UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE PÓS-GRADUAÇÃO EM
CIÊNCIAS DA COMPUTAÇÃO

**Luiz Carlos Pinto Silva Filho**

# Estudo de Casos com Aplicações Científicas de Alto Desempenho em Agregados de Computadores Multi-core

Dissertação apresentada à Universidade Federal de Santa Catarina como parte dos requisitos para a obtenção do título de Mestre em Ciências da Computação.

Áreas de concentração: Arquitetura de Computadores, Redes de Interconexão, Computação Paralela, Modelos de Programação Paralela, Computação de Alto Desempenho, Computação em Clusters, Avaliação de Desempenho, Modelagem de Sistemas Paralelos Distribuídos, Aplicações Científicas *Grand Challenge*

Prof. Dr. Mário A. R. Dantas

Florianópolis - SC
2008

**Luiz Carlos Pinto Silva Filho**

# Estudo de Casos com Aplicações Científicas de Alto Desempenho em Agregados de Computadores Multi-core

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Ciências da Computação e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciências da Computação.

---

**Prof. Dr. Mário Antônio Ribeiro Dantas**
*Coordenador do PPGCC*

**BANCA EXAMINADORA**

---

**Prof. Dr. Mário Antônio Ribeiro Dantas**
*Orientador*

---

**Prof. Dr. Nelson Francisco Favilla Ebecken**
COPPE / UFRJ

---

**Prof. Dr. Frank Augusto Siqueira**
INE / UFSC

---

**Prof. Dr. Luís Fernando Friedrich**
INE / UFSC

# *Agradecimentos*

Ao colega Rodrigo Paiva Mendonça, agradeço pela amizade e esforço dedicado junto comigo nos primeiros passos por esse caminho que agora chega ao seu derradeiro destino.

Agradeço ao meu parceiro Luiz Henrique B. Tomazella, que tornou-se um grande amigo e inseparável companheiro em discussões sobre sonhos e oportunidades de crescimento profissional. Tua dedicação para com nossos projetos de pesquisa e implacável prestatividade estão refletidas neste trabalho, que também tem a tua marca. Se Deus quiser, nossos sonhos serão realizados.

Ao meu orientador Mário A. R. Dantas, confesso-me profundamente grato por tantas coisas. Desde o primeiro momento, me acolheste como um verdadeiro padrinho. Essa jornada acadêmica não teria acontecido de maneira tão natural se você não estivesse sempre disposto a dar o apoio necessário, ensinando às lagartixas o caminho do teto. Além do mais, orgulho-me de ter feito contigo a dupla Mário e Luigi. E com sucesso!

Agradeço a postura colaborativa da EPAGRI S.A., que abriu espaço em seus ambientes de produção para nossos experimentos. Também agradeço à banca pelas contribuições, com perguntas e sugestões de grande valia, e especialmente ao professor Nelson Ebecken que se deslocou até a UFSC para participar da banca examinadora.

Palavras não são capazes de demonstrar minha gratidão a todas as pessoas que, de uma maneira ou outra, fizeram esta conquista acontecer. Se bastasse lembrá-las aqui neste espaço, não estariam presentes nos meus pensamentos e coração, distantes ou ao meu lado, mas juntos na grande luta que é a sobrevivência em sociedade do bicho homem.

Com enorme carinho e amor, agradeço à minha querida família, por se dedicarem tanto a jamais deixar faltar com palavras de força e afeto. Um carinho altruísta e equilibrado, um amor recíproco e incondicional, que permeará toda a minha existência, como um rio que se alaga para nutrir minha vida com liberdade e compreensão. Deste mesmo barco, agradeço à minha amada Luana, espírito de luz, que trouxe o mais lindo sorriso e a paz para meus dias, hoje e talvez sempre.

Enfim, sou sinceramente grato a todos, de coração.

*I can't buy what I want*
*because it's free!*
*(Eddie Vedder)*

# *Resumo*

Este trabalho de dissertação concentra seu esforço no estudo empírico de três casos com ambientes de *cluster* distintos e homogêneos, sendo que dois deles são ambientes operacionais de empresas. Tais agregados são compostos por computadores multiprocessados com processadores *mono-core* e *multi-core* (*dual-core* e *quad-core*), interconectados por redes Gigabit Ethernet, e outro ambiente interconectado por uma rede Myrinet.

O primeiro estudo de caso foi realizado em um ambiente experimental como um exercício empírico sobre a relação entre tecnologias de rede de interconexão e características inerentes a aplicações paralelas. Com isso, pretendeu-se entrar em contato com a realidade da computação paralela por meio de arquiteturas paralelas distribuídas como os agregados de computadores. Além disso, pode-se conhecer e analisar as diferenças em desempenho oferecido por sistemas de comunicação distintos, como a tecnologia de rede Myrinet face à tecnologia Ethernet, diante de aplicações de granularidades distintas, bem como compreender as métricas comumente adotadas em avaliações de desempenho.

Dentre as contribuições do trabalho de pesquisa e experimentação desenvolvido está a redução do tempo de execução de aplicações científicas *grand challenge*, como a modelagem numérica de previsão meteorológica. Sendo assim, busca-se como resultado a otimização de desempenho do ambiente de *cluster* em comparação à sua condição anterior, sem nenhuma especialização à aplicação em foco. Nesse sentido, dois estudos de casos foram realizados em agregados de computadores pertencentes a organizações, em uma aproximação com a realidade da computação de alto desempenho em ambientes de produção.

Com a realização deste estudo empírico como um todo, pode-se contrastar na prática os pontos estudados durante a revisão bibliográfica. Foi possível compreender melhor as vantagens e desvantangens envolvidas nesses ambientes enquanto sistemas paralelos distribuídos, com o foco voltado à modelagem de sistemas de alto desempenho em ambientes de produção. Durante o processo de otimização do desempenho, entrou-se em contato com os mecanismos de interação entre processos e os modelos de programação paralela envolvidos ao mesmo tempo em que investigou-se o impacto da tendência atual no que diz respeito a processadores *multi-core*, bem como os fatores redutores do desempenho (que resultam em *overhead*).

Enfim, o conhecimento adquirido com os estudos de casos possibilita uma melhor compreensão do processo e dos fatores envolvidos na implementação de ambientes de *cluster* adequados a cada aplicação paralela com demanda por alto desempenho, a fim de aproveitar melhor os recursos agregados. Além disso, a importância deste trabalho transcende à ciência da computação como disciplina acadêmica, pois a empresa parceira ganha em capacidade e qualidade na previsão meteorológica do tempo, seja para prevenir o impacto de desastres naturais ou para auxiliar na produção agrícola, e também em potencial de pesquisa no âmbito daquela área de atuação.

# *Abstract*

This dissertation concentrates its effort on the empirical study of three cases with distinct and homogeneous cluster configurations, two of them operational environments at organizations. Theses clusters are equipped with multiprocessor computers, including multiple single-core and multi-core processors (dual-core and quad-core), interconnected by Gigabit Ethernet networks, and one environment interconnected with a Myrinet device.

The first case study was performed on an experimental environment as an empirical exercise about the relationship between interconnect technologies and characteristics inherent to parallel applications, in order to get in touch with the reality of parallel computing through parallel distributed architectures such as a cluster. Furthermore, we could acknowledge and analyze the differences in performance offered by different communication systems, opposing Myrinet and Ethernet networking technologies before applications of different granularity, as well as understand common metrics adopted for performance assessments.

One of the contributions of this empirical and research work is to reduce the wall clock (or elapsed) time of grand challenge scientific applications, such as numerical weather prediction models. Therefore, it should result in a better performance of the cluster environment compared to its previous condition, with no adaptation for the running application. Based on that, two case studies were conducted on operational clusters belonging to organizations in order to interact with the reality of high performance computing in production environments.

Performing this empirical study as a whole confronts the knowledge obtained throughout the literature review putting them into practice. Moreover, we could accomplish a better understanding of the trade-offs involved in cluster environments as distributed parallel systems for production environments from the point of view of an architectural designer. During this optimization process, we could understand the mechanisms for processes interaction and parallel programming models as well as the factors for overhead growth and performance reduction.

Finally, the knowledge acquired with these case studies allow us to better comprehend the process and the factors involved in the implementation and adaptation of cluster environments to a specific high performance application, in order to better employ the aggregated computing resources. Furthermore, the importance of this work transcends computer sciences as an academic subject, because the partner organization gains capacity and quality for predicting weather conditions, either to prevent us from the impact of natural disasters or to enhance agricultural production, as well as gains in research potential within that specific area.

# *Sumário*

**Lista de Figuras**

**Lista de Tabelas**

**Glossário** p. 14

**1 Introdução** p. 16

1.1 Motivação . . . . . p. 18

1.2 Objetivos Gerais . . . . . p. 19

1.3 Objetivos Específicos . . . . . p. 19

1.4 Metodologia . . . . . p. 20

1.5 Trabalhos Relacionados . . . . . p. 22

1.5.1 Caracterização de aplicações científicas . . . . . p. 22

1.5.2 Agregados de computadores com processadores *multi-core* . . . p. 23

1.5.3 Redes de interconexão *network-on-chip* em *clusters* . . . . . p. 23

**2 Computação Paralela e Distribuída** p. 25

2.1 Taxonomia de Arquiteturas Paralelas . . . . . p. 27

2.1.1 Taxonomia de Flynn . . . . . p. 28

2.1.2 Taxonomia de Johnson . . . . . p. 29

2.1.3 Sistemas de Memória Compartilhada: Multiprocessadores . . . . p. 29

2.1.3.1 Memória Compartilhada Centralizada (UMA) . . . . . p. 30

2.1.3.2 Memória Compartilhada Distribuída (NUMA) . . . . . p. 30

2.1.4 Sistemas de Memória Distribuída: Multicomputadores . . . . . p. 31
2.2 Paradigmas de Programação Paralela . . . . . p. 32
2.2.1 Granularidade da Aplicação . . . . . p. 32
2.2.2 Programação Paralela Explícita e Implícita . . . . . p. 34
2.2.3 Paralelismo Funcional e de Dados (MPMD e SPMD) . . . . . p. 34
2.2.4 Memória Compartilhada e Passagem de Mensagens . . . . . p. 35
2.2.4.1 OpenMP . . . . . p. 36
2.2.4.2 MPI (*Message-Passing Interface*) . . . . . p. 37
2.2.4.3 PGAS (*Partitioned Global Address Space*) . . . . . p. 37
2.3 Sistemas Distribuídos . . . . . p. 38
2.3.1 Agregados Computacionais (*Clusters*) . . . . . p. 39
2.3.2 Grades Computacionais (*Grids*) . . . . . p. 39

**3 Computação de Alto Desempenho** p. 41
3.1 Métricas de Desempenho . . . . . p. 42
3.1.1 Tempo de Execução . . . . . p. 43
3.1.2 *Speedup* e Eficiência . . . . . p. 43
3.1.3 Escalabilidade . . . . . p. 44
3.1.3.1 Lei de Amdahl (*fixed-size speedup*) . . . . . p. 45
3.1.3.2 Lei de Gustafson (*fixed-time speedup*) . . . . . p. 46
3.2 Computação de Alto Desempenho em *Clusters* . . . . . p. 47
3.2.1 Redes de Interconexão . . . . . p. 48
3.2.2 Estudo de Caso: Myrinet vs. Ethernet . . . . . p. 51
3.2.3 Tendência: Processadores *Many-core* ou Nanotecnologia . . . . p. 55

**4 Estudos de Casos com Aplicações Científicas de Alto Desempenho** p. 59
*Weather Research and Forecasting Model* (WRF) . . . . . p. 60

4.1 Caso 1 . . . . . p. 64
4.1.1 Experimentação Investigativa (Caso 1A) . . . . . p. 65
4.1.1.1 Benchmark de rede: *b_eff* . . . . . p. 66
4.1.1.2 NAS Parallel Benchmarks (NPB) . . . . . p. 71
4.1.1.3 Conclusões . . . . . p. 79
4.1.2 Abordagem Proposta . . . . . p. 80
4.1.3 Experimentos (Caso 1B) . . . . . p. 81
4.1.3.1 Benchmark de rede: *b_eff* . . . . . p. 82
4.1.3.2 Modelo WRF . . . . . p. 84
4.1.4 Conclusões . . . . . p. 85
4.2 Caso 2 . . . . . p. 86
4.2.1 Experimentos . . . . . p. 87
4.2.2 Conclusões . . . . . p. 93

**5 Conclusões** p. 95

Perspectivas de Trabalhos Futuros . . . . . p. 96

**Referências** p. 97

**Anexo A: *namelist.input*** p. 101

**Anexo B: Lista de Publicações** p. 105

# *Lista de Figuras*


1 Evolução dos microprocessadores e a Lei de Moore . . . . . p. 26

2 Gráfico de consumo de energia por instrução (processadores Intel®) . . p. 27

3 Computador SISD (arquitetura seqüencial de von Neumann) (FOSTER, 1995) . . . . . p. 28

4 Sistema de memória compartilhada (GMSV ou DMSV) . . . . . p. 30

5 Sistema de memória compartilhada distribuída (DMSV) . . . . . p. 31

6 Sistema de memória distribuída (DMMP) . . . . . p. 31

7 Sistema distribuído baseado em *middleware* (TANEMBAUM; STEEN, 2002) p. 38

8 Gráfico do efeito Amdahl (*speedup* máximo indicado) . . . . . p. 46

9 Gráfico representativo de *scaled speedup* . . . . . p. 47

10 Família de redes de interconexão nos ambientes do TOP500 . . . . . p. 49

11 Rede de interconexão em barramento. . . . . p. 49

12 Rede de interconexão *crossbar*. . . . . p. 50

13 Fluxo dos modelos Ethernet e VIA. . . . . p. 52

14 Latência e taxa de transferência das redes Myrinet e Ethernet . . . . . p. 53

15 Tempo de execução dos algoritmos MM e TM com as redes Myrinet e Ethernet . . . . . p. 54

16 Speedup do algoritmo MM (Myrinet X Ethernet) . . . . . p. 54

17 Speedup do algoritmo TM (Myrinet X Ethernet) . . . . . p. 55

18 Evolução dos microprocessadores Intel® até sua edição *multi-core* . . . p. 56

19 Esquema ilustrativo da aplicação WRF (MICHALAKES et al., 2004) . . . p. 61

20 Região utilizada nos experimentos de previsão meteorológica . . . . . p. 63

21 *Zoom* da região utilizada nos experimentos de previsão meteorológica . p. 63
22 Ambientes de experimentação do Caso 1A . . . . . p. 65
23 Latência entre 2 processos do Caso 1A (msg. pequenas) . . . . . p. 67
24 Latência entre 2 processos do Caso 1A (msg. médias) . . . . . p. 67
25 Latência entre 2 processos do Caso 1A (msg. grandes) . . . . . p. 68
26 Taxa de transf. entre 2 processos do Caso 1A (msg. pequenas) . . . . . p. 69
27 Taxa de transf. entre 2 processos do Caso 1A (msg. médias e grandes) p. 69
28 Latência entre 8 processos (Caso 1A) . . . . . p. 70
29 Taxa de transferência entre 8 processos (Caso 1A) . . . . . p. 71
30 Tempo de execução: NAS EP classe B . . . . . p. 72
31 Tempo de execução: NAS FT classe B . . . . . p. 73
32 Tempo de execução: NAS CG classe B . . . . . p. 74
33 Tempo de execução: NAS IS classe B . . . . . p. 75
34 Tempo de execução: NAS MG classe B . . . . . p. 76
35 *Speedup* do NPB no ambiente N8xP1 (Caso 1A) . . . . . p. 78
36 Eficiência do NPB no ambiente N8xP1 (Caso 1A) . . . . . p. 78
37 Esquema ilustrativo do ambiente de experimentação do Caso 1B . . . . p. 81
38 Ambientes de experimentação do Caso 1B . . . . . p. 82
39 Latência coletada com o b_eff (Caso 1B) . . . . . p. 83
40 Taxa de transferência coletada com o b_eff (Caso 1B) . . . . . p. 83
41 Resultados da execução do WRF nos ambientes de *cluster* (Caso 1B) . p. 84
42 *Speedup* do WRF no sistema CMP-SMP (Caso 1B) . . . . . p. 85
43 Características do ambiente de experimentação do Caso 2 . . . . . p. 87
44 Esquema ilustrativo do ambiente de experimentação do Caso 2 . . . . . p. 88
45 Latência coletada com o b_eff (Caso 2) . . . . . p. 88
46 Taxa de transferência coletada com o b_eff (Caso 2) . . . . . p. 89

47 Duração de cada iteração com MPICH-SOCK, totalizando 34min16seg. p. 89

48 A utilização de oito núcleos em um mesmo nó com MPICH-SOCK. . . p. 90

49 Esquema ilustrativo: MPICH2 (BUNTINAS; MERCIER; GROPP, 2007) . . p. 91

50 A utilização de oito núcleos em um mesmo nó com MPICH-SSM. . . . p. 91

51 Duração de cada iteração com MPICH-SSM, totalizando 27min59seg. . p. 92

52 Duração de cada iteração com MPICH-NEMESIS, totalizando 25min44seg. p. 92

53 Comparação dos resultados com MPICH-SOCK, MPICH-SSM e MPICH-NEMESIS. . . . . p. 93

54 Comparação dos resultados com diversas MTU's (MPICH-NEMESIS). p. 93

55 Previsão meteorológica para 3 dias antes e depois do processo de otimização p. 93

# *Lista de Tabelas*


1 Taxonomia de Flynn (FLYNN, 1972) . . . . . p. 28

2 Taxonomia de computadores MIMD por Johnson (JOHNSON, 1988) . . p. 29

3 Classes do NAS Parallel Benchmarks 2.3 (tamanho do problema) . . . p. 72

4 Variação no tempo de execução do NPB no Sistema N8xP1 (Caso 1A) . p. 77

# *Glossário*

***cluster*:** é o termo inglês para designar um agregado computacional ou agregado de computadores. É usado como sinônimo, indistintamente, bem como ambiente de *cluster*;

**sistema:** é um sistema computacional como um todo, seja um *cluster* ou uma única máquina não-agregada;

**ambiente:** é usado como sistema computacional, mas também pode ser entendido como todo o conjunto de sistemas e aplicações utilizadas em um estudo de caso;

**nó:** um nó nada mais é do que um dos computadores agregados. É uma máquina autônoma com recursos computacionais próprios;

**computador:** há ambigüidade com o uso deste termo no seguinte sentido: pode ser entendido como um nó, uma máquina autônoma quando os experimentos são apresentados, mas na revisão bibliográfica também aparece como no termo computador paralelo, que engloba multiprocessadores e multicomputadores. Com isso em mente, a distinção fica facilitada;

**processador:** é o componente que provê os recursos de processamento, podendo ser mono ou multiprocessado. Pode ser visto como sinônimo de soquete, que é onde o microprocessador é encaixado no computador;

***core*:** em português, núcleo. É a unidade atômica de processamento, também conhecido como CPU (*Central Processing Unit*). O termo *core* se tornou popular com a tecnologia de processadores com múltiplos núcleos de processamento (*multi-core*). Antes, computadores multiprocessados eram equipados com múltiplos processadores *mono-core*, chamados SMP's;

***SMP*:** do termo inglês *Symmetric Multiprocessor*. É um computador com múltiplos processadores, sejam *mono-core* ou *multi-core*;

***CMP*:** do termo inglês *Chip-level Multiprocessor*. É um processador com múltiplos *cores* ou núcleos de processamento, conhecidos como *multi-core*;

***commodity*:** é um produto comercializável, como processadores ou computadores, geralmente produzidos em larga escala, que podem ser facilmente adquiridos a um preço competitivo. Um termo sinônimo em inglês, mais específico a produtos de tecnologia, é COTS (*Commercial Off-The-Shelf*);

***network-on-chip*:** ou NoC. É um paradigma de comunicação com múltiplos canais ponto-a-ponto implementados com base em microprocessadores de rede. O resultado é a separação entre computação e comunição, possibilitando *overlapping*. Há um modelo padrão chamado VIA (*Virtual Interface Architecture*) (DUNNING et al., 1998);

***overlapping*:** significa sobreposição. Em computação, é a habilidade de executar tarefas em paralelo, de forma independente. É capaz de reduzir o *overhead*;

***overhead*:** é um fator redutor do desempenho. Ocorre, por exemplo, quando é necessário acessar algum recurso computacional no ambiente paralelo e o tempo de execução total da aplicação aumenta, deixando algum processo da aplicação ocioso até que a condição de dependência seja satisfeita e seu fluxo de instruções continue.

# 1 Introdução

Computadores de alto desempenho, também conhecidos como supercomputadores, tornaram-se imprescindíveis como ferramenta de auxílio ou para a resolução de alguns problemas que ainda não podem ser resolvidos em um período aceitável de tempo, principalmente das áreas científica e de engenharia (KUMAR et al., 1994). São problemas fundamentais dessas áreas que geralmente envolvem modelagem numérica e simulações, conhecidos como *grand challenges* (em português, grandes desafios). A solução destes problemas, de enorme impacto econômico ou científico, pode ser auxiliada pela aplicação de técnicas e recursos computacionais de alto desempenho (KUMAR et al., 1994).

Pouco mais de uma década após seu surgimento, os agregados de computadores (ou *clusters*, em inglês) tornaram-se muito populares na comunidade acerca da computação de alto desempenho (ou *high performance computing*, em inglês) pois podem atingir configurações massivamente paralelas de forma distribuída. Hoje em dia, os *clusters* representam a maior fatia das soluções adotadas. Vide, por exemplo, a lista dos 500 supercomputadores mais rápidos do mundo, conhecida por TOP500 (MEUER et al., 2008), cuja atualização ocorre a cada seis meses. Em novembro de 2007, dos 500 supercomputadores, 406 ou 81,20% são classificados como *clusters*. E em junho de 2008, são 80%.

Apesar da consolidação dos *clusters* como solução para prover alto desempenho, a escolha dos seus componentes, tais como os processadores que equipam os computadores ou a rede de interconexão que efetivamente agrega os recursos computacionais, está submetida à oferta disponibilizada pelo mercado no momento de sua construção.

De fato, o mercado de computadores sofreu uma mudança com o lançamento dos processadores *multi-core*, que oferecem suporte nativo a processamento paralelo. Tendo em vista o acesso a estes processadores como *commodity*, sua inserção em ambientes de *cluster* já é fato. Também como *commodity*, as taxas de transferência da ordem de megabytes por segundo proporcionadas pelas redes de interconexão Gigabit Ethernet surgem como uma alternativa de baixo custo quando se pensa em compor um novo *cluster*.

Este trabalho de dissertação concentra seu esforço no estudo empírico de três ambientes de *cluster* distintos, sendo que dois deles são ambientes operacionais de empresas, equipados com computadores multiprocessados. Vale ressaltar que são experimentados computadores com múltiplos processadores *mono-core* (Intel® Xeon DP) e *multi-core* (AMD Opteron(TM) *dual-core* e *quad-core*), interconectados por redes Gigabit Ethernet, e outro ambiente interconectado por uma rede Myrinet (BODEN et al., 1995).

Para tanto, foi realizada uma pesquisa sobre a literatura relacionada à computação paralela, distribuída e de alto desempenho, concentrando o foco em ambientes de *cluster*. Conforme Zamani (2005), o desempenho de aplicações executadas em ambientes de *cluster* depende principalmente da escolha do modelo de programação paralela, das características da própria aplicação quanto às necessidades de computação e comunicação, e o desempenho do subsistema de comunicação. Portanto, estas variáveis condicionam o desempenho de aplicações executadas em agregados de computadores.

No Capítulo 3, é apresentado o primeiro estudo de caso, em um ambiente de *cluster* experimental. Foi realizado como um exercício empírico sobre a relação entre tecnologias de rede de interconexão e características inerentes a aplicações paralelas. Com isso, pretendeu-se entrar em contato com a computação de alto desempenho por meio de arquiteturas paralelas distribuídas como os ambientes de *cluster*. Ademais, com este estudo de caso, foi possível conhecer e analisar os mecanismos de interação entre processos e também tomar conhecimento das diferenças em desempenho oferecido por sistemas de comunicação distintos, como a tecnologia de rede Myrinet face à tecnologia Ethernet.

Já os outros dois estudos de casos, apresentados no Capítulo 4, foram realizados em agregados de computadores pertencentes a organizações, visando uma aproximação com a realidade da computação de alto desempenho em ambientes de produção.

Outro objetivo desses estudos é a compreensão das métricas envolvidas bem como analisar o comportamento do desempenho de agregados de computadores multiprocessados, tendo em mente as variáveis condicionantes citadas acima. Na busca pela otimização do desempenho destes ambientes para aplicações científicas *grand challenge*, como por exemplo a modelagem numérica de previsão meteorológica, tomou-se como alternativa uma abordagem híbrida no que tange ao modelo de programação paralela.

Portanto, dentre as contribuições do trabalho de pesquisa e experimentação desenvolvido, o desempenho desses ambientes de produção na execução de aplicações científicas deverá apresentar uma redução do tempo de execução em comparação à sua condição anterior, sem nenhuma especialização para a aplicação em questão.

## 1.1 Motivação

Desestimulada por limitações físicas e pelo alto consumo de energia, é provável que a produção de processadores seqüenciais torne-se diminuta dentro de poucos anos, à medida que a tecnologia de processadores *multi-core* seja viabilizada para processadores *many-core*, com até dezenas de núcleos de processamento em um mesmo processador. Porém, mais do que entender as vantagens e desvantagens desses processadores por si só, faz-se necessário compreender o impacto de sua inserção em ambientes de *cluster* para a computação de alto desempenho.

Diferentemente da situação inicial, em que os ambientes de *cluster* eram compostos apenas por processadores seqüenciais e a predição do seu desempenho já vinha sendo aprofundada há algum tempo pela comunidade científica, o contexto atual é novo e merece a devida atenção. É possível, por exemplo, que as limitações da tecnologia Ethernet relativas ao processamento dos eventos de comunicação sejam amenizadas com a presença de múltiplos núcleos de processamento em um mesmo computador, o que será abordado por um dos estudos de casos.

Além disso, com um crescente número de núcleos ou *cores* em um mesmo processador, a importância de sistemas de comunicação como as arquiteturas *network-on-chip* aumenta (FLICH et al., 2008), não para a interconexão entre computadores, e sim para a interconexão dos próprios núcleos de processamento dentro do mesmo processador e internamente ao computador. Portanto, a maior densidade de núcleos de processamento poderá sobrecarregar o subsistema de comunicação interna a cada computador agregado. Isso altera o comportamento do desempenho das aplicações em agregados de computadores e, por ser uma tecnologia relativamente recente, é um campo aberto para pesquisa.

Sendo assim, a motivação deste trabalho surgiu com os olhos voltados a esse novo contexto e evoluiu no sentido de melhor implementar e utilizar agregados de computadores *multi-core* para otimizar o desempenho de aplicações científicas. Além disso, a intenção era aproximar-se da realidade da computação de alto desempenho em ambientes de produção. O passo seguinte foi conhecer os paradigmas de programação paralela que melhor se adaptam aos ambientes computacionais em questão através da realização dos estudos empíricos com aplicações. Com isso, pretende-se compreender melhor as vantagens e desvantangens envolvidas nesses ambientes enquanto sistemas paralelos distribuídos, com o foco voltado à modelagem de sistemas de alto desempenho.

## 1.2 Objetivos Gerais

O presente trabalho tem como objetivo geral elaborar um estudo aprofundado sobre os mecanismos e componentes envolvidos na comunicação entre processos paralelos e distribuídos em ambientes de *cluster*, como por exemplo, a rede de interconexão ou o modelo de programação paralela, tendo em vista o suprimento das demandas por computação de alto desempenho no âmbito de aplicações científicas, como as aplicações *grand challenge*, em ambientes de produção.

Sendo assim, através da otimização do desempenho de ambientes de *cluster*, principalmente com técnicas e mecanismos vistos como melhores práticas, e não com a inserção de novos componentes de *hardware*, tem-se em vista a formação de *clusters* eficientes, de pequeno e médio porte, especializados para determinada aplicação científica com demanda por computação de alto desempenho.

## 1.3 Objetivos Específicos

- Estudar ambientes de computação de alto desempenho bem como conhecer as tecnologias de rede de interconexão para estes ambientes e sua relação com características inerentes às aplicações paralelas;
- Conhecer e analisar os mecanismos de interação entre processos e os modelos de programação paralela considerando arquiteturas paralelas distribuídas como ambientes de *cluster*;
- Investigar o impacto da presença de computadores multiprocessados no desempenho de ambientes de *cluster*, haja vista a tendência atual no que diz respeito à disponibilidade de processadores equipados com múltiplos e possivelmente numerosos núcleos de processamento;
- Compreender as métricas comumente adotadas em avaliações de desempenho, bem como a importância e a diversidade de algoritmos de *benchmarking*;
- Conhecer técnicas e mecanismos utilizados no processo de otimização do desempenho de agregados de computadores, com o foco voltado à modelagem de sistemas de alto desempenho;
- Envolver-se com aplicações paralelas científicas, inclusive com alguma aplicação *grand challenge*;

- Realizar estudos empíricos em ambientes operacionais de *cluster*, visando uma aproximação com a realidade da computação de alto desempenho em ambientes de produção.

## 1.4 Metodologia

Este trabalho de dissertação concentra seu esforço no estudo empírico de três ambientes de *cluster* distintos, sendo que dois deles são ambientes operacionais de empresas, equipados com computadores multiprocessados. Para tanto, foi realizada uma pesquisa sobre a literatura relacionada à computação paralela, distribuída e de alto desempenho, concentrando o foco em ambientes de *cluster*. Esta investigação, que contextualiza e embasa os estudos empíricos, é apresentada ao longo dos Capítulos 2 e 3.

Também no Capítulo 3, é apresentado o primeiro estudo de caso em um ambiente de *cluster* experimental. Foi realizado como um exercício empírico sobre a relação entre tecnologias de rede de interconexão e características inerentes a aplicações paralelas, com a tecnologia de rede Myrinet face à tecnologia Ethernet. Ainda no Capítulo 3, como um resultado do trabalho de revisão bibliográfica, são apresentados trabalhos científicos que apontam algumas tendências com relação à computação de alto desempenho em agregados de computadores, principalmente quanto ao impacto de processadores com uma maior densidade de núcleos de processamento nesse contexto.

Já os outros dois estudos de casos, apresentados no Capítulo 4, foram realizados em agregados de computadores pertencentes a organizações, em uma aproximação com a realidade da computação de alto desempenho em ambientes de produção. Dentre as contribuições do trabalho de pesquisa e experimentação desenvolvido, o desempenho desses ambientes de produção na execução de aplicações científicas deverá apresentar uma redução do tempo de execução em comparação à sua condição anterior, sem nenhuma especialização à aplicação em questão.

O processo adotado para a experimentação e validação dos resultados leva em consideração variações que dizem respeito aos processadores e à organização de *clusters*, e também à aplicação, com relação à granularidade e ao modelo de programação utilizado. Além disso, cada experimento é efetuado pelo menos 5 vezes, para prover consistência estatística ao cálculo da média aritmética. Para a análise desses resultados, toma-se como base as seguintes métricas de desempenho, detalhadas no Capítulo 3: tempo de execução, *speedup*, eficiência e escalabilidade.

Em nível de processadores, configurações com múltiplos processadores seqüenciais (*mono-core* SMP's) e também processadores com múltiplos núcleos de processamento (*multi-core* ou CMP-SMP's) serão submetidos à experimentação.

Em relação à organização de *clusters*, serão avaliadas algumas configurações no que diz respeito à distribuição dos processos e o número de núcleos de processamento utilizados por computador agregado, visando a otimização do desempenho. Além disso, alguns resultados coletados em um único computador multiprocessado com até 8 núcleos de processamento serão apresentados em comparação aos resultados obtidos com as configurações adotadas nos ambientes de *cluster* em estudo.

Em nível de aplicação, os experimentos irão abordar os modelos de programação paralela baseado em passagem de mensagem e baseado em memória compartilhada, bem como o modelo híbrido. Serão utilizadas categorias distintas de *workload*, desde *micro-benchmarks* até aplicações completas, para a avaliação de desempenho dos ambientes, servindo como um reforço na validação dos resultados empíricos. São eles:

1. ***b_eff (RABENSEIFNER; KONIGES, 2001):*** *micro-benchmark* de rede, utilizado para capturar características específicas dos ambientes de *cluster* estudados, relativos à comunicação entre processos de interesse primário, como latência e taxa de transferência (COULOURIS; DOLLIMORE; KINDBERG, 2005);

2. ***Algoritmos com matrizes (PACHECO, 1996):*** a dependência de dados dos algoritmos de multiplicação e transposição de matrizes resulta em comportamentos distintos quanto à demanda por comunicação entre os processos paralelos. Foi utilizado no estudo de caso realizado para entrar em contato com as diferentes tecnologias de rede de interconexão e sua relação com o desempenho de um ambiente de *cluster* experimental executando aplicações de granularidades distintas;

3. ***NAS Parallel Benchmarks (NPB) (BAILEY et al., 1991):*** conjunto de benchmarks específicos para a avaliação de computadores paralelos, utilizado no Caso 1 para investigar características do *cluster* e possibilidades de ganho de desempenho tendo em vista aplicações de diversas granularidades.

4. ***Weather Research and Forecasting Model (WRF) (MICHALAKES et al., 1999):*** aplicação de previsão numérica do tempo. Foi escolhida visando uma aproximação com a realidade da computação de alto desempenho em ambientes de produção, utilizando-se do mesmo modelo e conjunto de dados disponíveis operacionalmente.

Enfim, no Capítulo 5, são apresentadas as considerações finais e as indicações para trabalhos futuros. Em anexo, são citados os artigos científicos aprovados e publicados, e também os artigos submetidos à aprovação em congressos e conferências.

## 1.5 Trabalhos Relacionados

Os trabalhos relacionados à pesquisa desenvolvida nesta dissertação podem ser subdivididos em três categorias, descritas a seguir.

### 1.5.1 Caracterização de aplicações científicas

O seguintes trabalhos de caracterização das aplicações científicas estão relacionados com o presente trabalho pois servem como base para descrição das aplicações utilizadas nas análises.

A versão utilizada do *micro-benchmark* de rede *b_eff*, que foi desenvolvido por Rabenseifner e Koniges (2001), faz parte do conjunto *HPC Challenge Benchmark* (LUSZCZEK et al., 2006). Já os algoritmos de manipulação de matrizes foram desenvolvidos e caracterizados por Pacheco (1996).

Com ao relação ao NAS Parallel Benchmarks baseado em MPI, os trabalhos de Kim e Lilja (1998), Tabe e Stout (1999), Martin (1999) e Faraj e Yuan (2002) concentram-se na determinação das características com relação aos tipos de comunicação, tamanho das mensagens, quantidade e freqüência das fases de comunicação. Além disso, os trabalhos de Sun, Wang e Xu (1997) e Subhlok, Venkataramaiah e Singh (2002) também consideram o volume de dados comunicado, bem como a porcentagem de utilização de processador e memória a fim de caracterizar os algoritmos do NPB.

Quanto às características de funcionamento do modelo WRF, foram encontradas descrições e experimentos em trabalhos de Kerbyson, Barker e Davis (2007), National Center for Atmospheric Research (2007), Amstrong et al. (2006), Michalakes et al. (1999), Michalakes et al. (2004) e Zamani e Afsahi (2005), ajudando na caracterização apresentada no Capítulo 4.

Além de analisar o comportamento de processadores *multi-core* e seu impacto no desempenho, o trabalho de Alam et al. (2006) caracteriza diversas aplicações científicas, apresentando uma metodologia para validação dos resultados semelhante ao presente trabalho com relação à utilização de diferentes categorias de *workload* de origem científica,

desde micro-benchmarks até aplicações completas.

### 1.5.2 Agregados de computadores com processadores *multi-core*

A inserção de processadores *multi-core* em ambientes de *cluster* é investigada por alguns trabalhos.

Chai, Hartono e Panda (2006) detêm o foco sobre a comunicação entre processos intra-nó (residentes no mesmo computador) baseados em MPI e apresentam uma implementação específica para este tipo de comunicação, tendo como objetivo melhor desempenho e escalabilidade. Pourreza e Graham (2007) também analisam a comunicação intra-nó e apresentam os ganhos obtidos com a alocação estática de processos vizinhos em núcleos de processamento de uma mesma máquina. Além disso, estes dois trabalhos utilizam apenas *benchmarks* específicos de rede para a coleta dos resultados.

Já o trabalho desenvolvido por Chai, Gao e Panda (2007) apresenta modelos de avaliação de desempenho específicos para ambientes de *cluster* com processadores *multi-core* da fabricante Intel®, que diferem no nível arquitetural dos processadores *multi-core* AMD utilizados neste trabalho. Ademais, este trabalho também se assemelha ao presente trabalho no que tange às motivações e, em parte, à metodologia.

### 1.5.3 Redes de interconexão *network-on-chip* em *clusters*

A investigação de redes de interconexão baseadas na arquitetura *network-on-chip*, como Myrinet (BODEN et al., 1995) e Infiniband (CASSIDAY, 2000), em ambientes de *cluster* está relacionada com dois dos estudos empíricos realizados. Como há muitos trabalhos desenvolvidos sobre este assunto, apenas alguns trabalhos com um maior grau de identificação serão citados.

Lobosco, Costa e Amorim (2002) fazem um extenso estudo sobre o desempenho de uma rede Myrinet em comparação a duas redes do tipo Ethernet, uma FastEthernet e outra Gigabit Ethernet, sendo que a avaliação de desempenho é feita com o NAS Parallel Benchmaks.

Brightwell e Underwood (2004) qualificam as fontes de ganho em desempenho possíveis com o uso dessa arquitetura de rede, embora não sejam exploradas eficientemente em nível de aplicação. Para tal estudo, coletam resultados de ambientes de *cluster* com redes de interconexão baseadas na arquitetura *network-on-chip* e também Ethernet com

*benchmarks* de rede e com o NPB.

Pinto, Mendonça e Dantas (2008) analisam o ganho em desempenho proveniente do uso de redes de interconexão baseadas na arquitetura *network-on-chip* em comparação com redes do tipo Ethernet, considerando aplicações de granularidades distintas e suas curvas de *speedup* em função do crescimento do tamanho do problema.

# 2 *Computação Paralela e Distribuída*

Desde a invenção do computador, a busca por um maior desempenho computacional tem sido uma constante para engenheiros, cientistas e usuários de computadores. Nas últimas décadas, o acesso em larga escala a um crescente poder computacional acabou por incentivar a adoção do computador para a resolução de problemas de toda ordem, cada vez mais complexos. E essa tendência está longe do seu fim.

O meio tradicional para se obter alto desempenho é através de computadores paralelos, ou seja, computadores capazes de operar duas ou mais tarefas simultaneamente. Esta idéia não é nova, datando de meados de 1950 (ALMASI; GOTTLIEB, 1994). Hoje em dia, computadores paralelos de uso pessoal até computadores massivamente paralelos, com milhares de processadores, estão disponíveis no mercado. Um exemplo corrente são os processadores *multi-core* (com múltiplos núcleos de processamento), atualmente acessíveis como *commodity*, e que em poucos anos deverão colocar os computadores pessoais monoprocessados nas páginas de história da computação. Nesse sentido, a computação paralela é vista como uma evolução da computação seqüencial. Todavia, essa tendência nem sempre foi unânime.

A Lei de Grosch (PARHAMI, 2002), de 1950, afirma que a melhor razão custo/benefício (leia-se, custo/desempenho) é obtida com um único processador mais poderoso, e nunca com um conjunto de processadores menos poderosos de mesmo custo total. Esta lei foi considerada válida à época dos gigantescos *mainframes* e dos minicomputadores, até o final da década de 70, quando a computação paralela e mesmo os computadores baseados em microprocessadores estavam em fase de consolidação. Sua veracidade foi desacreditada em função das crescentes dificuldades de produzir processadores seqüenciais mais rápidos e complexos ao passo que sistemas com múltiplos processadores se mostravam como um caminho mais fácil e mais barato para conquistar melhor desempenho (ENSLOW, 1977).

A Lei de Moore (KUMAR et al., 1994; QUINN, 1994), de 1965, alega que o número

de transistores (componente fundamental de um processador) de um circuito integrado cresce exponencialmente, sem custos adicionais, dobrando a cada dezoito a vinte e quatro meses aproximadamente. Essa tendência vem sendo corroborada pela história dos microprocessadores, como mostra a evolução apresentada na Figura 1.

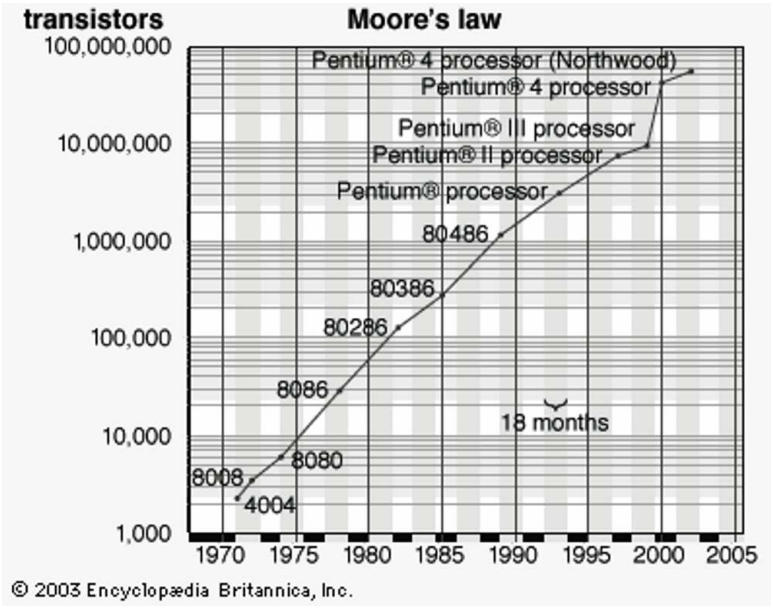



Figura 1: Evolução dos microprocessadores e a Lei de Moore

Apesar desse avanço constante, a capacidade computacional de um processador não cresce linearmente com o aumento do número de transistores. Limitações físicas, como a da velocidade finita de propagação por cabo, conhecida por *speed-of-light argument* em inglês, têm se mostrado como barreiras cada vez mais fortes, que dificultam ganhos de desempenho com o aumento do número de transistores. Por exemplo, temperaturas relativamente altas podem causar problemas de alto consumo de energia, altos custos para o resfriamento do processador e até mesmo de confiabilidade em função do perigo de interferências entre os componentes (CHAPARRO et al., 2007). Com o aumento de transistores por unidade de área, maior energia é consumida e, portanto, maior a temperatura do sistema. Os primeiros microprocessadores consumiam décimos de um watt, enquanto um Pentium® 4 operando a 2 GHz consome cerca de 100 watts. Em 2003, já haviam previsões de que a principal limitação em um futuro próximo estaria relacionada à energia, e não ao número de transistores (HENNESSY; PATTERSON, 2003).

Em função disso, houve uma guinada na indústria de processadores, como se pode ver na Figura 2 (INTEL®, 2006). Primeiro, o foco virou-se para a produção de microprocessadores mais econômicos em termos de consumo de energia, e então, em direção a modelos de processamento paralelo. Tome como exemplo o recente lançamento no mercado dos

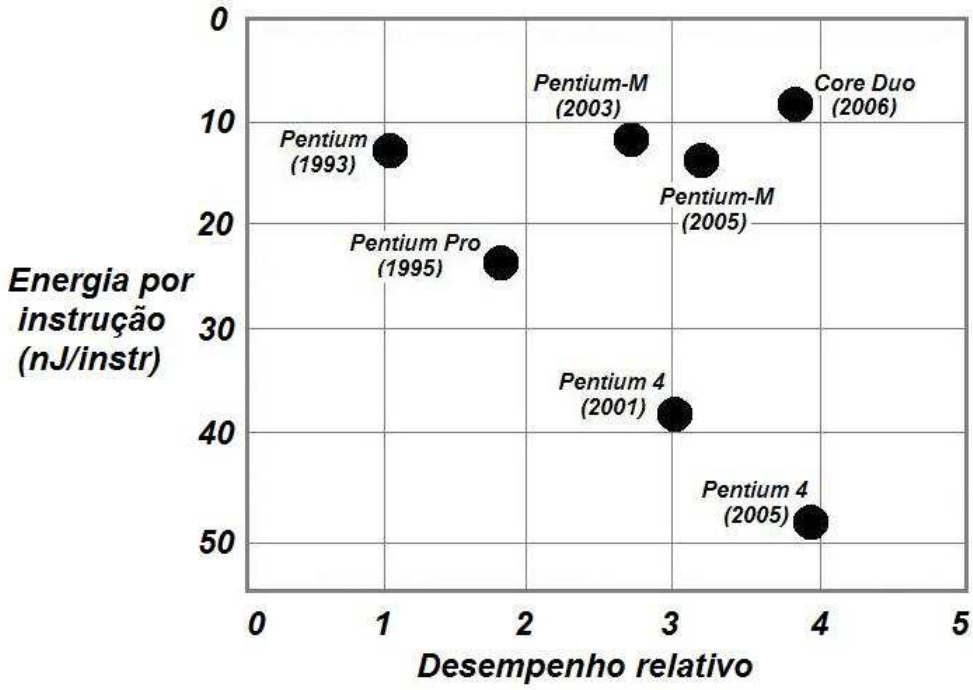



Figura 2: Gráfico de consumo de energia por instrução (processadores Intel®)

processadores *multi-core*, com o objetivo de prover mais capacidade de processamento tanto para computadores pessoais como servidores. Enfim, embora a Lei de Moore seja verdadeira no que diz respeito ao crescimento do número de transistores, as limitações físicas cada vez mais acentuadas não permitem que esta lei continue a incentivar modelos de processamento seqüenciais.

Em resumo, a capacidade do modelo de processamento seqüencial está se esgotando em função de limitações físicas intransponíveis, o que abre um caminho praticamente exclusivo (pelo menos por enquanto) para o modelo de processamento paralelo na incessante busca por um desempenho maior.

## 2.1 Taxonomia de Arquiteturas Paralelas

Sistemas de computação podem ser classificados à luz de vários aspectos arquiteturais. Apresentaremos dois esquemas formais de classificação de computadores paralelos, um proposto por Flynn e outro por Johnson. A seguir uma classificação tradicional que faz distinção quanto à forma como os processadores estão conectados à memória. Serão dados exemplos quando for conveniente. Ademais, estas classificações auxiliarão no entendimento dos paradigmas de programação paralela discutidos na seção seguinte.

### 2.1.1 Taxonomia de Flynn

Amplamente aceita, a Taxonomia de Flynn (FLYNN, 1972) leva em consideração os mecanismos de controle, a saber, o número de fluxos de instrução (*instruction streams*) e o número de fluxos de dados (*data streams*), para classificar os computadores. Sendo assim, são quatro as classes de computadores, indicadas na Tabela 1.

Tabela 1: Taxonomia de Flynn (FLYNN, 1972)

| | *Single instruction stream* | *Multiple instruction streams* |
|---|---|---|
| *Single data stream* | SISD | MISD |
| *Multiple data streams* | SIMD | MIMD |

A classe SISD (*Single Instruction, Single Data*) refere-se aos tradicionais computadores seqüenciais, baseados na arquitetura de von Neumann (FOSTER, 1995). A Figura 3 apresenta uma máquina seqüencial de von Neumann, na qual há uma única seqüência de instruções (uma unidade de controle e uma unidade de processamento) operando sobre uma única seqüência de dados (uma memória), sendo que uma instrução é decodoficada a cada unidade de tempo.

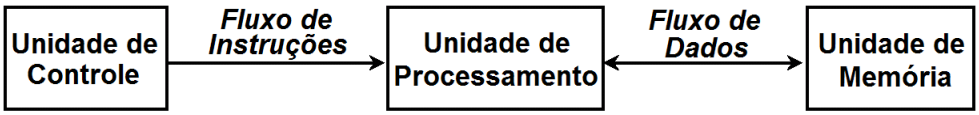



Figura 3: Computador SISD (arquitetura seqüencial de von Neumann) (FOSTER, 1995)

Processadores vetoriais, como os presentes em placas de processamento gráfico, são um exemplo da classe SIMD (*Single Instruction, Multiple Data*). Computadores SIMD possuem uma única unidade de controle para múltiplas unidades de processamento, ou seja, uma mesma instrução é executada por todas unidades de processamento mas sobre diferentes dados. Esse tipo de processador é comumente específico a uma determinada especialidade.

Para fins de completude do esquema, a classe MISD (*Multiple Instruction, Single Data*) também é descrita, já que é difícil e, na maioria dos casos, inútil se construir um computador deste tipo.

Enfim, sistemas pertencentes à classe MIMD (*Multiple Instruction, Multiple Data*) possuem múltiplos processadores que podem operar independentemente sobre diversos

fluxos de dados. Esta classe representa a grande maioria dos computadores paralelos atualmente e, por isso, merece uma atenção especial.

### 2.1.2 Taxonomia de Johnson

A forma como estão organizados processadores e memórias em computadores MIMD determinam como se dá a interação entre os processos em execução paralela. Tendo em vista a importância da classe MIMD, definida pela taxonomia de Flynn, Johnson propôs um esquema para o detalhamento desta classe (JOHNSON, 1988). Os critérios desta classificação levam em consideração aspectos arquiteturais que têm impacto relevante no desempenho do sistema, a saber: o mecanismo de comunicação/sincronização e a estrutura da memória. Sendo assim, os computadores MIMD são classificados em quatro classes, indicadas na Tabela 2.

Tabela 2: Taxonomia de computadores MIMD por Johnson (JOHNSON, 1988)

| | *Variáveis Compartilhadas* | *Passagem de Mensagens* |
|---|---|---|
| *Memória Global* | GMSV | GMMP |
| *Memória Distribuída* | DMSV | DMMP |

Esta taxonomia é interessante pois torna fácil a discussão das fontes de *overhead* envolvidas, seja por contenção do acesso à memória global, seja por sobrecarga da rede de interconexão decorrente de um maior volume de passagem de mensagem entre os processos.

Os sistemas classificados como GMSV, DMMP e DMSV são comuns e mapeiam os três grupos principais que serão detalhados em seguida. Já sistemas do tipo GMMP não são comuns, exceto quando se tem em mente sistemas com uma memória virtual que, embora global, provavelmente é distribuída, como em sistemas chamados de memória compartilhada virtual (*Virtual Shared Memory* ou VSM, em inglês) (RAINA, 1992).

### 2.1.3 Sistemas de Memória Compartilhada: Multiprocessadores

Como o próprio nome sugere, neste tipo de sistema há um único espaço de endereçamento, global e comum a todos os processadores. Geralmente, a memória consiste de vários módulos, de número não necessariamente igual ao número de processadores do sistema. A comunicação e a sincronização entre os processos em execução é feita através de variáveis compartilhadas, acessíveis a todos eles, o que facilita a programação des-

tes sistemas. A Figura 4 mostra a organização básica de um multiprocessador com sua memória global característica.

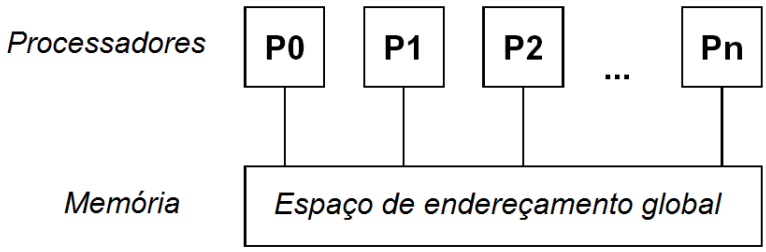



Figura 4: Sistema de memória compartilhada (GMSV ou DMSV)

Estes sistemas são considerados fortemente acoplados ou ligados. Ademais, há duas formas distintas de acesso à memória global, diretamente relacionadas ao modo como esta memória está interconectada aos processadores. Essa distinção é descrita a seguir.

#### 2.1.3.1 Memória Compartilhada Centralizada (UMA)

Nesse tipo de sistema, também referenciado como sistemas UMA (*Uniform Memory Access*), todos os processadores compartilham uma única memória centralizada de modo que o tempo de acesso é aproximadamente o mesmo para qualquer um deles e, por isso, o acesso à memória é dito uniforme. Este tipo de sistema é chamado de **SMP (*Symmetric Multiprocessor*)** (HENNESSY; PATTERSON, 2003). Porém, o número de processadores fica limitado a geralmente 64 processadores em função do congestionamento decorrente da existência de um único caminho de acesso à memória. Por isso, diz-se que a escalabilidade dos sistemas UMA é limitada.

#### 2.1.3.2 Memória Compartilhada Distribuída (NUMA)

Também conhecido como sistema NUMA (*Non-Uniform Memory Access*), nada mais é do que a combinação de dois ou mais subsistemas UMA, interligados por uma rede de interconexão, como mostra a Figura 5. Todos processadores têm acesso a qualquer parte da memória global mesmo que com tempos de acessos diferenciados em função da localização física da parte acessada, o que o torna um sistema assimétrico. Essa abordagem foi a saída adotada para melhorar a escalabilidade dos sistemas de memória compartilhada, possibilitando um número maior de processadores (HENNESSY; PATTERSON, 2003) nos também chamados multiprocessadores.

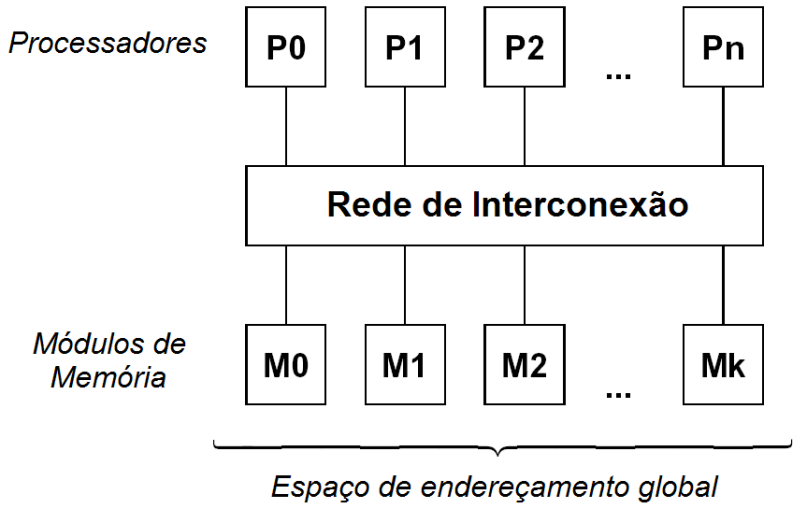



Figura 5: Sistema de memória compartilhada distribuída (DMSV)

### 2.1.4 Sistemas de Memória Distribuída: Multicomputadores

Em um sistema de memória distribuída, o espaço de endereçamento consiste de múltiplos espaços privados de memória, logicamente separados e que, portanto, não são diretamente acessíveis por processadores remotos. Ou seja, cada conjunto de processador e memória está localizado em computadores fisicamente distintos. Por isso, esses sistemas são chamados multicomputadores. A comunicação e sincronização entre os processos em execução dá-se através de mensagens trocadas pela rede de interconexão (FOSTER, 1995; JORDAN; ALAGHBAND, 2003), que interliga todos os computadores, como mostra a Figura 6. Sendo assim, a dificuldade de programação desses sistemas é maior, quando comparada aos sistemas de memória compartilhada, em função da necessidade de chamadas explícitas às primitivas de comunicação e sincronização.

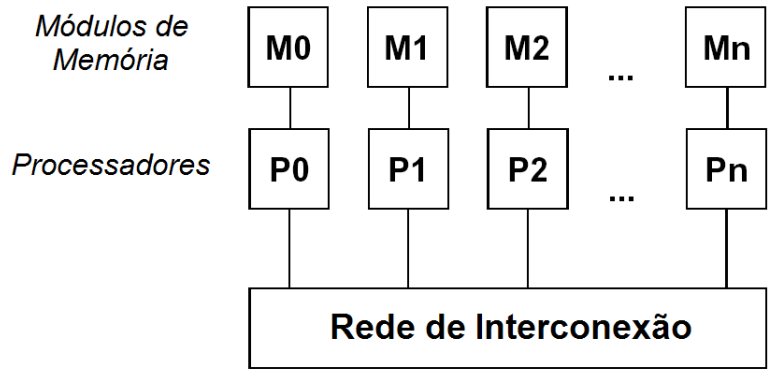



Figura 6: Sistema de memória distribuída (DMMP)

Diferentemente dos sistemas baseados em memória compartilhada, sistemas baseados em memória distribuída como os multicomputadores apresentam muito boa escalabilidade, suportando até milhares de processadores (KUMAR et al., 1994). São considerados fracamente acoplados ou ligados. O exemplo típico desse tipo de sistema paralelo são os *clusters* ou agregados de computadores, cuja importância vem crescendo principalmente para a resolução de problemas que demandam ambientes computacionais de alto desempenho. Os ambiente de *cluster* serão abordados em detalhes no decorrer deste trabalho.

## 2.2 Paradigmas de Programação Paralela

A fim de executar uma aplicação em paralelo, não basta ter à disposição um computador paralelo. É imprescindível que o código do problema em questão tenha sido paralelizado, manual ou automaticamente. Como mencionado anteriormente, os processadores em execução paralela devem interagir através do compartilhamento de informações para chegarem a um resultado final. Tais mecanismos de interação serão providos justamente pela linguagem de programação paralela.

Existem diversas técnicas para implementar um algoritmo em paralelo. A escolha do paradigma de programação paralela não é necessariamente dependente da arquitetura do computador paralelo. Por outro lado, como é de se esperar, a elaboração de uma linguagem de programação paralela, baseada em algum paradigma, geralmente é feita tendo em vista uma aplicação e um sistema computacional em específico, para os quais, provavelmente, terá um desempenho melhor. Além disso, as aplicações em si oferecem comportamentos diversos, resultando em tipos diferentes de paralelismo, e portanto também têm parte na definição do paradigma. Também podemos apontar a dificuldade de programação das linguagens disponíveis como outro fator importante na escolha do paradigma. Por exemplo, uma linguagem pode demandar mais esforço do programador, enquanto outras demandam menos mas geralmente resultam em códigos menos eficientes.

### 2.2.1 Granularidade da Aplicação

O grau de paralelismo explorado depende do modo como acontece a cooperação entre os processadores. Por sua vez, isso depende diretamente da aplicação, seja em função de características intrínsecas ao algoritmo, seja por causa da implementação específica do problema, ou ainda, do paradigma de programação adotado.

Esta questão está diretamente relacionada com o tempo gasto em trocas de in-

formações entre os processadores. Alguns processadores ou processos geram resultados intermediários que servirão de entrada para que outros processos continuem a busca pela solução final do problema, e portanto, os dados do problema em execução paralela precisam ser compartilhados.

A granularidade de uma aplicação paralela pode ser definida como a razão do tempo de computação em relação ao tempo gasto com comunicação (e sincronização) (KUMAR et al., 1994).

$$Granularidade = \frac{T_{comp}}{T_{comm}} \tag{2.1}$$

Sendo assim, se a necessidade de comunicação entre os processos é infreqüente, com longos períodos de processamento independente, diz-se que a aplicação é de granularidade grossa (*coarse-grained*), ou ainda, fracamente acoplada ou ligada. Por outro lado, quando há uma grande dependência de dados, a necessidade de comunicação entre os processos torna-se freqüente, seja para sincronização ou para troca de dados entre os processos, resultando em uma aplicação de granularidade fina (*fine-grained*), também chamada fortemente acoplada ou ligada (JORDAN; ALAGHBAND, 2003).

Entretanto, não há uma diferenciação precisa entre estes graus de paralelismo. Portanto, com base na definição formal acima, costuma-se classificar a granularidade de uma aplicação em três classes. Assim, definimos granularidade:

- **grossa (*coarse-grained*)**: necessidade de comunicação infreqüente, com longos períodos de computação independente de comunicação, e volume total de comunicação pequeno ou até mesmo desprezível.
- **média (*medium-grained*)**: necessidade de comunicação infreqüente, com períodos médios a longos de computação independente de comunicação, e volume total de comunicação considerável.
- **fina (*fine-grained*)**: necessidade de comunicação freqüente, com curtos períodos de computação seguidos de comunicação, e volume total de comunicação considerável.

### 2.2.2 Programação Paralela Explícita e Implícita

Quando a abordagem de programação paralela explícita é adotada, a forma como os processadores irão cooperar na resolução de um dado problema deve ser explicitamente especificada. Sendo assim, o problema é paralelizado manualmente. Neste caso, a função do compilador é simples e direta: gerar o código em linguagem de máquina tal qual foi definido pelo programador, cuja função é mais difícil.

Já a programação paralela implícita sobrecarrega o compilador com a paralelização de grande parte do código, enquanto o esforço investido pelo programador é aliviado. Embora a necessidade de implementação do problema ainda exista, tal processo é facilitado pois o programador pode utilizar uma linguagem de programação seqüencial. Por outro lado, a eficiência do código paralelo fica prejudicada com a geração automática de código devido à enorme dificuldade desta tarefa. O compilador deve analisar e entender as dependências entre diversas partes do código seqüencial para garantir um mapeamento eficiente para a arquitetura do computador paralelo. Além disso, um mesmo problema pode ser implementado de diversas formas em uma linguagem seqüencial, sendo que umas facilitam e outras dificultam a geração automática do código paralelizado.

Portanto, o esforço dispendido na implementação de algoritmos paralelos é, grosso modo, proporcional à eficiência e ao desempenho resultantes do código paralelo gerado (KUMAR et al., 1994).

### 2.2.3 Paralelismo Funcional e de Dados (MPMD e SPMD)

O paralelismo funcional ou de tarefas dá-se quando existem duas ou mais seqüências de instruções distintas e independentes que podem, portanto, serem executadas simultaneamente, seja sobre o mesmo fluxo de dados ou sobre diferentes fluxos de dados, o que é mais comum. Neste segundo caso, a aplicação segue o modelo de programação MPMD (*Multiple Program, Multiple Data*). Cada seqüência de instruções pode ser vista como um caminho de execução, uma tarefa distinta, comumente chamado de *thread*. Esse tipo de problema se adapta a computadores paralelos do tipo MIMD (vide Tabela 1), pois os computadores SIMD não são capazes de explorar eficientemente o paralelismo funcional por apresentarem apenas um fluxo de instruções.

O paralelismo de dados acontece quando diversos dados são processados por uma mesma seqüência de instruções. Ou seja, elementos de dados podem ser distribuídos para vários processadores que executam computações idênticas sobre esses dados. Algoritmos

com paralelismo de dados são mapeados naturalmente em computadores SIMD, resultando em um único fluxo de instruções síncrono. Por outro lado, a execução síncrona do início ao fim de um algoritmo em computadores MIMD, além de não ser necessária na maioria dos casos, dificulta sua utilização em função de sincronizações globais a cada instrução. Então, uma solução traduz-se no modelo de programação SPMD (*Single Program, Multiple Data*), segundo o qual cada processador executa o mesmo programa porém de modo assíncrono. A sincronização só ocorre quando há a necessidade de troca de informações. Assim, o paralelismo de dados pode ser eficientemente explorado também em computadores MIMD.

Os dois tipos de paralelismo, funcional e de dados, podem estar presentes em uma mesma aplicação. Com efeito, isso acontece em muitos problemas. O impacto do paralelismo funcional no desempenho é geralmente independente do tamanho do problema e, por isso, seu efeito torna-se limitado. Por outro lado, a quantidade de paralelismo de dados aumenta com o tamanho do problema, o que o torna ainda mais importante para a eficiente utilização de sistemas com muitos processadores.

Linguagens baseadas no paralelismo de dados são mais fáceis de programar, pois fornecem uma sintaxe de alto nível. O programador deve indicar como serão distribuídos as porções de dados entre os processadores e, então, o compilador se encarrega da tradução para o modelo SPMD, gerando automaticamente o código referente à comunicação. No entanto, o código resultante geralmente não é tão eficiente quanto códigos escritos com base em primitivas de baixo nível, como as fornecidas em PVM (GEIST et al., 1994) ou MPI (MESSAGE PASSING INTERFACE FORUM, 1995), pois aquele tipo de linguagem não oferece a mesma gama de padrões de comunicação que as linguagens baseadas em primitivas de baixo nível têm capacidade de representar (KUMAR et al., 1994).

### 2.2.4 Memória Compartilhada e Passagem de Mensagens

Estes dois paradigmas de programação paralela são os mais utilizados atualmente, e apresentam formas distintas para a interação entre os processos em execução paralela. Porém, vale ressaltar que estes paradigmas não estão necessariamente vinculados a uma ou outra arquitetura de computador paralelo. É possível executar uma aplicação baseada em passagem de mensagens em um computador com memória compartilhada, e existem linguagens que criam a abstração de uma memória compartilhada sobre um computador com memória distribuída.

Com o paradigma de memória compartilhada, como o próprio nome diz, os processos interagem uns com os outros através de um espaço de endereçamento comum a todos, onde

podem ler e escrever dados de forma assíncrona (FOSTER, 1995). Mecanismo de controle de acesso, como semáforos e bloqueios, são necessários para resolver problemas de exclusão mútua, visto que não existe a noção de que os dados pertencem a algum processo específico (KUMAR et al., 1994). Por outro lado, isto é uma vantagem à medida que facilita o processo de pragramação, já que não é necessário fazer chamadas de comunicações explícitas entre os processos como no paradigma baseado em passagem de mensagens.

A programação paralela baseada em passagem de mensagens é o modelo mais amplamente utilizado atualmente (FOSTER, 1995). Cada processo encapsula seus próprios dados, não havendo a noção de variáveis compartilhadas. A interação entre os processos dá-se enviando ou recebendo mensagens, que são chamadas explícitas às primitivas de comunicação. Apesar da maior dificuldade de programação deste paradigma, em geral, sua eficiência e portabilidade são melhores quando comparadas ao paradigma de memória compartilhada (KUMAR et al., 1994).

Independentemente do paradigma, a maioria das linguagens de programação paralela são essencialmente linguagens seqüenciais incrementadas com um conjunto especial de chamadas de sistema, fornecendo as funções necessárias para cada caso (KUMAR et al., 1994).

Nas próximas subseções, algumas dessas linguagens de programação paralela são apresentadas com mais detalhes.

#### 2.2.4.1 OpenMP

A extração de paralelismo com OpenMP se dá em nível de dados apenas, com o compartilhamento de um mesmo espaço de memória (em nível de processo) entre múltiplas tarefas (*threads*) iguais (PARHAMI, 2002).

Basicamente, o programador define qual é a região do código que deve ser executada em paralelo com a inserção de diretivas no código-fonte. Em geral, laços de repetição são bons candidatos à paralelização. Com as diretivas do OpenMP, deve-se indicar a execução paralela daquela região, bem como distinguir quais variáveis são compartilhadas e quais devem ter uma cópia para cada *thread*. Então, quando a execução da aplicação alcança aquela região do código, o fluxo de execução único é dividido com a criação de múltiplas tarefas (*threads*) que operam em paralelo.

Apesar da facilidade de programação com este tipo de linguagem, sua aplicabilidade se restringe a computadores baseados em compartilhamento físico de memória.

#### 2.2.4.2 MPI (*Message-Passing Interface*)

Quando o ambiente paralelo é distribuído, há a necessidade de troca de informações entre as tarefas distribuídas em computadores fisicamente autônomos. A solução mais utilizada é a passagem de mensagens (COULOURIS; DOLLIMORE; KINDBERG, 2005), que dispõe de um padrão universalmente consolidado: o MPI (MESSAGE PASSING INTERFACE FORUM, 1995).

A programação com MPI, que fornece primitivas de comunicação e sincronização entre processos, não é trivial. O programador deve explicitamente chamar a primitiva MPI desejada no momento adequado para extrair paralelismo com base neste modelo de programação. Isso torna a implementação mais difícil do que com o modelo de memória compartilhada, porém não há qualquer restrição quanto à arquitetura paralela do computador. Portanto, é possível extrair paralelismo com MPI praticamente em qualquer computador multiprocessado convencional.

Além disso, como a definição da interface MPI foi padronizada em meados dos anos 90, essa tecnologia já está bastante difundida e há um conjunto enorme de aplicações implementadas com MPI. Hoje em dia, há diversas bibliotecas implementadas segundo o padrão MPI, inclusive com melhorias de desempenho significativas. Dentre elas, a biblioteca MPICH (GROPP et al., 1996), que é amplamente utilizada no meio científico.

#### 2.2.4.3 PGAS (*Partitioned Global Address Space*)

As linguagens PGAS são trabalhos recentes que seguem o modelo de memória compartilhada distribuída (em inglês, *distributed shared memory* ou DSM). Este modelo cria uma abstração para o programador desenvolver a aplicação segundo o modelo de memória compartilhada, sem precisar se preocupar com a passagem explícita de mensagens em ambientes com memória distribuída. Porém, o modelo de passagem de mensangens não é evitado, embora seja transparente para o programador. Como as memórias não são fisicamente compartilhadas, a camada de abstração DSM ainda envia e recebe mensagens entre os computadores.

A sintaxe aproxima-se a de linguagens estruturadas conhecidas. Por exemplo, a linguagem UPC (*Unified Parallel C*) é semelhante à linguagem C (CHEN et al., 2003), enquanto que a linguagem Titanium é semelhante à linguagem Java (YELICK; AL, 2007). O desempenho dessas linguagens é comparável ao desempenho com MPI, mostrando-se como uma ferramenta promissora. Porém, na realidade, há diversas implementações do

modelo DSM, que ainda não conseguiram revolucionar a ampla utilização da abordagem com MPI para comunicação entre processos distribuídos.

## 2.3 Sistemas Distribuídos

A evolução das tecnologias de redes de interconexão e também de microprocessadores permitiu o que denominamos sistemas distribuídos. Segundo Tanembaum e Steen (2002), um sistema distribuído é uma coleção de computadores independentes que se parecem para seus usuários como um sistema singular e coerente.

Há dois aspectos envolvidos. Em relação ao *hardware*, os computadores são autônomos, podendo ser homogêneos ou heterogêneos em sua composição, multiprocessados ou não. Já o *software* provê a impressão de um sistema singular para o usuário.

Com relação à gerência do sistema como um todo, que se dá em nível de *software*, pode ser alcançada com um único sistema operacional distribuído ou cada computador pode ter seu próprio sistema operacional local. Como uma evolução dos sistemas operacionais baseados em serviços na rede e também do sistema operacional único e distribuído, atualmente é mais comum que os sistemas distribuídos sejam construídos sobre uma camada adicional chamada *middleware*, como apresenta a Figura 7 (TANEMBAUM; STEEN, 2002).

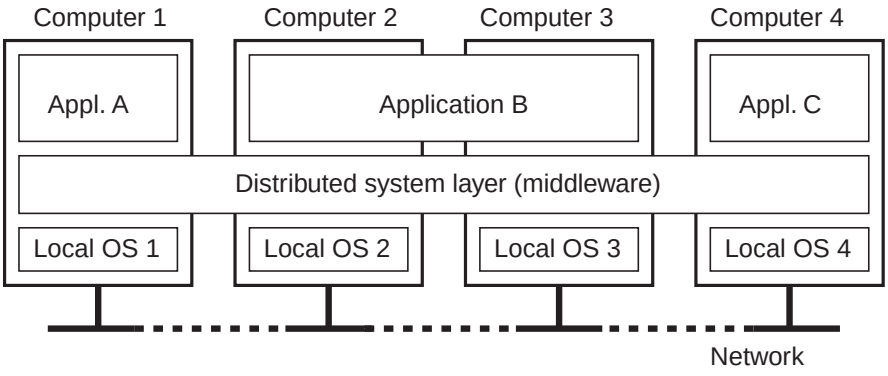



Figura 7: Sistema distribuído baseado em *middleware* (TANEMBAUM; STEEN, 2002)

Note que apenas os sistemas baseados em *middleware* realmente atendem à definição acima. Nas seções seguintes, serão apresentados dois tipos de sistemas distribuídos: os agregados de computadores (*clusters*) e as grades computacionais (*grids*).

### 2.3.1 Agregados Computacionais (*Clusters*)

Com a evolução das redes locais (LAN's) no que diz respeito à largura de banda e à escalabilidade (Ethernet comutada), a agregação de microcomputadores surge como uma alternativa economicamente mais acessível do que as MPP's (*massively parallel machines*, em inglês) para suprir as demandas por computação de alto desempenho (HENNESSY; PATTERSON, 2003). Tome como exemplo o primeiro *cluster* de PC's, construído em 1994 com 16 microcomputadores interconectados com uma rede Ethernet, que atingiu de forma sustentada um desempenho de cerca de 70 Megaflops. Àquela época, o custo de uma MPP com capacidade computacional equivalente era de 1 milhão de dólares americanos, enquanto que apenas 40.000 dólares americanos foram gastos na composição do *cluster*.

No entanto, a consolidação e a popularização dos *clusters* como uma solução de alto desempenho não aconteceu apenas devido ao custo mais baixo, mas também porque apresentam melhor disponibilidade (falhas ocorrem de maneira isolada) e facilidade de expansão do sistema (escalabilidade), quando comparados com computadores multiprocessados de memória compartilhada (HENNESSY; PATTERSON, 2003).

Por outro lado, os computadores agregados se comunicam através do barramento de E/S (ou I/O), que apresenta um desempenho pior do que o barramento de memória, responsável pela comunicação em um multiprocessador (HENNESSY; PATTERSON, 2003). Nesse sentido, dependendo da necessidade da aplicação com relação à interação entre os processos em execução, o custo da comunicação em um *cluster* pode se tornar um ponto crítico de perda de desempenho.

Enfim, nos próximos capítulos, serão abordados esses e outros aspectos sobre ambientes de *cluster*, já que são o foco deste trabalho de pesquisa e experimentação.

### 2.3.2 Grades Computacionais (*Grids*)

A computação em grade (*grid computing*, em inglês) é um paradigma de computação distribuída relativamente recente, que surgiu com os avanços da internet, da Web, e da própria tecnologia de redes de interconexão. Seu objetivo é prover o acesso remoto a recursos computacionais geograficamente distribuídos em geral, e não necessariamente com computadores de alto desempenho (KARNIADAKIS; KIRBY, 2002).

Para tanto, uma grade de computadores deve fornecer uma infra-estrutura de *hardware* e *software* para a localização eficiente dos recursos, negociação de acesso e a configuração

necessária para o acesso e uso efetivo dos recursos distribuídos. Geralmente, consiste de três partes: o cliente (que requisita algum recurso), o agente (que localiza e aloca o servidor mais adequado para o cliente, retornando para este os resultados) e o servidor dos recursos (KARNIADAKIS; KIRBY, 2002).

# 3 Computação de Alto Desempenho

Em especial, as áreas científica e de engenharia apresentam problemas com demanda por computação de alto desempenho (em inglês, *high performance computing* ou HPC), principalmente quando envolvem modelagem numérica e simulações. Alguns problemas fundamentais dessas áreas que ainda não podem ser resolvidos em um período aceitável de tempo são apontados como *grand challenges*. A solução destes problemas, de enorme impacto econômico ou científico, pode ser auxiliada pela aplicação de técnicas e recursos computacionais de alto desempenho (KUMAR et al., 1994).

Um desses problemas é a previsão meteorológica, ao qual maior atenção será dada neste trabalho. Neste caso, um modelo numérico de simulação é construído com base numa subdivisão tri-dimensional da atmosfera. Cada célula desta matriz representa, portanto, o estado daquele fragmento da atmosfera em um dado momento, com inúmeras informações pertinentes como temperatura, pressão, etc. Assim sendo, a previsão meteorológica dá-se em um processo iterativo de cálculos complexos sobre cada uma dessas células, simulando a progressão do tempo. A exatidão da previsão meteorológica depende basicamente de dois fatores: (1) o tamanho do espaço atmosférico modelado, e (2) a resolução da matriz representativa (número e tamanho das células). Grosso modo, quanto mais detalhado o espaço atmosférico, ou seja, quanto maior a resolução da matriz que representa este espaço, mais exata será a previsão meteorológica resultante e mais poder computacional será necessário. Conseqüentemente, computadores de alto desempenho são imprescindíveis para se obter o resultado esperado em um período aceitável de tempo, afinal de nada serve uma previsão meteorológica para 24 horas que demande 48 horas para ser computada.

Outros problemas caracterizados como *grand challenge* envolvem o estudo de fenômenos químicos, dinâmica dos fluidos computacional (em inglês, *computational fluid dynamics* ou CFD), aplicações para prospecção e exploração de petróleo, dentre outros.

Na maioria dos casos, o problema pode ser subdivido em múltiplas tarefas para serem

executadas paralelamente por processadores distintos, mas em um dado momento será necessário que tais tarefas compartilhem informações. Para tanto, as tarefas se comunicam entre si para a consecução de um resultado final, produto dos resultados parciais de cada uma delas. Sendo assim, as características do subsistema de comunicação, seja internamente (barramento) ou entre computadores (rede de interconexão), também são importantes, especialmente no caso de sistemas com muitos processadores disponíveis. Portanto, além de um alto poder de processamento puro, geralmente medido em FLOPS (*floating-point operations per second*) ou MIPS (*million instructions per second*), a computação de alto desempenho também engloba altas demandas por outros recursos, como comunicação entre processos, memória ou armazenamento.

Em resumo, o desempenho resultante de um sistema de computação de alto desempenho deve aproximar-se de seu potencial máximo, como um todo, ao mesmo tempo em que oferece a funcionalidade a que se destina, respeitando restrições como, por exemplo, o tempo máximo de execução da aplicação em foco. Nesse sentido, faz-se necessário dispor de métricas de desempenho, descritas a seguir, a fim de possibilitar a mensuração do desempenho em sistemas computacionais de maneira inequívoca.

Após a apresentação das métricas de desempenho em vista neste trabalho, maiores detalhes serão discutidos sobre a computação de alto desempenho em ambientes de *cluster*, que são o foco deste trabalho de pesquisa e experimentação. Nesse sentido, é apresentado um estudo de caso sobre o impacto da rede de interconexão para agregar computadores em um *cluster*. Enfim, com olhos especulativos, algumas tendências em relação com a computação de alto desempenho em ambientes de *cluster* serão discutidas.

## 3.1 Métricas de Desempenho

Como existem inúmeras formas de medir o desempenho de um sistema computacional, dependendo do objetivo de cada sistema, não há um conjunto padronizado de métricas. Portanto, é fundamental definir que as métricas de desempenho descritas a seguir têm como objetivo a mensuração do desempenho em sistemas homogêneos de computação paralela.

Métricas de desempenho servem para elucidar o comportamento de sistemas paralelos. Deve-se ter em mente que a métrica primária é tradicionalmente o tempo de execução, sendo que *speedup*, eficiência e escalabilidade também são importantes desde que resultem em um melhor tempo de execução (SAHNI; THANVANTRI, 1995).

### 3.1.1 Tempo de Execução

A primeira métrica, **tempo de execução** $T_{exec}$, refere-se ao tempo decorrido desde o início ($T_{inicio}$) até o fim da execução ($T_{fim}$) de uma determinada tarefa, seja em processamento seqüencial ou paralelo (KUMAR et al., 1994). Aqui, *tarefa* foi usada como sinônimo de carga (*workload*, em inglês), que faz restrição não só ao problema ou algoritmo em específico mas também a um tamanho fixo deste problema. O tempo de execução pode ser caracterizado, grosso modo, como uma função do tempo de computação propriamente dito da tarefa ($T_{comp}$) e do tempo de *overhead* ($T_{overhead}$), que pode ser visto como a combinação de qualquer excesso ou tempo gasto com o acesso ou utilização de algum recurso, tal como memória, comunicação, E/S (em inglês I/O) ou mesmo o eventual tempo de computação gasto indiretamente (JORDAN; ALAGHBAND, 2003). É importante perceber que o tempo de *overhead* somente fica caracterizado quando não estiver sobreposto (*overlapping*) ao tempo de computação da tarefa, isto é, quando a tarefa não continuar sua execução paralelamente. Enfim, o tempo de execução pode ser tomado como um indicativo direto do desempenho de sistemas paralelos (JORDAN; ALAGHBAND, 2003).

$$T_{exec} = T_{fim} - T_{inicio} = F(T_{comp}, T_{overhead}) \tag{3.1}$$

### 3.1.2 *Speedup* e Eficiência

Teoricamente, um problema pode ser dividido em $n$ subproblemas (doravante também chamados processos) que, portanto, podem ser resolvidos de modo concorrente com o uso de, normalmente, também $n$ processadores (QUINN, 1994). Sendo assim, em um caso ideal, o ganho de desempenho será linear pois o tempo de execução do problema em sua forma paralela, a saber $T_n$, será de $T_1/n$, onde $T_1$ é o tempo de execução do problema com o melhor algoritmo seqüencial, isto é, em um único processador. Portanto, a segunda métrica, ***speedup*** $S$, refere-se a este ganho de desempenho calculado com base na razão entre o tempo de execução do melhor algoritmo seqüencial e sua forma paralela (KUMAR et al., 1994).

$$S(n) = \frac{T_1}{T_n} \tag{3.2}$$

Ademais, não há unanimidade sobre a possibilidade de um *speedup* mais do que linear, ou seja, maior que o número de processadores utilizados. No entanto, tome como

exemplo um algoritmo de busca. O algoritmo seqüencial pode perder tempo examinando estratégias sem sucesso, uma após a outra. Já com uma versão paralela do mesmo algoritmo, várias estratégias são examinadas simultaneamente, o que pode resultar em uma solução rapidamente. Isto significa que, como regra geral, dependendo da instância do problema, é possível obter-se um *speedup* superlinear, porém estes casos são raros na prática (JORDAN; ALAGHBAND, 2003).

A partir do *speedup*, é possível derivar a terceira métrica, **eficiência** $E$, que denota quanto o sistema paralelo de computação foi explorado pelo algoritmo ou, em outras palavras, quantifica a adequação do sistema paralelo avaliado ao *workload* em questão. Do ponto de vista da aplicação, denota a qualidade da implementação paralela daquele algoritmo para aquele tamanho de problema. Eficiência é definida como a razão do *speedup* em relação ao número de processadores ou processos utilizados (KUMAR et al., 1994).

$$E(n) = \frac{S(n)}{n} = \frac{T_1}{nT_n} \tag{3.3}$$

Vale ressaltar que, neste trabalho, as métricas de *speedup* e eficiência serão expandidas para possibilitar uma análise comparativa também entre execuções paralelas. Dessa forma, será possível estabelecer o *speedup* e a eficiência relativos de um mesmo *workload* executando, por exemplo, com $n$ e com $2n$ processos.

$$S_{relativo}(n, 2n) = \frac{T_n}{T_{2n}} \tag{3.4}$$

Note também que a fórmula de eficiência $S/n$ vale para todos os tipos de *speedup* explicados nas próximas seções.

### 3.1.3 Escalabilidade

Na prática, tempo de execução $T_1/n$, speedup linear e superlinear, ou ainda, eficiência de 1 (ou 100%) raramente serão alcançados. Quando essas métricas são colocadas em função do aumento do número de processadores, ou seja, quando a capacidade computacional do sistema é escalada, tem-se a quarta métrica de desempenho levada em consideração neste trabalho. Entende-se por **escalabilidade** de um sistema paralelo sua capacidade de manter um *speedup* crescente proporcionalmente ao número de processadores utilizados, refletindo sua habilidade de efetivamente utilizar os crescentes recursos computacionais de processamento (KUMAR et al., 1994). A análise da escalabilidade de

um sistema não é trivial e deve levar em conta a variação ou não de parâmetros como, por exemplo, o tamanho do problema ou mesmo o número de processadores (QUINN, 1994).

#### 3.1.3.1 Lei de Amdahl (*fixed-size speedup*)

Um fator de influência para a análise de escalabilidade de um sistema é a existência de uma componente $C_{seq}$ necessariamente seqüencial no algoritmo, resultando em um gargalo (*bottleneck*, em inglês) que pode reduzir o desempenho (KUMAR et al., 1994). Este modelo de análise do desempenho é conhecido como **lei de Amdahl** (JORDAN; ALAGHBAND, 2003; QUINN, 1994). Segundo esta lei, o máximo *speedup* $S$ alcançável por um computador paralelo com $n$ processadores, para um tamanho fixo $w$ de *workload*, onde $0 <= C_{seq} <= 1$, é:

$$S_{max}(n) <= \frac{1}{C_{seq} + \frac{(1-C_{seq})}{n}} = \frac{n}{1 + (n-1)C_{seq}} \tag{3.5}$$

Isto significa que a componente seqüencial do algoritmo, por menor que seja em comparação à componente paralela $C_{par}$ ou $(1 - C_{seq})$, limita o speedup proporcionalmente à sua parte no tempo de execução. Portanto, o speedup não cresce linearmente com o aumento do número de processadores e a eficiência, por sua vez, diminui.

Tenha em mente que a lei de Amdahl sugere sua análise com o objetivo de reduzir o tempo de execução de um ***workload* de tamanho fixo**. Portanto, para determinar a **escalabilidade** quando o workload é fixo, deve-se analisar a relação entre tempo de execução e eficiência em função do aumento do número de processadores utilizados.

Por outro lado, a tendência é de que, com o mesmo número de processadores, o aumento do *workload* proporcione maiores *speedup* e eficiência, pois a componente seqüencial do algoritmo passa a ter menos impacto relativo sobre o tempo de execução total do problema. Isso é devido ao fato de que parte do *overhead* total é independente do aumento do *workload*. Já outra parte do *overhead* cresce com o aumento do workload mas em uma aceleração menor do que o ganho proporcionado pelo aumento do potencial de paralelismo, resultado do aumento dos subproblemas ou processos. Esse é chamado o **efeito Amdahl**, comum para uma grande classe de sistemas paralelos (KUMAR et al., 1994), que esclarece a regra geral de que o *speedup* é uma função crescente quando o *workload* é aumentado em $w$, como mostra a Figura 8 (QUINN, 1994).

Enfim, dado que um número crescente de processadores reduz a eficiência de um

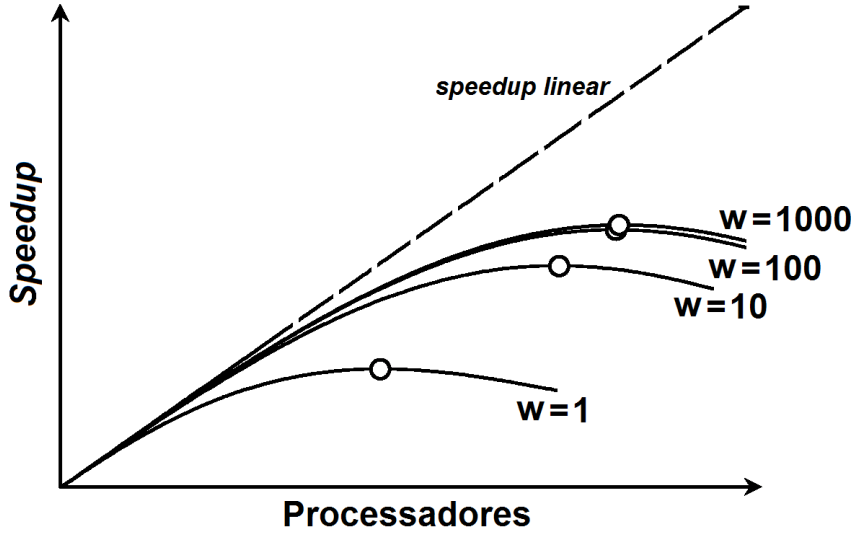



Figura 8: Gráfico do efeito Amdahl (*speedup* máximo indicado)

mesmo *workload* e que um *workload* de tamanho crescente aumenta a eficiência com um número fixo de processadores, deve ser possível **manter o nível de eficiência** com o aumento simultâneo de ambos os fatores citados. Se isso for realmente verificado, diz-se que tal sistema paralelo é **escalável** (KUMAR et al., 1994).

#### 3.1.3.2 Lei de Gustafson (*fixed-time speedup*)

O critério baseado em um nível estável de eficiência, no entanto, não se aplica a qualquer problema de computação paralela devido a necessidades específicas, por exemplo: restrições físicas como memória, ou ainda, limitações com relação ao tempo de execução. Sendo assim, a simulação geralmente deve obedecer a um tempo de execução pré-fixado e, quando este for o caso, deve-se adotar a análise de desempenho proposta pela **lei de Gustafson** (QUINN, 1994), de onde extrai-se a quinta métrica de desempenho. *Speedup* escalável (em inglês, ***scaled speedup***) é razão entre o tempo de execução de um workload de tamanho $n.w$ em um único processador e o tempo de execução do mesmo workload por $n$ processadores. Então:

$$S_{scaled}(n) = \frac{T_1(n.w)}{T_n(n.w)} = S(n) + n(1 - S(n)) \tag{3.6}$$

O gráfico resultante de $S_{scaled}(n)$ é mostrado na Figura 9.

Quando o objetivo do uso de um sistema paralelo na resolução de um problema apresenta um tempo de execução pré-determinado e fixo como alvo principal, é indicado o

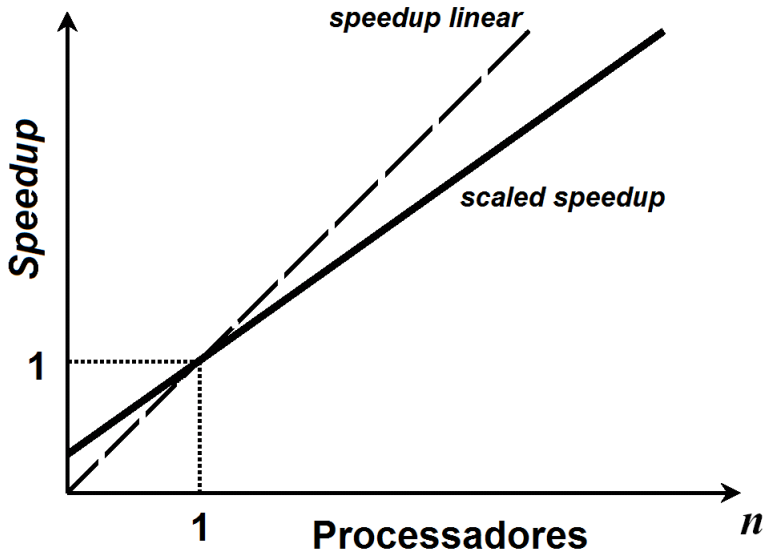



Figura 9: Gráfico representativo de *scaled speedup*

cálculo do *scaled speedup* $S_{scaled}(n)$ em detrimento do *speedup* $S(n)$ como métrica de desempenho (QUINN, 1994). Neste caso, portanto, a análise da **escalabilidade** deve levar em conta a capacidade do sistema de manter um *scaled speedup* crescente proporcionalmente ao aumento dos recursos computacionais. Lembre-se que o tamanho do workload é linearmente crescente em função do número de processadores, tendo em vista que o objetivo é conservar o **tempo de execução em um nível pré-fixado**.

## 3.2 Computação de Alto Desempenho em *Clusters*

Há cerca de quinze anos, eram utilizadas quase que exclusivamente máquinas massivamente paralelas (*massively parallel machines* ou MPP, em inglês), soluções proprietárias e de alto custo financeiro, para suprir a demanda por alto desempenho. No entanto, com o acesso facilitado a um crescente poder de processamento em computadores de menor porte, a agregação destes computadores mostrou-se como uma alternativa viável às MPP's, tanto do ponto de vista financeiro como da capacidade computacional.

Pouco mais de uma década após seu surgimento, os agregados de computadores (*clusters*, em inglês) tornaram-se muito populares na comunidade acerca da computação de alto desempenho (ou *high performance computing*, em inglês) pois podem atingir configurações massivamente paralelas de forma distribuída. Hoje em dia, os *clusters* representam a maior fatia das soluções adotadas. Vide, por exemplo, a lista dos 500 supercomputadores mais rápidos do mundo, conhecida por TOP500 (MEUER et al., 2008), cuja

atualização ocorre a cada seis meses. Em novembro de 2007, dos 500 supercomputadores, 406 ou 81,20% são classificados como *clusters*. E em junho de 2008, 400 deles são ambientes de *cluster*.

Apesar da consolidação dos *clusters* como solução para prover alto desempenho, a escolha dos computadores que o compõe está submetida à variabilidade do mercado, ou melhor, à variabilidade das configurações de computadores disponíveis no mercado ao longo dos anos. De fato, o mercado de computadores recentemente sofreu uma mudança drástica, do ponto de vista arquitetural, com o lançamento dos processadores *multi-core*, que oferecem suporte nativo a processamento paralelo. Tendo em vista o acesso a estes processadores como *commodity* (ou tecnologia de prateleira, em uma tentativa de aproximação ao português), sua inserção em ambientes de *cluster* já é fato. Também como *commodity*, as taxas de transferência da ordem de megabytes por segundo proporcionadas pelas redes de interconexão Gigabit Ethernet surgem como uma alternativa de baixo custo quando se pensa em construir um novo *cluster*.

### 3.2.1 Redes de Interconexão

O papel de uma rede de interconexão em arquiteturas MIMD com memória distribuída (isto é, a arquitetura típica de agregados de computadores) é conectar fisicamente todos núcleos de processamento, suportando comunicações diretas de envio e recebimento de dados entre os mesmos, de forma confiável (JORDAN; ALAGHBAND, 2003). Outros pontos importantes referem-se a custo financeiro, taxa de transferência, latência, concorrência e escalabilidade da rede que, de fato, agrega os computadores de um *cluster*. A arquitetura de rede implementada pela tecnologia Ethernet, principalmente Gigabit Ethernet, é amplamente utilizada na composição de clusters, assim como as redes Infiniband, mais eficientes e também economicamente menos acessíveis. A Figura 10 ilustra a porcentagem de cada família de rede na interconexão dos ambientes listados no TOP500 (MEUER et al., 2008).

Em sua configuração tradicional, as redes Ethernet tinham sua topologia classificada como um barramento (*bus*, em inglês), por meio de um equipamento chamado *hub* ou concentrador, que fornece acesso compartilhado ao meio físico como mostra a Figura 11. Em função das colisões decorrentes desse compartilhamento, tal abordagem arquitetural (baseada em *time-sharing*) não permite qualquer concorrência, restringindo sua escalabilidade a poucos computadores (JORDAN; ALAGHBAND, 2003). Por outro lado, comunicações de difusão são naturalmente eficientes, já que qualquer informação transmitida pelo meio

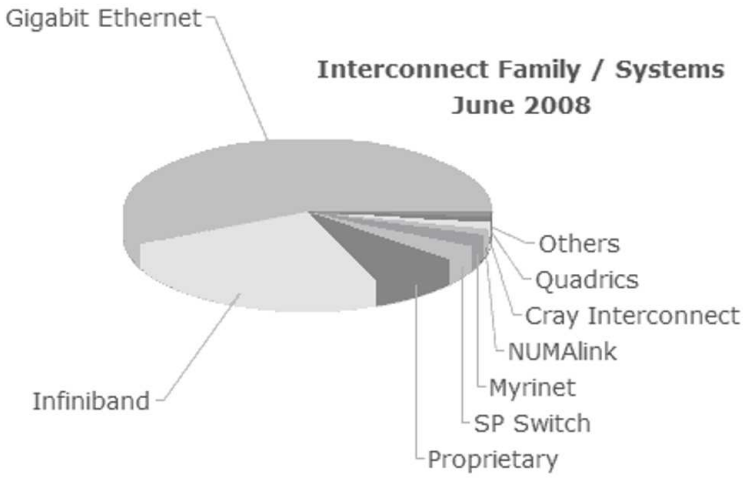



Figura 10: Família de redes de interconexão nos ambientes do TOP500

físico pode ser capturada por qualquer computador conectado à rede.

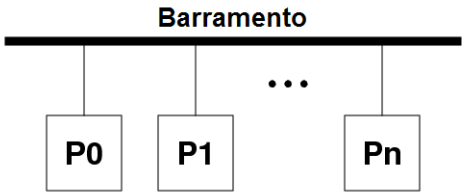



Figura 11: Rede de interconexão em barramento.

Uma alternativa consagrada às limitações citadas é a chamada Ethernet comutada (fisicamente interconectada com um *switch* nível 2), que permite certo nível de concorrência através de domínios de colisão independentes (TANEMBAUM, 1997). Isso possibilita operações de comunicação nos dois sentidos (duplex) e em paralelo, oferecendo as capacidades nominais da rede de forma escalável, geralmente até 32 computadores. O custo de implementação de uma rede Gigabit Ethernet para 8 computadores, por exemplo, pode ser tão baixo que geralmente seu custo total é menor do que o custo para adquirir um único computador do *cluster* (tomando-se como base um computador com custo de 2.000 dólares americanos).

Porém, dependendo do tamanho do sistema e também de características específicas à aplicação, a implementação de *clusters* com base na tecnologia Ethernet pode não ser adequada, degradando substancialmente o desempenho por se tornar o gargalo do sistema. Nestes casos, quando há a necessidade de se interconectar centenas ou milhares de computadores para a composição de um *cluster*, a utilização de redes de interconexão mais eficientes são imprescindíveis para o uso efetivo dos recursos computacionais agregados.

Essa categoria de redes de interconexão para *clusters*, tendo como exemplo Myrinet (BODEN et al., 1995) e Infiniband (CASSIDAY, 2000), apresentam uma topologia *crossbar*, em que todos os pontos estão completamente conectados uns com os outros, como mostra a Figura 12. Além disso, dispõem de um processador especializado e dedicado ao processo de comunicação, localizado na própria placa de rede, e fazem acesso direto à memória principal (DMA) sem a necessidade de interromper o sistema operacional. Essa categoria de redes de interconexão também é conhecida por arquitetura *network-on-chip*. No intuito de padronizar essa abordagem, surgiu o modelo VIA (Virtual Interface Architecture) como uma interface padrão para redes de alto desempenho (DUNNING et al., 1998).

Enfim, todas essas vantagens traduzem-se em maior desempenho quando comparado à tecnologia Ethernet, que também podem atingir taxas de transferência da ordem de gigabits/segundo. O preço por maior desempenho também se reflete no custo de implementação dessas redes, que podem representar a metade do custo total do ambiente de *cluster*, dependendo do número e complexidade dos nós.

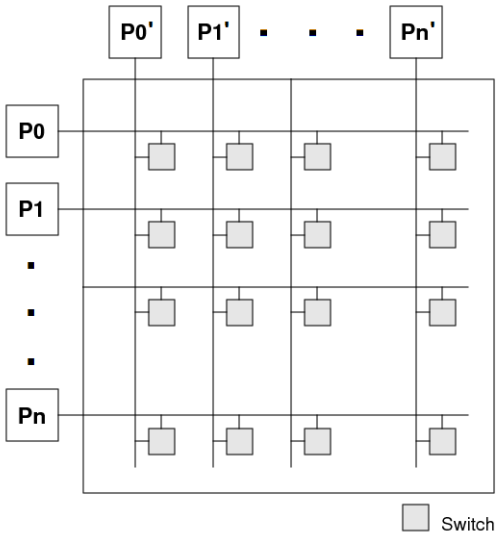



Figura 12: Rede de interconexão *crossbar*.

Portanto, tendo em mente as vantagens e desvantagens de cada arquitetura, a escolha da tecnologia de rede de interconexão mais adequada para a composição de um agregado de computadores é dependente diretamente dos objetivos do sistema em construção (se, por exemplo, existe a previsão de ampliação do número de computadores agregados) e, principalmente, dos requisitos da própria aplicação em foco (por exemplo, a necessidade de se executar a aplicação dentro de um tempo pré-definido).

Com o objetivo de investigar o impacto da rede de interconexão sobre o desempenho de aplicações científicas, foi desenvolvido o estudo de caso a seguir, com base em um mesmo

ambiente de *cluster* interconectado com duas redes de interconexão distintas, Ethernet e Myrinet.

### 3.2.2 Estudo de Caso: Myrinet vs. Ethernet

Foi realizado um estudo de caso para apresentar o impacto da rede de interconexão sobre o desempenho de agregados computacionais para aplicações científicas. Para tanto, não basta investigar apenas o comportamento da rede de interconexão ou da biblioteca de comunicação entre processos, pois as aplicações podem ser de naturezas diferentes e definir necessidades diversas. Portanto, o comportamento de algoritmos com graus de paralelismo distintos, isto é, de granularidades fina e grossa, foram observados em um mesmo agregado interconectado com duas redes distintas: a Ethernet e a Myrinet.

Os experimentos foram realizados em um agregado de cinco computadores: quatro máquinas equipadas com um processador Intel Pentium III 600 MHz e 256 MB de RAM, exercendo a função de escravas; e uma máquina mestre mais potente, equipada com um processador AMD Athlon XP 1.8GHz e 512 MB de RAM. Todas as máquinas operam com sistema Linux Ubuntu 6.06, baseado no kernel versão 2.6.17-11. Apenas processos básicos do sistema estão operantes, ou seja, as máquinas estão ociosas à espera dos experimentos. Com estas medidas, evitam-se questões relativas ao balanceamento de carga no sistema.

Este agregado computacional está interconectado por dois meios físicos distintos, isto é, por duas redes de interconexão: Fast Ethernet (placas 100BASE-T) e Myrinet (placas M3S-PCI64B). Em ambos os casos, o agregado dispõe de um segmento dedicado de rede, o que significa que está isolado de ruído proveniente de outros serviços de rede.

A biblioteca MPICH foi escolhida para configurar o sistema porque é uma biblioteca consagrada e preocupada com o fator desempenho. Entretanto, em função de uma incompatibilidade de versão da rede Myrinet à disposição com a implementação MPICH2 foi necessário adotar a implementação MPICH1.

Na implementação MPICH1 para Ethernet, a comunicação MPI se dá por meio de um canal TCP, que é um protocolo orientado à conexão residente no sistema operacional. Ou seja, quando o processo faz um pedido de comunicação para a biblioteca MPICH1, o pedido é repassado ao sistema operacional (GROPP et al., 1996). Assim, o sistema operacional é chamado à execução sempre que for necessário enviar ou receber pacotes de rede, concorrendo com os processos da aplicação distribuída pela utilização do processador principal.

Diferentemente, a biblioteca MPICH-GM para Myrinet favorece a configuração nativamente ponto-a-ponto da rede de interconexão Myrinet, de topologia *mesh* (BODEN et al., 1995), pois há uma conexão física dedicada de cada ponto com todos os outros pertencentes àquele segmento de rede, não existindo problemas de colisão. Outra diferença dá-se em nível de protocolos de comunicação, pois quando a biblioteca MPICH-GM recebe um pedido de comunicação de um processo, o que acontece é uma chamada direta à biblioteca de comunicação GM. Ela fornece a interface de acesso à rede Myrinet e implementa um serviço de rede não-orientado à conexão e externo ao sistema operacional. Portanto, o fluxo de comunicação é desviado do sistema operacional e a transmissão dos dados é feita via DMA para a placa de rede, enquanto que na rede Ethernet é necessário envolver o sistema operacional e também algum processador principal no processo de comunicação. Nas redes Myrinet, isto não é necessário pois o *switch* e as placas de rede Myrinet são equipadas com processador e memória especializados e dedicados, tornando-as capazes de encarregar-se do processamento de pacotes de rede. Na literatura, tal abordagem é conhecida como Virtual Interface Architecture (VIA) (DUNNING et al., 1998) e possibilita ganhos diferenciados no desempenho. A Figura 13 ilustra essas diferenças.

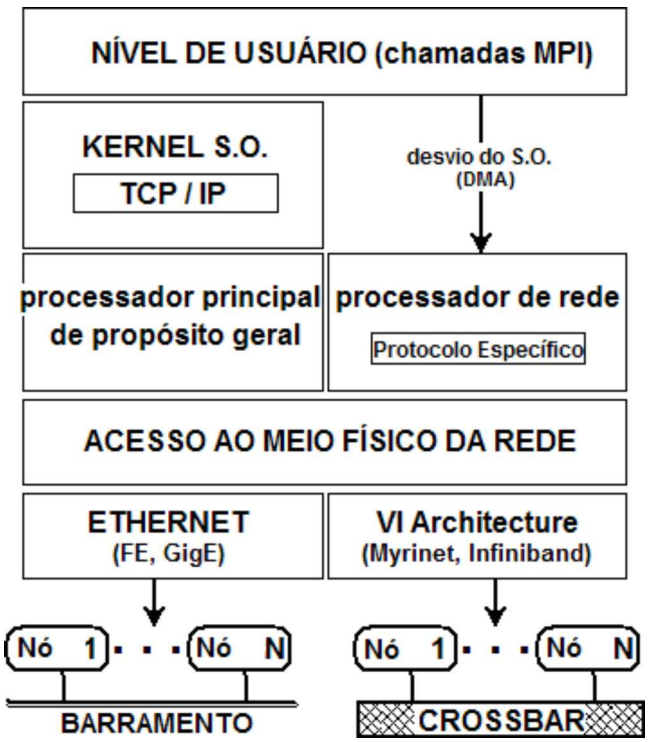



Figura 13: Fluxo dos modelos Ethernet e VIA.

Para melhor caracterizar as diferenças de capacidade das redes Fast Ethernet e Myrinet, a latência e a taxa de transferência (*round-trip time*) foram medidas com base nas

bibliotecas MPICH para cada ambiente de interconexão. O tamanho das mensagens utilizadas para estas medições corresponde ao tamanho, em bytes, de uma matriz N x N formada de elementos do tipo de dados inteiro (4 bytes).

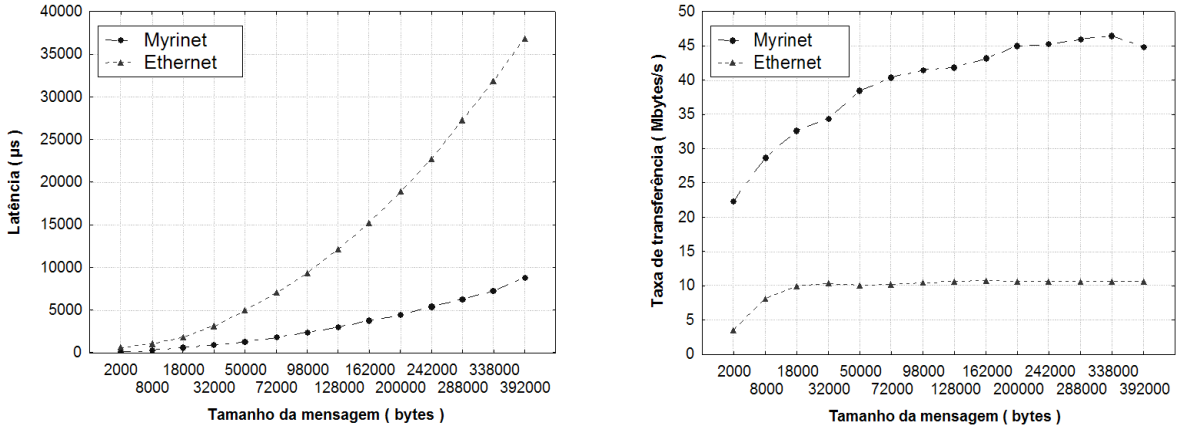



Figura 14: Latência e taxa de transferência das redes Myrinet e Ethernet

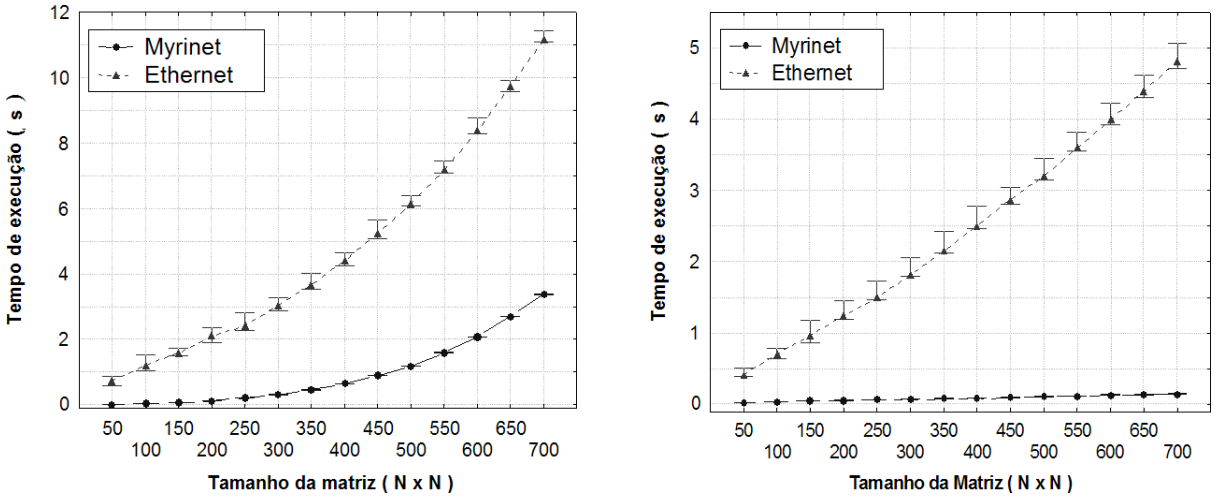
A Figura 14 ilustra o desempenho de cada rede de interconexão, mostrando que o agregado computacional vale-se de um ambiente de interconexão mais eficiente, Myrinet, e de outro mais limitado, Ethernet. Como era esperado, a rede Myrinet apresentou um desempenho diferenciado em função da presença de processador e memória dedicados em cada placa.

O ambiente de experimentação deste estudo de caso se constitui também em nível de aplicação. Foram adotados dois algoritmos largamente utilizados em aplicações científicas: a multiplicação de matrizes (MM), de granularidade grossa em função do grau nulo de dependência de dados e da alta demanda de processamento; e a transposição de matrizes (TM), de granularidade fina em função do alto grau de dependência de dados, uma vez que caracteriza-se pela redistribuição de elementos da matriz. Optou-se por implementações consagradas no meio acadêmico, publicadas por Pacheco (1996).

O tempo de execução foi coletado no nó mestre, ao final da execução de cada experimento. É importante ressaltar que a entrada de dados é uma matriz N x N gerada aleatoriamente, variando de 50 x 50 até 700 x 700. A Figura 15 apresenta o tempo médio, mínimo e máximo gasto na execução de cada algoritmo com o mesmo *cluster*, ora utilizando a rede de interconexão Ethernet, ora a Myrinet.

Segundo a Figura 15, a rede Myrinet obteve um desempenho melhor para ambos os algoritmos. Porém, o algoritmo TM (Figura 15(b)) apresentou um desempenho muito superior com a rede Myrinet por ser de granularidade fina. Em outras palavras, a dependência de dados deste algoritmo demanda uma rede de interconexão diferenciada.

Por outro lado, o algoritmo MM (Figura 15(a)) é de granularidade grossa e o ganho de desempenho em comparação à rede Ethernet é menor.




(a) Multiplicação de Matrizes - MM (b) Transposição de Matrizes - TM

Figura 15: Tempo de execução dos algoritmos MM e TM com as redes Myrinet e Ethernet

As Figuras 16 e 17 ilustram o *speedup* como métrica, considerando o ganho de desempenho da rede Myrinet em relação à rede Ethernet para ambos os algoritmos em função do crescimento do conjunto de dados operados.

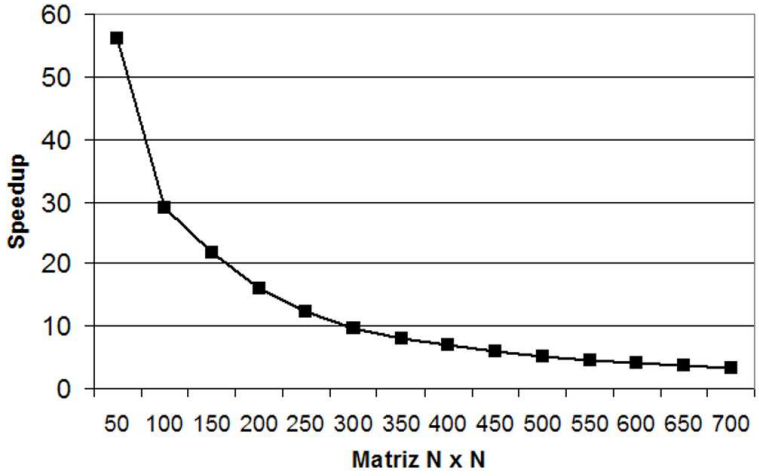



Figura 16: Speedup do algoritmo MM (Myrinet X Ethernet)

Como ilustra a Figura 16, o *speedup* diminui com o aumento do tamanho da matriz, ou seja, o ganho em função da rede de interconexão tende à equivalência. Isto se deve à granularidade grossa do algoritmo MM. Como não há dependência de dados, a necessidade de comunicação é muito pequena em comparação à necessidade por computação. Nesse caso, é esperado que o impacto de uma rede de interconexão mais eficiente seja menor com o crescimento dos dados de entrada.

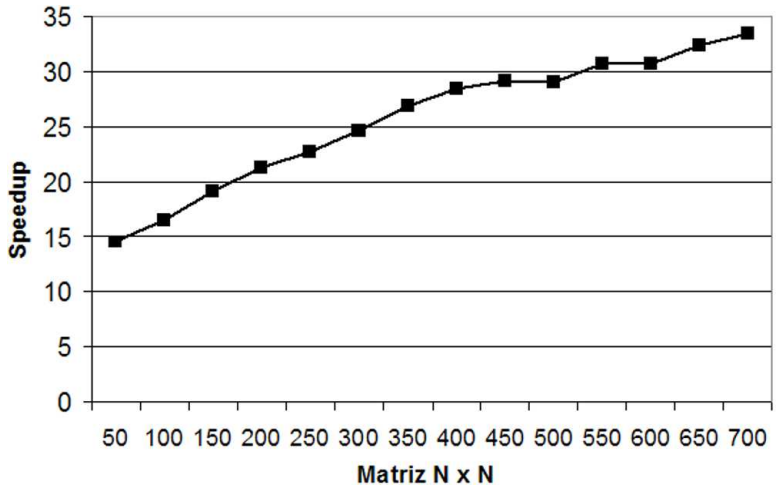



Figura 17: Speedup do algoritmo TM (Myrinet X Ethernet)

Por outro lado, no caso do algoritmo TM, vê-se o efeito contrário, conforme ilustra a Figura 17. O ganho em desempenho do algoritmo TM, de granularidade fina, mostra-se em crescimento com o aumento da matriz de entrada, ampliando as diferenças de desempenho das redes Ethernet e Myrinet. A dependência de dados existente neste algoritmo provoca maior demanda de comunicação entre processos. Neste caso, como os resultados indicam, uma rede de interconexão mais eficiente como a Myrinet é mais adequada para a agregação dos recursos computacionais de um *cluster*.

A partir dos resultados empíricos conclui-se que o desempenho da rede de interconexão tem um impacto reduzido sobre o desempenho de aplicações de granularidade grossa. Por outro lado, a rede de interconexão tem um papel muito importante no desempenho de aplicações de granularidade fina, pois uma grande dependência de dados é satisfeita de forma mais adequada com uma rede eficiente como a Myrinet.

Enfim, os experimentos confirmam que a relação entre rede de interconexão e granularidade de aplicações paralelas deve ser levada em consideração para compor eficientemente ambientes de *cluster*.

### 3.2.3 Tendência: Processadores *Many-core* ou Nanotecnologia

Nas últimas décadas, foi possível atingir consideráveis ganhos em desempenho computacional, sem gerar custos adicionais, com o aumento do número de transistores em processadores seqüenciais, como é observado pela Lei de Moore (KUMAR et al., 1994; QUINN, 1994). Não obstante, as limitações físicas da matéria sempre foram um fator limitante dos ganhos decorrentes desta abordagem, criando dificuldades econômicas e tecnológicas.

Hoje em dia, apesar do alto nível de maturidade tecnológica, o crescente consumo de energia em conseqüência do aumento do número de transistores tem dificultado a viabilidade econômica de tal abordagem. Além disso, essa maior concentração de energia pode prejudicar a confiabilidade do sistema, em função do perigo de interferência entre os componentes, e também, o conseqüente aumento da temperatura do processador gera mais custos para seu resfriamento (CHAPARRO et al., 2007).

Enfim, devido a limitações físicas e econômicas, a produção de microprocessadores com desempenho crescente não está mais apta a seguir as determinações da Lei de Moore, revelando um esgotamento da tecnologia de processamento seqüencial. A solução adotada pela indústria para driblar essas dificuldades foi a mudança para o paradigma de computação paralela, através de processadores multiprocessados, ou seja, com múlitplos núcleos de processamento.

A Figura 18 apresenta arquiteturas paralelas de microprocessadores Intel®. Como uma espécie de evolução, desde uma solução multiprocessada baseada em dois processadores seqüenciais distintos em uma mesma máquina até a configuração *multi-core* com dois núcleos de processamento em um mesmo processador, a tecnologia intermediária apresenta apenas um núcleo de processamento, porém com duas unidades controladoras independentes, possibilitando um ganho de desempenho não pela capacidade maior de processamento puro, mas por melhor utilizar o recurso através de dois fluxos de instruções (INTEL®, 2006).

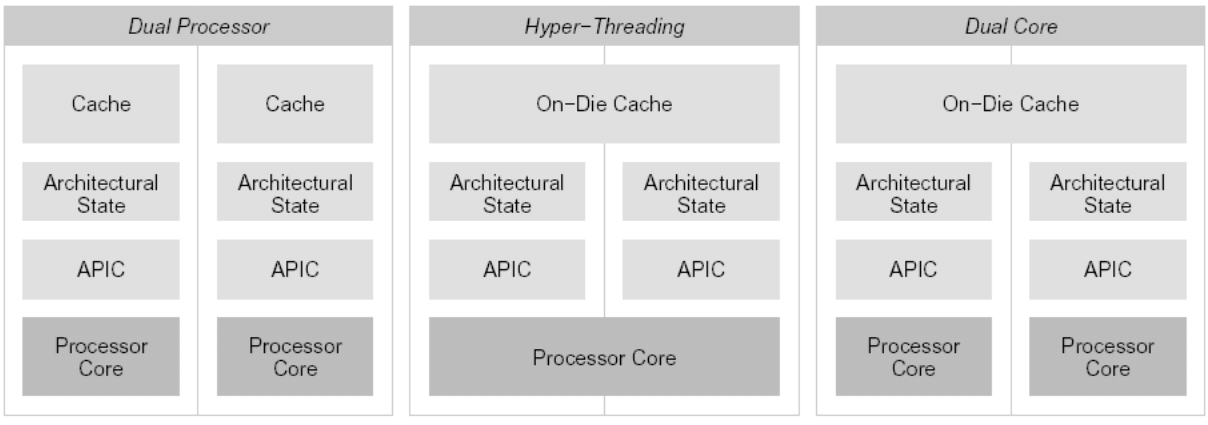



Figura 18: Evolução dos microprocessadores Intel® até sua edição *multi-core*

Com efeito, a complexidade dos processadores *multi-core* é maior, pois neste contexto há, por exemplo, necessidade de comunicação e sincronização destes núcleos. Sendo assim, a análise de desempenho destas arquiteturas também torna-se mais complexa e merece observação mais cuidadosa.

Com um crescente número de núcleos ou *cores* em um mesmo processador, a importância de sistemas de comunicação como as arquiteturas *network-on-chip* aumenta (FLICH et al., 2008), não para a interconexão entre computadores, e sim para a interconexão dos próprios núcleos de processamento dentro do mesmo processador e internamente ao computador. Portanto, é possível que a alta densidade de núcleos em processadores *multi-core* venha a se tornar um fator redutor do desempenho, com maior impacto do que a redução provocada pelas implicações da natureza paralela distribuída de ambientes de *cluster*, que requerem comunicação através de redes de interconexão, supostamente mais lentas do que a comunicação interna entre os núcleos de processamento.

Por outro lado, a presença de numerosos núcleos de processamento em um mesmo computador permite que alguns desses núcleos sejam alocados exclusivamente ao processamento decorrente da comunicação via Ethernet, por exemplo. Além disso, as redes Gigabit Ethernet já são capazes de oferecer altas taxas de transferência, embora a latência de acesso à rede deixe a desejar. Apesar da eficiência comprovada das redes de interconexão *network-on-chip*, seu custo financeiro é muito mais alto do que redes como a Ethernet.

A ampla aceitação e a popularização dos processadores *multi-core* em diversos segmentos do mercado resultará em uma queda de preço, naturalmente. Nesse contexto, se considerarmos o preço por núcleo de processamento, e não por processador, essa queda será ainda mais significativa. Com o passar do tempo, é intuitivo pensar que múltiplos e numerosos núcleos de processamento estarão disponíveis em um mesmo computador a um preço comparável ao custo dos melhores processadores seqüenciais de hoje em dia.

Como conseqüência, poderá haver uma competição com as redes de interconexão com arquitetura *network-on-chip*, que transferem grande parte do processamento relativo à comunicação para os chamados processadores de rede, localizados na própria placa de rede (*Network Interface Card* ou NIC, em inglês) (BODEN et al., 1995; CASSIDAY, 2000).

Outro paradigma que está em evolução, podendo se tornar uma alternativa viável para a computação de alto desempenho em um futuro próximo, é a computação em nanoescala, em nível atômico (BRéCHIGNAC, 2008).

A abordagem *bottom-up*, também chamada nanotecnologia, considera por exemplo a própria natureza da molécula, de pequena escala, em avanços na eletrônica molecular para permitir a produção de circuitos na escala de nanômetros (CHEN, 2004). Mas, a realidade é que esta abordagem está longe de se tornar acessível no mercado.

Já a abordagem *top-down*, chamada de nanoCMOS, é complexa pois não é mais válida

a noção de um único tipo ideal de transistor usado em todo o *chip*, como nos microprocessadores comuns (SINNOTT; MILLAR; ASENOV, 2008). Porém, com um processador nanoCMOS, é possível por exemplo que alguns dos problemas discutidos acima tornem-se ultrapassados, já que o poder computacional desta abordagem é promissor (MAEX, 2004).

Hoje em dia, técnicas da nanotecnologia já são aplicadas na tecnologia nanoCMOS, trazendo à prática uma abordagem híbrida em sistemas integrados heterogêneos. Estas abordagens devem se aproximar ainda mais na tentativa de resolver problemas de variabilidade e predição e de tolerância a falhas, intrínsecos a este tipo de tecnologia (MAEX, 2004), para avançar com segurança na produção de componentes comercializáveis.

# 4 Estudos de Casos com Aplicações Científicas de Alto Desempenho

A principal aplicação científica estudada nos casos apresentados a seguir é o WRF (*Weather Research and Forecasting Model*), um modelo numérico para previsão meteorológica do tempo em mesoescala que vem se tornando cada vez mais importante entre a comunidade desta área de atuação, tanto em ambientes operacionais como em ambientes científicos de pesquisa atmosférica.

O modelo WRF é uma aplicação *grand challenge*, cuja demanda é pela redução do seu tempo de execução para que, por exemplo, seja possível aumentar a resolução aplicada ao conjunto de dados ou mesmo a região escolhida para a previsão do tempo. Sendo assim, após realizar um estudo empírico sobre determinado ambiente de *cluster*, o resultado deverá traduzir-se em um menor tempo de execução do modelo.

Seu desenvolvimento é um esforço conjunto de um consórcio de agências governamentais, em sua maioria estadunidenses, dentre elas: o NCAR (*National Center for Atmospheric Research*), o NOAA (*National Oceanic and Atmospheric Administration's*), o NCEP (*National Centers for Environmental Prediction*) e o FSL (*Forecast System Laboratory*); a AFWA (*Department of Defense's Air Force Weather Agency*) e o NRL (*Naval Research Laboratory*); o CAPS (*Center for Analysis and Prediction of Storms*) da Universidade de Oklahoma e a FAA (*Federal Aviation Administration*) (NATIONAL CENTER FOR ATMOSPHERIC RESEARCH, 2007). Também estão envolvidos neste projeto diversos cientistas da comunidade científica mundial, o que dinamiza e intensifica as atividades em prol de uma aplicação que ofereça os últimos avanços em pesquisas da área.

Neste capítulo, o modelo WRF será apresentado em maiores detalhes, de modo a especificar as características de funcionamento do modelo e também com relação ao escopo da previsão do tempo realizada pela EPAGRI S.A. (Empresa de Pesquisa Agropecuária e Ex-

tensão Rural de Santa Catarina). Esta é uma empresa pública, responsável pela previsão meteorológica do tempo no estado de Santa Catarina, que utiliza os resultados gerados pelo modelo WRF para auxiliar a equipe de meteorologistas neste processo. Enfim, serão apresentados dois estudos de casos realizados nos ambientes de *cluster* da EPAGRI S.A. (doravante denominada empresa).

Inicialmente, a empresa dispunha de um único ambiente, utilizado na execução dos experimentos do primeiro caso do estudo empírico desenvolvido, cujo objetivo era propor uma alternativa para reduzir o tempo de execução do modelo numérico WRF sem a adição de novos componentes de *hardware*. Naquele momento, também foi possível realizar alguns experimentos com duas máquinas mais recentes, equipadas com processadores *dual-core*. Esta primeira parte do estudo, motivada pela necessidade de um maior desempenho em um ambiente de produção operacional, pode ser vista como um exercício de fixação do conhecimento teórico obtido sobre ambientes de *cluster* por meio de um estudo empírico.

Posteriormente, com a necessidade por uma previsão de tempo com maior resolução, o que demanda um poder computacional maior, a empresa adquiriu um novo ambiente de *cluster*. Em retribuição à postura colaborativa para com nossa pesquisa, desenvolvemos a segunda parte do estudo empírico apresentado neste trabalho, com o objetivo de otimizar o desempenho do *cluster* recém implantado, antes de colocá-lo em produção com a última versão disponível do modelo WRF.

A principal contribuição desta dissertação está no Caso 2. Foi realizada uma pesquisa aprofundada sobre a própria aplicação WRF e também uma análise detalhada sobre os problemas encontrados durante o processo de otimização no sentido de buscar alternativas viáveis e eficazes para reduzir o tempo de execução da aplicação. Com isso, pretende-se conhecer as melhores práticas (técnicas e mecanismos) utilizadas em processos de otimização do desempenho de agregados de computadores *multi-core*, com o foco voltado à modelagem de sistemas de alto desempenho em ambientes de produção.

# *Weather Research and Forecasting Model* (WRF)

Modelos numéricos como o WRF são muito importantes para auxiliar os profissionais de meteorologia no processo de previsão meteorológica do tempo. Porém, a existência de diversos modelos e a conseqüente falta de integração em torno de um modelo utilizado universalmente, por assim dizer, era um freio para o desenvolvimento científico e tecnológico desta área de atuação. A fim de avançar no entendimento meteorólogico assim

como acelerar a implementação operacional de resultados alcançados em pesquisas, um consórcio de agências governamentais, principalmente estadunidenses, uniu forças para o desenvolvimento da próxima geração de modelos de previsão do tempo em mesoescala, lançando, em 1999, o WRF.

A aplicação WRF consiste de um conjunto de componentes, como: o próprio modelo numérico WRF, pré-processadores para os dados de entrada seja para previsões com dados reais (observados) ou idealizados, pós-processadores para análise e visualização, e um programa para assimilação de dados variáveis tri-dimensionais (3DVAR). A Figura 19 apresenta um esquema ilustrativo da aplicação WRF como um todo (NATIONAL CENTER FOR ATMOSPHERIC RESEARCH, 2007).

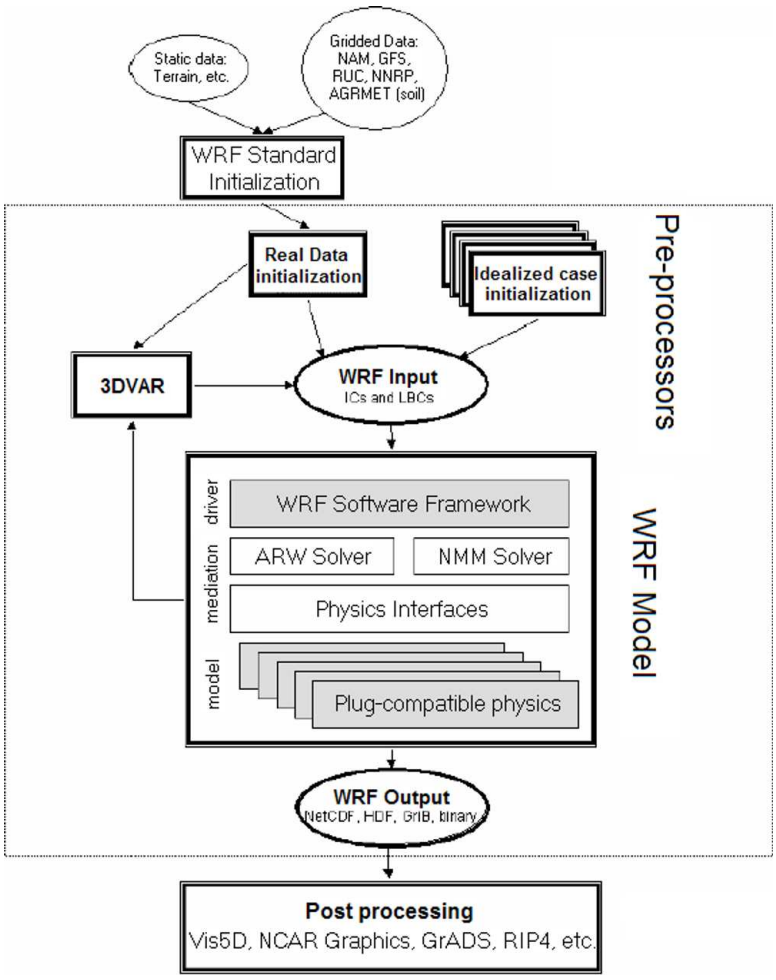



Figura 19: Esquema ilustrativo da aplicação WRF (MICHALAKES et al., 2004)

O desenvolvimento da aplicação WRF foi modelado segundo alguns objetivos como (NATIONAL CENTER FOR ATMOSPHERIC RESEARCH, 2007):

- **flexibilidade:** o WSF (*WRF Software Framework*) permite que módulos de física

sejam facilmente conectados através de uma interface padrão, de modo que novos componentes possam ser desenvolvidos e testados pela comunidade;

- **portabilidade:** todo o código implementado no modelo WRF deve ser passível de execução nas principais plataformas computacionais existentes;
- **eficiência:** o código é desenvolvido de modo que seja eficiente em ambientes de computação massivamente paralelos;
- **código centralizado:** o código-fonte é mantido centralizado, evitando problemas de compatibilidade. Isto significa que um novo componente de física deve ser aprovado antes de ser incorporado ao modelo.

Todos esses cuidados em nível de *software* permitem que o WRF seja capaz de se tornar, de fato, o estado da arte em simulação da previsão meteorológica do tempo, o que também é objetivo do projeto. Existem muitas opções de configuração do modelo com relação à física utilizada nas simulações, compartilhando a experiência da ampla comunidade envolvida, tanto para fins operacionais como de pesquisa.

Deve ficar esclarecido que o presente estudo leva em consideração tão somente o modelo WRF, não contabilizando as operações de pré ou pós-processamento. Sendo assim, reduzir o tempo de execução do modelo numérico torna-se ainda mais importante quando se tem em mente que existem outras etapas no processo, a fim de efetivamente auxiliar os meteorologistas na previsão do tempo.

Além disso, todos os experimentos são realizados com base na mesma configuração do modelo WRF (relativa aos componentes de física) e mesmo conjunto de dados de entrada (do dia 29 de maio de 2008) utilizados no ambiente de produção da empresa na previsão meteorológica.

O conjunto de dados de entrada do modelo WRF é uma matriz tridimensional que representa a atmosfera de uma determinada região, desde metros até milhares de quilômetros, com diversas informações como, por exemplo, a topografia da região em foco e também dados de observatórios para alimentar a simulação com uma condição inicial. Em todos os experimentos, o domínio utilizado é de 100 por 126, com uma resolução de 15 km, o que representa uma área territorial de 12.600 quilômetros quadrados. Na vertical, o domínio é de 37, totalizando 466.200 elementos na matriz.

As Figuras 20 e 21 apresentam a extensão do conjunto de dados de entrada utilizados nos experimentos. A região abrange completamente os estados de Santa Catarina, Rio

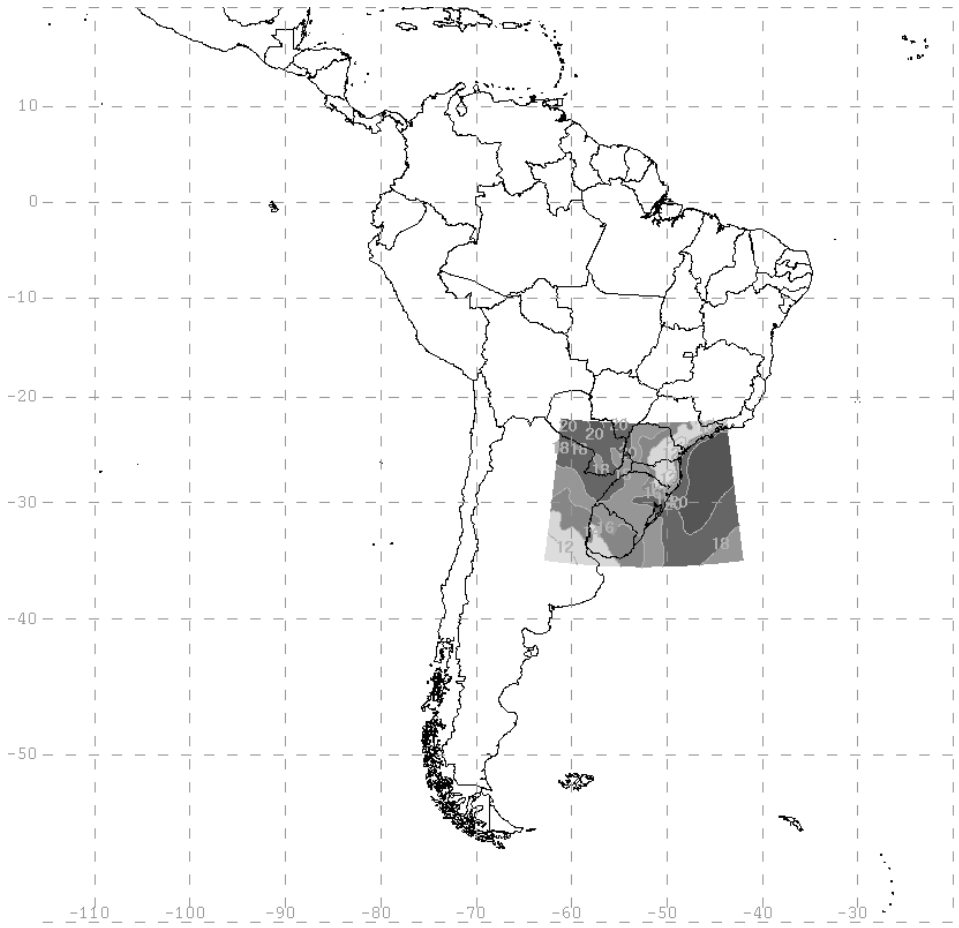



Figura 20: Região utilizada nos experimentos de previsão meteorológica

Grande do Sul e Paraná, e o Uruguai; e em parte, os estados de São Paulo, Rio de Janeiro, Minas Gerais e Mato Grosso do Sul, e ainda, parte do Paraguai e da Argentina.

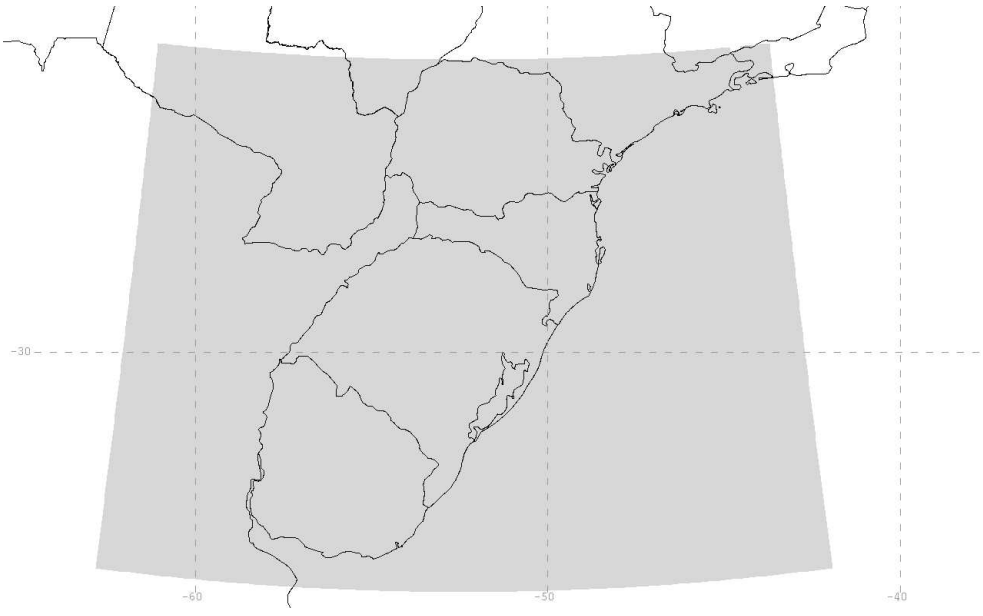



Figura 21: *Zoom* da região utilizada nos experimentos de previsão meteorológica

Na execução paralela do modelo WRF, cada processo recebe uma sub-matriz de tamanho aproximadamente igual, que diminui com o aumento do número de processos. A principal atividade que demanda comunicação entre os processos é a redistribuição dos dados laterais, nos quatro limites lógicos de cada sub-matriz, ocorrendo a cada iteração com mensagens entre 10 e 100 kilobytes (KERBYSON; BARKER; DAVIS, 2007).

Cada iteração avança o tempo de simulação em 75 segundos, totalizando 96, 576 e 3456 iterações nas previsões para 2, 12 e 72 horas, respectivamente. Além disso, a cada 12 iterações (ou 15 minutos de simulação) ocorre uma iteração de radiação física, que se soma ao tempo de processamento da iteração ordinária, e a cada 145 (ou 3 horas de simulação) ocorrem picos por causa da geração do arquivo de saída.

É utilizado o *solver* ARW (*Advanced Research WRF*), originalmente chamado "em", de massa Euleriana (vide Figura 19). Ademais, o arquivo de configuração (*namelist.input*) utilizado nos experimentos pode ser visto no Anexo A.

## 4.1 Caso 1

Neste primeiro caso do estudo empírico, o objetivo é propor uma alternativa para reduzir o tempo de execução do modelo numérico WRF sem a adição de novos componentes de *hardware* ao ambiente de *cluster* em produção com a versão 2.2 do modelo.

Primeiramente, investigou-se empiricamente alternativas para alcançar o objetivo, inclusive face a duas máquinas mais recentes, equipadas com processadores *dual-core*. A metodologia de experimentação busca validar os resultados da alternativa proposta através de algoritmos de *benchmarking* com caracterísitcas distintas em relação à granularidade, que é a relação entre tempo gasto com computação e comunicação. Foram, então, realizados experimentos com o modelo WRF no ambiente de *cluster* configurado segundo a proposta e também, a título de comparação, em um computador com dois processadores *quad-core*. Enfim, a proposta mostrou-se pertinente na busca pela redução do tempo de execução do modelo WRF, sem adicionar novos componentes ao ambiente de *cluster*.

Esta primeira parte do estudo, motivada pela necessidade de um maior desempenho em um ambiente de produção operacional, pode ser vista como um exercício de fixação do conhecimento teórico obtido sobre ambientes de *cluster* por meio de um estudo empírico.

### 4.1.1 Experimentação Investigativa (Caso 1A)

Foram avaliadas três configurações distintas de um mesmo *cluster* no que diz respeito à distribuição dos processos e o número de núcleos de processamento utilizados por computador agregado. Além disso, foi possível realizar experimentos em computadores mais recentes, equipados com processadores *dual-core* e até 8 núcleos de processamento, ampliando a análise em nível de ambientes paralelos e de processadores.

De antemão, alguns termos devem ser definidos. Um ***core*** ou **núcleo** é a unidade atômica de processamento dos sistemas. Um soquete contém um ou mais *cores*. Um **nó** é uma máquina independente com um ou mais soquetes, que compartilham recursos como a memória principal e o acesso à rede de interconexão. Um ***cluster***, **sistema** ou **ambiente** é um conjunto de nós interconectados. No entanto, note que o sistema N1xP8 não é um *cluster*, e sim um nó multiprocessado.

A nomenclatura dos sistemas apresentados na Figura 22 seguem um padrão conforme o número de nós (N) e o número de núcleos de processamento ativos por nó (P). Por exemplo, o sistema N8xP1 é composto por 8 computadores (N = 8), cada um com 2 núcleos de processamento, porém apenas 1 núcleo é alocado para a aplicação (P = 1). O segundo núcleo de cada computador está, portanto, ocioso em relação à aplicação.

| SISTEMA (nós x processos) / INFO | N8xP1 | N4xP2 | N2xP4 | N1xP8 |
|---|---|---|---|---|
| Interconexão | Gigabit Ethernet (GigE) | | GigE | HyperTransport |
| | 3Com Switch 3812 | | back-to-back | Crossbar |
| MTU | 1500 | | 1500 | N/A |
| *Modelo do processador* | 32-bit Intel Xeon (DP) | | 64-bit AMD Opteron 880 | |
| *Velocidade do processador* | 2.66GHz | | 2.4Ghz | |
| *Tecnologia Manuf.* | 130nm | | 90nm | |
| *# Cores por soquete* | 1 | | 2 | |
| *# Cores por nó* | 2 | 2 | 4* | 8 |
| *# nós* | 8 | 4 | 2 | 1 |
| *# Total de cores (Ativo/Ocioso)* | *8/8* | *8/0* | *8/4* | *8/0* |
| | POR CORE | | | |
| *Cache L1 (I/D)* | 12KB/8KB | | 64KB/64KB | |
| *Cache L2* | 512KB | | 1MB | |
| *Cache L3* | - | | - | |
| *DRAM* | 1GB | 512MB | 1GB | |
| *Velocidade DRAM* | 533MHz | | 800Mhz | |
| *BogoMIPS* | ~ 5300 | | ~ 4800 | |

* O sist.N2xP4 é composto de dois nós: um com 4 cores em 2 soquetes e outro com 8 cores em 4 soquetes

Figura 22: Ambientes de experimentação do Caso 1A

A fim de entender devidamente as configurações descritas na Figura 22, note que todos os sistemas apresentam no máximo 8 núcleos chamados ativos, ou seja, haverão no máximo 8 núcleos dedicados ao processamento da aplicação. Por outro lado, os núcleos de processamento chamados ociosos não participam do processamento relativo à aplicação, ou seja, estão ociosos em relação à aplicação apenas. Isso não significa que estarão ociosos

de fato, pois podem estar executando processos do sistema operacional, por exemplo. Os sistemas N8xP1 e N2xP4 apresentam *cores* ociosos, enquanto os sistemas N4xP2 e N1xP8 não. Note que o sistema N8xP1 apresenta um *core* ocioso em cada nó, enquanto no sistema N4xP2 um mesmo nó possui 4 *cores* ociosos em relação ao processamento da aplicação.

Os sistemas baseados em processadores Xeon (INTEL®, 2004) rodam Linux kernel 2.6.8.1 e os sistemas baseados em processadores Opteron (AMD, 2007) rodam Linux kernel 2.6.22.8. Todos os sistemas operacionais estão com suporte a SMP ativado. Além disso, estão dedicados aos experimentos, isto é, não estão operando quaisquer outros serviços, exceto a configuração mínima necessária à execução dos próprios experimentos.

A implementação da biblioteca MPI-2 (MESSAGE PASSING INTERFACE FORUM, 1997) utilizada foi a MPICH2 (GROPP et al., 1996), versão 1.0.6p1, com base no mecanismo de comunicação por soquete (–with-device=ch3:sock) para todos os sistemas.

#### 4.1.1.1 Benchmark de rede: *b_eff*

Com o objetivo de caracterizar os subsistemas de comunicação entre processos de cada um dos quatro sistemas descritos acima, foi executado o benchmark *b_eff* (RABENSEIFNER; KONIGES, 2001) para coletar as duas métricas de desempenho de redes de interconexão (descritas anteriormente) em função do tamanho das mensagens. Com este intuito, a versão do *b_eff*, parte do *HPC Challenge Benchmark* (LUSZCZEK et al., 2006), foi adaptada pois originalmente os dados de latência e taxa de transferência são coletados apenas para mensagens de 8 e de 2000000 bytes.

Dessa maneira, foi possível coletar a latência e a taxa de transferência de mensagens entre 2 bytes até 16 megabytes para processos se comunicando. Foram efetuados experimentos com 2 processos e também com 8 processos, pois não é raro todos processos se comunicarem simultaneamente.

Os experimentos com 2 processos se comunicando levam em consideração duas formas de comunicação. Na comunicação chamada *one-way*, um processo envia mensagens enquanto o segundo apenas as recebe. Na comunicação do tipo *two-way*, os dois processos enviam e recebem mensagens simultaneamente. No caso com 8 processos, todos enviam e recebem mensagens simultaneamente.

Os resultados são apresentados segundo o tamanho das mensagens. Foram consideradas pequenas, mensagens entre 2 e 512 bytes. De tamanho médio, mensagens entre 1 e 64 kilobytes. E grandes, as mensagens entre 128 kilobytes e 16 megabytes.

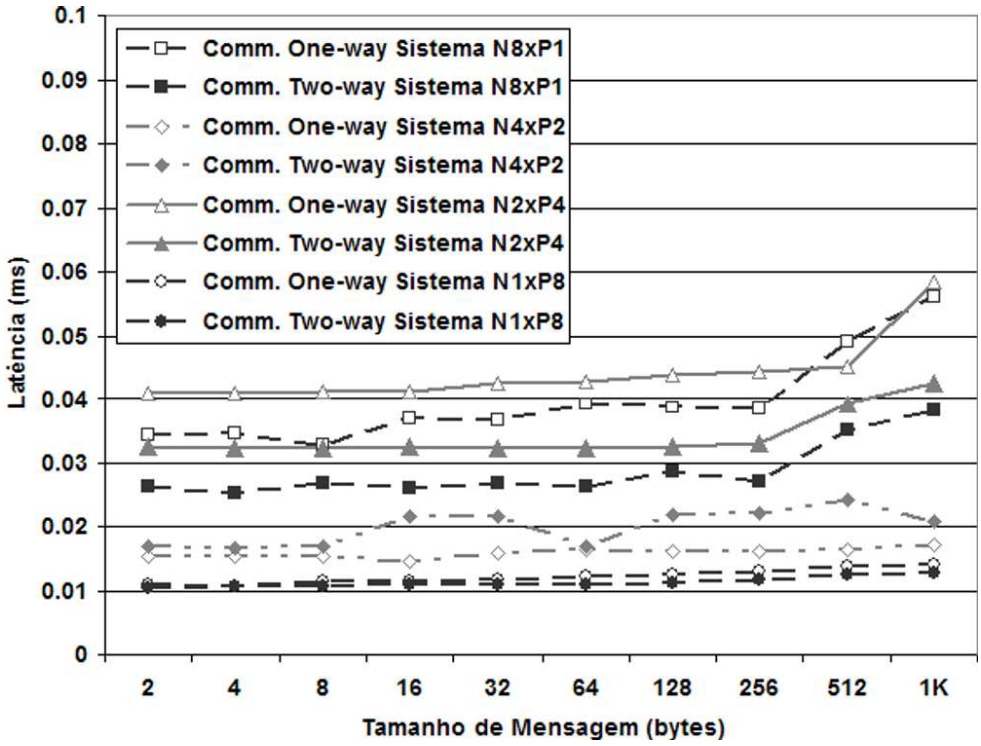



Figura 23: Latência entre 2 processos do Caso 1A (msg. pequenas)

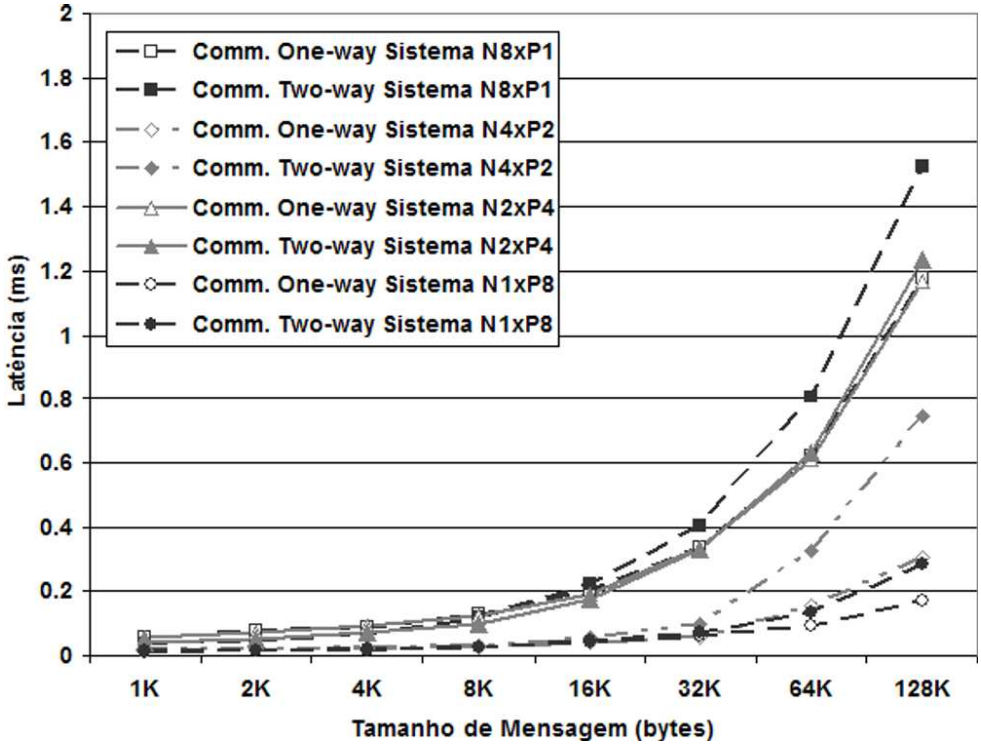



Figura 24: Latência entre 2 processos do Caso 1A (msg. médias)

Nas Figuras 23, 24 e 25, os resultados mostram que os sistemas N8xP1 e N2xP4 têm comportamentos similares, apresentando maior latência na comunicação de mensagens de qualquer tamanho, tanto nos modos de comunicação *one-way* como *two-way*, em com-

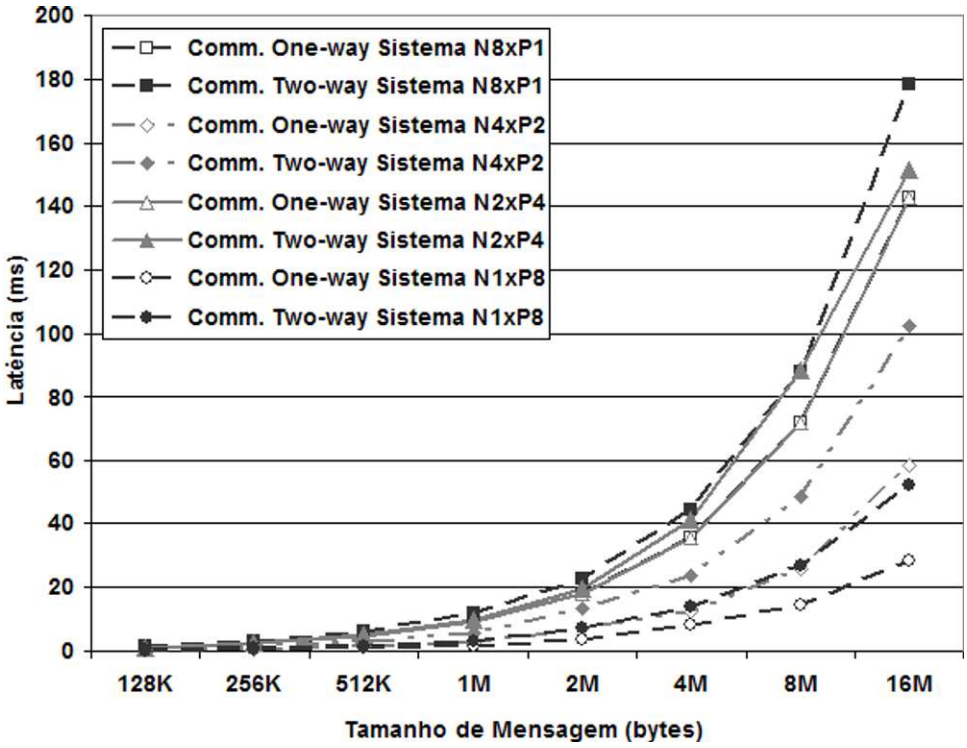



Figura 25: Latência entre 2 processos do Caso 1A (msg. grandes)

paração aos sistemas N4xP2 e N1xP8. Isso se explica pois, como há apenas dois processos, a comunicação nos sistemas N4xP2 e N1xP8 acontece internamente aos nós, sem acesso à interface de rede, enquanto os outros dois sistemas se comunicam via rede Ethernet. O sistema N1xP8 apresentou o melhor desempenho de latência dentre todos os sistemas.

Com base nas Figuras 26 e 27, podemos perceber que a taxa de transferência para comunicação *one-way* ou *two-way* dos sistemas N4xP2 e N1xP8 são superiores a dos sistemas N8xP1 e N2xP4 para qualquer tamanho de mensagem. Além disso, o comportamento da taxa de transferência da comunicação *two-way* do sistema N1xP8 e da comunicação *one-way* do sistema N4xP2 são semelhantes. No sistema N4xP2, o padrão de crescimento da taxa de transferência para comunicação *two-way* e *one-way* são similares para mensagens pequenas e médias, mas para mensagens maiores que 32KB, seu comportamento assume um padrão constante e seu desempenho começa a diminuir.

Nos sistemas N8xP1 e N2xP4, o padrão da taxa de transferência tanto para comunicação *one-way* como para *two-way* são muito parecidos. Contudo, a taxa de transferência é maior para comunicação *two-way* do que para comunicação *one-way* para mensagens de até 8KB no sistema N2xP4, e até 64KB no sistema N8xP1. Isso se deve, em parte, ao algoritmo usado pelo *b_eff*, que calcula a taxa de transferência para comunicação *one-way* com base na latência máxima capturada, enquanto a taxa de transferência para

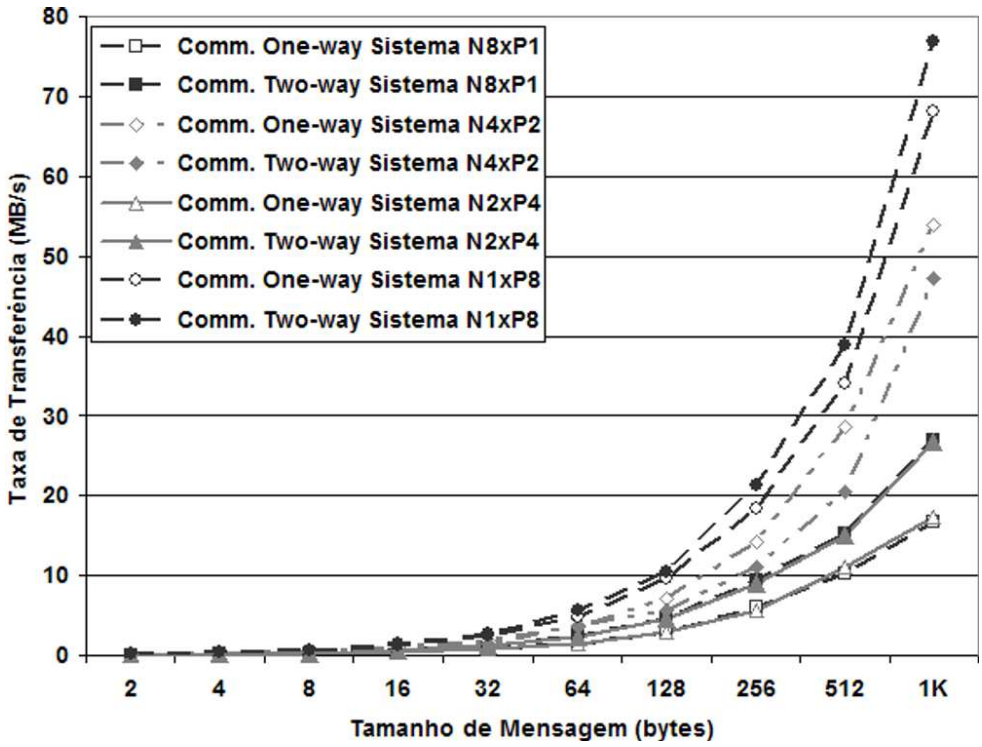



Figura 26: Taxa de transf. entre 2 processos do Caso 1A (msg. pequenas)

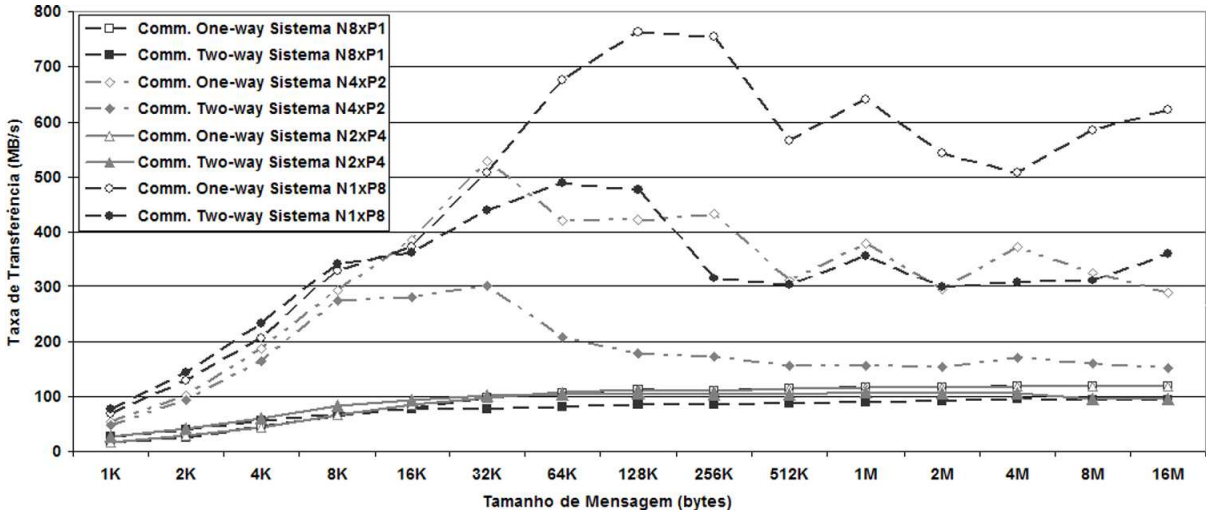



Figura 27: Taxa de transf. entre 2 processos do Caso 1A (msg. médias e grandes)

comunicação *two-way* é calculada com base na latência média. De qualquer forma, para mensagens grandes, a taxa de transferência para comunicação *two-way* representa até 80% da taxa de transferência da comunicação *one-way*. Enfim, os resultados da comunicação entre apenas 2 processos indicam que os núcleos ociosos dos sistemas N8xP1 e N2xP4 não produzem ganhos consideráveis para o desempenho da comunicação, nem *one-way* nem *two-way*.

As Figuras 28 e 29 apresentam os resultados de latência e taxa de transferência de 8

processos se comunicando simultaneamente. Com base nestes gráficos, é possível ter uma visão mais ampla das capacidades de comunicação dos sistemas do que com apenas dois processos se comunicando.

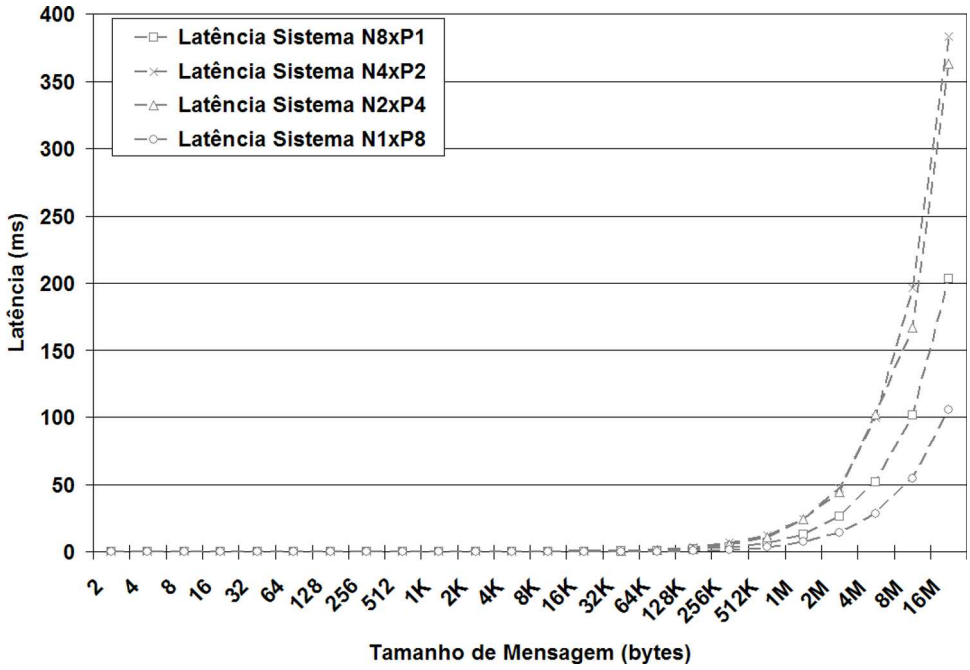



Figura 28: Latência entre 8 processos (Caso 1A)

Com bases nos gráficos, é possível perceber que o pior desempenho de comunicação neste caso é do sistema N4xP2, que não apresenta núcleos de processamento ociosos. Um agravante é que os processos competem localmente pelo acesso à memória e à rede devido à arquitetura de interconexão com topologia em barra dos processadores Intel®. Diferentemente, o sistema N1xP8 apresenta o melhor desempenho de comunicação entre processos, embora também não disponha de núcleos ociosos. Porém, tenha em mente que neste sistema não existe a necessidade de acesso à rede de interconexão por ser um único computador com 8 núcleos de processamento. De qualquer forma, há uma queda brusca nas taxas de transferência para mensagens grandes (maiores que 64KB) ao passo que a latência cresce consideravelmente, mas continua sendo a menor dentre os sistemas em estudo.

Os sistemas N8xP1 e N2xP4 estão configurados com núcleos ociosos em relação à aplicação (no caso o *b_eff*). A impressão inicial é de que seu desempenho deve, de alguma forma, se aproximar. No entanto, os gráficos da taxa de transferência do sistema N2xP4 apresentam um comportamento descrecente e seu desempenho é bem inferior ao do sistema N8xP1, que por sua vez apresenta um gráfico crescente da taxa de transferência. O sistema N8xP1 apresenta o segundo melhor desempenho dentre os quatro sistemas, e o melhor

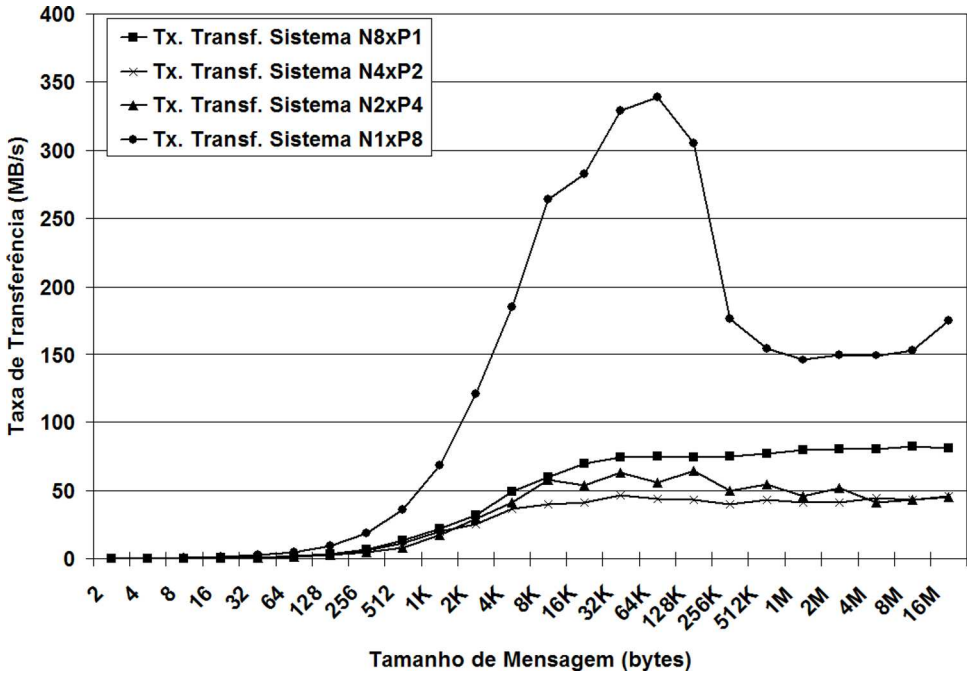



Figura 29: Taxa de transferência entre 8 processos (Caso 1A)

desempenho dentre os sistemas interconectados por redes Gigabit Ethernet.

Essa enorme diferença entre os sistemas que possuem núcleos de processamento ociosos é devido à forma como estão distribuídos estes núcleos. No sistema N2xP4, apenas uma máquina possui núcleos ociosos e, portanto, o desempenho acaba sendo prejudicado pela máquina que está com todos seus núcleos ocupados com a aplicação. Por outro lado, no sistema N8xP1, cada máquina possui um núcleo ocioso, o que resulta em um menor impacto do *overhead* de comunicação sobre o desempenho.

#### 4.1.1.2 NAS Parallel Benchmarks (NPB)

O NAS Parallel Benchmarks (NPB) é um conjunto de algoritmos, derivados de aplicações reais de dinâmica dos fluidos computacional (CFD) utilizadas pela NASA, que foram desenvolvidos especialmente para a avaliação de desempenho de computadores paralelos (BAILEY et al., 1991). O NPB é composto por 8 algoritmos, 5 deles são *kernel benchmarks* (EP, FT, IS, CG e MG) e os outros 3 são *benchmarks* de aplicações simuladas (SP, BT e LU). Este trabalho abrange somente os cinco *kernels*, pois apresentam granularidades bem distintas. O conceito de granularidade está associado à razão do tempo gasto com computação em relação ao tempo gasto com comunicação entre os processos distribuídos.

Os algoritmos foram compilados de forma idêntica em todos os sistemas, com diretivas de otimização *-O3*. Optamos pela Classe B para todos os algoritmos, pois não foi possível

rodar alguns deles para a Classe C (vide Tabela 3).

Tabela 3: Classes do NAS Parallel Benchmarks 2.3 (tamanho do problema)

| **Benchmark** | **Classe A** | **Classe B** | **Classe C** |
|---|---|---|---|
| **EP** | $2^{28}$ | $2^{30}$ | $2^{32}$ |
| **FT** | $256^2$x128 | 512x$256^2$ | $512^3$ |
| **CG** | 14000 | 75000 | 150000 |
| **MG** | $256^3$ | $256^3$ | $512^3$ |
| **IS** | $2^{23}$ | $2^{25}$ | $2^{27}$ |

Os algoritmos do NPB e seus resultados serão apresentandos segundo uma ordem descendente de granularidade dos algoritmos. Como anteriormente explicado, granularidade refere-se à razão de computação em relação à comunicação realizada pelo algoritmo.

Com base na Figura 30, percebe-se que, para o algoritmo EP, o tempo de execução alcançado com os sistemas N2xP4 e N1xP8 supera os tempos dos sistemas N8xP1 e N4xP2. Note que os sistemas que obtiveram o menor tempo de execução operam com processadores AMD Opteron(TM), mais recentes e, além disso, sua freqüência de acesso à memória é maior do que os processadores Intel® Xeon®.

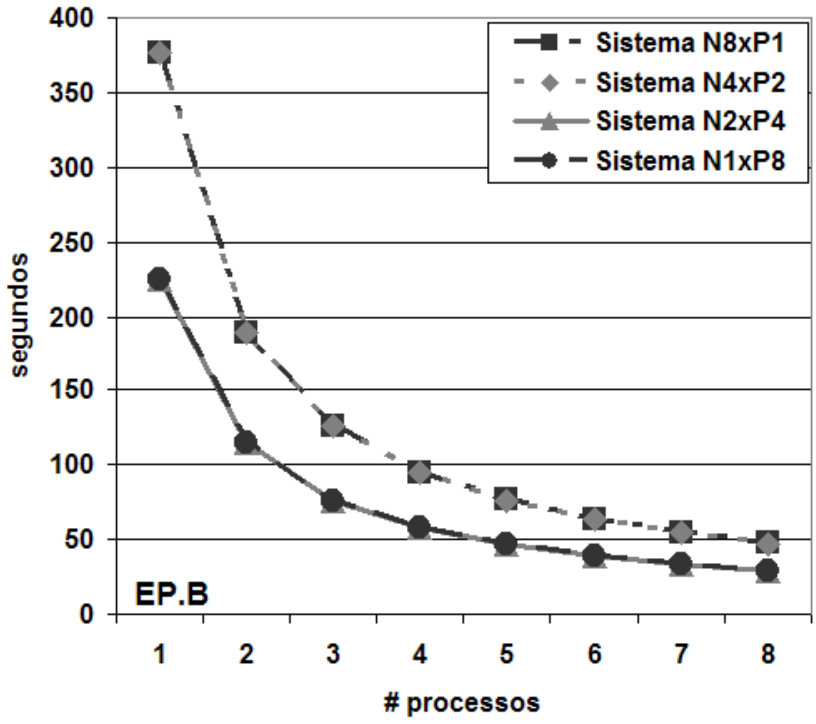



Figura 30: Tempo de execução: NAS EP classe B

O algoritmo EP (*Embarrassingly Parallel*) é de granularidade grossa, pois sua ne-

cessidade de comunicação é quase desprezível, limitando-se à distribuição de dados em um momento inicial e ao reagrupamento do resultado parcial de cada processo ao final de sua execução. Em função dessa necessidade mínima de comunicação demandada pelo algoritmo EP ao passo que o tempo de sua execução é gasto quase que exclusivamente com computação, tomamos estes resultados como base para definirmos o desempenho por núcleo (ou *per-core*) dos sistemas. Basicamente, assumiremos que o desempenho por núcleo dos sistemas N2xP4 e N1xP8 é maior do que dos sistemas N8xP1 e N4xP2.

O algoritmo FT (*FFT 3D PDE*) resolve equações diferenciais parciais utilizando-se de FFT's 1D em série. Apresenta necessidade tanto por computação sobre ponto-flutuante assim como por comunicação, embora através de mensagens grandes, da ordem de megabytes, em uma freqüência baixa. Isso ocorre porque os autores do NPB tomaram o cuidado de agregar as mensagens em nível de aplicação para minimizar o custo de comunicação. O resultado é um algoritmo de granularidade média com comunicações de todos para todos perfeitamente balanceadas, o que significa que todo processo envia e recebe a mesma quantidade de dados que os outros. Além disso, apesar da demanda por taxa de transferência crescer proporcionalmente ao desempenho *per-core* do sistema, não há um aumento significativo das necessidades de comunicação em função do crescimento do número de processadores usados.

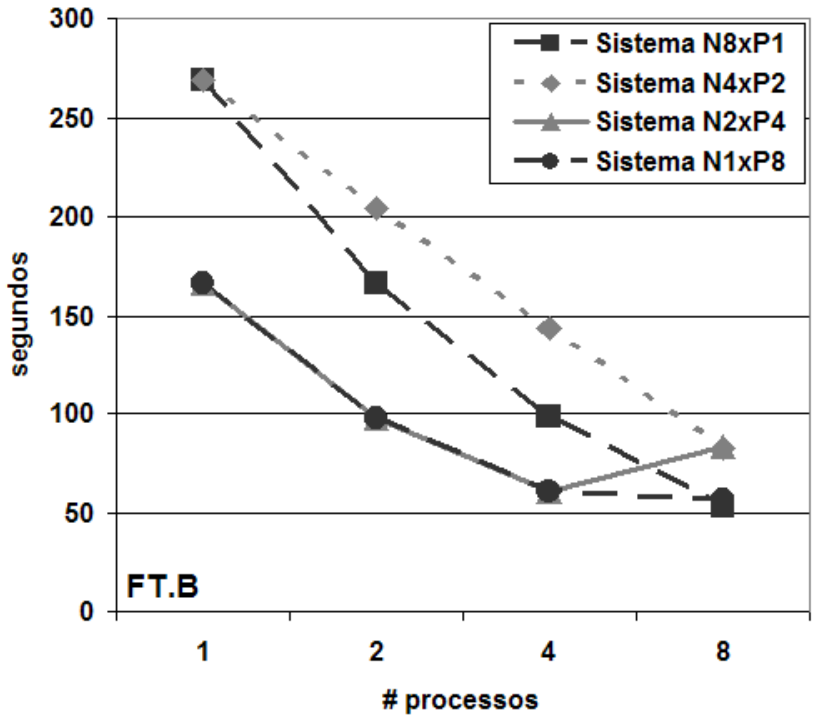



Figura 31: Tempo de execução: NAS FT classe B

Como se pode ver na Figura 31, os sistemas N8xP1 e N4xP2 apresentam uma melhor escalabilidade do que os sistemas N2xP4 e N1xP8, pois os ganhos são mais acentuados com o aumento do número de processos. Interessantemente, o tempo de execução do algoritmo FT no sistema N8xP1, com um núcleo ocioso por máquina, acaba sendo praticamente o mesmo do sistema N1xP8, que é na realidade uma única máquina com oito núcleos de processamento, e não um *cluster* interconectado via Gigabit Ethernet. Dado que o desempenho per-core do sistema N1xP8 é maior assim como seu desempenho de comunicação, como foi apresentado pelos resultados coletados com o *b_eff*.

Já o algoritmo CG (*Conjugate Gradient*) é um método de gradiente conjugado que consiste de computações sobre ponto-flutuante e testa freqüentes comunicações do tipo ponto-a-ponto de longa distância e organizadas irregularmente. Embora o algoritmo CG apresente necessidade de computação considerável, é caracterizado como de granularidade fina em função da grande quantidade de mensagens trocadas. O tamanho médio das mensagens é o menor dentre os cincos algoritmos estudados, com uma predominância de mensagens pequenas e o restante das mensagens tendendo a grandes.

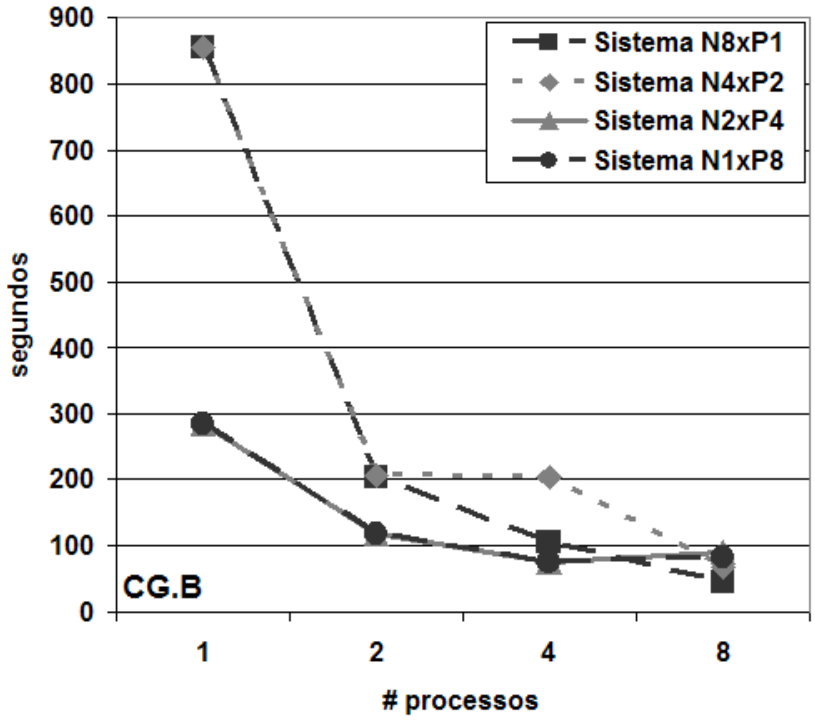



Figura 32: Tempo de execução: NAS CG classe B

De acordo com a Figura 32, percebe-se uma nítida aproximação entre os resultados dos quatro sistemas, salvo a execução com um processo apenas. Quando o algoritmo CG é executado com 4 processos, há uma grande diferença no tempo de execução dos sistemas

N8xP1 e N4xP2, ambos com configurações semelhantes, exceto pela presença de núcleos ociosos em relação à aplicação no sistema N8xP1. Evidencia-se neste caso uma enorme vantagem decorrente da presença de um núcleo de processamento ocioso em cada nó do *cluster*. Para 8 processos, é possível perceber uma pequena vantagem do sistema N8xP1 sobre os outros sistemas.

Ademais, a Figura 32 mostra ainda que, com o aumento do número de processos, o tempo de execução do algoritmo CG em todos os sistemas se aproxima, neutralizando as diferenças relativas à capacidade de processamento puro ao mesmo tempo em que põe em cheque a capacidade do sistema de comunicação entre processos.

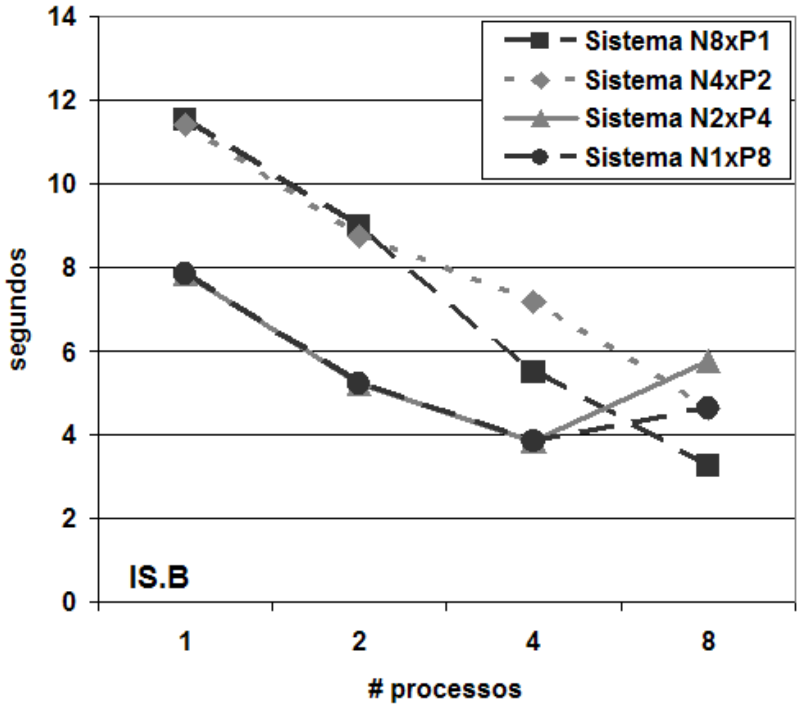



Figura 33: Tempo de execução: NAS IS classe B

O algoritmo IS (*Integer Sort*) é o único que não opera sobre dados do tipo ponto-flutuante, pois que é um ordenador de inteiros. Predominam comunicações de redução e de todos para todos não-balanceadas. O padrão de comunicação é dependente do conjunto de dados. Ademais, o tamanho médio das mensagens é mediano, apresentando muitas mensagens pequenas e de até poucos kilobytes, e também mensagens grandes, da ordem de megabytes, enquanto outros tamanhos de mensagem são raros. Sendo assim, a granularidade do algoritmo IS é fina, embora o volume de dados comunicado para a classe B seja menor do que do algoritmo CG, pois freqüentemente é necessária a comunicação entre os processos para que prossigam a computação do algoritmo.

Conforme a Figura 33, os sistemas N8xP1 e N4xP2 apresentam tempo de execução quase que linearmente descrescente com o aumento do número de processos, refletindo melhor escalabilidade do que os sistemas N2xP4 e N1xP8. Note que o gráfico do tempo de execução nos sistemas N2xP4 e N1xP8 apresentam um crescimento mais acentuado em comparação com os algoritmos FT e CG, quando o número de processos é aumentado de 4 para 8. Esta perda de desempenho constatada com o algoritmo IS é atribuída ao *overhead* causado pela alta freqüência de mensagens trocadas entre os processos em comunicações coletivas, ou seja, entre todos os processos simultaneamente.

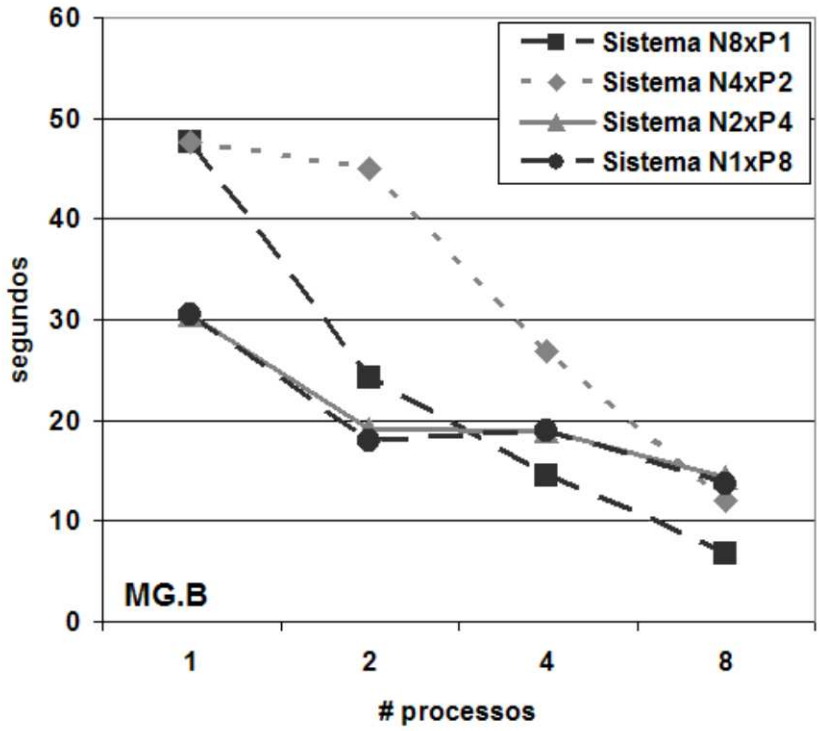



Figura 34: Tempo de execução: NAS MG classe B

O algoritmo MG (*Multigrid*) busca aproximar uma solução para problemas de Poisson. Assim como o algoritmo IS, apresenta granularidade fina com fases curtas de computação intercaladas por comunicações, mas as semelhanças reduzem-se a isso. O algoritmo MG caracteriza-se por testar comunicações tanto de longa como de curta distância com uma freqüência alta e predominância de comunicações do tipo ponto-a-ponto entre processos vizinhos. Além disso, a freqüência em que as comunicações ocorrem apresenta uma distribuição quase uniforme. O tamanho médio das mensagens é mediano pois as mensagens utilizadas variam segundo um padrão uniforme, ou seja, um número considerável de mensagens de uma variedade de tamanhos, de pequenas a grandes, são trocadas entre os processos.

Mais uma vez, os resultados dos sistemas N8xP1 e N4xP2 apresentam ganhos consideráveis com o aumento do número de processos, revelando boa escalabilidade, enquanto nos sistemas N2xP4 e N1xP8 os resultados ficam próximos uns dos outros. Note que, estranhamente, o tempo de execução é quase idêntico para 2 e 4 processos, e não diminui nem 30% em relação ao resultado coletado com 8 processos.

A Tabela 4 apresenta resultados específicos ao sistema N8xP1 em comparação à configuração original do *cluster*. O intuito é quantificar em termos de execução as vantagens da presença de um núcleo de processamento ocioso em cada nó do *cluster* em comparação à alocação de todos os núcleos de processamento disponíveis para a execução da aplicação.

Tabela 4: Variação no tempo de execução do NPB no Sistema N8xP1 (Caso 1A)

| **Benchmark** | **8 processos** | **16 processos** | **Variação** |
|---|---|---|---|
| **EP** | 47,696 s | 25,860 s | - 45,78% |
| **FT** | 53,232 s | 57,492 s | 8,00% |
| **CG** | 45,268 s | 49,988 s | 10,43% |
| **IS** | 3,234 s | 4,066 s | 25,73% |
| **MG** | 6,798 s | 7,724 s | 13,62% |

Segundo a Tabela 4, exceto na execução do algoritmo EP, há uma perda de desempenho quando todos os núcleos de processamento estão alocados para a aplicação. A diferença no tempo de execução chega a 25% em comparação ao tempo obtido com o mesmo *cluster* configurado com um núcleo ocioso por máquina.

Com base na Figura 35, percebe-se ganhos em desempenho com o aumento do número de processos porém, exceto para o algoritmo de granularidade grossa EP, quando todos os 16 núcleos de processamento disponíveis no sistema N8xP1 estão alocados para a aplicação, o *speedup* (relativo ao tempo de execução com 1 processo) diminui. Além disso, note que o algoritmo EP apresenta um *speedup* linear, exceto no caso com 16 processos. Já o algoritmo CG apresenta *speedup* superlinear.

A Figura 36 mostra o gráfico da eficiência de cada um dos algoritmos, sendo que quase todos apresentam padrões decrescentes com o aumento do número de processos, salvo os casos de 4 para 8 processos dos algoritmos CG e MG. De forma mais acentuada, todos os algoritmos sofreram perda de eficiência quando se passou de 8 para 16 processos executando a aplicação, inclusive o algoritmo EP. O algoritmo CG, que apresentou *speedup* mais do que linear, foi o único a obter eficiência maior que 1 (ou 100%), embora apresente uma queda brusca de eficiência quando há 16 processos executando a aplicação.

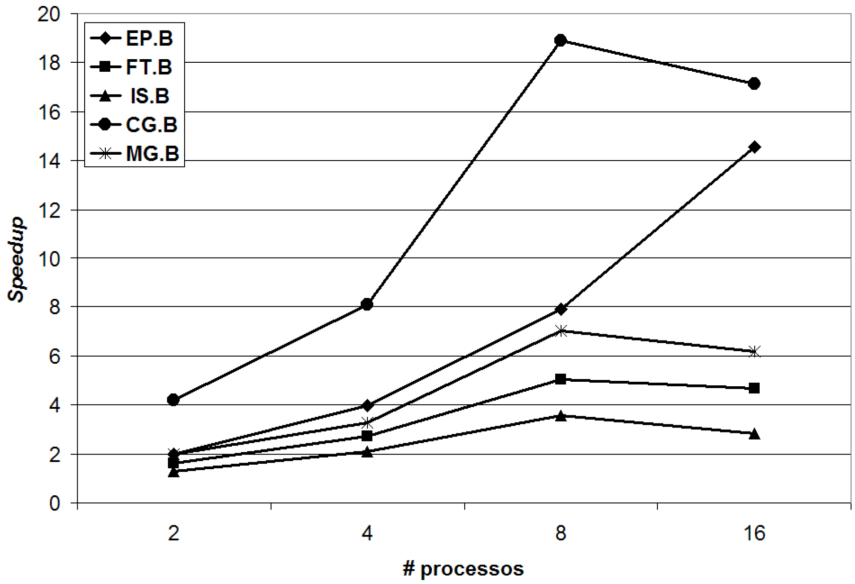



Figura 35: *Speedup* do NPB no ambiente N8xP1 (Caso 1A)

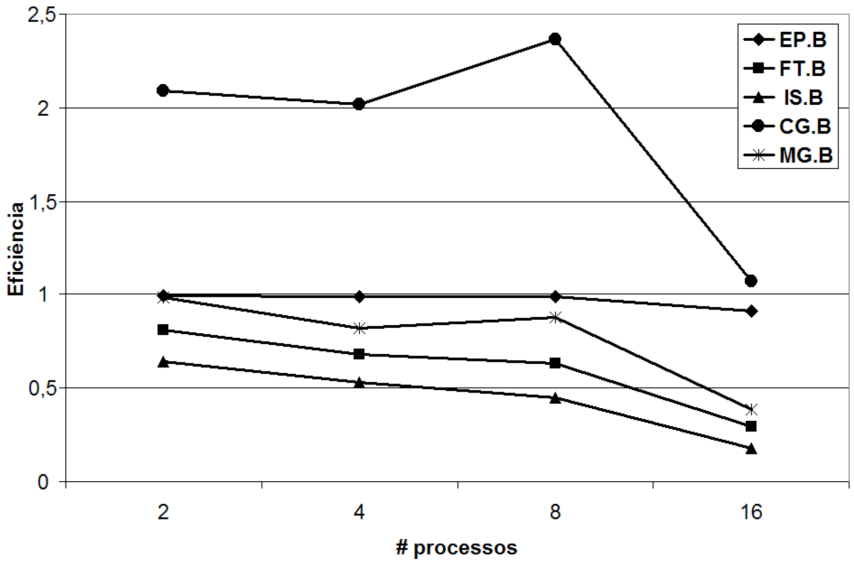



Figura 36: Eficiência do NPB no ambiente N8xP1 (Caso 1A)

Tanto os resultados de *speedup* quanto de eficiência, apontam uma queda no desempenho quando se passa de 8 para 16 processos alocados para a aplicação, principalmente com os algoritmos de granularidade média e fina. Portanto, os resultados apontam que a escalabilidade do *cluster* N8xP1, com um núcleo ocioso em relação à aplicação por máquina, é melhor do que a escalabilidade deste mesmo *cluster* quando todos os núcleos de processamento estão ocupados com a aplicação.

#### 4.1.1.3 Conclusões

Primeiramente, o sucesso da alternativa experimentada está na distribuição adequada dos núcleos ociosos. Os resultados mostram que, em um sistema homogêneo, de nada adianta deixar núcleos ociosos em uma máquina enquanto outra máquina está com todos núcleos executando processos da aplicação, pois esta última vai acabar determinando o desempenho do sistema como um todo. Além disso, os resultados indicam que é possível obter ganho de desempenho no ambiente de *cluster* em estudo, atingindo de 8% até 25% de redução no tempo de execução exceto com algoritmos de granularidade grossa, quando pelo menos um núcleo de processamento por máquina não é alocado para a aplicação, como no sistema N8xP1.

Com o algoritmo EP, de granularidade grossa e necessidade de comunicação mínima, foi possível constatar que o desempenho por núcleo dos sistemas N2xP4 e N1xP8 (com processadores AMD) é superior ao dos sistemas N8xP1 e N4xP2 (com processadores Intel®). Porém, há de se ter em mente que a tecnologia dos processadores usados são de gerações distintas. De qualquer forma, esperava-se que o impacto dessa diferença no tempo de execução fosse diminuir com os algoritmos de menor granularidade já que, nestes casos, o processo de comunicação se torna uma fonte considerável de *overhead*.

No entanto, foi surpreendente constatar com um algoritmo de granularidade média, o FT, que o sistema N8xP1, configurado com um núcleo ocioso por máquina, proporcionou um tempo de execução praticamente igual ao sistema N1xP8, que não é um *cluster* interconectado por Gigabit Ethernet, e sim uma única máquina com oito núcleos de processamento interconectados em topologia *crossbar*, com um banco de memória dedicado para cada dois núcleos operando a uma freqüência 50% maior do que a freqüência de acesso à memória do *cluster* N8xP1 (vide Figura 22). Além disso, os resultados coletados com o *b_eff* mostram que o desempenho da comunicação neste sistema multiprocessado é muito superior ao desempenho da comunicação entre processos das configurações de *cluster* interconectadas por Gigabit Ethernet.

Enfim, o tempo de execução obtido pelo *cluster* configurado com um núcleo ocioso por máquina foi o menor dentre todos os sistemas em quatro dos cinco algoritmos NPB experimentados. Portanto, mesmo executando a aplicação com apenas 8 núcleos de processamento em contrapartida à utilização dos 16 disponíveis, obteve-se uma configuração de *cluster* mais eficiente e com melhor escalabilidade.

Sendo assim, a investigação realizada com o intuito de ganhar desempenho em ambi-

entes de *cluster* teve sucesso com essa abordagem, mostrando-se uma alternativa interessante para experimentação com o modelo WRF, a fim de reduzir seu tempo de execução no ambiente de *cluster* em questão.

### 4.1.2 Abordagem Proposta

Devido à ausência de um processador de rede, todo processamento decorrente da comunicação via Ethernet é atribuída a um processador principal, isto é, concorrendo pelos mesmos processadores utilizados pela aplicação. No caso de um único processador por computador, a aplicação deve necessariamente parar sua execução para dar lugar ao processamento relativo à comunicação, decorrente da sua própria necessidade de interação entre os processos distribuídos, o que causa uma grande perda de desempenho.

Quando todos os núcleos são alocados para a execução da aplicação, ocorrem inúmeras trocas de contexto entre os processos dos diferentes níveis, já que a efetivação da comunicação não se dá em nível de aplicação. Conforme Hennessy e Patterson (2003), uma troca de contexto de um processo para outro tipicamente requer de centenas a milhares de ciclos de processamento, provocando um *overhead* considerável.

Porém, ainda mais com o advento dos processadores *multi-core*, o acesso a computadores com múltiplos núcleos está se consolidando como uma tecnologia de prateleira (*commodity*) de baixo custo. Quando tais computadores são agregados via Ethernet para a composição de um *cluster*, surge a possibilidade de melhorar o desempenho do sistema através da sobreposição do processamento da aplicação e do processamento decorrente da comunicação entre os processos da aplicação. Sendo assim, dependendo da aplicação, o tempo gasto com o processamento decorrente da comunicação poderá ocorrer simultaneamente, e seu impacto sobre o tempo de execução final da aplicação poderá ser reduzido.

Enfim, a abordagem proposta sugere que não sejam alocados aos processos da aplicação todos os núcleos de processamento disponíveis em cada computador, de modo que os núcleos então ociosos em relação à aplicação podem se ocupar com as tarefas decorrentes de sua execução, como o processamento de pacotes e protocolos de comunicação. Portanto, em função desse paralelismo, espera-se que o tempo gasto com o processamento decorrente da comunicação entre processos tenha seu impacto reduzido, traduzindo-se em um menor tempo de execução da aplicação. Esta é a idéia norteadora da abordagem proposta neste estudo de caso.

### 4.1.3 Experimentos (Caso 1B)

O modelo WRF foi avaliado no mesmo ambiente de *cluster*, apresentado na Figura 37 em um esquema ilustrativo, com três configurações distintas. Uma delas segue a abordagem proposta. Também foram realizados experimentos com o modelo WRF em um único computador multiprocessado com até 8 núcleos de processamento, a titulo de comparação.

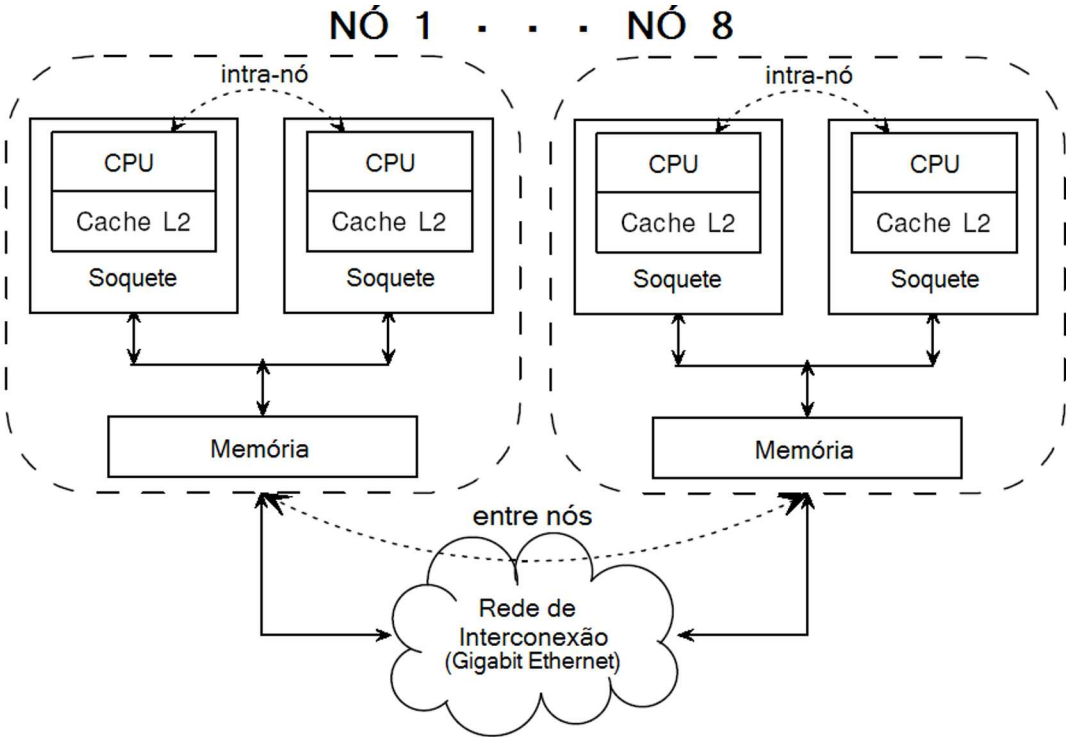



Figura 37: Esquema ilustrativo do ambiente de experimentação do Caso 1B

A nomenclatura dos sistemas apresentados em detalhes na Figura 38 seguem um padrão conforme o número de nós (N) e o número de núcleos de processamento ativos por nó (P). Por exemplo, o sistema N8xP1 é composto por 8 computadores (N = 8), cada um com 2 núcleos de processamento, porém apenas 1 núcleo é alocado para a aplicação (P = 1). O segundo núcleo de cada computador está, portanto, ocioso em relação à aplicação. Note que o sistema CMP-SMP não é um *cluster*, e sim um nó multiprocessado com dois processadores *quad-core*.

Os ambientes baseados em processadores Xeon (INTEL®, 2004) rodam Linux kernel 2.6.8.1 e o sistema CMP-SMP, com processadores *quad-core* Opteron (AMD, 2007), roda Linux kernel 2.6.22.8, todos estão com suporte a SMP ativado e a memória *swap* desativada. Além disso, todos os sistemas estão isolados de ruídos externos e dedicados aos experimentos, isto é, não estão operando quaisquer outros serviços, exceto a configuração

| INFO / SISTEMA | N8xP2 | N8xP1 | N4xP2 | CMP-SMP |
|---|---|---|---|---|
| Interconexão | Gigabit Ethernet (GigE) | | | Crossbar |
| | 3Com Switch 3812 | | | HyperTransport |
| MTU | 1500 | | | N/A |
| *Modelo do processador* | 32-bit Intel Xeon (DP) | | | 64-bit AMD Opteron 2350 |
| *Veloc. do processador* | 2.66 GHz | | | 2 Ghz |
| *Tecnologia Manuf.* | 130nm | | | 65nm |
| *# Cores por soquete* | 1 | | | 4 |
| *# Cores por nó* | 2 | 2 | 2 | 8 |
| *# nós* | 8 | 8 | 4 | 1 |
| *# Cores Ativo/Ocioso* | *16/0* | *8/8* | *8/0* | *variável* |
| | POR CORE | | | |
| *Cache L1 (I/D)* | 12KB/8KB | | | 64KB/64KB |
| *Cache L2* | 512KB | | | 512KB |
| *Cache L3* | - | | | 2MB |
| *DRAM* | 512MB | 1GB | 512MB | 1GB |
| *Velocidade DRAM* | 533MHz | | | 1000Mhz |
| *BogoMIPS* | ~ 5300 | | | ~ 4000 |

Figura 38: Ambientes de experimentação do Caso 1B

mínima necessária à execução dos experimentos com a biblioteca MPICH2.

É importante ressaltar que as aplicações usadas nestes experimentos são baseadas exclusivamente em MPI para a extração do paralelismo, embora haja trabalhos (RABENSEIFNER; WELLEIN, 2003; CAPPELLO; ETIEMBLE, 2000) que indiquem um modelo de programação híbrida, com OpenMP para paralelismo intra-nó e MPI entre nós, como o meio mais eficiente de utilizar agregados de computadores multiprocessados. Porém, como o foco deste trabalho está em analisar a sobreposição de computação e comunicação, levou-se em consideração apenas a comunicação MPI por soquete entre os processos, seja entre nós ou intra-nó. Além disso, programação paralela com MPI deve continuar importante por razões de portabilidade e até mesmo pela enorme quantidade de aplicações baseadas em MPI.

#### 4.1.3.1 Benchmark de rede: *b_eff*

Para caracterizar cada ambiente (exceto o sistema CMP-SMP) em relação à comunicação entre processos com métricas de interesse primário, como latência e taxa de transferência (COULOURIS; DOLLIMORE; KINDBERG, 2005), foi executado o algoritmo de benchmark de comunicação *b_eff* (RABENSEIFNER; KONIGES, 2001), que possui uma versão disponível como parte do HPC Challenge Benchmark (LUSZCZEK et al., 2006). No entanto, esta versão testa a taxa de transferência e latência apenas para mensagens de tamanho 8 e 2.000.000 bytes. Sendo assim, adaptamos o algoritmo *b_eff* para avaliar as características de comunicação para uma gama de tamanhos de mensagens.

| LATÊNCIA (ms) | # processos | modo de comunic. | 0 | 1K | 8K | 32K | 64K | 128K | 1M | 4M |
|---|---|---|---|---|---|---|---|---|---|---|
| N8xP2 | 2 | uni-direcional | 0.0153 | 0.0173 | 0.0250 | 0.0557 | 0.1552 | 0.3029 | 2.6355 | 12.5876 |
| | 2 | bi-direcional | 0.0168 | 0.0207 | 0.0304 | 0.0998 | 0.3246 | 0.7457 | 5.8836 | 23.6697 |
| | 16 | bi-direcional | 0.0469 | 0.0576 | 0.2327 | 1.0482 | 2.3646 | 5.0371 | 40.4980 | 162.1403 |
| N8xP1 | 2 | uni-direcional | 0.0342 | 0.0561 | 0.1281 | 0.3371 | 0.6215 | 1.1798 | 9.0810 | 35.8172 |
| | 2 | bi-direcional | 0.0261 | 0.0381 | 0.1211 | 0.4015 | 0.8061 | 1.5252 | 11.7471 | 44.2522 |
| | 8 | bi-direcional | 0.0289 | 0.0417 | 0.1300 | 0.4360 | 0.8788 | 1.7858 | 12.7988 | 51.4637 |
| N4xP2 | 2 | uni-direcional | 0.0153 | 0.0173 | 0.0250 | 0.0557 | 0.1552 | 0.3029 | 2.6355 | 12.5876 |
| | 2 | bi-direcional | 0.0168 | 0.0207 | 0.0304 | 0.0998 | 0.3246 | 0.7457 | 5.8836 | 23.6697 |
| | 8 | bi-direcional | 0.0337 | 0.0463 | 0.1992 | 0.7589 | 1.6351 | 3.2785 | 24.2303 | 99.6214 |

Figura 39: Latência coletada com o b_eff (Caso 1B)

| TX. TRANSF. (MB/s) | # processos | modo de comunic. | 0 | 1K | 8K | 32K | 64K | 128K | 1M | 4M |
|---|---|---|---|---|---|---|---|---|---|---|
| N8xP2 | 2 | uni-direcional | 0.1173 | 54.0247 | 292.4233 | 528.6114 | 421.2688 | 422.7265 | 378.9459 | 371.8689 |
| | 2 | bi-direcional | 0.1143 | 47.3600 | 272.5599 | 301.3550 | 206.6780 | 178.6099 | 156.8393 | 171.6436 |
| | 16 | bi-direcional | 0.0396 | 16.3430 | 33.6024 | 30.9186 | 27.1158 | 25.5238 | 25.6144 | 25.2320 |
| N8xP1 | 2 | uni-direcional | 0.0548 | 16.6795 | 63.5115 | 97.2331 | 104.8752 | 110.8938 | 115.5708 | 117.1184 |
| | 2 | bi-direcional | 0.0802 | 26.8313 | 67.4022 | 77.7124 | 81.6810 | 85.7770 | 88.7027 | 95.0442 |
| | 8 | bi-direcional | 0.0611 | 21.8215 | 59.9733 | 74.0701 | 75.1796 | 74.3785 | 79.4108 | 80.0891 |
| N4xP2 | 2 | uni-direcional | 0.1173 | 54.0247 | 292.4233 | 528.6114 | 421.2688 | 422.7265 | 378.9459 | 371.8689 |
| | 2 | bi-direcional | 0.1143 | 47.3600 | 272.5599 | 301.3550 | 206.6780 | 178.6099 | 156.8393 | 171.6436 |
| | 8 | bi-direcional | 0.0519 | 19.6169 | 39.4759 | 46.1811 | 43.6014 | 42.9530 | 41.3949 | 44.3636 |

Figura 40: Taxa de transferência coletada com o b_eff (Caso 1B)

Com base nos resultados coletados, conforme as Figuras 39 e 40, verifica-se que as configurações N4xP2 e N8xP1 apresentam resultados de latência e taxa de transferência semelhantes para a comunicação entre dois processos, tanto uni-direcional como bi-direcional, pois estão localizados no mesmo nó e os processos utilizam, portanto, o barramento interno do computador como rede de interconexão. Já no ambiente N8xP1, a comunicação entre dois processos é cerca de duas vezes pior porque a comunicação se dá através da rede Gigabit Ethernet, já que os processos estão localizados em computadores distintos.

Porém, vale ressaltar que em uma aplicação com 8 ou 16 processos, por exemplo, é provável que mais do que dois processos se comuniquem simultaneamente, possivelmente todos. Por isso, foram coletados dados referentes ao pior caso, em que todos os processos se comunicam ao mesmo tempo e nos dois sentidos, a fim de melhor caracterizar os ambientes em termos de latência e taxa de transferência.

Neste caso, os resultados são bem diferentes em comparação à situação em que apenas dois processos se comunicam. O ambiente de *cluster* N8xP1 passa a apresentar o melhor desempenho tanto em latência como em taxa de transferência, e o ambiente N8xP2 apresenta os piores valores em ambas as métricas.

A maior demanda por uma característica de comunicação ou outra dependerá da aplicação paralela, porém, essa caracterização mais ampla do desempenho da comunicação entre processos, através da taxa de transferência e latência de todos processos se comunicando simultaneamente, permite revelar vantagens e desvantagens não percebidas quando

apenas 2 processos se comunicam.

#### 4.1.3.2 Modelo WRF

Como já mencionado, o WRF utiliza uma matriz tri-dimensional para a representação da atmosfera, desde metros até milhares de quilômetros, com diversas informações como de topografia e dados de observatórios para alimentar a simulação com uma condição inicial. As simulações aqui apresentadas rodam o modelo já compilado (ifort 9.0 sem suporte a OpenMP) e disponível no ambiente em sua versão 2.2, tomando como entrada o conjunto de dados da forma que é utilizado neste ambiente de produção da empresa.

Na execução paralela do modelo WRF, cada processo recebe uma sub-matriz de tamanho aproximadamente igual, que diminui com o aumento do número de processos. Quanto à comunicação, a redistribuição dos dados laterais, nos quatro limites lógicos de cada sub-matriz, é a principal atividade, ocorrendo a cada iteração com mensagens entre 10 e 100 kilobytes (KERBYSON; BARKER; DAVIS, 2007). Cada iteração avança o tempo de simulação em 75 segundos, totalizando 96 e 576 iterações nas previsões para 2 e 12 horas, respectivamente. Além disso, a cada 12 iterações (ou 15 minutos de simulação) ocorre uma iteração de radiação física, que se soma ao tempo da iteração ordinária, e a cada 145 (ou 3 horas de simulação) ocorrem picos por causa da geração do arquivo de saída.

| *2 horas* | N8xP2 (16 proc.) | N8xP1 (8 proc.) | N4xP2 (8 proc.) |
|---|---|---|---|
| Tempo de execução (min:seg) | 24:12 | 18:03 | 24:50 |
| Redução de tempo | 0% | 25,4% | -2,6% |
| *Speedup* Relativo | 1 | 1,34 | 0,98 |

| *12 horas* | N8xP2 (16 proc.) | N8xP1 (8 proc.) | N4xP2 (8 proc.) |
|---|---|---|---|
| Tempo de execução (h:mm:ss) | 2:01:07 | 1:33:49 | 2:01:54 |
| Redução de tempo | 0% | 22,5% | -0,7% |
| *Speedup* Relativo | 1 | 1,29 | 0,99 |

Figura 41: Resultados da execução do WRF nos ambientes de *cluster* (Caso 1B)

Conforme a Figura 41, as previsões de tempo para 2 e 12 horas, apresentaram uma

redução maior do que 20% no tempo de execução. Os resultados obtidos com o WRF mostram um ganho de desempenho quando o *cluster* é configurado segundo a proposta, atingindo um *speedup* relativo de 1,29 em comparação à previsão do tempo para 12 horas no ambiente de *cluster* original, com 16 processos. Observe que o tempo de execução aumenta quando todos os processadores disponíveis são alocados a processos da aplicação, como é o caso dos sistemas N8xP2 e N4xP2.

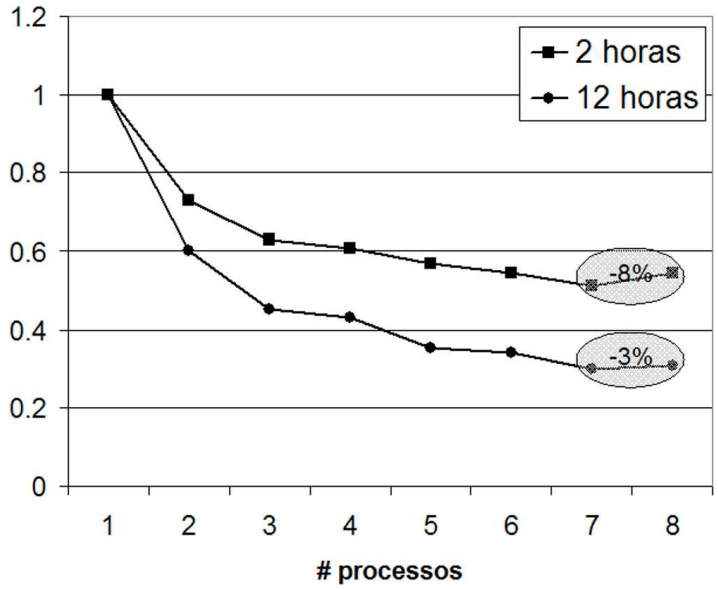



Figura 42: *Speedup* do WRF no sistema CMP-SMP (Caso 1B)

Em adição aos experimentos em ambientes de *cluster*, a Figura 42 apresenta os resultados obtidos em uma única máquina multiprocessada, com dois processadores AMD *quad-core*. Pode-se observar que, embora a interação entre os processos se dê sem necessidade de acesso à rede, o tempo de execução aumenta quando todos os processadores disponíveis são alocados a processos da aplicação. Esta constatação vem para reforçar as indicações dos resultados anteriores de que a abordagem proposta é pertinente, oferecendo melhores eficiência e desempenho, traduzidos em um menor tempo de execução.

### 4.1.4 Conclusões

Como indicação deste primeiro caso do estudo empírico, foi possível comprovar o sucesso da abordagem proposta em reduzir o tempo de execução do modelo numérico WRF de previsão de tempo sem, no entanto, investir em novos equipamentos para o *cluster*, tal como uma rede de interconexão mais eficiente, Myrinet ou Infiniband, por exemplo. Vale ressaltar que a abordagem aproveita-se da tendência atual com relação a computadores multiprocessados.

Embora apenas 8 núcleos tenham sido alocados para a aplicação, houve ganho de desempenho em relação à configuração de *cluster* no qual todos os 16 núcleos de processamento disponíveis foram alocados a processos da aplicação. O ganho expressivo no desempenho do modelo WRF, com uma redução de mais de 20% no seu tempo de execução, também foi verificado para o algoritmo de granularidade fina IS do NAS Parallel Benchmark. Como foi mostrado no Caso 1A, a eficiência e a escalabilidade do cluster configurado segundo a proposta melhorou em relação à configuração original. A abordagem se mostrou ineficiente apenas para o caso com o algoritmo de granularidade grossa EP. Portanto, a eficiência da abordagem dependerá da aplicação paralela.

Além disso, a caracterização das capacidades de comunicação de cada ambiente com o benchmark de rede *b_eff*, através da taxa de transferência e latência de todos processos se comunicando simultaneamente, também revelou vantagens do *cluster* configurado segundo a abordagem, não percebidas quando apenas 2 processos se comunicam.

Enfim, os resultados dos casos 1A e 1B indicam ganho de desempenho com a implementação da abordagem proposta, alcançando assim o objetivo de propor uma alternativa para reduzir o tempo de execução do modelo WRF sem, no entanto, adquirir novos componentes para o ambiente de *cluster* em questão.

## 4.2 Caso 2

Este segundo caso foi desenvolvido com o objetivo de otimizar o desempenho do modelo no ambiente de *cluster* recém adquirido pela empresa, antes de colocá-lo em produção com a última versão disponível do WRF.

A principal contribuição da presente dissertação está neste estudo de caso. Foi realizada uma pesquisa mais aprofundada sobre a própria aplicação WRF, de relevante importância para a comunidade científica, e também uma análise mais detalhada sobre as melhores práticas e os problemas encontrados durante o processo de otimização. Nesse sentido, foi-se em busca de alternativas viáveis e eficazes a fim de tirar a máxima eficiência deste ambiente de *cluster*, reduzindo o tempo de execução do modelo WRF em comparação à configuração original, sem nenhuma especialização para a aplicação em questão.

Conforme Zamani (2005), o desempenho de aplicações executadas em ambientes de *cluster* depende principalmente da escolha do modelo de programação paralela, das características da própria aplicação quanto às necessidades de computação e comunicação, e do desempenho do subsistema de comunicação.

Embora as duas últimas características sejam praticamente inquestionáveis, a escolha pelo modelo de passagem de mensagem ou por um modelo híbrido (por exemplo, MPI+OpenMP) para prover melhor desempenho em agregados de computadores multiprocessados ainda está em debate (ZAMANI, 2005). Este segundo estudo de caso leva em consideração a utilização da abordagem híbrida com o modelo baseado em memória compartilhada para a comunicação de processos localizados no mesmo nó do *cluster* como alternativa para a otimização de desempenho de aplicações científicas.

Vale relembrar que o presente estudo de caso leva em consideração tão somente o modelo WRF, não contabilizando as operações de pré ou pós-processamento. Além disso, todos os experimentos são realizados com base na mesma configuração do modelo WRF (relativa aos componentes de física) e mesmo conjunto de dados de entrada (do dia 29 de maio de 2008) utilizados no ambiente de produção da empresa na previsão meteorológica. Neste segundo caso, as simulações são configuradas para realizar a previsão do tempo para 72 horas (3 dias). O arquivo de configuração encontra-se disponível no Anexo A.

### 4.2.1 Experimentos

A Figura 43 apresenta características do ambiente em foco, descrevendo-o em termos de componentes. Já a Figura 44 ilustra o ambiente de *cluster*, também mostrando como estão organizados os núcleos de processamento em cada nó.

| INFO / SISTEMA | Ambiente 2 |
|---|---|
| ***# Nós*** | 6 |
| ***# Cores por nó*** | 8 |
| **Interconexão** | Gigabit Ethernet |
| | 3Com Switch 3812 |
| **MTU** | 1500 |
| ***Modelo do processador*** | 64-bit AMD Opteron 2350 |
| ***Veloc. do processador*** | 2 Ghz |
| ***Tecnologia Manuf.*** | 65nm |
| ***Cache L1 (I/D)*** | 64KB/64KB |
| ***Cache L2*** | 512KB |
| ***Cache L3*** | 2MB |
| ***# Cores por soquete*** | 4 |
| ***DRAM*** | 8GB |
| ***Velocidade DRAM*** | 1000Mhz |

Figura 43: Características do ambiente de experimentação do Caso 2

O processo de otimização do ambiente de *cluster* parte de uma configuração semelhante à implementada no caso anterior. Os computadores rodam Debian Linux kernel 2.6.18-6-amd64 e o ambiente de *cluster* está isolado de ruídos externos e dedicado aos

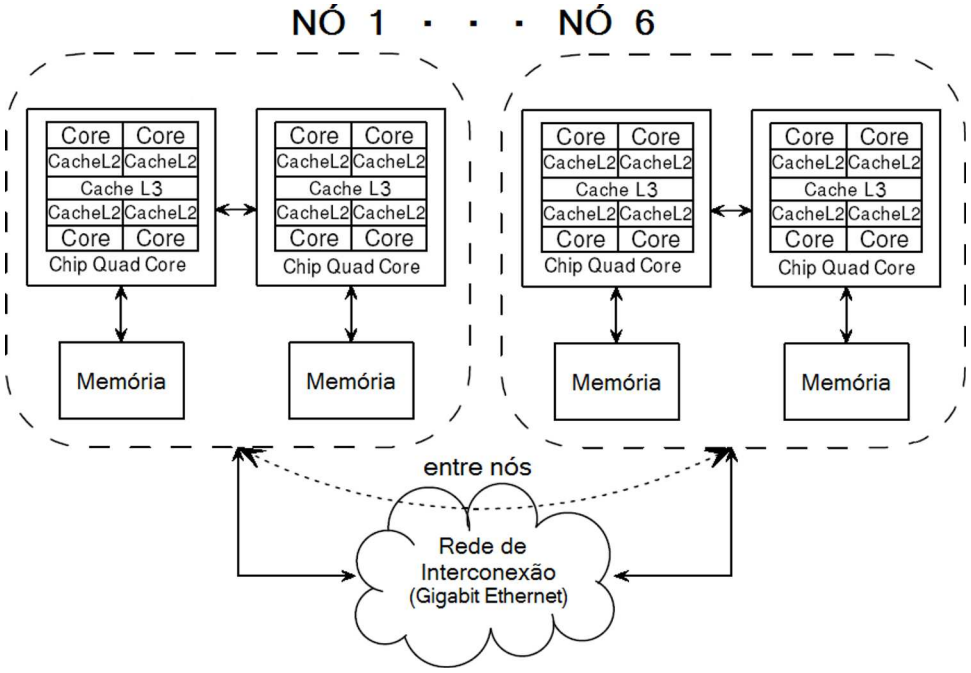



Figura 44: Esquema ilustrativo do ambiente de experimentação do Caso 2

experimentos, pois não estão operando quaisquer outros serviços, exceto a configuração mínima necessária à execução dos experimentos com a biblioteca MPICH2 versão 1.0.6p1. Além disso, tomou-se o cuidado de desativar a utilização da memória virtual (*swap*).

Quanto ao subsistema de comunicação, foi utilizada a rede Gigabit Ethernet para interconectar fisicamente os nós. As Figuras 45 e 46 ilustram os dados coletados com o benchmark de rede b_eff para 2 processos em um mesmo nó e em nós distintos, 8 processos em um mesmo nó, e todos os 48 processadores se comunicando. A comunicação foi mediada pelo canal CH3:SOCK do MPICH2 (doravante denominado MPICH-SOCK), que utiliza soquetes para a comunicação entre processos.

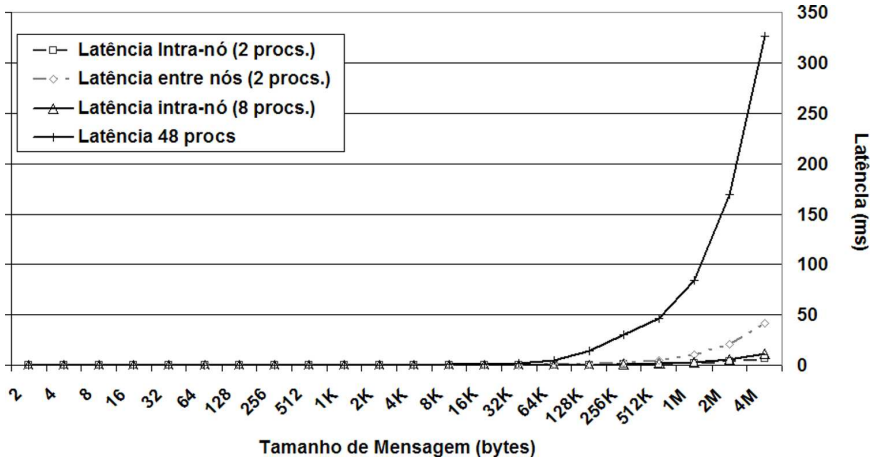



Figura 45: Latência coletada com o b_eff (Caso 2)

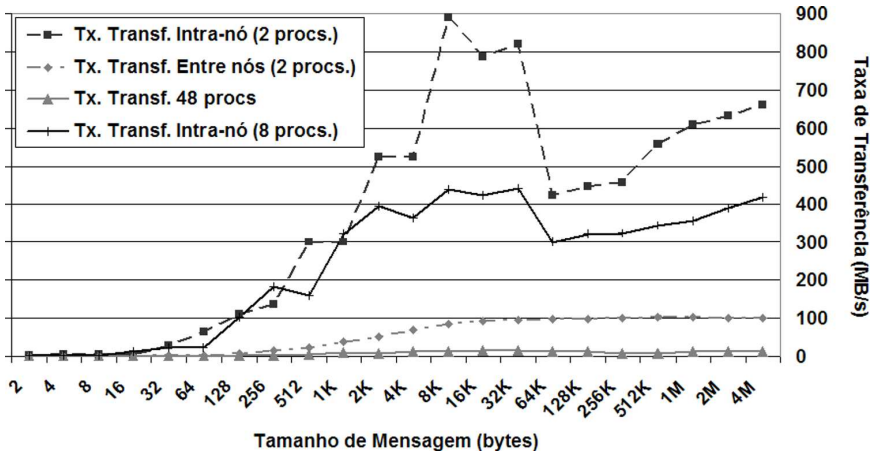



Figura 46: Taxa de transferência coletada com o b_eff (Caso 2)

Quanto à aplicação, convém relembrar que o WRF utiliza uma matriz tri-dimensional para a representação da atmosfera, com diversas informações sobre, por exemplo, a topografia e a situação meteorológica da região em questão, a fim de alimentar a simulação com uma condição inicial. Na execução paralela do modelo WRF, cada processo recebe uma sub-matriz da atmosfera de tamanho aproximadamente igual, diminuindo com o aumento do número de processos. A redistribuição dos dados laterais, nos quatro limites lógicos de cada sub-matriz, é a principal atividade que demanda comunicação entre os processos, ocorrendo a cada iteração com mensagens entre 10 e 100 kilobytes (KERBYSON; BARKER; DAVIS, 2007). Cada iteração avança o tempo de simulação em 75 segundos. Além disso, a cada 12 iterações (ou 15 minutos de simulação) ocorre uma iteração de radiação física, que se soma ao tempo de processamento da iteração ordinária, e a cada 145 (ou 3 horas de simulação) ocorrem picos por causa da geração do arquivo de saída.

As simulações apresentadas a seguir rodam o modelo WRF versão 3, compilado com gfortran 4.1.2.

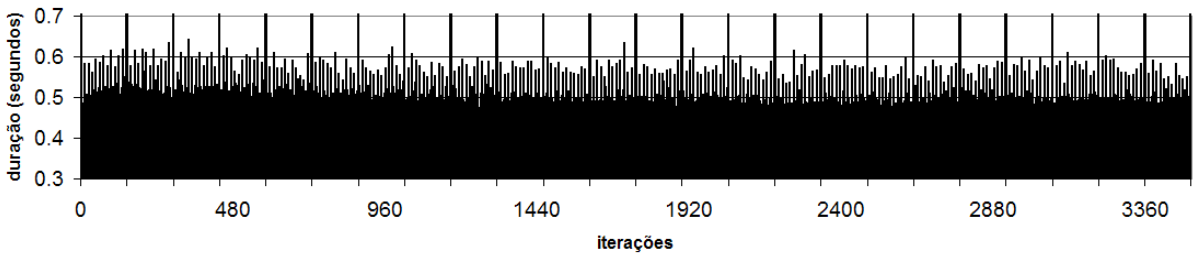



Figura 47: Duração de cada iteração com MPICH-SOCK, totalizando 34min16seg.

A Figura 47 apresenta a duração de cada iteração ao longo de toda a execução do modelo WRF com o MPICH-SOCK. Os 48 processadores disponíveis no ambiente estão

alocados a processos da aplicação. Os picos que extrapolam o eixo Y do gráfico referem-se ao agrupamento dos dados resultantes para a geração do arquivo de saída do modelo, que duram cerca de 8 segundos neste caso.

Porém, como ilustra a Figura 48, os processadores estão subutilizados nesta configuração com o MPICH-SOCK. Durante a maior parte da execução do modelo, percebe-se que os processadores não utilizam nem 50% de sua capacidade de processamento.

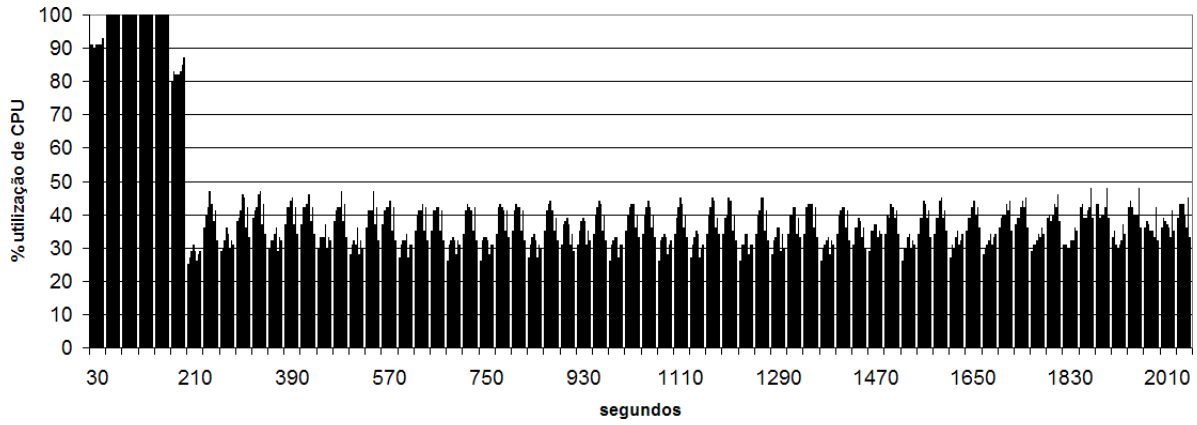



Figura 48: A utilização de oito núcleos em um mesmo nó com MPICH-SOCK.

Por não haver distinção entre processos localizados no mesmo nó ou em nós diferentes com o MPICH-SOCK, toda operação com o MPI é tratada como uma operação de comunicação comum, como se estivessem em nós diferentes. Sendo assim, é gasto tempo com a criação de soquetes para a interação entre os processos localizados no mesmo nó, além de envolver desnecessariamente o sistema operacional neste processo. O resultado é a subutilização dos processadores na execução da aplicação.

Como já foi explicado, o MPI é uma interface padrão que provê as primitivas de comunicação e sincronização necessárias à execução paralela e distribuída de uma aplicação, tendo a melhoria da portabilidade e do desempenho como principais metas. Porém, a efetivação da comunicação depende da implementação de uma biblioteca como o MPICH2, o qual implementa ambos os padrões MPI existentes: MPI-1 (MESSAGE PASSING INTERFACE FORUM, 1995) e MPI-2 (MESSAGE PASSING INTERFACE FORUM, 1997). Nesse sentido, a biblioteca MPICH2 implementa a comunicação efetiva com base em camadas, como ilustra a Figura 49.

A camada ADI3 apresenta a interface MPI para aplicação e a interface ADI3 para um *device* ADI3. O esquema apresenta o *device* CH3, que pode ser usado para qualquer tipo de rede, e também o *device* para redes Infiniband (CASSIDAY, 2000), que permite tirar vantagem das características específicas dessa rede de interconexão. É possível também

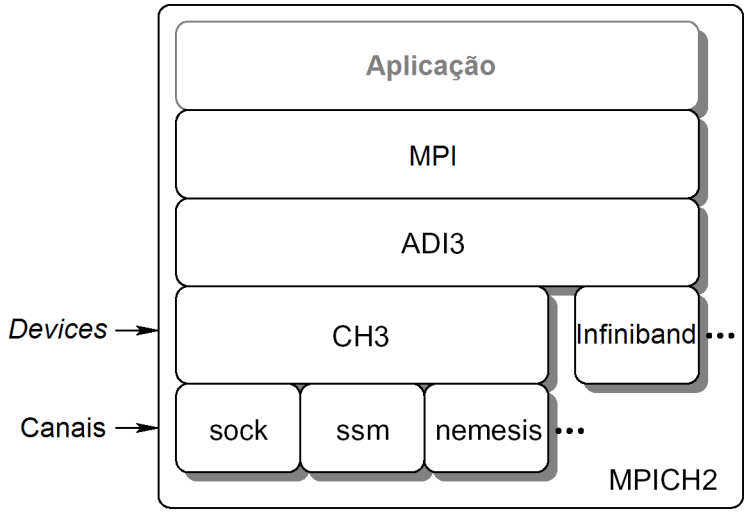



Figura 49: Esquema ilustrativo: MPICH2 (BUNTINAS; MERCIER; GROPP, 2007)

implementar um subsistema de comunicação acoplando um canal ao *device* CH3, como é caso do MPICH-SOCK, por exemplo.

Como solução para o problema da subutilização dos processadores na execução da aplicação, optou-se por um canal de comunicação do MPICH2 que faz essa distinção, usando memória compartilhada entre processos localizados no mesmo nó, e comunicação por soquete entre nós, como é o caso do canal SSM (doravante chamado MPICH-SSM).

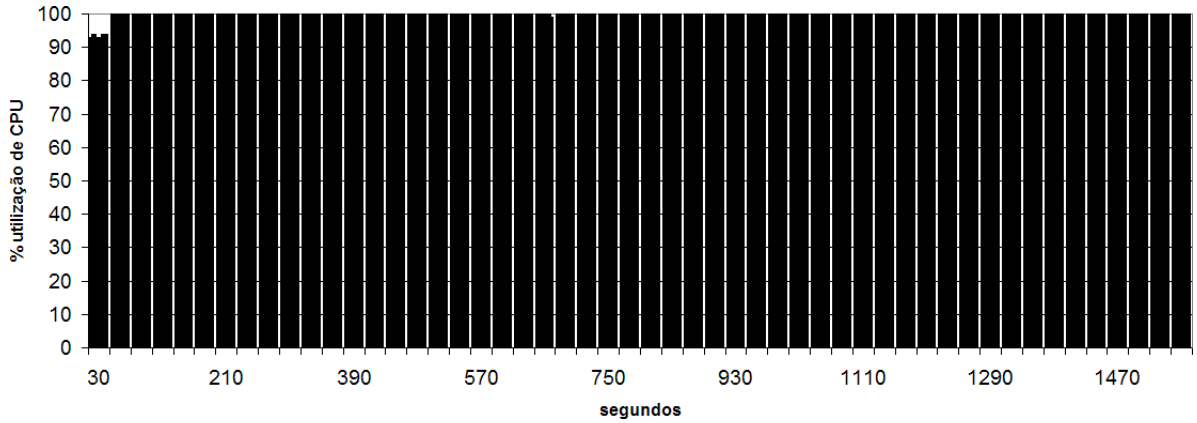



Figura 50: A utilização de oito núcleos em um mesmo nó com MPICH-SSM.

A Figura 50 apresenta a utilização dos processadores de um dos 6 nós do ambiente de *cluster* durante a execução do modelo WRF alocando todos os processadores disponíveis. Percebe-se que o MPICH-SSM solucionou satisfatoriamente o problema da subutilização dos processadores, já que os dados mostram todos processadores rodando a 100% de sua capacidade durante praticamente toda a execução do modelo.

Além disso, a Figura 51 ilustra o tempo de execução que baixou de, aproximadamente,

34 minutos para cerca de 28 minutos, uma diferença em torno de 18% em comparação à execução com o MPICH-SOCK. Vale ressaltar que os picos que extrapolam o eixo Y do gráfico, relativos ao agrupamento dos dados resultantes para a geração do arquivo de saída do modelo, duram cerca de 6 segundos com o MPICH-SSM.

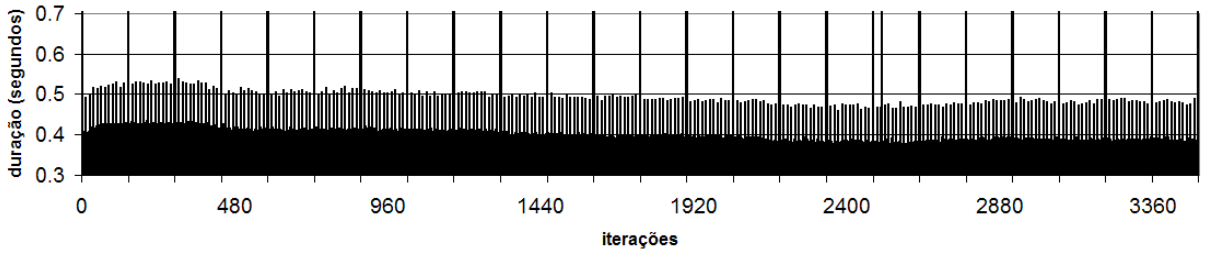



Figura 51: Duração de cada iteração com MPICH-SSM, totalizando 27min59seg.

A Figura 49 também apresenta o canal NEMESIS, que foi desenvolvido para ser um subsistema de comunicação escalável, baseado em memória compartilhada para processos localizados em um mesmo nó do *cluster* e de alto desempenho. De fato, Buntinas, Mercier e Gropp (2007) apresentam uma avaliação de desempenho do canal NEMESIS da biblioteca MPICH2 (doravante chamado MPICH-NEMESIS), que mostrou-se eficiente e com *overhead* muito pequeno. Em função dessas características, a utilização do MPICH-NEMESIS surgiu como uma alternativa interessante para o presente estudo de caso.

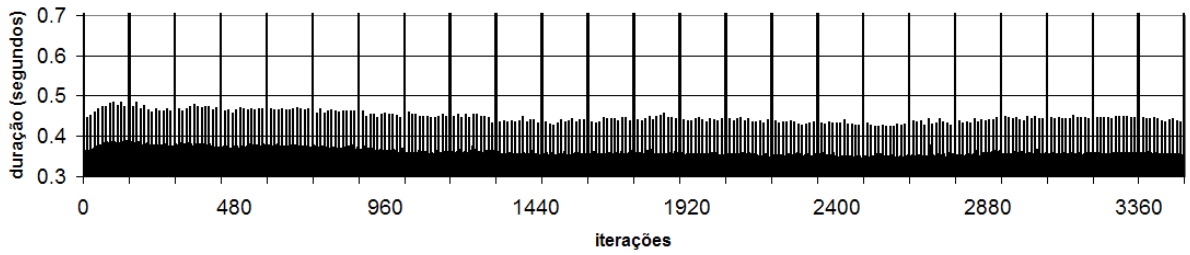



Figura 52: Duração de cada iteração com MPICH-NEMESIS, totalizando 25min44seg.

Realmente, como mostra a Figura 52, a duração das iterações do modelo WRF com o MPICH-NEMESIS diminuiu em comparação à execução com MPICH-SOCK (vide Figura 47) e com MPICH-SSM (vide Figura 51). Ademais, assim como com o MPICH-SSM, os picos que extrapolam o eixo Y do gráfico (geração do arquivo de saída do modelo) duram cerca de 6 segundos com o MPICH-NEMESIS.

Uma tabela comparativa entre as alternativas experimentadas é apresentada na Figura 53. Em comparação à execução com MPICH-SOCK, os resultados indicam redução no tempo de execução do modelo WRF de cerca de 18% com a utilização do MPICH-SSM e

de cerca de 25% com o MPICH-NEMESIS.

| *Previsão 72 horas* | MPICH-SOCK | MPICH-SSM | MPICH-NEMESIS |
|---|---|---|---|
| Tempo Execução (48 procs.) | 34min16seg | 27min59seg | 25min44seg |
| Redução de tempo | 0% | 18,34% | 24,90% |
| Speedup Relativo | 1 | 1,22 | 1,33 |

Figura 53: Comparação dos resultados com MPICH-SOCK, MPICH-SSM e MPICH-NEMESIS.

Enfim, os resultados empíricos indicam a adoção do MPICH-NEMESIS para extrair um maior desempenho deste ambiente de *cluster*, traduzindo-se em um menor tempo de execução do modelo WRF na previsão meteorólogica para 3 dias ou 72 horas.

| *Previsão 72 horas (NEMESIS)* | MTU 1500 | MTU 9000 | MTU 4500 |
|---|---|---|---|
| Tempo Execução (48 procs.) | 25min44seg | 25min00seg | 24min36seg |
| Redução de tempo | 0% | 2,85% | 4,40% |
| Speedup Relativo | 1 | 1,03 | 1,05 |

Figura 54: Comparação dos resultados com diversas MTU's (MPICH-NEMESIS).

Em adição à escolha do subsistema de comunicação da biblioteca MPICH2, o tamanho máximo das mensagens em nível de enlace, no caso a MTU (*Maximum Transmission Unit*) da rede Gigabit Ethernet, é outro aspecto que pode melhorar o desempenho do ambiente. Porém, como mostra a Figura 54, seu impacto é bem menor do que a escolha do subsistema de comunicação, por exemplo. Obteve-se pouco mais de 4% de redução no tempo de execução com a adequação do tamanho da MTU ao modelo WRF.

### 4.2.2 Conclusões

Com base nos resultados coletados neste estudo empírico, apresentados resumidamente na Figura 55, indica-se a adoção do MPICH-NEMESIS com a MTU da rede Gigabit Ethernet configurada para 4500 bytes para extrair um maior desempenho deste ambiente de *cluster* na execução do modelo WRF na previsão meteorólogica para 3 dias.

| *OTIMIZAÇÃO* | Antes | Depois |
|---|---|---|
| Tempo Execução (48 procs.) | 34min16seg | 24min36seg |
| Redução de tempo | 0% | 28,21% |
| Speedup Relativo | 1 | 1,39 |

Figura 55: Previsão meteorológica para 3 dias antes e depois do processo de otimização

A realização destes experimentos possibilitou uma maior compreensão sobre sistemas paralelos distribuídos. O *cluster* utilizado é um modelo híbrido no que diz respeito à sua composição, pois é um multicomputador de multiprocessadores, ou seja, para melhor atingir o objetivo de reduzir o tempo de execução da aplicação de alto desempenho, foi necessário implantar dois modelos de interação entre os processos da aplicação. Nesse sentido, portanto, os processos distribuídos entre os computadores agregados comunicam-se por meio de passagem de mensagens, enquanto o modelo de memória compartilhada permite a interação entre os processos localizados em um mesmo computador do *cluster*.

Enfim, o objetivo em vista foi alcançado com sucesso e, como resultado disso, a configuração proposta foi adotada como solução para o ambiente de produção que está sendo colocado em operação na empresa. Ademais, vale ressaltar que a empresa parceira torna-se o segundo centro brasileiro de meteorologia a dispor do modelo WRF em sua versão mais recente para suas atividades operacionais e de pesquisa.

# 5 Conclusões

Apesar da consolidação dos *clusters* como solução para prover alto desempenho, a escolha dos seus componentes, tais como os processadores ou a rede de interconexão que efetivamente agrega os recursos computacionais, está submetida à oferta disponibilizada pelo mercado no momento de sua construção. De fato, com o lançamento dos processadores *multi-core* como *commodity*, sua inserção em ambientes de *cluster* já é realidade e este novo contexto merece a devida atenção.

Neste trabalho de dissertação, foi realizado um estudo empírico sobre três casos distintos com ambientes de *cluster* homogêneos.

O primeiro estudo de caso foi realizado em um ambiente experimental como um exercício empírico sobre a relação entre tecnologias de rede de interconexão e características inerentes a aplicações paralelas. Com isso, foi possível conhecer e analisar as diferenças em desempenho de sistemas de comunicação distintos, como a tecnologia de rede Myrinet face à tecnologia Ethernet, diante de aplicações de granularidades distintas, bem como compreender as métricas comumente adotadas em avaliações de desempenho.

Já os outros dois estudos de casos foram realizados em agregados de computadores pertencentes a organizações, em uma aproximação com a realidade da computação de alto desempenho em ambientes de produção. Com o objetivo de construir sistemas de alto desempenho adequados à execução de aplicações científicas *grand challenge*, como a modelagem numérica de previsão meteorológica, foram implementadas técnicas para reduzir seu tempo de execução em comparação à condição anterior do ambiente.

Durante o processo de otimização do desempenho, foi possível entender os mecanismos de interação entre processos e os modelos de programação paralela envolvidos. No último estudo de caso, por exemplo, foi necessário implantar um modelo híbrido de interação entre os processos da aplicação (passagem de mensagens + memória compartilhada) para atingir uma redução de cerca de 30% no tempo de execução. Além disso, investigou-se o impacto da tendência atual no que diz respeito a processadores *multi-core*, bem como os

fatores redutores do desempenho (que resultam em *overhead*).

Dessa forma, portanto, pode-se contrastar na prática os pontos estudados durante a revisão bibliográfica. Além disso, compreendeu-se melhor as vantagens e desvantangens envolvidas nesses ambientes enquanto sistemas paralelos distribuídos, com o foco na modelagem de sistemas de alto desempenho. Enfim, o conhecimento adquirido com os estudos de casos possibilita uma melhor compreensão do processo e dos fatores envolvidos na implementação de ambientes de *cluster* adequados a cada aplicação paralela com demanda por alto desempenho, aproveitando melhor os recursos agregados.

Ademais, a importância deste trabalho transcende à ciência da computação como disciplina acadêmica, pois a empresa parceira ganha em capacidade e qualidade na previsão meteorológica do tempo com a última versão do WRF, seja para prevenir o impacto de desastres naturais ou para auxiliar na produção agrícola, ganhando também em potencial de pesquisa no âmbito daquela área de atuação.

# Perspectivas de Trabalhos Futuros

Como sugestão para possíveis trabalhos futuros estão:

- a aproximação com a realidade de outras aplicações de alto desempenho, envolvidas com atividades operacionais e de produção provenientes de outras áreas de atuação;
- a utilização de um protocolo leve para a comunicação entre os nós, já que a rede que interconecta os computadores agregados é local e isolada, não oferecendo maiores riscos de perda de dados, por exemplo;
- a extensão dos experimentos a outros agregados de computadores, se possível de grande escala, a fim de confrontar características relativas à escalabilidade, bem como utilizar redes de interconexão *network-on-chip* a fim de melhor quantificar os benefícios alcançados com as técnicas de otimização do desempenho utilizadas;
- uma investigação aprofundada sobre o comportamento de agregados de computadores com processadores *multi-core*, inclusive com a quantificação do *overhead* gerado por um crescente número de núcleos de processamento concorrendo por recursos computacionais compartilhados, a fim de minimizar a perda de eficiência.

# *Referências*

ALAM, S. R. et al. Characterization of scientific workloads on systems with multi-core processors. *IEEE International Symposium on Workload Characterization*, p. 225–236, 2006.

ALMASI, G. S.; GOTTLIEB, A. *Highly Parallel Computing*. 2ª. ed. [S.l.]: The Benjamin/Cummings Publishing Company Inc., 1994.

AMD. *AMD Opteron(TM) Processor Product Data Sheet*. [S.l.], Publication 23932, 2007.

AMSTRONG, B. et al. Hpc benchmarking and performance evaluation with realistic applications. *SPEC Benchmark Workshop*, 2006.

BAILEY, D. H. et al. The nas parallel benchmarks. *International Journal of Supercomputer Applications*, v. 5, n. 3, p. 63–73, 1991.

BODEN, N. et al. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, v. 15, n. 1, p. 29–36, 1995.

BRéCHIGNAC, C. Preface. *International Journal of Nanotechnology*, v. 5, n. 6/7/8, p. 569–570, 2008.

BRIGHTWELL, R.; UNDERWOOD, K. An analysis of the impact of mpi overlap and independent progress. *International Conference on Supercomputing*, 2004.

BUNTINAS, D.; MERCIER, G.; GROPP, W. Implementation and evaluation of shared-memory communication and synchronization operations in mpich2 using the nemesis communication subsystem. *Parallel Computing*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 33, n. 9, p. 634–644, 2007.

CAPPELLO, F.; ETIEMBLE, D. Mpi versus mpi+openmp on the ibm sp for the nas benchmarks. *Supercomputing*, 2000.

CASSIDAY, D. Infiniband architecture tutorial. *Hot Chips 12*, 2000.

CHAI, L.; GAO, Q.; PANDA, D. Understanding the impact of multi-core architecture in cluster computing: A case study with intel dual-core system. *CCGRID '07: IEEE International Symposium on Cluster Computing and the Grid*, IEEE Computer Society, p. 471–478, 2007.

CHAI, L.; HARTONO, A.; PANDA, D. Designing high performance and scalable mpi intra-node communication support for clusters. *IEEE International Conference on Cluster Computing*, 2006.

CHAPARRO, P. et al. Understanding the thermal implications of multi-core architectures. *IEEE TPDS*, v. 18, n. 8, p. 1055–1065, 2007.

CHEN, W. et al. A performance analysis of the berkeley upc compiler. *17th Annual International Conference on Supercomputing (ICS)*, 2003.

CHEN, Y. Nanoscale molecular electronic circuits. *Nano and Giga Challenges in Microelectronics*, 2004.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Distributed systems: Concepts and Design*. 4ª. ed. [S.l.]: Addison Wesley, 2005.

DUNNING, D. et al. The virtual interface architecture. *IEEE Micro*, v. 18, n. 2, p. 66–76, 1998.

ENSLOW, P. Multiprocessor organization survey. *ACM Computing Survey*, v. 9, n. 1, 1977.

FARAJ, A.; YUAN, X. Communication characteristics in the nas parallel benchmarks. *Parallel and Distributed Computing and Systems*, 2002.

FLICH, J. et al. On the potential of noc virtualization for multicore chips. *International Workshop on Multi-Core Computing Systems (MuCoCoS)*, 2008.

FLYNN, M. J. Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, v. 21, n. 9, p. 948–960, 1972.

FOSTER, I. *Designing and Building Parallel Programs*. 1ª. ed. [S.l.]: Addison-Wesley Publishing Company Inc., 1995.

GEIST, A. et al. *PVM: Parallel Virtual Machine*. [S.l.]: MIT Press, 1994.

GROPP, W. et al. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel Computing*, v. 22, n. 6, p. 789–828, 1996.

HENNESSY, J. L.; PATTERSON, D. A. *Computer Architecture - A Quantitative Approach*. 3ª. ed. [S.l.]: Morgan Kaufmann Publishers, 2003.

INTEL®. *Intel® Xeon® Processor with 533 MHz FSB at 2GHz to 3.20GHz Datasheet*. [S.l.], Publication 252135, 2004.

INTEL®. *Intel® Developer Forum*. 2006. Justin Rattner Keynote Speech, http://www.intel.com/pressroom/kits/events/idfspr_2006/.

JOHNSON, E. E. Completing an mimd multiprocessor taxonomy. *ACM SIGARCH Computer Architecture News*, v. 16, n. 3, p. 44–47, 1988.

JORDAN, H.; ALAGHBAND, G. *Fundamentals of Parallel Processing*. 1ª. ed. [S.l.]: Prentice Hall, 2003.

KARNIADAKIS, G.; KIRBY, R. *Parallel Scientific Computing in C++ and MPI*. 1ª. ed. [S.l.]: Cambridge Press, 2002.

KERBYSON, D.; BARKER, K.; DAVIS, K. Analysis of the weather research and forcasting (wrf) model on large-scale systems. 2007.

KIM, J.; LILJA, D. Characterization of communication patterns in message-passing parallel scientific application programs. *Communication, Architecture, and Applications for Network-Based Parallel Computing*, p. 202–216, 1998.

KUMAR, V. et al. *Introduction to Parallel Computing*. 1ª. ed. [S.l.]: The Benjamin/Cummings Publishing Company Inc., 1994.

LOBOSCO, M.; COSTA, V. S.; AMORIM, C. L. de. Performance evaluation of fast ethernet, giganet and myrinet on a cluster. *International Conference on Computational Science*, p. 296–305, 2002.

LUSZCZEK, P. et al. The hpc challenge (hpcc) benchmark suite. *IEEE SC06 Conference Tutorial*, 2006.

MAEX, K. Nanotechnology and nanocmos. *Nano and Giga Challenges in Microelectronics*, 2004.

MARTIN, R. *A Systematic Characterization of Application Sensitivity to Network Performance*. Tese (Doutorado), Berkeley, 1999.

MESSAGE PASSING INTERFACE FORUM. *MPI: A Message-Passing Interface Standard, Rel. 1.1*. www.mpi-forum.org, June 1995.

MESSAGE PASSING INTERFACE FORUM. *MPI-2: Extensions to the Message-Passing Interface*. www.mpi-forum.org, June 1997.

MEUER, H. et al. *TOP500 Supercomputer Sites*. 2008. Disponível em: <www.top500.org>.

MICHALAKES, J. et al. Design of a next-generation regional weather research and forecast model. *Towards Teracomputing, World Scientific*, p. 117–124, 1999.

MICHALAKES, J. et al. The weather research and forecast model: Software architecture and performance. *ECMWF Workshop on the Use of High Performance Computing in Meteorology*, 2004.

NATIONAL CENTER FOR ATMOSPHERIC RESEARCH. *A Description of the Advanced Research WRF Version 2*. [S.l.], January 2007.

PACHECO, P. *Parallel Programming with MPI*. 1ª. ed. [S.l.]: Morgan Kaufmann, 1996.

PARHAMI, B. *Introduction to Parallel Processing*. 1ª. ed. [S.l.]: Kluwer Academic Publishers, 2002.

PINTO, L. C.; MENDONçA, R. P.; DANTAS, M. A. R. The impact of interconnection networks and applications granularity to compound cluster configurations. *IEEE Symposium on Computers and Communications*, 2008.

POURREZA, H.; GRAHAM, P. On the programming impact of multi-core, multi-processor nodes in mpi clusters. *High Performance Computing Systems and Applications*, 2007.

QUINN, M. J. *Parallel Computing*. 2ª. ed. [S.l.]: McGraw-Hill Inc., 1994.

RABENSEIFNER, R.; KONIGES, A. E. The parallel communication and i/o bandwidth benchmarks: b_eff and b_eff_io. *Cray User Group Conference, CUG Summit*, 2001.

RABENSEIFNER, R.; WELLEIN, G. Communication and optimization aspects of parallel programming models on hybrid architectures. *International Journal of High Performance Computing Applications*, v. 17, n. 1, p. 49–62, 2003.

RAINA, S. *Virtual Shared Memory: A Survey of Techniques and Systems*. [S.l.], CSTR-92-36, University of Bristol, UK, 1992.

ROSSETTO, A. G. M. et al. Summit - a framework for coordinating applications execution in mobile grid environments. *IEEE/ACM International Conference on Grid Computing*, p. 129–136, 2007.

SAHNI, S.; THANVANTRI, V. *Parallel computing: Performance metrics and models*. [S.l.], Research Report, Computer Science Department, University of Florida, 1995.

SINNOTT, R.; MILLAR, C.; ASENOV, A. Supercomputing at work in the nanocmos electronics domain. *European Research Consortium for Informatics and Mathematics*, n. 74, p. 22–23, 2008.

SUBHLOK, J.; VENKATARAMAIAH, S.; SINGH, A. Characterizing nas benchmark performance on shared heterogeneous networks. *IEEE International Parallel and Distributed Processing Symposium*, p. 86–94, 2002.

SUN, Y.; WANG, J.; XU, Z. Architetural implications of the nas mg and ft parallel benchmarks. *Advances in Parallel and Distributed Computing*, p. 235–240, 1997.

TABE, T.; STOUT, Q. *The use of MPI communication library in the NAS parallel benchmarks*. [S.l.], 1999. Technical Report CSE-TR-386-99, University of Michigan, 1999.

TANEMBAUM, A. *Redes de Computadores*. 3ª. ed. [S.l.]: Pearson, 1997.

TANEMBAUM, A.; STEEN, M. van. *Distributed systems: Principles and Paradigms*. 4ª. ed. [S.l.]: Prentice Hall, 2002.

YELICK, K.; AL et. Parallel languages and compilers: Perspective from the titanium experience. *International Journal of High Performance Computing Applications*, v. 21, n. 3, p. 266–290, 2007.

ZAMANI, R. *Communication Characteristics of Message-Passing Applications, and Impact of RDMA on their Performance*. Tese (Doutorado), Kingston, Ontario, Canada, 2005.

ZAMANI, R.; AFSAHI, A. Communication characteristics of message-passing scientific and engineering applications. *International Conference on Parallel and Distributed Computing and Systems (PDCS)*, p. 644–649, 2005.

# *Anexo A:* namelist.input

&time_control
run_days = 0,
run_hours = 72,
run_minutes = 0,
run_seconds = 0,
start_year = 2008, 2008, 2001,
start_month = 05, 04, 06,
start_day = 29, 07, 11,
start_hour = 00, 00, 12,
start_minute = 00, 00, 00,
start_second = 00, 00, 00,
end_year = 2008, 2008, 2001,
end_month = 06, 04, 06,
end_day = 01, 10, 12,
end_hour = 00, 00, 12,
end_minute = 00, 00, 00,
end_second = 00, 00, 00,
interval_seconds = 10800
input_from_file = .true.,.false.,.false.,
history_interval = 180, 60, 60,
frames_per_outfile = 1000, 1000, 1000,
restart = .false.,
restart_interval = 5000,
io_form_history = 2
io_form_restart = 2
io_form_input = 2
io_form_boundary = 2
debug_level = 0
auxinput1_inname = “met_em.d⟨domain⟩.⟨date⟩”

auxinput1_inname =“wrf_real_input_em.d⟨domain⟩.⟨date⟩”

&domains
time_step = 75,
time_step_fract_num = 0,
time_step_fract_den = 1,
max_dom = 1,
s_we = 1, 1, 1,
e_we = 126, 112, 94,
s_sn = 1, 1, 1,
e_sn = 100, 97, 91,
s_vert = 1, 1, 1,
e_vert = 37, 28, 28,
num_metgrid_levels = 27
dx = 15000, 10000, 3333,
dy = 15000, 10000, 3333,
grid_id = 1, 2, 3,
parent_id = 0, 1, 2,
i_parent_start = 0, 31, 30,
j_parent_start = 0, 17, 30,
parent_grid_ratio = 1, 3, 3,
parent_time_step_ratio = 1, 3, 3,
feedback = 1,
smooth_option = 0
p_top_requested = 5000
interp_type = 1
lowest_lev_from_sfc = .false.
lagrange_order = 1
force_sfc_in_vinterp = 1
zap_close_levels = 500
sfcp_to_sfcp = .false.
adjust_heights = .false.
eta_levels = 1.000, 0.990, 0.978, 0.964, 0.946,
0.922, 0.894, 0.860, 0.817, 0.766,
0.707, 0.644, 0.576, 0.507, 0.444,

0.380, 0.324, 0.273, 0.228, 0.188,
0.152, 0.121, 0.093, 0.069, 0.048,
0.029, 0.014, 0.000,
eta_levels = 1.000, 0.993, 0.983, 0.970, 0.954,
0.934, 0.909, 0.880, 0.845, 0.807,
0.765, 0.719, 0.672, 0.622, 0.571,
0.520, 0.468, 0.420, 0.376, 0.335,
0.298, 0.263, 0.231, 0.202, 0.175,
0.150, 0.127, 0.106, 0.088, 0.070,
0.055, 0.040, 0.026, 0.013, 0.000

&physics
mp_physics = 8, 3, 3,
ra_lw_physics = 1, 1, 1,
ra_sw_physics = 1, 1, 1,
radt = 15, 30, 30,
sf_sfclay_physics = 2, 1, 1,
sf_surface_physics = 2, 1, 1,
bl_pbl_physics = 2, 1, 1,
bldt = 0, 0, 0,
cu_physics = 3, 1, 0,
cudt = 5, 5, 5,
isfflx = 1,
ifsnow = 0,
icloud = 1,
surface_input_source = 1,
num_soil_layers = 4,
ucmcall = 0,
mp_zero_out = 0,
maxiens = 1,
maxens = 3,
maxens2 = 3,
maxens3 = 16,
ensdim = 144,
/

```
&fdda
/

&dynamics
w_damping = 0,
diff_opt = 1,
km_opt = 4,
diff_6th_opt = 0,
diff_6th_factor = 0.12,
damp_opt = 0,
base_temp = 290.
zdamp = 5000., 5000., 5000.,
dampcoef = 0.01, 0.01, 0.01
khdif = 0, 0, 0,
kvdif = 0, 0, 0,
non_hydrostatic = .true., .true., .true.,
pd_moist = .false., .false., .false.,
pd_scalar = .false., .false., .false.,
/

&bdy_control
spec_bdy_width = 5,
spec_zone = 1,
relax_zone = 4,
specified = .true., .false.,.false.,
nested = .false., .true., .true.,
/

&grib2
/

&namelist_quilt
nio_tasks_per_group = 0,
nio_groups = 1,
```

# *Anexo B: Lista de Publicações*

**# Uma Abordagem para Composição de Clusters Eficientes na Execução do Modelo Numérico WRF de Previsão do Tempo**

*Autores:* Luiz Carlos Pinto, Luiz H. B. Tomazella, M. A. R. Dantas

*Evento:* IX Workshop em Sistemas Computacionais de Alto Desempenho - Simpósio em Sistemas Computacionais (**WSCAD-SSC**)

*Local:* Campo Grande, MS, Brasil

*Ano:* 2008

*CAPES Qualis CC:* Nacional C

**# An Experimental Study on How to Build Efficient Multi-Core Clusters for High Performance Computing**

*Autores:* Luiz Carlos Pinto, Luiz H. B. Tomazella, M. A. R. Dantas

*Evento:* IEEE 11th International Conference on Computational Science and Engineering (**CSE**)

*Local:* São Paulo, SP, Brasil

*Ano:* 2008

*CAPES Qualis CC:* Não avaliado

**BEST PAPER AWARD:** *to appear in special issue of the International Journal of Computational Science and Engineering (IJCSE) and International Journal of High Performance Computing and Networking (IJHPCN)*

**# The Impact of Interconnection Networks and Applications Granularity to Compound Cluster Configurations**

*Autores:* Luiz Carlos Pinto, Rodrigo P. Mendonça, M. A. R. Dantas

*Evento:* IEEE 13th Symposium on Computers and Communications (**ISCC**)

*Local:* Marrakech, Marrocos

*Ano:* 2008

*CAPES Qualis CC:* Internacional A

**# Building Efficient Multi-Core Clusters for High Performance Computing**

*Autores:* Luiz Carlos Pinto, Luiz H. B. Tomazella, M. A. R. Dantas

*Evento:* IEEE 13th Symposium on Computers and Communications (**ISCC**)

*Local:* Marrakech, Marrocos

*Ano:* 2008

*CAPES Qualis CC:* Internacional A

**# Análise da relação entre rede de interconexão e granularidade de aplicações para a formação eficiente de agregados de computadores**

*Autores:* Luiz Carlos Pinto, Rodrigo P. Mendonça, M. A. R. Dantas

*Evento:* III Congresso Sul Catarinense de Computação (**Sulcomp**)

*Local:* Criciúma, SC, Brasil

*Ano:* 2007

*CAPES Qualis CC:* Não avaliado

**# Impacto da rede de interconexão para a formação eficiente de agregados de computadores**

*Autores:* Luiz Carlos Pinto, Rodrigo P. Mendonça, M. A. R. Dantas

*Evento:* 5a. Escola Regional de Redes de Computadores (**ERRC**)

*Local:* Santa Maria, RS, Brasil

*Ano:* 2007

*CAPES Qualis CC:* Não avaliado