup

Como forma de medir o ganho obtido com a implementação em ambiente paralelo, realizou-se o cálculo do *speedup* [55]. Este é dado pela divisão do tempo de execução serial pelo tempo de execução em paralelo, conforme a expressão 4.1:

$$Speedup = \frac{T_{serial}}{T_{paralelo}} \tag{4.1}$$

Realizaram-se análises de *speedup* comparando a implementação em ambiente de linguagem interpretada (Mathematica) com a implementação em ambiente paralelo (MPI) e também comparando a implementação em ambiente de um único processador (C) com a implementação em ambiente paralelo (MPI).

4.6.1 Mathematica x Paralelismo com MPI x Grid

Primeiramente foram realizadas as análises de *speedup* comparando a implementação em ambiente de linguagem interpretada com a implementação em ambiente paralelo com MPI. Primeiramente são apresentados os gráficos de *speedup* dos testes de entrada única, nas figuras 4.31, 4.32 e 4.33.

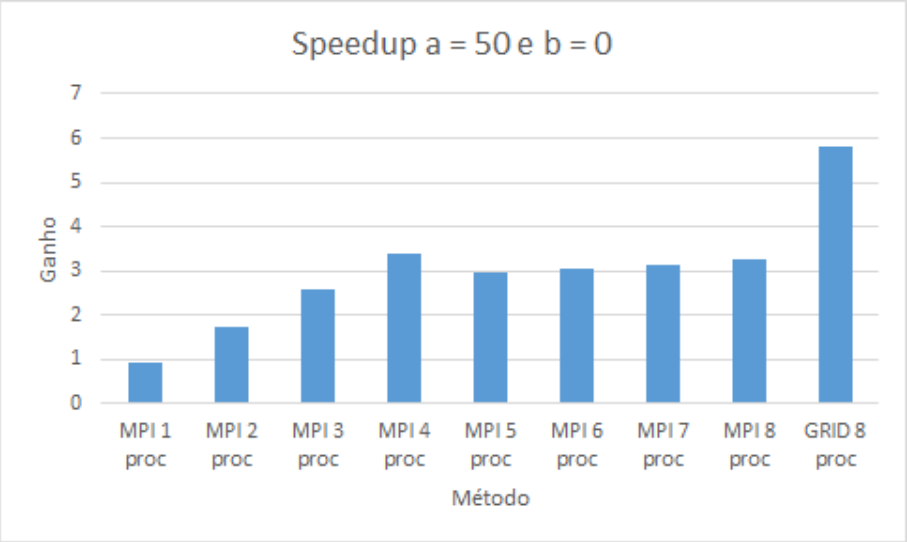

Figura 4.31: *Speedup Mathematica x MPI x Grid (50x0).*

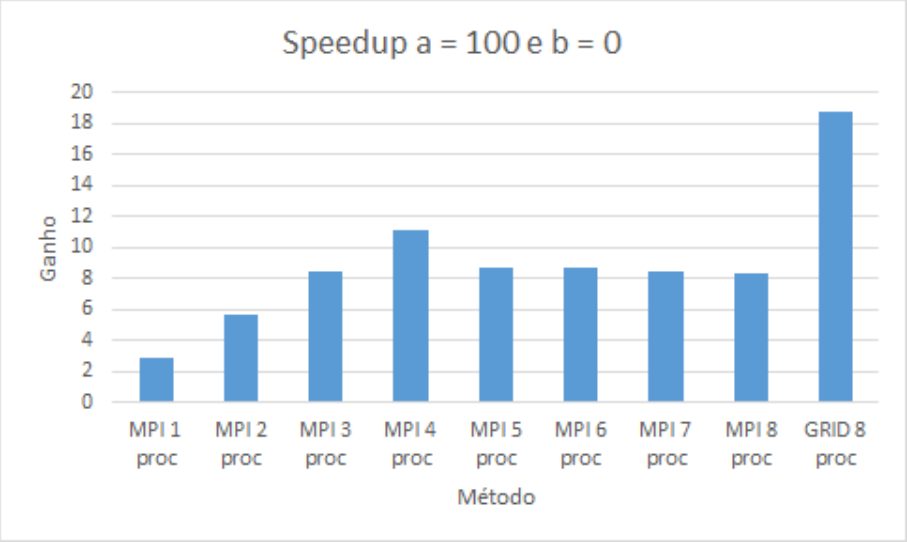

Figura 4.32: *Speedup Mathematica x MPI x Grid (100x0).*

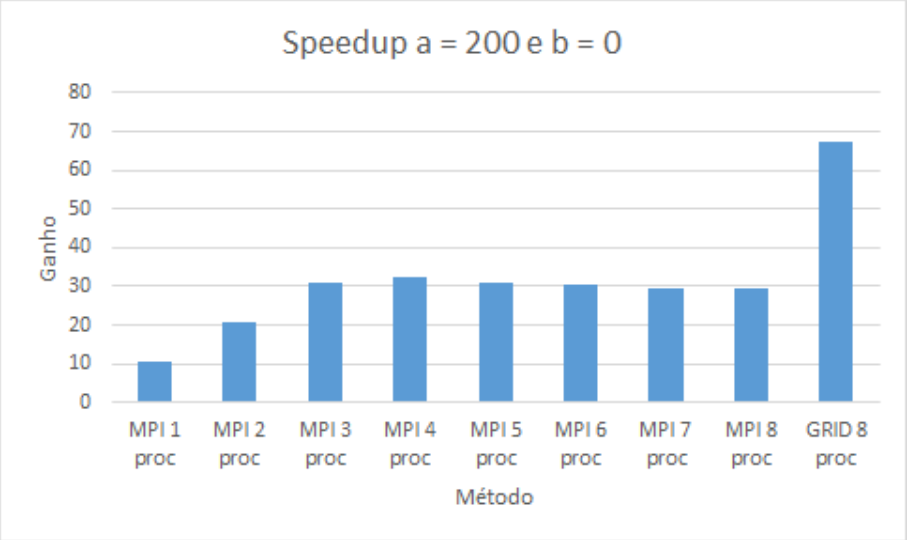

Figura 4.33: *Speedup Mathematica x MPI x Grid (200x0).*

O *speedup* médio para o MPI nessas configurações foi de 2.62641 para as entradas $a = 50$ e $b = 0$, de 7.78583 para as entradas $a = 100$ e $b = 0$ e de 26.8089 para as entradas $a = 200$ e $b = 0$. O *speedup* médio para o Grid nessas configurações foi de 5.787695 para as entradas $a = 50$ e $b = 0$, de 18.781614 para as entradas $a = 100$ e $b = 0$ e de 67.587408 para as entradas $a = 200$ e $b = 0$.

Em seguida são apresentados os gráficos de *speedup* dos testes de entradas iguais representados nas figuras 4.34, 4.35 e 4.36.

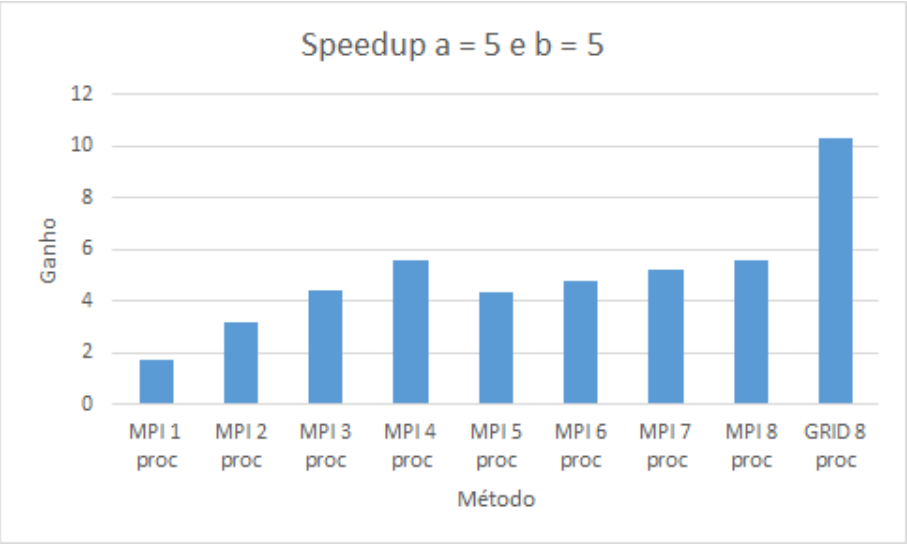

Figura 4.34: *Speedup Mathematica x MPI x Grid (5x5).*

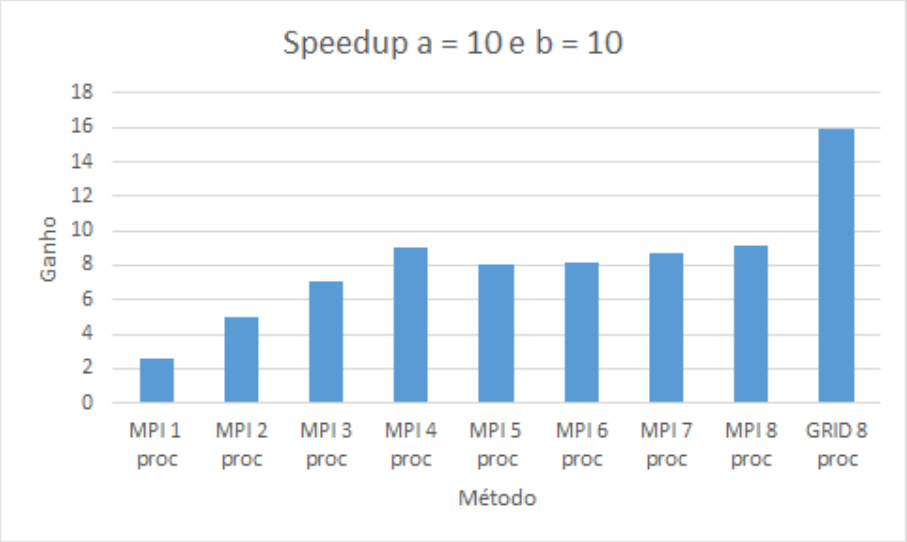

Figura 4.35: *Speedup Mathematica x MPI x Grid (10x10).*

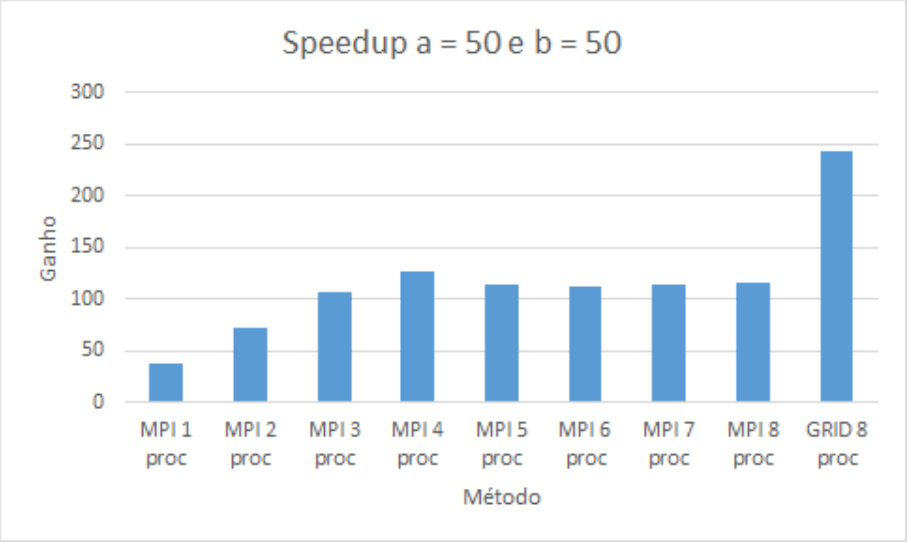

Figura 4.36: *Speedup Mathematica x MPI x Grid (50x50).*

O *speedup* médio para o MPI nessas configurações foi de 4.35035 para as entradas $a = 5$ e $b = 5$, de 7.21969 para as entradas $a = 10$ e $b = 10$ e de 100.149 para as entradas $a = 50$ e $b = 50$. O *speedup* médio para o Grid nessas configurações foi de 10.356253 para as entradas $a = 5$ e $b = 5$, de 15.895168 para as entradas $a = 10$ e $b = 10$ e de 243.136952 para as entradas $a = 50$ e $b = 50$.

Por fim, são apresentados os gráficos de *speedup* dos testes para entradas diferentes, representados nas figuras 4.37, 4.38 e 4.39.

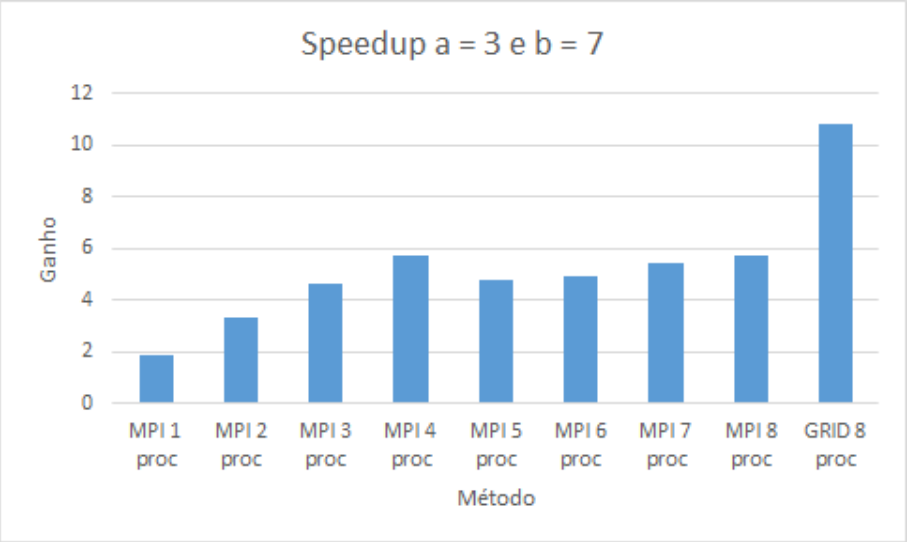

Figura 4.37: *Speedup Mathematica x MPI x Grid (3x7).*

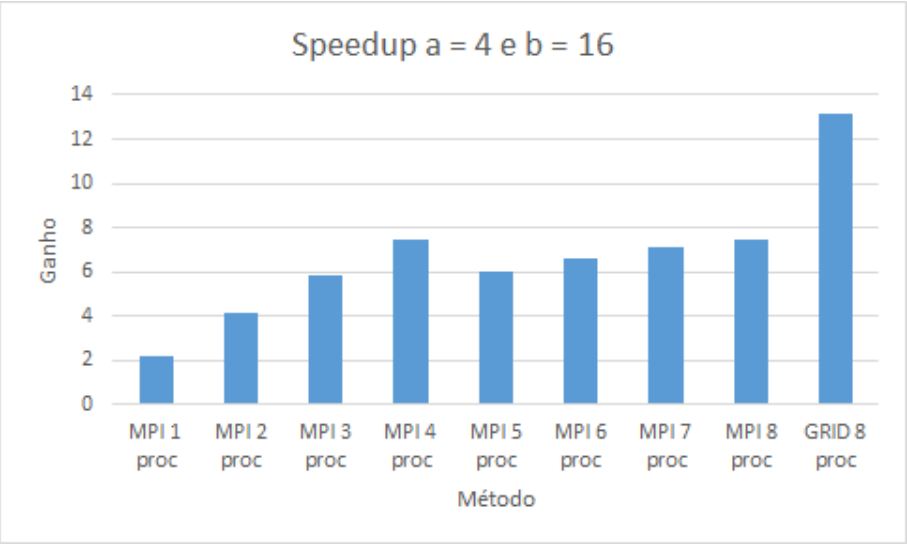

Figura 4.38: *Speedup Mathematica x MPI x Grid (4x16).*

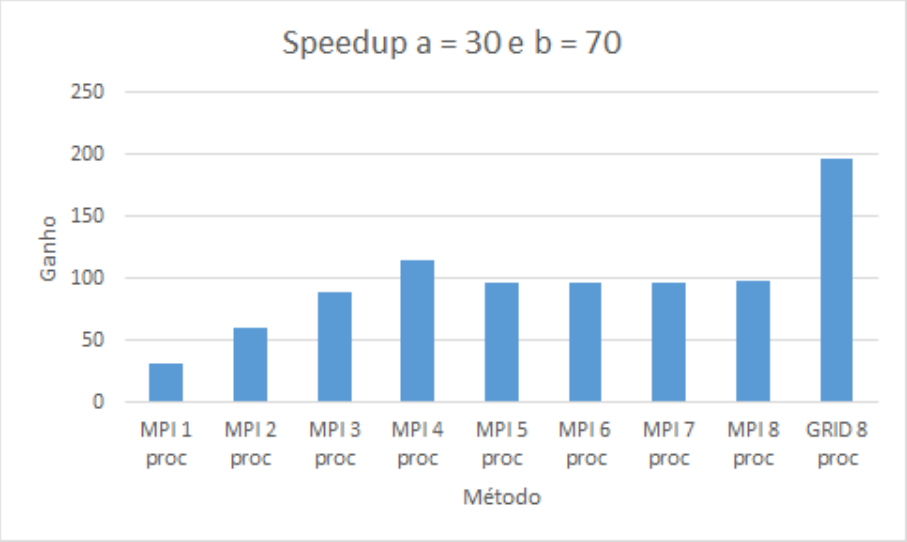

Figura 4.39: *Speedup Mathematica x MPI x Grid (30x70).*

O *speedup* médio para o MPI nessas configurações foi de 4.54809 para as entradas $a = 3$ e $b = 7$, de 5.84917 para as entradas $a = 4$ e $b = 16$ e de 85.3276 para as entradas $a = 30$ e $b = 70$. O *speedup* médio para o Grid nessas configurações foi de 10.857231 para as entradas $a = 3$ e $b = 7$, de 13.144492 para as entradas $a = 4$ e $b = 16$ e de 196.539134 para as entradas $a = 30$ e $b = 70$.

Com base nos gráficos das figuras nota-se que o gráfico de *speedup* se mantêm de forma crescente, com pico de ganho nas execuções com 4 processadores (em um universo de 8 processadores possíveis contemplados pelos testes), posteriormente com uma leve queda

e por fim novamente um aumento de ganho. A causa deste declínio no ganho se dá pelo fato de o processador da máquina de testes possuir 4 processadores físicos e 8 *threads*, em que os testes com 4 processadores ocupam os 4 núcleos físicos completamente. A partir das execuções com número maior de processadores, o uso dos processadores físicos fica dividido entre as *threads*, o que causa uma condição de concorrência e acaba por diminuir o ganho dos resultados.

Ainda que exista esta pequena queda de ganho, verifica-se nos testes a tendência de que o aumento do número de processadores tende a aumentar o ganho, diminuindo os tempos de execução dos testes. O ganho em relação aos testes realizados no Mathematica varia bastante, em que o menor fica perto de 2 vezes e o mais expressivo chega perto de 140 vezes.

No caso do ambiente de grande porte do GridUnesp verifica-se que o ganho é ainda mais expressivo, tornando a paralelização uma estratégia vantajosa. Para os casos do GridUnesp comparados com o Mathematica aferiu-se ganhos de 5.7 a 243.4 vezes.

4.6.2 C x MPI x Grid

Concluindo as análises de *speedup*, foram comparadas a implementação em ambiente de um único processador (C) com a implementação em ambiente paralelo com MPI. Primeiramente são apresentados os gráficos de *speedup* dos testes de entrada única nas figuras 4.40, 4.41 e 4.42.

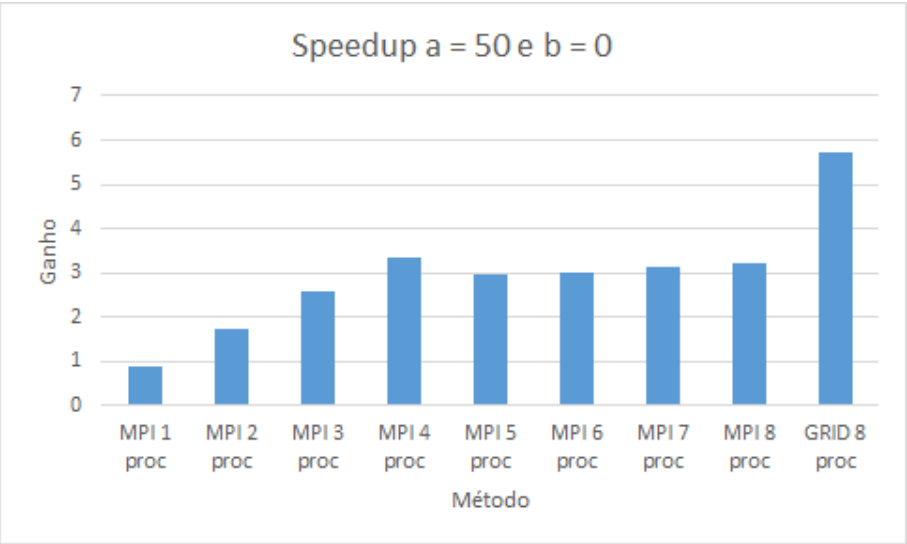

Figura 4.40: *Speedup C x MPI x Grid (50x0).*

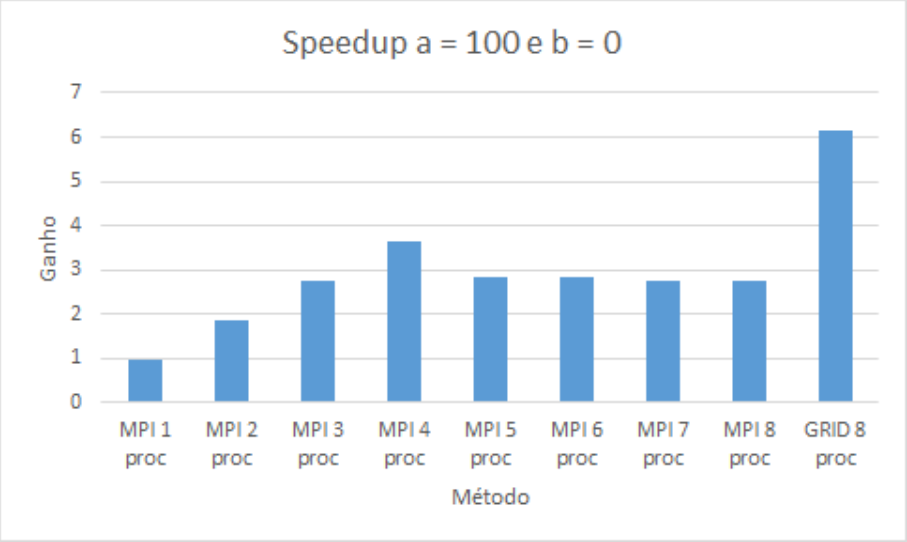

Figura 4.41: *Speedup C x MPI x Grid (100x0).*

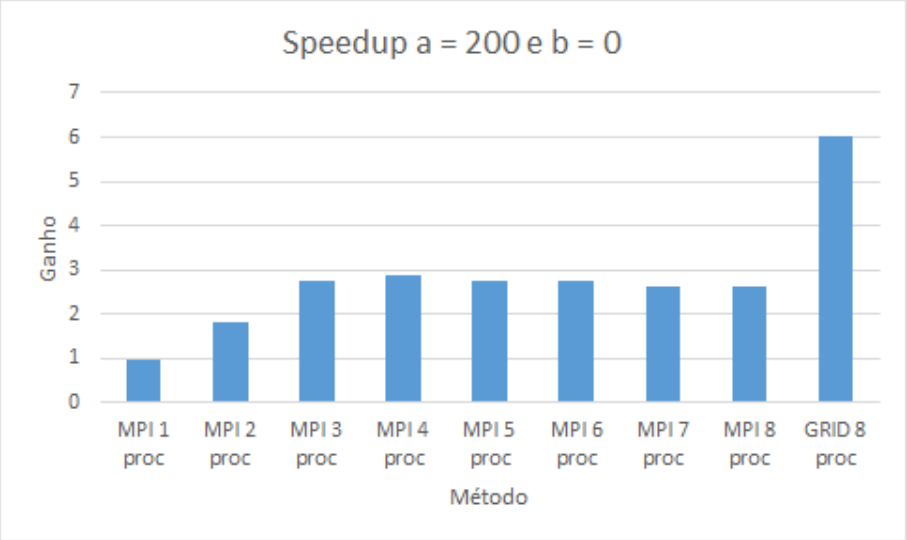

Figura 4.42: *Speedup C x MPI x Grid (200x0).*

O *speedup* médio para o MPI nessas configurações foi de 2.59774 para as entradas $a = 50$ e $b = 0$, de 2.54263 para as entradas $a = 100$ e $b = 0$ e de 2.39282 para as entradas $a = 200$ e $b = 0$. O *speedup* médio para o Grid nessas configurações foi de 5.724513 para as entradas $a = 50$ e $b = 0$, de 6.133540 para as entradas $a = 100$ e $b = 0$ e de 6.032495 para as entradas $a = 200$ e $b = 0$.

Em seguida são apresentados os gráficos de *speedup* dos testes de entradas iguais, representados nas figuras 4.43, 4.44 e 4.45.

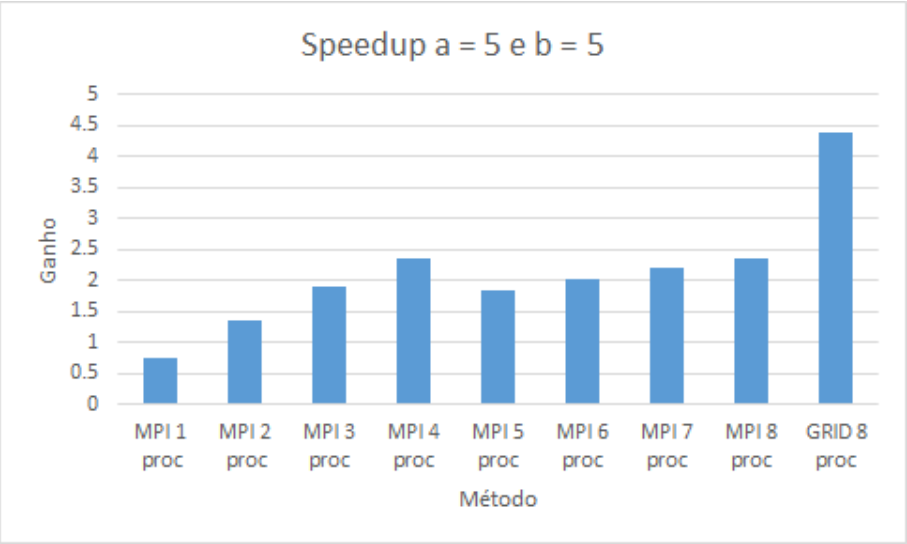

Figura 4.43: *Speedup C x MPI x Grid (5x5).*

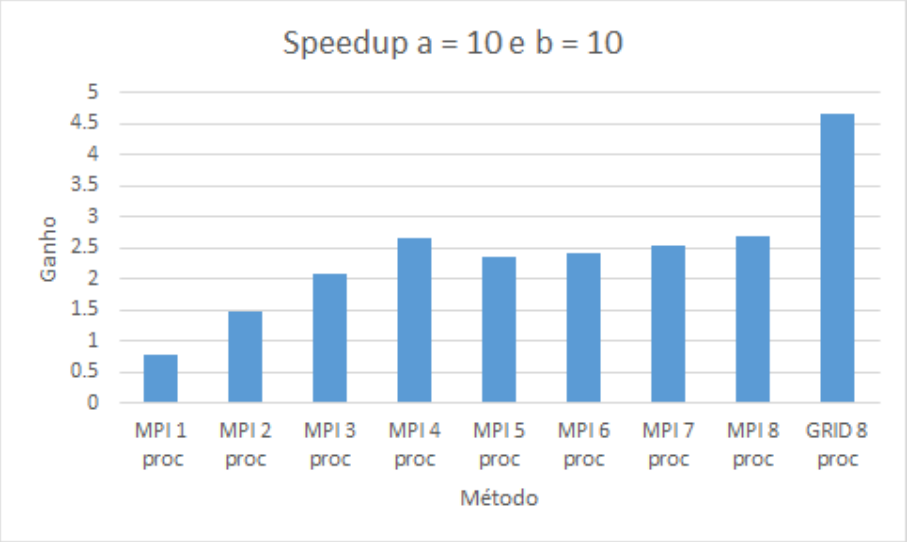

Figura 4.44: *Speedup C x MPI x Grid (10x10).*

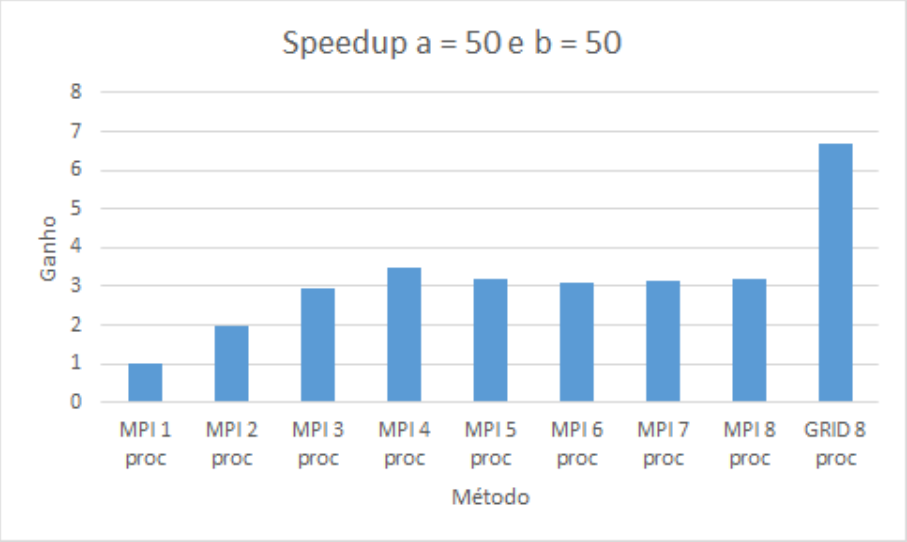

Figura 4.45: *Speedup C x MPI x Grid (50x50).*

O *speedup* médio para o MPI nessas configurações foi de 1.84989 para as entradas $a = 5$ e $b = 5$, de 2.12267 para as entradas $a = 10$ e $b = 10$ e de 2.75801 para as entradas $a = 50$ e $b = 50$. O *speedup* médio para o Grid nessas configurações foi de 4.403761 para as entradas $a = 5$ e $b = 5$, de 4.673359 para as entradas $a = 10$ e $b = 10$ e de 6.695754 para as entradas $a = 50$ e $b = 50$.

Por fim são, apresentados os gráficos de *speedup* dos testes de entradas diferentes, representados nas figuras 4.46, 4.47 e 4.48.

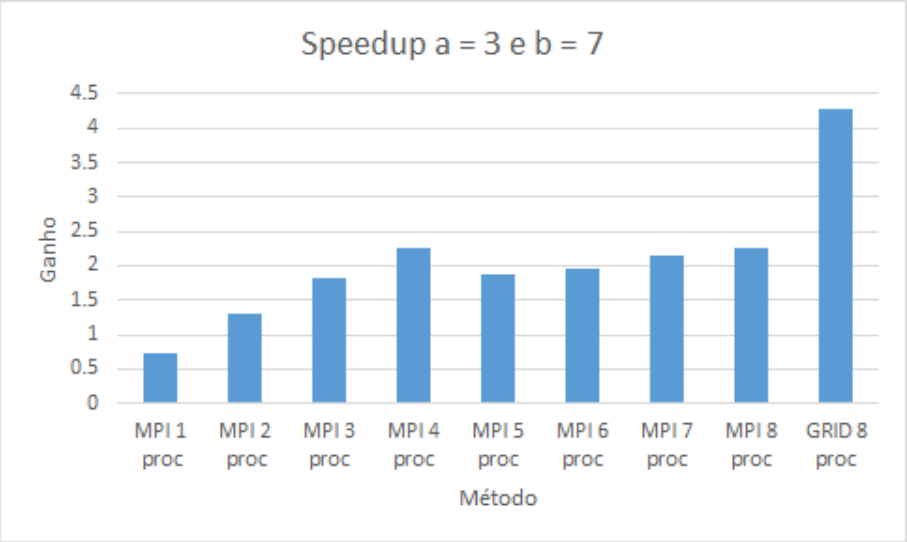

Figura 4.46: *Speedup C x MPI x Grid (3x7).*

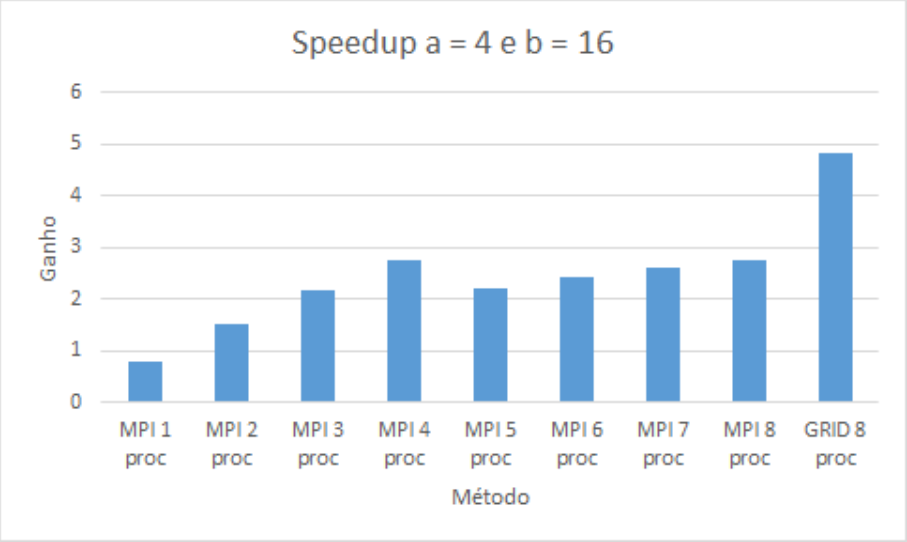

Figura 4.47: *Speedup C x MPI x Grid (4x16).*

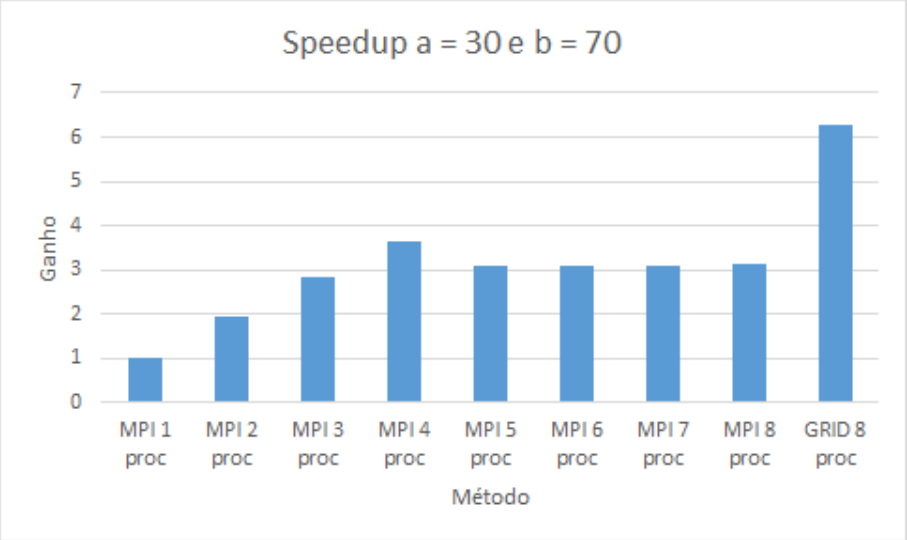

Figura 4.48: *Speedup C x MPI x Grid (30x70).*

O *speedup* médio para o MPI nessas configurações foi de 1.79386 para as entradas $a = 3$ e $b = 7$, de 2.15577 para as entradas $a = 4$ e $b = 16$ e de 2.73029 para as entradas $a = 30$ e $b = 70$. O *speedup* médio para o Grid nessas configurações foi de 4.282325 para as entradas $a = 3$ e $b = 7$, de 4.844542 para as entradas $a = 4$ e $b = 16$ e de 6.288816 para as entradas $a = 30$ e $b = 70$.

Da mesma forma que as análises de *speedup* entre a implementação em ambiente de linguagem interpretada e a implementação de ambiente paralelo apontaram uma tendência crescente no ganho com o aumento do número de processadores, as análises de *speedup*

entre a implementação em ambiente de um único processador e a implementação de ambiente paralelo seguem a mesma tendência, com apenas algumas variações, de acordo com os valores de entrada. Os ganhos são menores em relação ao ambiente de linguagem interpretada, ficando entre 1 e 3.5 nos testes realizados.

No caso do ambiente de grande porte do GridUnesp, quando com parado com a implementação em C, verifica-se que o ganho também é expressivo. Para estes casos, nos resultados obtidos no GridUnesp quando comparados com os resultados obtidos em C aferiu-se ganhos de 4.2 a 6.7 vezes.

Capítulo 5

Conclusões

5.1 Considerações Finais

Neste trabalho apresentou-se uma conceituação sobre mecânica quântica, informação, computação e entropias quânticas, bem como computação paralela e MPI. Propôs-se e realizou-se uma simulação com implementação em ambiente paralelo sobre o efeito da entropia de emaranhamento dos fótons em uma transmissão de dados e a comparação com a implementação em um ambiente de um único processador.

Este trabalho rediscute e amplia os trabalhos anteriores de C. Brandão e M.F. Borges [56, 6], em que os resultados contribuíram com as pesquisas em âmbito da computação quântica acerca dos fenômenos que ocorrem com os fótons em uma transmissão de dados.

Além disso, a inclinação da curva na entropia de Von Neumann, observada nas simulações anteriores, pode significar uma tendência de diminuição de entropia e de auto-organização dos fótons.

5.2 Contribuições

No presente trabalho criou-se uma ferramenta que possibilita a realização de testes que mais se aproximam de uma transmissão real, com a possibilidade de um aumento significativo no número de fótons nas entradas do divisor de feixe, com execução da simulação em tempo hábil. Apesar de haver trabalhos que utilizaram um número razoável de fótons nas entradas do divisor de feixe, o elevado tempo de execução de suas simulações tornou-os inviáveis.

Com todo ferramental físico e matemático e com o suporte computacional, foi possível o desenvolvimento completo e mais robusto de um ambiente de simulação. O trabalho também visou apresentar algoritmos paralelos mais eficientes que os algoritmos sequenciais para a realização de simulações nesta área. Vale salientar que o ambiente de simulação desenvolvido mostrou-se altamente escalável, com resultados expressivos em sistemas paralelos de grande porte.

5.3 Trabalhos Futuros

Como sugestão para trabalhos futuros pode-se citar diferentes abordagens de paralelização utilizando MPI ou ainda diferentes ferramentas de paralelização como o OpenMP, de modo a apresentar resultados mais eficientes que os obtidos no presente trabalho.

Traçar comparações entre implementações em diferentes linguagens de programação, como Java, Fortran, ou ainda linguagens interpretadas como R também se figuram como sugestão para trabalhos futuros.

De modo a expandir o conhecimento sobre o comportamento do emaranhamento dos fótons, pode-se citar a utilização de estados gaussianos, uma vez que neste trabalho foram apresentados estados-Fock.

Por fim, vale a pena salientar a possibilidade de investigar a implementação de determinados trechos das simulações sob o paradigma GPU.

Referências Bibliográficas

[1] MARIANTONI, M. et al. Implementing the quantum von neumann architecture with superconducting circuits. *Science*, v. 334, n. 6052, p. 61–65, 2011. Disponível em: <http://www.sciencemag.org/content/334/6052/61.abstract>.

[2] LUCERO, E. et al. Computing prime factors with a Josephson phase qubit quantum processor. fev. 2012. Disponível em: <http://arxiv.org/abs/1202.5707>.

[3] DICARLO, L. et al. Demonstration of two-qubit algorithms with a superconducting quantum processor. *Nature*, Macmillan Publishers Limited. All rights reserved, v. 460, n. 7252, p. 240–244, jul. 2009. ISSN 0028-0836. Disponível em: <http://dx.doi.org/10.1038/nature08121>.

[4] RIVEST, R.; SHAMIR, A.; ADLEMAN, L. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, v. 21, p. 120–126, 1978.

[5] BRANDAO, C. *Ensaios sobre computação e informação quânticas: fundamentação e simulações sobre o efeito da entropia*. Dissertação (Mestrado) — UNESP, São José do Rio Preto, 2010.

[6] BORGES, M. F.; GODOY, R.; PRATAVIEIRA, G. A. Towards a quantum information process using tsallis entropy. *International Journal of Applied Mathematics*, n. 21, p. 645–655, 2008.

[7] GODOY, R. *Espaço de Hilbert e Quantificação de emaranhamento via entropia não extensiva*. Dissertação (Mestrado) — UNESP, São José do Rio Preto, 2005.

[8] NEUMANN, J. *The Mathematical Foundations of Quantum Mechanics*. [S.l.]: Princeton University Press, 1955.

[9] TSALLIS, C.; BARRETO, F. C. S.; LOH, E. D. Generalization of the planck radiation law and application to the cosmic microwave background radiation. *Phys. Rev.*, n. 52, p. 1447, 1995.

[10] EISBERG, R.; RESNICK, R. *Física Quântica*. [S.l.]: Rio de Janeiro: Campus, 1994.

[11] HEISENBERG, W. *The phisical principles of the quantum theory*. [S.l.]: Dover Publications, Inc, New York, 1949.

[12] EINSTEIN, A. Über einen die Erzeugung und Verwandlung des Lichtes betreffenden heuristischen Gesichtspunkt. *Annalen der Physik*, v. 322, p. 132–148, 1905.

[13] WHEELER, J.; ZUREK, W. *Quantum Theory and Measurement*. [S.l.]: Books on Demand, 1983. (Princeton Series in Physics). ISBN 9780608095820.

[14] BROGLIE, L. A tentative theory of light quanta. *Philosophical Magazine*, n. 47, p. 446–458, 1924.

[15] BENNETT, C.; DIVINCENZO, D. P. Quantum information and computation. *Nature*, n. 404, p. 247–255, 2000.

[16] CUNHA, M. T. *Noções de Informação Quântica*. [S.l.]: IMPA, Rio de Janeiro, 2007.

[17] TUOMI, I. The lives and death of moore's law. *First Monday*, v. 7, n. 11, 2002.

[18] NIELSEN, M. A.; CHUANG, I. *Quantum computation and Quantum information*. [S.l.]: Cambridge University Press, 2002.

[19] FEYMAN, R. P. Simulating physics with computers. *Int. J. Theoretical Physics*, n. 21, p. 467–488, 1982.

[20] GERSHENFELD, N. A.; CHUANG, I. L. Bulk spin-resonance quantum computation. *Science*, v. 275, n. 5298, p. 350–356, 1997. Disponível em: <http://www.sciencemag.org/content/275/5298/350.abstract>.

[21] LINDEN, N.; BARJAT, H.; FREEMAN, R. An implementation of the deutsch–jozsa algorithm on a three-qubit nmr quantum computer. *Chemical Physics Letters*, Elsevier, v. 296, n. 1, p. 61–67, 1998.

[22] DORAI, K.; KUMAR, A. et al. Quantum entanglement in the nmr implementation of the deutsch-jozsa algorithm. *Pramana-Journal of Physics*, Indian Academy of Sciences, v. 56, n. 5, p. L705–L713, 2001.

[23] BLOCH, F.; SIEGERT, A. Magnetic resonance for nonrotating fields. *Phys. Rev.*, American Physical Society, v. 57, p. 522–527, Mar 1940. Disponível em: <http://link.aps.org/doi/10.1103/PhysRev.57.522>.

[24] GERJUOY, E. Shor's factoring algorithm and modern cryptography. an illustration of the capabilities inherent in quantum computers. *Am. J. Phys.*, n. 73, p. 521–540, 2005.

[25] GROVER, L. K. Quantum mechanics helps in searching for a needle in a haystack. *Phys. Rev. Lett.*, American Physical Society, v. 79, n. 2, p. 325–328, Jul 1997.

[26] VANDERSYPEN, L. M. K. et al. Experimental realization of shor's quantum factoring algorithm using nuclear magnetic resonance. *Nature*, n. 414, p. 883–887, 2001.

[27] CHUANG, I. L.; GERSHENFELD, N.; KUBINEC, M. Experimental implementation of fast quantum searching. *Phys. Rev. Lett.*, American Physical Society, v. 80, n. 15, p. 3408–3411, Apr 1998.

[28] BRICKMAN, K.-A. et al. Implementation of grover's quantum search algorithm in a scalable system. *Phys. Rev. A*, American Physical Society, v. 72, n. 5, p. 050306, Nov 2005.

[29] WALTHER, P. et al. Experimental one-way quantum computing. *Nature*, n. 169, p. 434, 2005.

[30] SOMAROO, S. et al. Quantum simulations on a quantum computer. *Phys. Rev. Lett.*, American Physical Society, v. 82, n. 26, p. 5381–5384, Jun 1999.

[31] CUNHA, M. *Emaranhamento: caracterização, manipulação e consequências.* Tese (Doutorado) — UFMG, Belo Horizonte, 2005.

[32] SALLES, A. *Emaranhamento Quântico: Detecção, Dinâmica e Não-localidade.* Tese (Doutorado) — UFRJ, Rio De Janeiro, 2009.

[33] SANTOS, D. C. *Em busca de um entendimento completo acerca do emaranhamento.* Dissertação (Mestrado) — UFMG, Belo Horizonte, 2006.

[34] VIDIELLA, A. Entanglement and nonextensive statisics. *Physics Letters A*, n. 260, p. 335–339, 1999.

[35] YOUNG, T. On the theory of light and colours. *Philosophical Transactions of the Royal Society of London*, n. 92, p. 12, 1802.

[36] EINSTEIN, A.; PODOLSKY, B.; ROSEN, N. Can quantum-mechanical description of physical reality be considered complete? *Phys. Rev.*, American Physical Society, v. 47, p. 777–780, May 1935. Disponível em: <http://link.aps.org/doi/10.1103/PhysRev.47.777>.

[37] BELL, J. S. On the Einstein Podolsky Rosen paradox. *Physics*, v. 1, n. 3, p. 195–200, 1964.

[38] BELL, J. *Speakable and Unspeakable in Quantum Mechanics.* [S.l.]: Cambridge University Press, 1993.

[39] ASPECT, A.; DALIBARD, J.; ROGER, G. Experimental test of bell's inequalities using time- varying analyzers. *Phys. Rev. Lett.*, American Physical Society, v. 49, n. 25, p. 1804–1807, Dec 1982.

[40] AGRAWAL, P.; PATI, A. Perfect teleportation and superdense coding with w states. *Physical Review A*, APS, v. 74, n. 6, p. 062320, 2006.

[41] MURALIDHARAN, S.; PANIGRAHI, P. K. Perfect teleportation, quantum-state sharing, and superdense coding through a genuinely entangled five-qubit state. *Physical Review A*, APS, v. 77, n. 3, p. 032321, 2008.

[42] WANG, C.; DENG, F. G.; LONG, G. L. Multi-step quantum secure direct communication using multi-particle green–horne–zeilinger state. *Optics communications*, Elsevier, v. 253, n. 1, p. 15–20, 2005.

[43] CUNHA ET.AL, M. *Entropia: introdução à Teoria Matemática da (des)Informação.* UFMG, Belo Horizonte, 2004.

[44] SHANNON, C. E. A mathematical theory of communication. *Bell Sys. Tech. J.*, n. 27, p. 379–423 and 623–656, 1948.

[45] TSALLIS, C. Possible generalizations of boltzmann - gibbs statistics. *J. Stat. Phys.*, n. 52, p. 479–487, 1988.

[46] TSALLIS, C.; GELL-MANN, M.; SATO, Y. *Extensivity and Entropy Production*. [S.l.]: Europhysics News, 2005.

[47] TSALLIS, C. Nonextensive statical mechanics and thermodynamics. *Braz. J. Stat. Phys.*, v. 29, 1999.

[48] FLYNN, M. J. Readings in computer architecture. In: HILL, M. D.; JOUPPI, N. P.; SOHI, G. S. (Ed.). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2000. cap. Very high-speed computing systems, p. 519–527. ISBN 1-55860-539-8. Disponível em: <http://dl.acm.org/citation.cfm?id=333067.333226>.

[49] GRANLUND, T.; the GMP development team. *GNU MP: The GNU Multiple Precision Arithmetic Library*. 5.1.2. ed. [S.l.], 2013. Http://gmplib.org.

[50] FOUSSE, L. et al. Mpfr: A multiple-precision binary floating-point library with correct rounding. *ACM Trans. Math. Softw.*, ACM, New York, NY, USA, v. 33, n. 2, p. 13, 2007. ISSN 0098-3500.

[51] GNU. *GCC Optimization Options*. http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html - acessado em 06/02/2015, 2015.

[52] GNU. *GCC i386 and x86 Options*. http://gcc.gnu.org/onlinedocs/gcc-4.0.2/i386-and-x86_002d64-Options.html - acessado em 06/02/2015, 2015.

[53] MACHADO, J. M.; VERARDI, S. L. L.; SHIYOU, Y. An application of the adomian's decomposition method to the analysis of mhd duct flows. *Magnetics, IEEE Transactions on*, IEEE, v. 41, n. 5, p. 1588–1591, 2005.

[54] MARASCO, A. Nonlinear hydrodynamic models of traffic flow in the presence of tollgates. *Mathematical and computer modelling*, Elsevier, v. 35, n. 5, p. 549–559, 2002.

[55] XAVIER, C.; IYENGAR, S. *Introduction to Parallel Algorithms*. [S.l.]: Wiley, 1998. (A Wiley interscience publication). ISBN 9780471251828.

[56] BRANDAO, C.; BORGES, M. F. Quantum entanglement and information: the effect of entropy revisited. *International Journal of Pure and Applied Mathematics*, v. 68, p. 25–35, 2011.