FRANCISCO PEREIRA JUNIOR

SELEÇÃO DE VARIÁVEIS E CARACTERÍSTICAS COMO APLICAÇÃO PARALELA PARA *CLUSTER* MPI

MARINGÁ
2006

FRANCISCO PEREIRA JUNIOR

SELEÇÃO DE VARIÁVEIS E CARACTERÍSTICAS COMO APLICAÇÃO PARALELA PARA *CLUSTER* MPI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. João Angelo Martini

MARINGÁ
2006

Dados Internacionais de Catalogação-na-Publicação (CIP)
(Biblioteca Central - UEM, Maringá – PR., Brasil)

P436s

Pereira Junior, Francisco
Seleção de variáveis e características como aplicação paralela para *Cluster* MPI / Francisco Pereira Junior. - Maringá, PR : [s.n.], 2006.
118 f. : il. color.

Orientador : Prof. Dr. João Angelo Martini.
Dissertação (mestrado) - Universidade Estadual de Maringá. Programa de Pós-graduação em Ciência da Computação, 2006.

1. Computação paralela - *Cluster* MPI - Seleção de variáveis e características. 2. Computação paralela - Cluster MPI - Seleção de variáveis e características - Processamento de alto desempenho. 3. Algoritmo - Otimização. I. Universidade Estadual de Maringá. Programa de Pós-graduação em Ciência da Computação. II. Título.

CDD 21.ed.004.3682

FRANCISCO PEREIRA JUNIOR

SELEÇÃO DE VARIÁVEIS E CARACTERÍSTICAS COMO APLICAÇÃO PARALELA PARA *CLUSTER* MPI

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Estadual de Maringá, como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Aprovado em 27/10/2006.

BANCA EXAMINADORA

Prof. Dr. João Angelo Martini (Orientador)
Universidade Estadual de Maringá (PCC - UEM)

Prof. Dr. Ronaldo Augusto de Lara Gonçalves
Universidade Estadual de Maringá (PCC - UEM)

Prof. Dr. Márcio Augusto de Souza
Universidade Estadual de Ponta Grossa (DEINFO - UEPG)

Dedicatória

Dedico este trabalho aos meus pais, Ivone e Francisco, e a minha namorada Josane.

Agradecimentos

Registro aqui os meus agradecimentos, primeiramente a Deus, pela sabedoria, proteção, amor e presença;

À minha família, especialmente aos meus pais, Ivone e Francisco, pelo exemplo de vida e de dignidade; A minha irmã, Fernanda, e meu cunhado, Fernando, pela torcida; à minha namorada, Josane, pelo apoio, incentivo, carinho e principalmente sua compreensão na minha constante ausência;

Ao meu orientador, Prof. Dr. João Angelo Martini, por ter acreditado em meu trabalho, pela dedicação, paciência e ensinamentos;

À Universidade Estadual de Maringá e ao programa de Pós-graduação em Ciência da Computação (PCC), pela acolhida e infra-estrutura; à Maria Inês Davanço (secretária do PCC) por sua atenção, simpatia e presteza; ao Prof. Dr. Ronaldo Augusto de Lara Gonçalves pelo incentivo e atenção;

Aos amigos, André Schwerz (Andrezão), Rafael Liberato (Rafa), Igor Wiese (Igão) e Fábio Gorino, companheiros no estudo, nas discussões e no lazer, devo muito a vocês;

Aos professores da Coordenação de Informática da UTFPR (Adriana, Adriane, Alessandro, Alexandre, André, Antônio, Elias, Eidy, Fabrício, Gabriel, Guilherme, Jean, Lígia, Luciano, Maurício, Pozza, Ricardo, Vanderley), que entenderam que eu não poderia contribuir o quanto era necessário durante o desenvolvimento deste trabalho. Especialmente agradeço aos professores: Fabrício e Vanderley, por confiar e me recomendar para este mestrado; Antonio e Gabriel, pela disposição e torcida; também agradeço ao professor Antonio Airton Carneiro de Freitas, pelas idéias e conhecimento compartilhado;

Aos meus alunos (estagiários e acima de tudo amigos) Ana Paula Chaves, Jocimara Segantini Ferranti, Marlos Henrique dos Santos e Henrique Shishido pelo apoio e comprometimento;

enfim... meu amor, papai, mamãe, amigos, obrigado a todos vocês... CONSEGUIMOS.

"Quanto maior a esfera de conhecimento maior a
superfície de contato com o desconhecido."
(autor desconhecido)

Resumo

A metodologia de seleção de variáveis e características aplicada à modelagem de sistemas objetiva a redução da quantidade de variáveis de entrada de um modelo. Esta análise se justifica quando o número de variáveis chega na ordem de dezenas, centenas ou milhares, ou o conjunto de dados representado por elas alcança grandes dimensões. A resolução computacional desta metodologia, baseada em cálculos complexos e repetitivos, demandam algoritmos com alto custo de processamento. Neste escopo, no intuito de diminuir o tempo total consumido originalmente pelos algoritmos, e também maximizar o uso dos recursos de processamento, foram implementados 3 algoritmos seriais e 2 paralelos. Para uma melhor avaliação do comportamento dos algoritmos pelas abordagens utilizadas, foram realizadas variações no ambiente de execução (quantidade de processadores) e no conjunto de dados. As versões seriais foram evolutivamente melhoradas até uma versão considerada ótima, com reduções de até 56,98% de tempo. Este melhor algoritmo serial foi usado como parâmetro para a construção dos algoritmos paralelos. Os algoritmos paralelos destacam-se pela inclusão das primitivas da biblioteca de comunicação MPI, pelas abordagens utilizadas no processo de divisão e distribuição de tarefas, e pela execução em dois tipos diferentes de *cluster*. A abordagem com subtarefa de tamanho fixo atenuou significativamente a diferença de desempenho entre os *clusters* homogêneo e heterogêneo. Técnicas de otimização, reestruturação de algoritmo e computação paralela baseada em *cluster* MPI contribuíram para diminuir o tempo total de execução da aplicação em aproximadamente 93%. Com alto índice de processamento útil dos nós de processamento, e os resultados obtidos neste experimento real, comprova-se a eficácia destas técnicas.

Palavras-chave: Computação Paralela, Cluster, MPI, Seleção de Variáveis e Características.

Abstract

The methodology of variables and features selection applied to the modeling of systems aims at the reduction of the model entrance variables amount. This analysis is justified when the number of variables reaches the sets of tens, hundreds or thousands, or the data set represented by them reaches great dimensions. The computational resolution of this methodology, based on complex and repetitive calculations, demand algorithms with high processing cost. In this target, intentending to reduce the total time consumed originally by the algorithms, as well as to maximize the use of the processing resources, 3 serial algorithms and 2 parallels have been implemented. For a better evaluation of the algorithms behavior for the used approaches, there have been made variations in the execution environment (amount of processors) and in the data set. The serial versions were gradually improved up to a version considered excellent, with reductions of up to 56,98% of time. This better serial algorithm was used as parameter for the construction of the parallel algorithms. The parallel algorithms outstand for the inclusion of the primitive of the communication library MPI, for the approaches used in the division and distribution process of tasks, and for the execution in two different types of cluster. The approach with sub-task of fixed size significantly attenuated the difference in performance between homogeneous and heterogeneous clusters. Optimization techniques, algorithm restructuring and parallel computation based in cluster MPI, have contributed to reduce the total time of the application execution in approximately 93%. With the processing nodes high index of useful processing, and the results obtained in this real experiment, the effectiveness of these techniques could be proved.

Keywords: Parallel Computation, Cluster, MPI, Variables and Features Selection.

Lista de Figuras

Lista de Tabelas

Listagem

Lista de Siglas e Abreviaturas

APDF	Algoritmo Paralelo Divisão Fixa
APDI	Algoritmo Paralelo Divisão Implícita
APS	*American Phytopathological Society*
BOVESPA	Bolsa de Valores do Estado de São Paulo
CENAPAD	Centro Nacional de Processamento de Alto Desempenho
CLUEM	*Cluster* Universidade Estadual de Maringá
CLUTF	*Cluster* Universidade Tecnológica Federal
CONAB	Companhia Nacional de Abastecimento
COW	*Clusters of Workstation*
DSM	*Distributed Shared-Memory*
EMBRAPA	Empresa Brasileira de Pesquisa Agropecuária
IA	Inteligência Artificial
INMET	Instituto Nacional de Meteorologia
MAPA	Ministério da Agricultura, Pecuária e Abastecimento
MIMD	*Multiple Instruction Multiple Data*
MISD	*Multiple Instruction Single Data*
MPI	*Message Passing Interface*
MPP	*Massive Parallel Processing*
OMM	Organização Meteorológica Mundial
OSCAR	*Open Source Cluster Application Resources*
PVM	*Parallel Virtual Machine*
RNA	Redes Neurais Artificiais
SIMD	*Single Instruction Multiple Data*
SIMEPAR	Sistema Meteorológico do Paraná
SISD	*Single Instruction Single Data*
VLSI	*Very Large Scale Integration*
VS1	Versão Serial Original
VS2	Versão Serial Organizada
VS3	Versão Serial Otimizada

Sumário

1 Introdução

A otimização que vem ocorrendo nas últimas décadas no processo de fabricação dos microprocessadores, tem barateado o seu custo. Essa otimização permite a construção de microprocessadores cada vez menores e, ao mesmo tempo, possibilita o surgimento de microprocessadores mais rápidos. Os grandes avanços em hardware como: válvulas, transistores, circuitos integrados, *Very Large Scale Integration* (VLSI), *pipeline*, hierarquia de memória cache, *hyper-threading*, *dual core*, proporcionaram maior desempenho e elevaram a computação como um meio para solucionar problemas outrora inimagináveis.

Desde os primeiros computadores, a evolução dos recursos computacionais tenta acompanhar a crescente complexidade dos problemas científicos. Mesmo assim, esta evolução é ainda insuficiente para solucionar alguns problemas que são provenientes de áreas distintas, como: engenharias em geral, biologia molecular, computação gráfica, previsão meteorológica, inteligência artificial, entre outras. Os ditos problemas grandes ou complexos chegam a consumir horas ou dias de processamento das arquiteturas uniprocessadas convencionais.

Impulsionada pela grande demanda de computação e por restrições físicas como espaço, quantidade de transistores, dissipação de calor e também velocidade finita da luz, que dificultam o aumento de velocidade em um único processador, a computação paralela acena como alternativa para amenizar estes problemas. Assim, o processamento ou a computação paralela implica na divisão de uma aplicação em tarefas menores, de maneira que ela possa ser executada

cooperativamente por vários elementos de processamento (EPs).

Dentre as arquiteturas específicas para execução de aplicações paralelas, os *clusters* apresentam uma das melhores relações custo/benefício. Por meio das bibliotecas de comunicação, eles tornam-se ambientes promissores na solução dos problemas de grande escala.

Há diversas áreas de aplicação que são beneficiadas pela computação paralela, como a Inteligência Artificial (IA) e suas Redes Neurais Artificiais (RNAs), que se destacam pelas suas peculiaridades e também pela possibilidade de previsão de valores, com base no conhecimento adquirido em informações do passado. A união da área de IA, juntamente com o processamento paralelo, motivou a criação de um modelo de previsão micro-climática em parceria com a Empresa Brasileira de Pesquisa Agropecuária (EMBRAPA Soja). O intuito deste modelo é prever a umidade relativa do ar, que é fator predisponente à Ferrugem Asiática, fitopatologia que ataca os cultivares da Soja, acarrentando enormes prejuízos. Desta forma, o conhecimento das condições meteorológicas influencia diretamente na tomada de decisões e também no planejamento de operações cotidianas das atividades agrícolas. Um estudo do impacto sócio-econômico da Ferrugem Asiática no agronegócio brasileiro foi elaborado e está disponível no apêndice A, e ele justifica a análise das séries temporais meteorológicas usadas neste trabalho.

Para que as RNAs possam prever valores, elas passam por um processo de treinamento ou aprendizado, que, para alcançar melhores resultados, necessita de um conjunto adequado de dados de entrada, composto de variáveis qualificadas. Teoricamente, um grande número de variáveis e características deveria prover maior detalhamento ao modelo, mas na prática é comum aparecer muitas características irrelevantes ou redundantes em termos de conteúdo informacional. Então, encontrar um subconjunto adequado (ótimo) de variáveis alivia em grande parte a necessidade demasiada de processamento e também facilita o

aprendizado das RNAs.

O processo de seleção das variáveis também apresenta alto custo computacional para certos domínios. Assim, diante deste contexto, o presente trabalho busca propor estratégias de redução do tempo de execução de uma metodologia de seleção de variáveis e características, possibilitando maximizar o uso dos recursos disponíveis na arquitetura. Quando se busca alto desempenho, as minúcias da programação precisam ser atentamente analisadas. Para códigos repetitivos que demandam horas ou até dias de processamento, a economia de uma simples instrução, que é repetida milhares de vezes, implica na antecipação do seu tempo final de execução. Para isto, uma versão serial do algoritmo da metodologia de seleção foi implementado na Linguagem C e evolutivamente otimizado, no intuito de reduzir a quantidade de instruções desnecessárias.

Em seguida, o algoritmo foi paralelizado e executado em dois tipos diferentes de *clusters*, um homogêneo e outro heterogêneo. A distribuição de tarefas entre os nós de processamento pode ser determinante na eficiência de um algoritmo. Assim, duas abordagens foram utilizadas: a primeira faz uma divisão implícita de forma igualitária; já a segunda divide a tarefa em subtarefas de tamanho fixo. Busca-se, com isso, analisar e avaliar o comportamento de uma aplicação real de computação paralela em um *cluster* MPI.

1.1 Objetivos

O objetivo principal deste trabalho é reduzir o tempo de processamento de uma metodologia de seleção de variáveis e características. Para isso, o trabalho foi fundamentado na otimização algorítmica e na utilização de computação paralela baseada em *cluster* MPI. Como objetivos secundários têm-se:

- implementar a metodologia para uma base de dados real e relevante;

- apresentar a diversidade de fatores influenciadores no quesito consumo computacional e fornecer estratégias diferenciadas para a resolução de aplicações reais;
- analisar o desempenho e o comportamento do algoritmo de seleção de variáveis e características em um ambiente de computação paralela, alterando o número de nós de processamento envolvidos na computação e o tamanho do problema tratado;
- analisar a diferença de desempenho de aplicações executadas em *cluster* homogêneos e heterogêneos;
- apresentar soluções para melhor aproveitar os recursos computacionais disponíveis.

1.2 Organização do Trabalho

Este trabalho está organizado em capítulos, cujos conteúdos são estruturados na tentativa de conduzir de forma clara e concisa diversos assuntos pertinentes ao entendimento da proposta principal aqui exposta.

O segundo capítulo, **Arquiteturas Paralelas**, traz uma sucinta revisão histórica dos computadores, apontando os principais marcos e dando ênfase às arquiteturas paralelas. Descreve também as razões para o estabelecimento de arquiteturas de alto desempenho, as justificativas para o uso de *cluster* de computadores, algumas classificações de arquiteturas, incluindo a taxonomia de Flynn e listam-se áreas para aplicação dos *clusters*.

Dedica-se o terceiro capítulo, **Programação Paralela**, à discussão de assuntos relacionados ao desenvolvimento de sistemas paralelos. Inicialmente, descreve-se em detalhes algumas técnicas clássicas de análise e otimização de algoritmos e, em seguida, expõe-se as ferramentas necessárias para a obtenção

do paralelismo.

O quarto capítulo, **Seleção de Variáveis e Características**, apresenta de uma forma generalizada os conceitos e características básicos das Redes Neurais, passando por uma explanação sobre as séries temporais e posteriormente, uma discussão acerca da metodologia que intitula este capítulo.

No quinto capítulo, **Experimentos e Resultados**, descrevem-se em detalhes os algoritmos, as técnicas e os conjuntos de dados utilizados na resolução computacional do problema do filtro de variáveis e apresentam-se os resultados tanto da avaliação da arquitetura quanto da metodologia.

Finalmente, no sexto e último capítulo, **Considerações Finais**, são colocadas as conclusões extraídas deste trabalho e apresentadas as principais contribuições, sugerindo-se temas a serem desenvolvidos em trabalhos futuros.

2 *Arquiteturas Paralelas*

Vislumbradas pela evolução dos computadores, diversas áreas do conhecimento elegeram a computação como meio para equacionar seus problemas, que outrora eram inviáveis devido a quantidade de cálculos exigidos. A elaboração de softwares para representar fielmente os experimentos, em geral, eleva a complexidade dos algoritmos ou exige um conjunto de dados muito grande e isto dita a quantidade de operações necessárias para solucionar computacionalmente o problema. A crescente necessidade por poder computacional vem gerando uma demanda por processadores cada vez mais rápidos, e a computação paralela passa a ser uma alternativa essencial para muitas áreas relacionadas à pesquisa.

Este capítulo discute sucintamente a evolução histórica dos computadores, abordando arquiteturas paralelas, que são foco deste trabalho, algumas classificações de arquiteturas, e a demanda de processamento gerada pelos softwares, principalmente os científicos.

2.1 Breve Histórico da Evolução dos Computadores

O projeto do matemático Howard Aiken, em 1943, abriu caminho para o surgimento dos primeiros grandes computadores. Nomeado de MARK I, o computador realizava, em um dia, cálculos que antes consumiriam seis meses. Juntamente com o início da Segunda Guerra Mundial, Alan Turing uti-

lizou seu projeto Colossus para decifrar mensagens interceptadas de exércitos inimigos. Colossus, primeiro computador a utilizar válvulas, procurava coincidências entre as mensagens cifradas e os códigos conhecidos, em uma taxa de 5.000 caracteres por segundo.

Com um projeto iniciado em 1942 e finalizado em 1945, coincidindo com o final da Segunda Guerra Mundial, o *Eletronic Numerical Integrator and Calculator* (ENIAC), projeto de John Mauchly e Presper Eckert com aporte financeiro do exército americano, usando válvulas eletrônicas, alcançou *clock* de 100 KHz. O seu sucessor, *Electronic Discrete Variable Automatic Computer* (EDVAC), foi modelo para a construção de outros computadores. Planejado por John von Neumann, principal ícone do desenvolvimento de computadores do pós-guerra, a máquina *Institute of Advanced Studies* (IAS) já processava informações de forma binária e armazenava tanto dados quanto o programa em uma memória interna. Os complexos cálculos de balística passaram a ser executados em 30 segundos, contra 12 horas dos métodos anteriores. Findando a primeira geração de computadores a *International Business Machines* (IBM®) lançou o modelo 704 que contemplava componentes de hardware exclusivos para dados em ponto-flutuante.

A geração posterior dos computadores é caracterizada pela substituição das válvulas pelos transistores e o início do uso de computadores para fins comerciais. Em 1956, surgiu o primeiro computador transistorizado. Inventado em 1948, nos Laboratórios da Bell®, o transistor era mais rápido e mais confiável. Com uma dimensão aproximada de 100 vezes menor que uma válvula, não precisava de tempo para aquecimento e consumia menos energia. Alguns computadores desta geração eram programados em linguagem montadora e já faziam cálculos em microssegundos (milionésimos de segundos). Fabricados pela *Digital Electronic Computer* (DEC®), surgem em 1961 os computadores da família *Programmed Data Processor* (PDP), dando origem à indústria

dos chamados minicomputadores. O modelo 6600 da companhia *Control Data Corporation* (CDC), em 1964, foi o primeiro expoente dos computadores que poderiam executar instruções em paralelo. Exclusivamente projetado para processamento científico, se bem programado, poderia executar 10 instruções ao mesmo tempo [Tanenbaum 2001].

Os circuitos integrados, associação de vários transistores em um mesmo *chip*, e a miniaturização de outros componentes eletrônicos, propiciaram um novo avanço e com eles surgiram os computadores de terceira geração. Ainda mais confiáveis e menores, pela proximidade dos componentes, os circuitos integrados tornaram os computadores mais compactos, rápidos, baratos e com baixo consumo de energia. Os representantes mais importantes foram os modelos da família System/360, fabricados pela IBM®, projetados tanto para aplicações científicas quanto para aplicações comerciais. Uma outra inovação desta geração foi a implantação do conceito de multiprogramação e um grande espaço de endereçamento de memória.

A próxima geração dos computadores foi marcada pela técnica VLSI, que possibilitou a inclusão de dezenas de milhares e, posteriormente, centenas de milhões de transistores em um único *chip*. Surge então o microprocessador e com ele a era da computação pessoal. Assim, os fabricantes dividiram esforços na fabricação de supercomputadores e computadores pessoais. Os supercomputadores eram dominantes na área científica, que é baseada em cálculos complexos, e nas empresas, com vários terminais acoplados executando aplicações comerciais. Os *Personal Computers* (PCs) alavancaram as aplicações integradas como editores de texto, planilhas eletrônicas, manipuladores de imagens, entre outros.

Ao longo do tempo, uma série de técnicas e recursos sofisticados da supercomputação foram incorporados, sucessivamente, aos microprocessadores utilizados em computadores pessoais e estações de trabalho. Acreditava-se, na

década de 80, que o aumento do desempenho de uma máquina dependia apenas e tão somente da criação de processadores mais rápidos e eficientes, porém havia outros empecilhos, tais como:

- alto custo e longos prazos, na pesquisa e projeto de novas arquiteturas de hardware;
- necessidade de se trabalhar com tecnologia de ponta;
- limitação e dependência de outros componentes como memória e dispositivos de armazenamento;
- uma esperada saturação da tecnologia, ou seja, em um determinado momento a tecnologia aplicada aos processadores não conseguirá miniaturizar e colocar tantos transistores quanto necessários para aumentar o poder de processamento do *chip* em questão.

A computação passou por um processo evolutivo intenso, em termos de hardware e software, a fim de proporcionar maior desempenho e ampliar o leque de aplicações que podem ser computacionalmente resolvidas de maneira eficiente. Diversas técnicas e tecnologias foram propostas para se chegar o mais próximo possível do estado da arte e construir componentes complexos em *chips* cada vez menores [Stallings 2002]. Os grandes avanços ocorridos no processo de fabricação de microprocessadores atuais estão baseados no aumento do *pipeline*, melhor organização da hierarquia de memória cache, *hyper-threading*, *dual core*, dentre outros.

Estas inovações trazem mais eficiência e velocidade aos microprocessadores, oportunizando aos softwares, principalmente os de propósito geral, um excelente ambiente de execução. Entretanto, essa evolução não é suficiente para suprir a exigência de alguns softwares específicos que consomem facilmente os recursos computacionais oferecidos pelas máquinas convencionais (modelo de

von Neumann). O tratamento de grandes problemas, principalmente das áreas científicas como: processamento de imagens e sinais, inteligência artificial, mineração de dados, física molecular, previsão meteorológica, estudos sísmicos, pesquisas militares, aerodinâmica de aviões e mísseis, exploração de petróleo, diagnóstico médico, entre outras, demandam o uso de máquinas com múltiplos processadores ou ainda supercomputadores proprietários, fornecidos por grandes empresas. Ambas as soluções oferecem grande desempenho, mas esbarram no elevado custo e na baixa escalabilidade.

Baseado nas dificuldades apresentadas, Donald Becker e Thomas Sterling, em 1993, na NASA, tiveram a idéia de juntar computadores pessoais para um trabalho cooperativo [Sterling et al. 2003] [Bell and Gray 2002]. Com o grande sucesso desta experiência, batizada de *cluster*, conseguiram alcançar níveis de processamento equivalentes aos de supercomputadores da época, mas por uma fração do seu preço. Segundo [Franco 2004], *cluster* é um conjunto de máquinas ou nós, normalmente sem necessidade de periféricos, interligadas via rede, que trabalham em conjunto, trocando informações entre si para solucionar uma determinada tarefa.

Como já foi mencionado, a estrutura do *cluster* apresenta vantagens competitivas em relação a outros ambientes multiprocessados, tanto de memória compartilhada quanto distribuídas, além do custo, que em geral é menor, oferece uma maior flexibilidade com um proporcional poder de processamento [Stallings 2002]. Isto é possível pela utilização de componentes padrão de mercado, ligados por uma rede tradicional de interconexão, que, através de softwares livres e programação paralela, fornecem um ambiente propício para a comunicação e sincronização das tarefas em execução [Sterling et al. 2003].

Assim, o aprimoramento dos computadores pessoais, a redução dos preços e a excelente relação custo/benefício apresentada pelos *clusters* tiveram um impacto notório nos centros de pesquisas, que, novamente, vislumbraram boas

oportunidades para solucionar seus problemas de grande escala.

2.2 Classificação de Arquiteturas

Existem diversas maneiras de classificar uma arquitetura de computadores, e a proposta mais comumente utilizada até hoje foi formalizada por Flynn, em 1972. A chamada Taxonomia de Flynn organiza os computadores em classes e define que o processo computacional é uma relação entre fluxos de instruções e fluxos de dados [Flynn 1972]. Segundo [Tanenbaum 2001], "[...] um fluxo de instruções corresponde a um contador de programa, um sistema com *n* CPUs possui *n* contadores de programa, e conseqüentemente *n* fluxos de instruções". O autor ainda conceitua um fluxo de dados como um conjunto de operandos, ou variáveis. As quatro combinações possíveis da relação são descritas a seguir:

- *Single Instruction Single Data* (SISD): caracterizada por possuir fluxo único de instruções e fluxo único de dados, corresponde ao modelo clássico de von Neumann. Um processador (elemento de processamento) executa, seqüencialmente, um conjunto de instruções sobre um conjunto de dados, realizando uma operação de cada vez. Os microcomputadores uniprocessados atuais estão classificados nesta categoria.

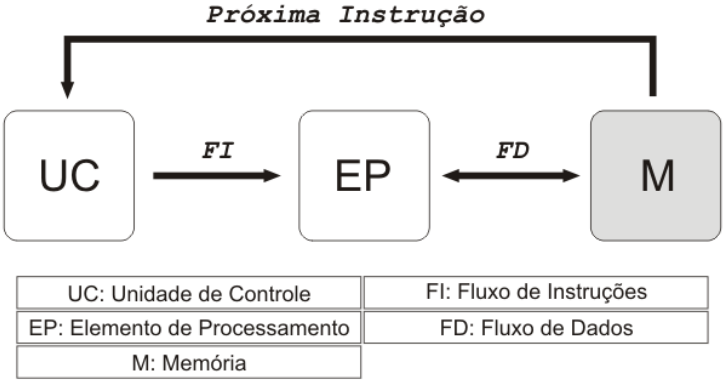

Figura 2.1: Arquitetura SISD.

- *Single Instruction Multiple Data* (SIMD): caracterizada por possuir fluxo

único de instruções e múltiplos fluxos de dados, envolve inúmeros processadores escravos sob domínio de uma única unidade de controle. São exemplos desta classe os supercomputadores vetoriais e os processadores matriciais, usados principalmente em processamento científico.

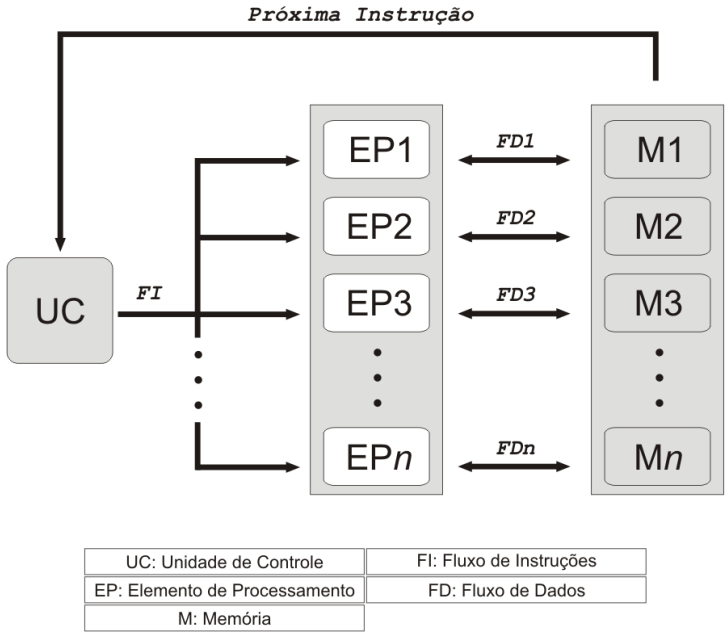

Figura 2.2: Arquitetura SIMD.

- *Multiple Instruction Single Data* (MISD): caracterizada por possuir múltiplos fluxos de instruções e um único fluxo de dados, envolve diversos processadores, executando diferentes instruções em um único conjunto de dados. Não há um exemplar para esta categoria, sendo considerada apenas um modelo teórico.

- *Multiple Instruction Multiple Data* (MIMD): caracterizada por possuir múltiplos fluxos de instruções e múltiplos fluxos de dados, envolve múltiplos processadores, executando diferentes instruções em diferentes conjuntos de dados, de maneira independente. É modelo para a maioria dos computadores paralelos.

Sabe-se que a comunicação e o sincronismo dos processos em execução são diretamente relacionados à organização da memória. Sendo assim, a classe

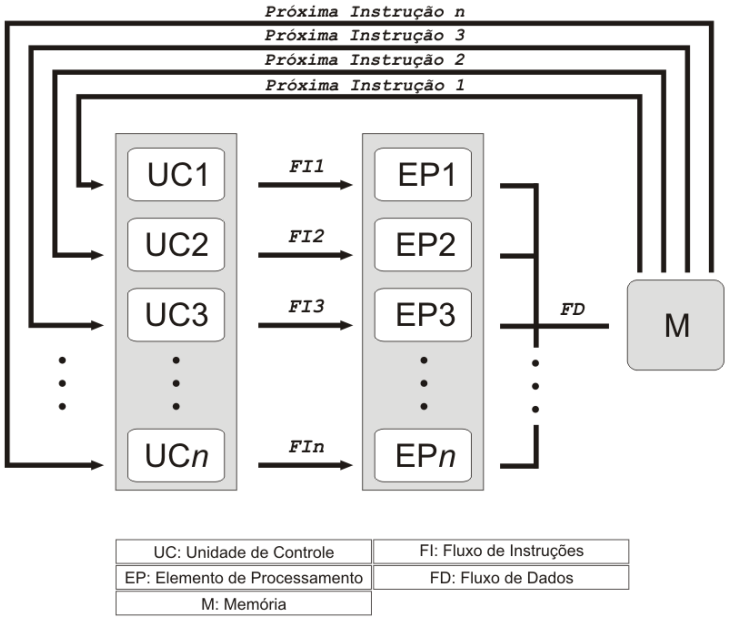

Figura 2.3: Arquitetura MISD.

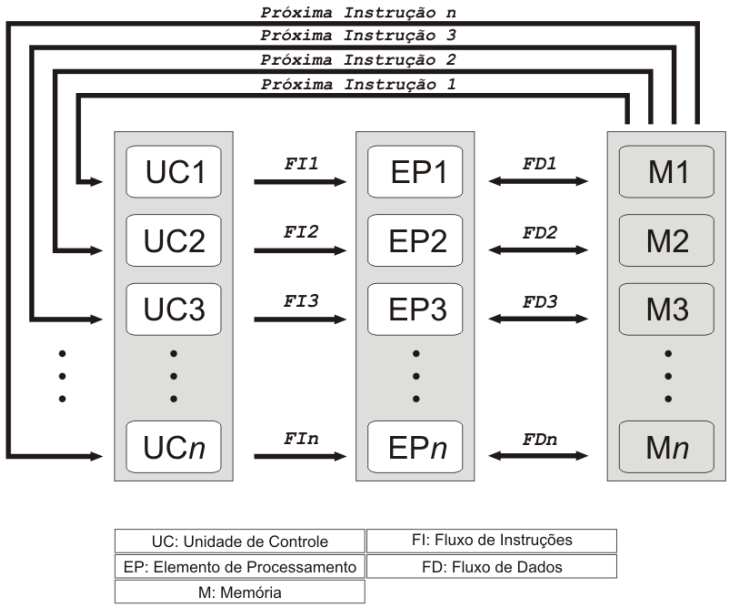

Figura 2.4: Arquitetura MIMD.

MIMD, proposta por Flynn, pode ainda, segundo [Tanenbaum 2001], ser subdividida em duas novas categorias, como mostra a Figura 2.5.

Os multiprocessadores (arquitetura de memória compartilhada ou centralizada) são chamados de sistemas fortemente acoplados, possuem memória global e única, que é compartilhada por todos os processadores para realizar a comunicação entre os processos. É necessária a utilização de técnicas especiais

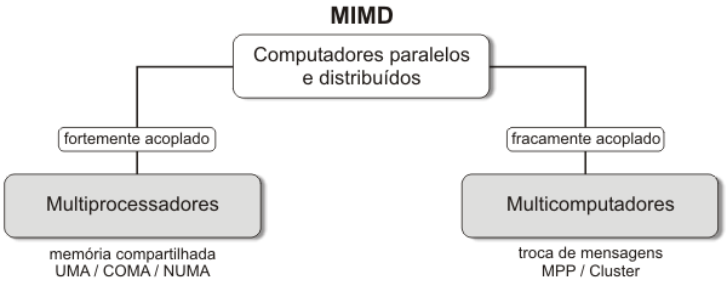

Figura 2.5: Subdivisão da classe MIMD.

para que este compartilhamento não se torne o gargalo da arquitetura.

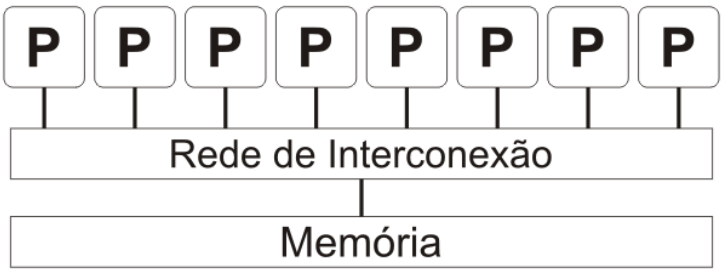

Figura 2.6: Multiprocessadores.

Os multicomputadores (arquitetura de memória distribuída) são chamados de sistemas fracamente acoplados, possuem memória distribuída, e os processos têm apenas acesso à memória local. Comunicam-se por troca de mensagens através de uma rede de interconexão e esta troca é caracterizada como sendo uma transferência explícita de dados entre os processadores.

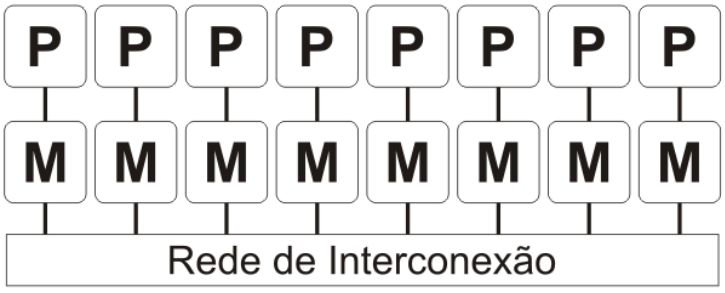

Figura 2.7: Multicomputadores.

[Patterson and Hennessy 2005] ainda adiciona uma categoria híbrida, chamada (arquitetura de memória compartilhada distribuída) ou *distributed shared-memory* (DSM), na qual a comunicação é feita por meio de um espaço de endereçamento compartilhado. Isto é, as memórias fisicamente separadas podem ser

endereçadas como um único espaço de endereços logicamente compartilhado.

2.3 Aplicações de *Cluster*

O uso de *clusters* vem sendo explorado nas mais diferentes comunidades, devido a tal solução oferecer um equilíbrio entre o desempenho desejado e o custo do sistema [Bell and Gray 2002]. Os *clusters*, porém, não são utilizados somente em sistemas em que se busca alto desempenho. Sua aplicabilidade estende-se a outras áreas e pode ser dividida, quanto à sua funcionalidade, em quatro categorias básicas [Baker 2000]: Alta Disponibilidade, Balanceamento de Carga, Computação Distribuída e Computação de Alto Desempenho.

- A primeira categoria, *cluster* de Alta Disponibilidade, deve ser utilizada quando o sistema precisa estar disponível o maior tempo possível, ou seja, responder às solicitações recebidas em 99,9% dos casos, ou mais. A forma para se atingir alta disponibilidade é através do uso de técnicas de redundância, em que se busca eliminar os pontos únicos de falha do sistema (*Single Point Of Failure* (SPOF)). Para isso, criam-se réplicas dos serviços ou se duplicam componentes de hardware como interfaces de rede, discos rígidos, fontes de alimentação, entre outros [Jalote 1994].

- A nomenclatura de *cluster* para Balanceamento de Carga é atribuída quando um serviço é requisitado por muitos usuários e existem diversas máquinas responsáveis por responder pelo mesmo serviço. Para garantir que todas as máquinas recebam uma carga de trabalho equivalente, *daemons* (programas de monitoramento) estão ativos em cada servidor, monitorando a utilização da CPU e outros recursos. Com o resultado da monitoração, os servidores são capazes de realizar uma redistribuição da carga de tarefas, balanceando a utilização de todos os nós.

- *Cluster* para Computação Distribuída, também denominado *Grid*, é um ambiente dinâmico, em que um conjunto de nós de processamento pode ser adicionado ou removido sempre que necessário. Portanto, o *Grid* Computacional é uma rede de computadores geograficamente distribuída na qual o indivíduo, que pode ser uma única máquina ou um outro *cluster*, conecta-se para dispor seu poder computacional.

- O *cluster* para Computação de Alto Desempenho é o mais comum entre as comunidades científicas que têm tarefas típicas que exigem alto poder de processamento. Sua função é conceitualmente simples: dado um problema complexo e identificado como "paralelizável", um servidor (mestre) deve ser responsável por dividir este problema em inúmeros pedaços a serem processados paralelamente em nós escravos (nós dedicados ao processamento). Assim que cada nó encontrar a sua solução, ele a envia ao servidor para que a solução completa do problema possa ser remontada. O grande desafio é encontrar uma solução arquitetural universal para esse tipo de problema, visto que cada caso exige um modo de divisão próprio e uma implementação de software específica.

Outra classificação possível a um *cluster* é quanto a sua estrutura: pilhas de PCs (*Beowulf*[1]) ou *Clusters of Workstation* (COWs). A diferença básica está na utilização direta ou não dos nós, ou seja, o nó é ou não dedicado ao processamento. A arquitetura *Beowulf* utiliza todos os seus nós exclusivamente para computação e os COW's são estações de trabalhos que em momentos de ociosidade são utilizadas para adicionar processamento ao sistema.

Um *cluster* pode ainda ser chamado de homogêneo ou heterogêneo. O *cluster* é dito homogêneo quando todos os seus nós são formados por máquinas semelhantes e utilizam uma única tecnologia de comunicação entre os nós. Já

[1] *Beowulfs* são *clusters* construídos com componentes padrão de mercado e voltados exclusivamente para processamento paralelo de dados. Utilizam, na sua grande maioria, sistema operacional Linux.

um *cluster* heterogêneo pode ser formado por diferentes máquinas e diferentes tecnologias de interconexão.

3 *Programação Paralela*

A arquitetura de computadores, discutida no capítulo 2, é apenas uma fração da área computacional. Os sistemas operacionais e os softwares aplicativos complementam o ambiente e pressionam o processo evolutivo do hardware. Mesmo para programação seqüencial, que é largamente utilizada, o desenvolvimento de programas de computador é custoso e demanda tempo. Desenvolver sistemas de alto desempenho baseados em paralelismo é ainda mais complicado, visto que, necessitam de uma arquitetura especial de hardware e software.

> [...] paralelismo é uma estratégia utilizada em computação para obter-se, mais rapidamente, resultados de tarefas grandes e complexas. Segundo esta estratégia, uma tarefa grande pode ser dividida em várias tarefas pequenas, que serão distribuídas entre vários processadores e executadas simultaneamente [...] [CENAPAD-SP 2006].

Os supercomputadores, *Massive Parallel Processing* (MPP), e os *clusters* precisam ser programados, e softwares específicos devem ser construídos para usufruir de todos os benefícios da arquitetura paralela e alcançar o objetivo proposto de diminuir o tempo total de execução de uma aplicação. A troca do paradigma seqüencial pelo paralelo não é intuitiva aos programadores, e a já conhecida estrutura da arquitetura de von Neumann, em que um único processador pode acessar certa quantidade de memória é derrubada. Na computação paralela é como se várias instâncias do paradigma seqüencial executassem ao mesmo tempo, em um trabalho cooperativo sobre o mesmo problema.

Muitos são os fatores que podem influenciar positiva e negativamente na eficiência de um software, e este capítulo discute as características fundamentais

referentes à programação paralela, tais como: técnicas para análise e otimização de algoritmos, ferramentas para a obtenção de paralelismo em linguagens de programação, ambientes de troca de mensagens, divisão de tarefas, análise e medidas de desempenho e ferramentas para gerenciamento de ambientes paralelos.

3.1 Análise e Otimização de Algoritmos

> [...] a principal razão para a computação paralela é reduzir o tempo de execução, não havendo qualquer sentido em usar um conjunto de processadores antes de executar a aplicação tão rápido quanto possível em um único processador. Portanto, o primeiro passo na paralelização de um programa é a otimização do seu código serial [Cunha 2004].

A otimização serial, além de propiciar uma implementação mais eficiente, orienta os desenvolvedores na identificação das principais rotinas a se beneficiarem do processo de paralelização [Cunha et al. 2001].

O levantamento do perfil de tempos de execução, ou *profiling*, pode ser feito por ferramentas que identificam as subrotinas consumidoras de tempo de CPU. Algumas destas ferramentas estão incorporadas ao próprio sistema operacional, como por exemplo o "gprof", e desempenham adequadamente sua função.

A busca por um melhor desempenho de um algoritmo está diretamente ligada ao comportamento das instruções no processador e no reuso da hierarquia de memória [Goedecker and Hoisie 2001] [Corrêa 2001]. Quando se deseja diminuir o tempo total de execução de uma aplicação, cada detalhe do programa precisa ser minuciosamente analisado. Em suma, para conseguir um algoritmo ótimo, deve-se reduzir ao máximo o número de instruções por ele processadas, sem alterar a consistência do seu resultado final.

O executável de um programa nada mais é do que uma seqüência de instruções interpretáveis pelo hardware, que é resultante do processo de compilação de um texto (código-fonte) escrito pelo programador. Apesar de compiladores

modernos tentarem implicitamente detectar, reordenar e melhorar a ordem de execução de um programa, algumas instruções possuem peculiaridades que podem desperdiçar processamento e a explícita atuação do programador sobre o código pode apresentar bons resultados.

Embora imperceptível aos softwares de propósito geral, uma correta análise dos algoritmos e o uso de técnicas de otimização podem diminuir consideravelmente o tempo total de execução de uma aplicação [Cunha 2004]. Em softwares científicos, de pouca interação com o usuário e grande demanda de processamento, esses valores ficam ainda mais evidentes. Normalmente essas aplicações gastam a maior parte do seu tempo em laços que são repetidos centenas, milhares ou milhões de vezes.

O conhecimento da aplicação (regras do negócio) e os detalhes do hardware utilizado, auxiliam muito na escolha da melhor técnica a ser aplicada ao algoritmo. O maior conhecimento nas regras subsidia, dentre outras ações, inversões de passos, eliminação de redundâncias e junção de tarefas. Já os detalhes da arquitetura facilitam a exploração de recursos de hardware como: reuso de memória cache, balanceamento entre as tarefas e comunicação dos dados.

Em suma, as técnicas de otimização de algoritmos discutidas neste trabalho podem ser aplicadas a qualquer tipo de codificação, serial ou paralela, e estão focadas na redução de chamadas a funções, redução de condicionais e reestruturação de laços.

3.1.1 Redução de Chamadas a Funções

Para programação em alto nível, de sistemas que não são expressivos no quesito consumo de processamento, chamadas a funções pré-definidas são comuns, devido à organização de código que proporcionam. Esta prática, entretanto, pode consumir precioso tempo em aplicações que demandam alto processa-

mento. Dois são os vilões: o primeiro é a quantidade de chamadas; o segundo, o custo computacional.

A redução de chamadas desnecessárias a uma função demonstra economia significativa de tempo, e a reorganização do algoritmo e o uso de variáveis auxiliares ou até mesmo variáveis indexadas e arquivos textos podem culminar na eliminação deste excesso. Quando em uma linguagem de alto nível precisa-se calcular a média sobre um conjunto de dados, a qualquer momento do programa é chamado o método responsável para tal tarefa. Se este valor, contudo, não for guardado, volta-se a chamá-lo quando o resultado for novamente requisitado. Esta é uma prática normal em sistemas comuns, mas extremamente custosa caso essa ação se repita inúmeras vezes.

Hipoteticamente, se um programa necessitar de 5 segundos para ser executado, e depois de reduzida a quantidade de chamadas passar a gastar 4 segundos, provavelmente passará despercebido pelo usuário, mesmo que isto represente 20% de tempo gasto desnecessariamente. Já em programas que consomem horas ou dias de processamento, esta percentagem extra de tempo é expressiva e inaceitável.

O *overhead* ocasionado pelas chamada de uma função é pequeno, mas no montante final deve ser considerado. A diminuição desse custo de invocação pode ser efetivado com o uso de *inlining* [Corrêa 2001]. Entende-se por *inlining* o processo de substituição de uma chamada de subrotina ou função do usuário pela sua definição. Um laço que contenha internamente uma chamada pode ser um excelente candidato a *inlining*, visto que, o processo salva o código fonte da subrotina ou função no local onde ela é invocada. Mas o exagero das trocas pode aumentar consideravelmente o tamanho do código executável, o que pode acarretar impactos negativos ao desempenho do programa, devido à necessidade de paginação do programa executável [Severance and Dowd 1998]. Outro bom candidato à *inlining* é a substituição de uma chamada de função pré-definida por

uma instrução mais simples, por exemplo, trocar uma instrução de potenciação $pow(x,2)$ pela multiplicação correspondente $(x*x)$.

Em contrapartida aos benefícios citados, o procedimento de substituição de chamadas à função normalmente proporciona aumento da ilegibilidade e uma maior dificuldade na manutenção dos códigos, visto que essas trocas, replicam pedaços de códigos por todo o programa. E uma necessidade de correção, que outrora era pontual, passa a estar pulverizada. Sendo assim, estas só devem ser efetuadas em posições estratégicas, que realmente trarão benefícios de diminuição de tempo.

3.1.2 Redução de Condicionais

A quantidade média de ocorrência de instruções de desvios (instruções condicionais) em um programa qualquer é da ordem de uma para cada cinco instruções [Gonçalves et al. 2006]. A decodificação de uma instrução condicional gera dois caminhos que podem ser seguidos: o tomado e o não tomado, e o processador antecipa-se na execução de um deles. Nos melhores casos, a taxa de acerto destas previsões chegam a 98%, segundo [Gonçalves et al. 2006], mas o transtorno gerado para o processador, quando há um erro do desvio previsto, é muito grande.

Nos processadores superescalares, as instruções são carregadas para o primeiro de vários estágios da arquitetura, e quando esta passar para o próximo estágio, outra instrução da seqüência é carregada e assim sucessivamente. O problema das instruções condicionais é que as subseqüentes são carregadas antes mesmo do resultado da condição, através da escolha de um dos caminhos. A quantidade de estágios dos processadores atuais varia, mas a exemplo do Intel® Pentium4, que possui 20 estágios, há o carregamento de instruções que vão sendo processadas sem a certeza de que o fluxo esteja no caminho certo. E no caso de um caminho tomado errado, o esvaziamento do pipeline e o rei-

nício do carregamento das instruções gastam tempos extras de processamento. Baseado neste contexto, evitar condicionais, principalmente os aninhados, pode melhorar o tempo de execução da aplicação.

3.1.3 Reestruturação de Laços

Nos laços de repetição, três são os principais fatores que precisam de atenção. O primeiro é quanto ao reaproveitamento da memória cache; o segundo é sobre a instrução condicional implícita que vai gerar sua parada; e o terceiro diz respeito às instruções internas ao laço. Técnicas que tentam otimizar laços se baseiam na economia de instruções, que irão se repetir durante todas as iterações, ou seja, economizar uma instrução dentro de um laço pode representar a economia de milhares de instruções na aplicação.

Por problemas arquiteturais, a memória cache tem um tamanho reduzido, mas em compensação, uma velocidade altíssima, se comparada a outras memórias [Tanenbaum 2001]. Além disso, o aproveitamento desse recurso melhora a execução de uma aplicação [Severance and Dowd 1998]. Se um laço cabe inteiramente na cache, há um ganho substancial no tempo de execução, pois em uma eventual necessidade de acesso à memória principal, serão desperdiçados preciosos ciclos de processamento.

Todo laço de repetição exige uma condição de parada e esta precisa ser bem analisada, pois a cada iteração uma instrução condicional verifica sua continuidade. O problema agrava-se com o uso de comparações compostas ou cálculos nos elementos testados, visto que para uma comparação $(A == B * 2)$, a multiplicação será realizada a cada iteração. Caso não exista nenhuma dependência interna, essa operação poderá ser extraída do laço, dando lugar a um valor constante $(A == C)$ onde $C = B * 2$. Este é um pequeno detalhe que também economiza várias instruções.

Da mesma forma que as condições de parada, instruções internas ao laço serão executadas em todas as suas iterações. Então, evitar condicionais, eliminar chamadas desnecessárias a funções, extrair possíveis cálculos e reorganizar a estrutura interna do laço, segundo [Severance and Dowd 1998], representam ganhos de desempenho. Algumas técnicas tradicionais de reestruturação de laços são descritas a seguir [Gonçalves et al. 2006].

1. O **desenrolamento de laço** é a técnica que diminui a quantidade de comparações que são executadas como condição de parada. Normalmente, uma variável de controle é incrementada ou decrementada em uma unidade a cada iteração e esta é aplicada a uma expressão condicional que representa sua saída ou uma nova execução. No entanto, colocando mais instruções similares dentro do mesmo laço, possibilita-se que a variável de controle seja alterada em um passo maior, e a comparação passa a ser feita uma vez para cada grupo de instruções. Conforme a implementação mostrada na Listagem 3.1, o laço original manipula um vetor indexado pela variável de controle *x*, indo de 0 a 99 e executando a linha interna uma vez para cada posição do vetor. Já no laço desenrolado, cinco execuções internas são realizadas e o incremento da variável de controle também segue este valor. Neste exemplo, reduz-se de 100 para 20 as comparações feitas como condição de parada.

```
1  // Laço normal                          // Laço desenrolado
2  for( x = 0; x < 100; x = x + 1 )        for( x = 0; x < 100; x = x + 5 )
3  {                                       {
4      vetor[x] = 10;                          vetor[x] = 10;
5  }                                           vetor[x + 1] = 10;
6                                              vetor[x + 2] = 10;
7                                              vetor[x + 3] = 10;
8                                              vetor[x + 4] = 10;
9                                          }
```

Listagem 3.1: Desenrolamento de laço.

2. A **extração de condicionais** consiste em analisar e remover execuções de instruções do tipo *if-then-else*. As comparações são instruções custosas ao processador e reescrever códigos, principalmente internos a laços, de forma a economizar tais instruções, é um modo de otimizar o algoritmo. Na implementação mostrada na Listagem 3.2, internamente ao laço existe uma comparação que será testada em toda iteração, porém será verdadeira apenas uma vez. Ao retirar tal comparação, aplica-se uma economia de 99 instruções de desvio.

```
1 // Laço normal
2 for( x = 0; x < 100; x = x + 1 )
3 {
4     vetor[x] = 10;
5     if( x == 55 )
6     {
7         vetor[x] = 11;
8     }
9 }
```

```
// Laço com condicional extraído
for( x = 0; x < 100; x = x + 1 )
{
    vetor[x] = 10;
}
vetor[55] = 11;
```

Listagem 3.2: Extração de Condicionais.

3. O **descascamento de laço** é a técnica de extrair instruções que estariam internas ao laço. Por si só esta técnica pode não aumentar o desempenho do algoritmo, mas utilizada em conjunto com a fusão de laço proporciona o esperado ganho. Ao retirar algumas iterações do laço, pode-se aproveitar sua estrutura para executar instruções de outro laço ou adaptá-lo ao tamanho da memória cache. Como na Listagem 3.3, o laço original, que vai de 0 a 52, é "descascado" e três das suas iterações são executadas fora de sua estrutura.

4. A **fusão de laço** é o aproveitamento de dois ou mais laços que percorrem a mesma quantidade de iterações. Ao aplicar essa técnica nos locais possíveis sem que ocorra inconsistência dos dados, economiza-se a quan-

```
1  // Laço normal                          // Laço descascado
2  for( x = 0; x < 53; x = x + 1 )         for( x = 0; x < 50; x = x + 1 )
3  {                                       {
4      vetor[x] = 10;                          vetor[x] = 10;
5  }                                       }
6                                          vetor[50] = 10;
7                                          vetor[51] = 10;
8                                          vetor[52] = 10;
```

Listagem 3.3: Descascamento de laço.

tidade de comparações da condição de parada. Cabe ressalva que ao fundir dois laços em um, aumentando conseqüentemente seu tamanho, este pode perder o benefício de caber na cache. A exemplo da implementação da Listagem 3.4, dois laços que originalmente executam 100 vezes são agrupados de forma a economizar instruções.

```
1  // Laços normal                         // Fusão dos dois laços
2  for( x = 0; x < 100; x = x + 1 )        for( x = 0; x < 100; x = x + 1 )
3  {                                       {
4      vetor[x] = 10;                          vetor[x] = 10;
5  }                                           vetor2[x] = 20;
6  for( x = 0; x < 100; x = x + 1 )        }
7  {
8      vetor2[x] = 20;
9  }
```

Listagem 3.4: Fusão de laços.

5. A **fissão de laço**, ao contrário da fusão, separa pedaços de um grande laço em partes menores. A intenção é o aproveitamento da memória cache caso o laço original inteiro exceda seu tamanho ou uma fusão com outras estruturas. Possíveis problemas de dependência de dados devem ser analisados. Como mostrado na Listagem 3.5, um laço que executa suas instruções por 100 iterações é dividido em duas partes menores.

6. A **reversão de laço** inverte a sua ordem de execução. É uma técnica que

```
1  // Laço normal                          // Fissão em dois laços menores
2  for( x = 0; x < 100; x = x + 1 )        for( x = 0; x < 50; x = x + 1 )
3  {                                       {
4      vetor[x] = 10;                          vetor[x] = 10;
5  }                                       }
6                                          for( x = 50; x < 100; x = x + 1 )
7                                          {
8                                              vetor[x] = 10;
9                                          }
```

Listagem 3.5: Fissão de laços.

apresenta duas vantagens. Primeiro, a comparação executada na condição de parada é mais rápida quando comparada com 0 (ao invés de incrementar a variável de controle de 0 a 100, decrementa-se de 100 a 0). A segunda vantagem é relacionada ao reaproveitamento da mesma estrutura para uma provável fusão com algum outro de ordem semelhante. A Listagem 3.6 demonstra o laço original e o laço reverso.

```
1  // Laço normal                          // Laço reverso
2  for( x = 0; x < 100; x = x + 1 )        for( x = 99; x >= 0; x = x - 1 )
3  {                                       {
4      vetor[x] = 10;                          vetor[x] = 10;
5  }                                       }
```

Listagem 3.6: Reversão de laço.

7. O **intercâmbio de laços** é a permuta de laços aninhados. Esta técnica é utilizada para percorrer matrizes que, dependendo da linguagem, possuem formas diferentes de serem alocadas na memória (na ordem de linhas ou colunas). Também auxiliam nos casos em que o laço mais interno, muito grande, não esteja aproveitando a memória cache, e uma troca com o laço mais externo pode otimizar o uso da arquitetura. Na Listagem 3.7 há um exemplo de intercâmbio de laços.

As técnicas descritas nesta seção devem ser usadas com bom senso porque contribuem para a ilegibilidade do código, mas se o objetivo for o desempenho

```
1  // Laços aninhados                        // Laços trocados
2  for( x = 0; x < 100; x = x + 1 )          for( y = 0; x < 50; y = y + 1 )
3  {                                         {
4     for( y = 0; y < 50; y = y + 1 )           for( x = 0; x < 100; x = x + 1 )
5     {                                         {
6         matriz[x][y] = 10;                        matriz[x][y] = 10;
7     }                                         }
8  }                                         }
```

Listagem 3.7: Intercâmbio de laços.

do algoritmo, vale a eliminação de quaisquer instruções dispensáveis. Estas técnicas podem ser combinadas entre si e a característica do ambiente computacional e a particularidade de cada aplicação dita a melhor escolha. No capítulo 5 será apresentado o resultado obtido pela inserção de várias destas técnicas em uma aplicação de seleção de variáveis e características.

3.2 Ferramentas para Programação Paralela

Apontada como promissora na resolução de aplicações de grande escala, a computação paralela tem como princípio básico decompor grandes e/ou complexos algoritmos em pequenas tarefas, executadas simultaneamente por vários processadores [Pacheco 1997]. Para [Tanenbaum 2001], a execução paralela envolve basicamente:

i) o particionamento de um programa em tarefas menores;

ii) o mapeamento das tarefas nos processadores do sistema;

iii) a troca de dados entre as diferentes tarefas do programa (comunicação);

iv) a sincronização na execução das tarefas.

A obtenção do paralelismo dá-se de duas formas: implícita, quando o compilador é o responsável por encontrar as partes do programa que podem ser executadas em paralelo; e explícita, quando o programador analisa e introduz certas funções especialmente projetadas para esta atividade. O paralelismo explícito

apesar de não ser uma tarefa trivial [Patterson and Hennessy 2005], apresenta inúmeras vantagens sobre o implícito, principalmente porque inclui em maior grau o conhecimento dos programadores na aplicação, proporcionando programas geralmente mais eficientes.

A divisão do problema em subproblemas deixa evidente a necessidade de mecanismos que se comuniquem entre si para que haja sincronização entre os processos em execução, pois todos estão em um trabalho conjunto para resolver este determinado problema. Outro ponto importante é o mapeamento dos processos nos processadores do sistema. De acordo com [Bal et al. 1989] [Grama et al. 2003], processadores são as unidades físicas de processamento, e processos são entidades totalmente independentes, que executam código de programa e que possuem dados e um estado próprio.

Uma das maiores diferenças entre um sistema paralelo e um sistema monoprocessador é a forma de comunicação entre os processos [Tanenbaum 2001]. Basicamente são dois os tipos de comunicação: via memória compartilhada (ver Figura 3.1), no qual os processos de um programa compartilham uma área de memória comum onde dados podem ser lidos e escritos por quaisquer processos; e via troca de mensagens (ver Figura 3.2), conjunto de processos que somente possuem acesso à memória local e as informações são enviadas da memória local de um processo para a memória local de outro processo, remotamente, através do envio e recebimento explícito de mensagens.

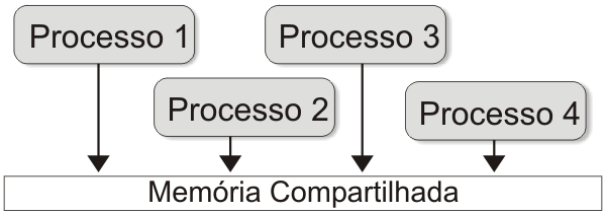

Figura 3.1: Comunicação via Memória Compartilhada, concorrência no acesso à memória.

Genericamente a comunicação via troca de mensagens é implementada utilizando bibliotecas de comunicação. Os dois sistemas mais conhecidos são

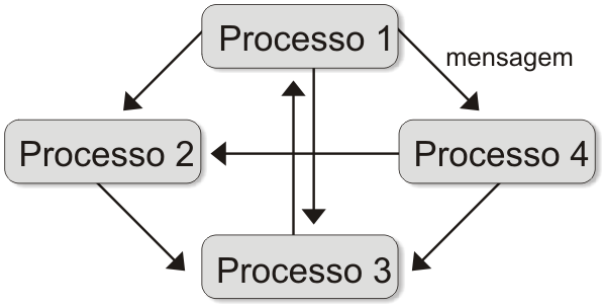

Figura 3.2: Comunicação via Troca de Mensagens, cada processador com sua própria memória local.

Parallel Virtual Machine (PVM) e *Message Passing Interface* (MPI) e estão disponíveis em diversas plataformas, suportando linguagens como Fortran 77, Fortran 90, ANSI C e C++, existindo também implementações para outras linguagens, como Java (mpiJava) [Aveleda 2003]. Este trabalho faz uso de MPI, e na subseção seguinte maiores detalhes do padrão serão apresentados.

Segundo [Grama et al. 2003], a transmissão das mensagens pode ser realizada de duas maneiras: i) com primitiva bloqueante, o processo transmissor fica bloqueado até receber o aviso de recebimento do processo receptor (ver Figura 3.3a); ii) com primitiva não-bloqueante (ver Figura 3.3b), o processo transmissor envia uma mensagem ao receptor mas segue normalmente sua execução.

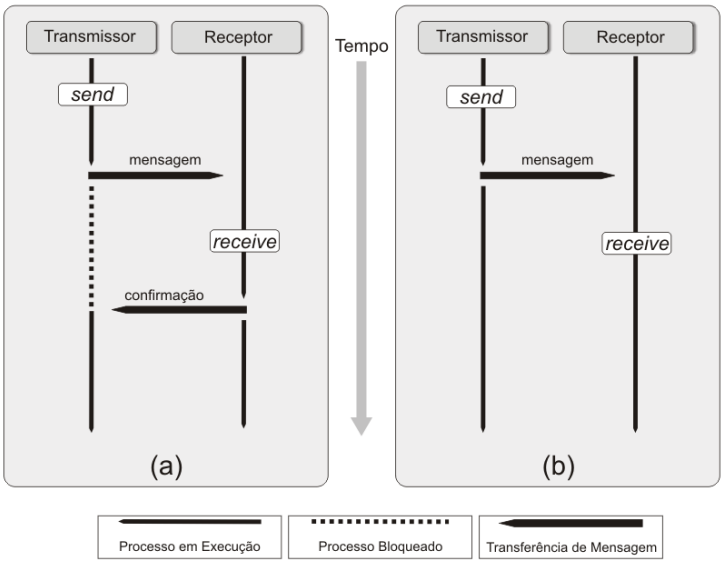

Figura 3.3: Primitivas Send/Receive a)Bloqueante b)Não-Bloqueante.

3.2.1 MPI

Inicialmente projetada por cada fabricante de máquina paralela, que, conforme a sua necessidade, desenvolvia seu próprio ambiente, o método de passagem de mensagem pecou pela falta de padrão e prejudicou a portabilidade entre os sistemas. Objetivando acabar com este problema, um comitê de aproximadamente 40 instituições, incluindo os fabricantes de máquinas, universidades, laboratórios governamentais e membros mundiais, desenvolveu em 1994 um conjunto de funções independente de hardware e plataforma [CENAPAD-NE 1998] [Grama et al. 2003] [Pacheco 1997] [Pacheco 1995]. Deu-se a este conjunto o nome de MPI, que pelo volume de referências em trabalhos encontrados na literatura, pode ser considerada, atualmente, a biblioteca de comunicação de maior expressividade, apresentando-se como uma das alternativas mais viáveis ao paralelismo. [Foster 1995] afirma que a biblioteca MPI pode ser vista como uma coleção de processos que executam programas escritos em uma linguagem seqüencial comum, e que realizam chamadas a rotinas da biblioteca para enviar e receber mensagens.

A biblioteca MPI conta com aproximadamente 125 funções de programação e ferramentas de análise de desempenho [Grama et al. 2003], e de acordo com [CENAPAD-NE 1998], pode ser definida como um conjunto de rotinas para facilitar a comunicação (troca de dados e sincronização) entre os processos paralelos. [Pacheco 1995] identifica duas características que podem ser apresentadas como limitações de MPI: o programador é responsável diretamente pela paralelização; e em alguns ambientes específicos, o custo da comunicação pode tornar-se extremamente proibitivo pela quantidade de transmissão de mensagens necessárias à aplicação. Segundo [Castelo 1999] [CENAPAD-NE 1998] [Pacheco 1997], o padrão MPI apresenta outras características listadas como pontos fortes:

i) eficiência - cuidadosamente projetado para executar eficientemente em máquinas diferentes, o padrão MPI define a mesma *Application Programming Interface* (API) para os usuários, contudo cada fabricante é livre para implementar as rotinas, utilizando características exclusivas de sua arquitetura;

ii) facilidade - é baseado em práticas comuns de paralelismo e de outras bibliotecas de troca de mensagens, e também acrescenta algumas extensões para permitir maior flexibilidade;

iii) portabilidade - é compatível com sistemas de memória distribuída, memória compartilhada, MPP e outras arquiteturas especiais;

iv) transparência - permite que um programa seja executado em sistemas heterogêneos sem mudanças significativas;

v) segurança - provê uma interface de comunicação confiável, eximindo o usuário de possíveis preocupações com falhas;

vi) escalabilidade - suporta comunicação coletiva com a criação de subgrupos de processos, melhorando ainda mais a troca de mensagens.

Funções simples como inicializar, encerrar, recolher informações do ambiente, enviar e receber mensagens (ver Tabela 3.1), solucionam a grande maioria dos problemas propostos [Grama et al. 2003] [Franco 2004]. Todavia, para permitir implementações mais otimizadas, existem funções avançadas que facilitam principalmente as operações globais como a comunicação coletiva, a junção e a redução.

3.2.2 Divisão de Tarefas

O processamento paralelo de programas requer a utilização de técnicas de decomposição para dividir, mapear e garantir o equilíbrio da carga computacional e das estruturas de dados [Hwang 1993], e a eficiência de um aplicativo pode

Tabela 3.1: Funções básicas do padrão MPI.

Função	Descrição
MPI_Init	inicializa o ambiente MPI;
MPI_Comm_size	contabiliza o número de processos de um grupo;
MPI_Comm_rank	identifica um processo MPI dentro de um determinado grupo;
MPI_Send	rotina para envio de mensagens no MPI;
MPI_Recv	rotina de recepção de mensagens no MPI;
MPI_Finalize	finaliza um processo no MPI;

ser seriamente afetada pela má distribuição dos dados e/ou da computação.

Conforme [Grama et al. 2003], a obtenção de paralelismo é estratégica para se alcançar eficiência que, de acordo com [Franco 2004], pode se dar de duas formas: paralelismo funcional, elegendo as fases do algoritmo que podem ser executadas em paralelo; e paralelismo de dados, identificando os dados que podem ser processados em paralelo. Na primeira abordagem, o enfoque inicial é no processamento que será realizado, e acontecerá um particionamento do programa em tarefas (funções) independentes que poderão executar procedimentos bem diferentes entre si. Já na segunda abordagem, o enfoque dá-se nos dados que serão processados e na sua divisão em subconjuntos, que, posteriormente, serão distribuídos entre os processadores que executarão a mesma seqüência de instrução sobre eles. Dependendo da aplicação, uma abordagem poderá ser mais adequada do que outra, e em muitas aplicações, uma combinação dos paradigmas pode apresentar bons resultados.

Devido à latência provocada pela comunicação entre os processos, a divisão de uma tarefa em subtarefas em um nível de granularidade adequado é fundamental para o desempenho de um algoritmo [Lu and Chung 1998]. A granularidade, também chamada de nível de paralelismo, está relacionada com o tamanho médio das subtarefas que podem ser executadas simultaneamente em diferentes processadores. [Grama et al. 2003] define que granularidade é a quantidade de trabalho realizada em uma unidade processadora em relação à

quantidade de informação trocada.

Quando as tarefas são divididas em um conjunto pequeno de subtarefas, e cada subtarefa possui uma grande quantidade de instruções seqüenciais, tem-se granularidade grossa. A vantagem é a redução dos custos de comunicação entre os processos, aproveitando o tempo para fazer processamento. Já na granularidade fina, as tarefas são formadas por um grande número de subtarefas, sendo que cada subtarefa possui uma quantidade reduzida de instruções. Como vantagens, pode-se facilitar o balanceamento de carga e aumentar o número de processos concorrentes, porém aumenta-se a necessidade de comunicação entre os processos e conseqüentemente o custo de rede [Lu and Chung 1998] [Patterson and Hennessy 2005]. A granularidade média representa um conjunto intermediário das anteriores.

3.3 Análise e Medidas de Desempenho

Para alcançar a excelência de um programa paralelo, é de suma importância explorar as características das arquiteturas paralelas e o paralelismo implícito nas aplicações [Oliveira 2003]. Assim, os desenvolvedores utilizam as técnicas de avaliação de desempenho como auxiliar na elaboração de códigos mais eficientes. [Junior 2002] [Oliveira 2003] [Júnior 1997] e outras inúmeras referências são enfáticas em afirmar que não existe uma medida universal, aplicável a todos os sistemas, mas sim, medidas consideradas padrão, porque geralmente são as mais usadas, e outras medidas mais específicas.

Inúmeros são os fatores passíveis de avaliação e também influenciadores no desempenho de um sistema, principalmente em sistemas paralelos. Quando se trata de uma aplicação real as características se amplificam, porque o comportamento da aplicação é naturalmente inesperado. Sendo assim, a maior gama de detalhes possíveis precisam ser avaliados, e a compreensão do sistema analisado

pode priorizar alguns fatores em detrimento de outros.

> [...] no início da década de 90 o desempenho era expresso adequadamente através do número de operações de ponto flutuante por segundo (*FLoating point OPeration* (FLOP)), nos anos seguintes os fabricantes e organizações buscaram formas de tentar expressar o desempenho computacional do equipamento como um todo e não apenas do processador. Neste intuito, outros aspectos relevantes como o desempenho do sistema de discos, a velocidade de acesso à memória, o desempenho em redes de alta velocidade e a capacidade dos equipamentos executarem diversas tarefas simultaneamente passaram a ser medidos e os resultados obtidos ponderados em índices que expressam o desempenho geral do equipamento [Vazquez 2002].

[Junior 2002] vai além e subdivide a avaliação de desempenho em análise, com o objetivo de interpretar os resultados e identificar os pontos de melhoria; e em predição, apresentando ao usuário se o desempenho é adequado ou não em relação ao esperado. Ainda classifica as métricas quanto à dependência, ou não, à velocidade de execução da aplicação. As medidas diretamente ligadas ao tempo são *speedup* e eficiência; já as medidas independentes, com objetivo quantificador são, entre outras: ocupação de registradores, falta de dados em cache e movimentação de memória, para a avaliação de hardware; e informações sobre a execução dos programas, quantidade de ocorrências de uma função, *overhead* de comunicação, para softwares.

Independente da classificação utilizada, é certo que bons parâmetros de desempenho auxiliam tanto a construção de sistemas eficientes, quanto a avaliação do resultado final da arquitetura.

3.3.1 Medidas de Desempenho

Quando, em um sistema paralelo, utilizam-se quatro processadores, logo se cria a expectativa de que qualquer aplicação executada neste sistema será processada quatro vezes mais rápida. Este desempenho ideal ou linear, na prática, raramente é alcançado [Patterson and Hennessy 2005], pois para que a execução paralela atinja este desempenho esperado, o código do programa original-

mente desenvolvido de forma seqüencial deve, de alguma maneira, ser 100% paralelizado.

Entretanto, a característica primordial da computação paralela é dividir uma tarefa em subtarefas e então executá-las paralelamente em diversas unidades de processamento [CENAPAD-SP 2006]. Isto já comprova a dificuldade de se obter o ganho linear de desempenho, pois existe uma fração de tempo seqüencial necessária para o particionamento do problema, um custo de transmissão dos dados pela rede de interconexão e um custo para sincronização dos resultados parciais, enviados pelos nós escravos ao nó mestre.

A redução no tempo total de processamento de uma aplicação, embora não seja um critério único e suficiente na avaliação do desempenho, é a principal razão para se utilizar as técnicas de computação paralela [Cunha 2004]. A escolha certa da medida ou do conjunto de medidas é fator chave para representar o comportamento real do sistema e identificar possíveis razões de tempos de execução ruins ou fornecer parâmetros para a geração de códigos otimizados [Junior 2002]. Algumas métricas são descritas a seguir:

- *speedup* (ou aceleração): mede a razão entre o tempo consumido pela execução de um algoritmo ou aplicação em um único processador, pelo seu tempo de execução em paralelo.

$$speedup_{(n)} = \frac{tempoExecução_{(serial)}}{tempoExecução_{(paralelo)}}$$

- eficiência: é a medida da fração de tempo que um elemento de processamento é utilizado de forma proveitosa. A eficiência é igual ao *speedup* dividido pela quantidade de processadores utilizados, representado por (n).

$$eficiência_{(n)} = \frac{speedup_{(n)}}{n}$$

Um problema recorrente no processo de avaliação de desempenho é o re-

ferencial a ser tomado para a definição das medidas. Vários autores, entre eles [Sahni and Thanvantri 1996] [Hwang 1993] [Gustafson 1988], tratam do assunto e defendem maneiras distintas para o cálculo. O *speedup*, por exemplo, pode ser expresso com base apenas no tempo de processamento, denominado por [Sahni and Thanvantri 1996] de *speedup* Real, *speedup* Absoluto e *speedup* Relativo, ou agregado a outros valores para tornar o fator mais significativo.

O *speedup* Real é medido com base no tempo do melhor programa serial (MPS) em um processador, pelo tempo de execução do programa em paralelo (PP) em *n* processadores.

$$speedupReal = \frac{T(MPS\,em\,1\,processador)}{T(PP\,em\,n\,processadores)}$$

O *speedup* Absoluto é a razão do tempo do melhor programa serial no processador mais rápido, pelo programa paralelo em *n* processadores.

$$speedupAbsoluto = \frac{T(MPS\,no\,processador\,mais\,rápido)}{T(PP\,em\,n\,processadores)}$$

O *speedup* Relativo é o tempo de execução do programa paralelo sendo executado em um único processador pelo seu tempo de execução em *n* processadores.

$$speedupRelativo = \frac{T(PP\,em\,1\,processador)}{T(PP\,em\,n\,processadores)}$$

Usado isoladamente, o *speedup* pode não representar fielmente o desempenho de uma arquitetura paralela, mas, se combinado com outras medidas, pode traduzir melhor a realidade de desempenho de um sistema. Outras definições que podem auxiliar no estudo do comportamento das arquiteturas paralelas são:

- lei de amdahl: define que o *speedup* de um programa fica limitado pela fração não paralelizável (ou estritamente serial) existente em todo algoritmo [Amdahl 1967].

- escalabilidade: indica a variação do tempo de execução e da aceleração, com o acréscimo do número de processadores e/ou tamanho do problema.

3.4 Ferramentas para Gerenciamento de Ambientes Paralelos

3.4.1 Oscar

O pacote *Open Source Cluster Application Resources* (OSCAR) ou Recursos para Aplicações de *cluster* de Código Aberto, é uma coleção de *scripts* para construção e configuração de *clusters*. Por ser uma tarefa repetitiva, e envolver conhecimento sobre vários aspectos de sistemas operacionais, estes *scripts* foram a forma de automatizar todo o processo.

A plataforma necessária para sua instalação é o sistema operacional Linux, nas distribuições Red Hat, Fedora e Mandriva. Inicialmente executado a partir do nó mestre, através de uma interface visual, uma seqüência ordenada de passos deve ser realizada, como pode ser visto na Figura 3.4. Nos primeiros passos, copiam-se todos os pacotes, necessários e compatíveis, da instalação do sistema operacional para o disco local, configuram-se os endereços de rede, nomes das máquinas, discos de *boot* e habilitam-se os usuários. Posteriormente, através dos discos de *boot*, cada futuro nó escravo, copia sua respectiva imagem e configuração do nó mestre.

Ainda nesta interface existem *scripts* que executam testes de comunicação, leitura e escrita entre os nós pertencentes ao *cluster*. Este *script* foi utilizado para a construção do *cluster* homogêneo citado na seção 5.1.

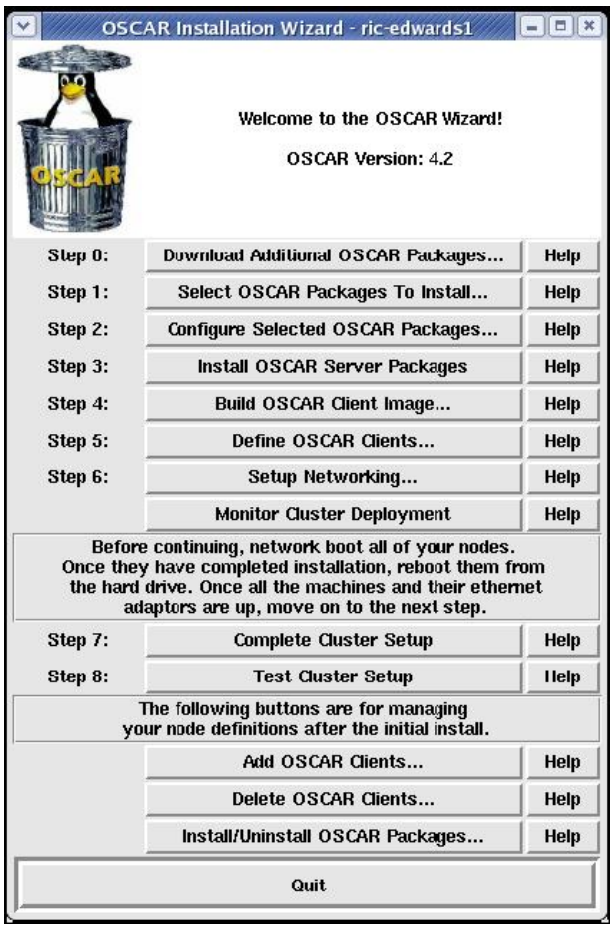

Figura 3.4: Interface do OSCAR (*script* para automatizar a construção de *clusters*).

3.4.2 Ganglia

Ganglia é um pacote de monitoramento, distribuído e escalável, para sistemas de computação de alto desempenho, como *clusters* e *grids*. É executado via *daemons* (processos em *backgrounds*) nos nós escravos. Para garantir escalabilidade e baixa sobrecarga, o Ganglia faz a coleta das métricas em intervalos de tempos aleatórios. O nó mestre possui uma interface via navegador de internet que monitora todos esses *daemons* exibindo informações como percentagem de uso de CPU, quantidade utilizada de memória e de rede, entre outros. Este monitoramento não mostra informações a respeito de uma aplicação específica e sim uma visão do *cluster* como um todo.

Uma vez em que os recursos do *cluster* foram alocados apenas para o experimento, a função deste monitoramento foi a visualização gráfica em tempo de

execução de nós ociosos ou sobrecarregados, permitindo algumas correções no algoritmo baseado nestas informações.

4 *Seleção de Variáveis e Características*

As Redes Neurais Artificiais (RNAs) são aplicadas para resolução de problemas cotidianos de diversas áreas do conhecimento, e têm nas séries temporais promissores conjuntos de dados de entrada. De um modo geral, o objetivo de uma RNA é classificar e reconhecer padrões, predizer e generalizar informações, estabelecendo uma relação entre a entrada e uma correspondente saída ou resposta. O processo de aprendizado de uma RNA consiste em estudar o comportamento de um sistema, e então, aplicar este conhecimento adquirido a novos dados de entrada para predizer uma saída apropriada.

Com um alto custo computacional de treinamento, o sucesso do aprendizado depende fortemente da qualidade e da quantidade dos dados disponíveis e, portanto, um bom acervo de dados é fundamental para a confiabilidade do resultado final.

A seleção de variáveis e características é um passo essencial para áreas importantes de pesquisas como a modelagem de sistemas complexos, mineração de dados e reconhecimento de padrões. Os conjuntos de dados destas áreas geralmente apresentam alta dimensão e isto cria problemas para os algoritmos de aprendizagem. Logo, a redução da dimensão ou a seleção de um subconjunto de variáveis e características tornam o modelo mais parcimonioso. Sozinha, a tradicional análise de relevância dos dados é ineficiente para selecionar o melhor conjunto, assim, utiliza-se um método que, além da relevância, também analisa a redundância. Portanto, o algoritmo de aprendizado de uma RNA fica melhor,

mais rápido e mais fácil de ser entendido. Além disso, pode evitar o problema de excesso de treinamento (*overfitting*).

Neste capítulo, inicialmente, apresenta-se uma breve contextualização sobre séries temporais e RNA. Mesmo o desenvolvimento da RNA não fazendo parte do escopo deste trabalho, este tópico é discutido a fim de facilitar o entendimento e justificar as motivações para o uso da metodologia de seleção de variáveis e características descritas posteriormente.

4.1 Séries Temporais

Segundo [Morettin and Toloi 1985], "uma série temporal é qualquer conjunto de observações ordenadas no tempo". Muitos podem ser os exemplos de séries temporais: valores horários da umidade relativa do ar na cidade de Londrina, índice mensal de chuva na região norte, valores diários da cotação do dólar, produção diária de automóveis na companhia ABC, faturamento mensal da empresa XYZ, índice diário das vendas de ações na Bolsa de Valores de São Paulo (BOVESPA), número de atendimentos diários em um pronto socorro, índice semestral de aprovados no curso de Ciência da Computação, dentre outros.

O interesse pela trajetória, espaçamento e quantidade dos valores de uma variável ao longo do tempo é dependente da aplicação. A análise de séries temporais tem sido muito usada em diversos problemas do mundo real, eliminando riscos e auxiliando o planejamento e a tomada de decisões. De acordo com [Morettin and Toloi 1985], a motivação pela observação de séries temporais pode se dar em:

1. investigar o mecanismo gerador da série temporal;
2. fazer previsões de valores futuros;
3. descrever o seu comportamento;

4. procurar a existência de tendências, ciclos e variações;

5. procurar periodicidades relevantes nos dados.

Ainda, a avaliação simultânea de várias séries de um mesmo domínio pode demonstrar a influência de uma variável sobre a outra, ou mesmo, a independência entre elas. As RNAs podem auxiliar nas observações dos dados, visto que baseado em suas premissas básicas, buscam automatizar os mesmos objetivos.

4.2 Redes Neurais

As RNAs são técnicas computacionais representadas por um modelo matemático inspirado na estrutura neural de organismos inteligentes, capazes de adquirir conhecimento através de experiências [Haykin 1999]. O cérebro de um mamífero tem bilhões de neurônios, já uma grande RNA possui somente centenas ou milhares de unidades de processamento, justificada pelo alto custo computacional empregado por esses modelos. O processo de desenvolvimento de uma RNA consiste, resumidamente, na coleta e separação de dados, treinamento da rede e integração.

Conforme [Haykin 1999], as duas primeiras etapas, coleta e separação de dados, requerem uma cuidadosa análise para minimizar ambigüidades e desacertos nos dados. Além disso, os dados coletados devem ser significativos e cobrir amplamente o domínio do problema [Zuben 2003]. Paralelamente à qualidade dos dados, que é essencial segundo [Bailey and Thompson 1990], a quantidade de variáveis e o tamanho da amostra também devem ser analisados, visto que esta decisão pode influenciar positiva ou negativamente na eficiência da RNA. As conseqüências decorrentes dos erros nesta etapa vão da ineficiência de predição da rede até o *overfitting*. Assim, o bom conhecimento do domínio da aplicação em conjunto com uma metodologia de seleção podem definir a melhor escolha.

A próxima etapa, treinamento ou aprendizagem, tem como finalidade ajustar os parâmetros ou pesos sinápticos a fim de capacitar a rede com a associação das entradas a uma saída prevista.

> [...] quando uma entrada é apresentada à rede, uma saída será produzida e posteriormente comparada com a saída desejada. Se a saída produzida for diferente da saída desejada, um ajuste de pesos sinápticos deverá ser realizado de forma que a rede armazene o conhecimento desejado e, conseqüentemente, reflita o aprendizado das funções para as quais ela é projetada. Assim, pode-se concluir que o aprendizado consiste em um processo iterativo de atualização dos pesos sinápticos e, é devido a esse processo que uma RNA é capaz de aprender e generalizar [Haykin 1999].

[Haykin 1999] classifica o procedimento de ajuste dos pesos de duas formas: aprendizado supervisionado e aprendizado não-supervisionado. A diferença entre eles é que, no primeiro paradigma, o supervisor, ou agente externo, é responsável pela adequação dos pesos sinápticos, observando os valores da saída efetiva com a saída esperada, diminuindo o erro existente até que a saída esteja dentro da faixa considerada satisfatória. Já no segundo paradigma, fica a cargo do algoritmo de aprendizagem fazer todo o processo de ajuste de forma autônoma.

A última etapa aqui elencada é a integração, ou seja, é basicamente a utilização do modelo, ou do conhecimento adquirido com os exemplos nos treinamentos, para que, dentre outras coisas, as RNAs consigam prever situações futuras baseado nas informações do passado (séries temporais).

A utilização de RNAs vem surgindo como alternativa para a solução de problemas tratados de forma ineficiente pelo enfoque algorítmico tradicional, pois baseia-se em características comportamentais como capacidade de aprender e generalizar a partir de estímulos (dados) a ela apresentados [Zuben 2003]. Percebe-se então que todo o conhecimento adquirido na etapa de aprendizagem é influenciado pela qualidade e quantidade do histórico de dados atribuído como entrada. Em muitas situações, o tempo de treinamento de uma RNA se torna inaceitável, mesmo utilizando PC's com grande poder de processamento

[Haykin 1999].

Teoricamente, mais variáveis e características deveriam prover um maior poder de discriminação, entretanto, na prática, esse excesso ocasiona um aumento ainda maior do custo de processamento e, muitas vezes, gera confusão durante o ajuste (treinamento) do modelo [Blum and Langley 1997]. Tais motivos justificam a necessidade de escolher modelos parcimoniosos de qualidade.

4.3 Metodologia de Seleção

Neste trabalho, utiliza-se uma metodologia de seleção de variáveis e características apresentada em [Freitas et al. 2006]. Este método baseia-se na análise de relevância e redundância [Yu and Liu 2004], via informação mútua estimada diretamente dos dados.

A seleção de variáveis e características também objetiva a transferência eficiente das informações contidas nos dados para o modelo. Estes tipos de métodos geralmente consistiam em formalizar estatisticamente um subconjunto mínimo (ótimo) de variáveis relevantes associadas a um determinado algoritmo, aplicado a um domínio particular de interesse [Kohavi and John 1997], e segundo [Guyon and Elisseeff 2003], muitos destes problemas foram caracterizados como de difícil solução.

Este conceito evoluiu e o aprendizado do modelo passou a ser considerado mais eficiente e efetivo quando é utilizado um subconjunto ótimo, composto somente por características relevantes e não redundantes. A referência [Yu and Liu 2004] apresentou um método de seleção de variáveis e características que utiliza os conceitos de relevância e redundância aplicados em conjunto.

Os conceitos de relevância e redundância são geralmente ligados ao conceito de correlação, o qual, no plano linear pode ser estimado via coeficiente de correlação. Duas variáveis são consideradas totalmente redundantes se seus va-

lores forem completamente correlacionados e não apresentarem relação linear se o coeficiente de correlação for zero. Entretanto, quando se trata de sistemas não lineares, geralmente a dependência entre variáveis é estimada via informação mútua. As definições formais de relevância, quatidade de informação que a variável carrega e redundância, quantidade de informação de uma variável contida em outra, podem ser vistas em [Yu and Liu 2004].

No contexto de seleção de variáveis e características de sistemas não lineares, estimar a informação mútua entre variáveis, diretamente de dados discretos, sem a necessidade de fazer hipóteses sobre a distribuição a priori dos dados, tem vital importância prática. Isto pode ser alcançado com a utilização da inequação de Cauchy-Schwartz, que é uma substituta do divergente de Kullback-Leibler, integrada a uma janela de Parzen [Parzen 1962].

Considerando $a_i \varepsilon R^m, i = 1,2 \ldots N$ um conjunto de amostras discretas pertencente a uma variável randômica $Y \varepsilon R^m$, uma janela de Parzen pode ser associada a um kernel Gaussiano para estimar a função de densidade de probabilidades (fdp) de Y diretamente dos dados

$$f_y(y) = \frac{1}{N} \sum_{i=1}^{N} G(y - a_i, \sigma^2 I) \tag{4.1}$$

em que G é uma função Gaussiana, a_i é o centro i e I é uma matriz identidade quadrada de ordem m.

Por outro lado, a equação generalizada de entropia de Renyi [Renyi 1976] é dada por

$$H_{R\alpha} = \frac{1}{1-\alpha} \log \left(\int f_y(y)^{\alpha} dy \right) \quad \alpha > 0, \alpha \neq 1 \tag{4.2}$$

Logo, a entropia de Renyi de segunda ordem ($\alpha = 2$) pode ser expressa em

termos de

$$H_{R2} = \log\left(\int f_y(y)^2 dy\right) \tag{4.3}$$

Seja a relação entre duas funções tipo Gaussiana, apresentada a seguir, e que já é bem conhecida

$$\int_{-\infty}^{+\infty} G(x-a_i, \sum 1) G(x-a_j, \sum 2) dx = G(a_i - a_j, \sum 1 + \sum 2) \tag{4.4}$$

em que a_i e a_j são amostras que representam as variáveis analisadas, entendidas como os centros das Gaussianas G da janela de Parzen, e $\sum 1$ e $\sum 2$ são as respectivas matrizes de covariâncias.

Combinando as equações 4.1, 4.3 e 4.4, resulta em

$$\begin{aligned} H(\{a_i\}) &= -\log P(\{a_i\}) \\ P(\{a_i\}) &= \frac{1}{N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} G(a_i - a_j, 2\sigma^2 I) \end{aligned} \tag{4.5}$$

em que o termo $P(\{a_i\})$ é interpretado como toda a energia potencial contida na amostra desde que $G(a_i - a_j, 2\sigma^2 I)$ seja interpretada como a energia potencial contida nas amostras a_i em relação ao campo potencial da amostra a_j e vice-versa [Xu et al. 1998]. A equação 4.5 será utilizada na expressão que estimará a informação mútua, ou seja, a dependência entre as variáveis analisadas.

O divergente de Kullback-Leibler [Principe et al. 1999] é uma medida tradicional de informação mútua (dependência) entre duas variáveis aleatórias e é dado por: $K(f,g) = \int f(x) \log(f(x)/g(x)) dx$, onde $f(x)$ e $g(x)$ são duas funções de densidade de probabilidade (fdps). Entretanto, nem a medida anterior e nem o seu equivalente para a entropia de Renyi podem ser integrados com uma janela de Parzen. A referência [Xu et al. 1998] apresentou um método que integra a janela de Parzen à inequação de Cauchy-Schwartz, para estimar a informação mútua diretamente de dados discretos. Esta equação, para duas variáveis aleató-

rias Y_1 e Y_2 (com fdps $f_{y1}(Y_1)$ e $f_{y2}(Y_2)$ e fpd conjunta $f_{y1,y2}(Y_1,Y_2)$) é expressa pela equação

$$C(Y_1,X_2) = \log \frac{(\int\int f_{y1,y2}(Y_1,Y_2)^2 dy_1 dy_2)(\int\int f_{y1}(Y_1)^2 f_{y2}(Y_2)^2 dy_1 dy_2)}{(\int\int f_{y1,y2}(Y_1,Y_2) f_{y1}(Y_1) f_{y2}(Y_2) dy_1 dy_2)^2} \quad (4.6)$$

Considerando o conjunto de dados $\{a_i, i = 1,2\ldots N\}$, composto pelos subconjuntos a_{i1} e a_{i2}, onde $a_i = [a_{i1}a_{i2}]^T$, e utilizando a equação 4.6, a dependência entre estes dois subconjuntos de dados pode ser estimada por meio da equação a seguir

$$C(a_{i1},a_{i2}) = \log \frac{P(\{a_i\})P_1(\{a_{i1}\})P_2(\{a_{i2}\})}{P_c(\{a_i\})^2} \quad (4.7)$$

em que $P(\{a_i\})$ é o total da informação potencial, ou seja, a_i é um vetor formado pelos vetores das duas variáveis analisadas, ou $a_i = [a_{i1}a_{i2}]^T$; $P_l(j,\{a_i\}) = \frac{1}{N}\sum_{i=1}^{N} G(a_{jl} - a_{il}, 2\sigma^2 I_l)$ é o potencial marginal parcial $(l = 1,2)$ utilizado no cálculo do potencial de informação marginal $P_l(\{a_i\}) = \frac{1}{N}\sum_{j=1}^{N} P_l(j,\{a_i\})$, que para a_{i1} e a_{i2}, resulta no cálculo de $P(\{a_{i1}\})$ e $P(\{a_{i2}\})$; o potencial da informação marginal $(l = 1,2)$ também é utilizado no cálculo do potencial de informação cruzado dado por $P_c(\{a_i\}) = \frac{1}{N}\sum_{j=1}^{N} P_1(j,\{a_i\})P_2(j,\{a_i\})$.

A independência entre duas variáveis aleatórias, de acordo com a equação anterior, requer potenciais de informação marginais pequenos, potenciais de informação conjuntos pequenos e potenciais de informação cruzados grandes. Observa-se que a variável aleatória discreta pode ser do tipo escalar ou vetorial e que a equação 4.7 pode também ser estendida a mais de duas variáveis aleatórias.

Para a análise de relevância e redundância são definidos dois tipos de correlação, de acordo com a referência [Yu and Liu 2004]:

1. **C-correlação**: denotada por $SU_{i,c}$, que é a correlação entre as variáveis de

entrada F_i e a variável dependente (classe C);

2. **F-correlação**: que é a correlação entre qualquer par de variáveis de entrada F_i e F_j $(i \neq j)$.

A determinação de um subconjunto ótimo de variáveis de entrada relevantes é feita a partir do cálculo da C-correlação e da F-correlação. Assume-se que uma variável de entrada F_i é relevante se for altamente correlacionada com a classe C. Já a análise de redundância é feita por meio da avaliação da F-correlação. O algoritmo apresentado na listagem 4.1 faz a seleção de variáveis de acordo com a referência [Yu and Liu 2004], a partir dos cálculos da C-correlação e da F-correlação.

```
1  Entrada:  S(F1, F2, ..., FN, C) // conjunto de dados de treinamento;
2            P                     // patamar pré-definido;
3  Saída:    Sotimo                // subconjunto selecionado como ótimo;
4
5
6  Início
7      Para i de 1 até N
8          Calcule SUi,c para cada Fi;
9          Se (SUi,c > P)
10             Adiciona Fi à Slista;
11     Fim Para
12     Ordene Slista decrescente de valores de SUi,c;
13     Atribua a Fj o primeiro elemento de Slista;
14     Repita
15         Atribua a Fi o próximo elemento de Slista;
16         Se (Fi  <>  Vazio) então faça
17             Repita
18                 Se (SUi,j  >=  SUi,c) então faça
19                     Remove Fi de Slista;
20                 Atribua a Fi o próximo elemento de Slista;
21             Até que (Fi  =  Vazio);
22         Atribua a Fj o próximo elemento de Slista;
23     Até que (Fj  =  Vazio);
24     Atribua a Sótimo os elementos de Slista;
25 Fim
```

Listagem 4.1: Algoritmo para a Seleção de Variáveis e Características.

O algoritmo tem como primeiro passo (linhas 7 a 11) os cálculos de $(SU_{i,c})$

para cada F_i e caso esteja dentro do patamar aceitável (P) é adicionado a S_{lista}. No segundo passo (linha 12), S_{lista} é ordenada de forma decrescente das variáveis relevantes $(SU_{i,c})$. O terceiro passo (linhas 13 a 24) consiste em escolher as variáveis predominantes por meio do conceito de cobertura de Markov [Yu and Liu 2004], ou seja, são eliminadas as variáveis redundantes. Uma a uma as duplas $(SU_{i,j})$ são comparadas a $(SU_{i,c})$ e caso apresentem valores maiores ou iguais, são removidas de S_{lista} e uma nova dupla é carregada até que todas as variáveis sejam testadas. Por fim, os sobreviventes em S_{lista} são atribuídos a S_{otimo} e apresentados como os selecionados.

5 *Experimentos e Resultados*

Os experimentos realizados neste trabalho são baseados na metodologia apresentada na seção 4.3, tendo como princípio básico, reduzir a quantidade de variáveis utilizadas por uma aplicação qualquer. Em domínios nos quais o problema apresenta diversas características, dados meramente quantitativos misturam-se aos qualitativos e a utilização de uma espécie de filtro, fundamentado pelos conceitos de redundância e relevância [Yu and Liu 2004], possibilita a escolha de um subconjunto ótimo de fatores que melhor represente o domínio.

Os experimentos consistem em encontrar a Entropia, que representa o quanto de informação que a variável carrega (relevância), e a Informação Mútua, que é a quantidade de informação de uma variável contida em outra (redundância), de uma série temporal meteorológica. O sistema desenvolvido foi dividido em três etapas distintas e independentes:

1. preparação;
2. processamento;
3. interpretação.

A primeira etapa implica na preparação dos dados que, posteriormente, serão usados pelo sistema. Entende-se por preparação o procedimento de importar, validar e organizar os dados para que estes, padronizados, sejam mais facilmente lidos na próxima etapa.

A segunda etapa, processamento, é a implementação da metodologia em si. Assim, são executados os cálculos pertinentes à metodologia de seleção, e os resultados dos cálculos são armazenados em um arquivo texto formatado. Foi nesta etapa que os conceitos descritos nos capítulos 2 e 3 foram aplicados, resultando em 3 implementações seriais e em 2 implementações paralelas.

As versões seriais foram evolutivamente melhoradas até uma versão considerada ótima, baseada nos princípios de otimização vistos na seção 3.1. Sendo assim, o melhor programa serial foi usado como base para a construção dos programas paralelos.

Os algoritmos paralelos destacam-se pela inclusão das primitivas da biblioteca de comunicação MPI (subseção 3.2.1) e pelas abordagens utilizadas no processo de divisão de tarefas (subseção 3.2.2). Uma das abordagens apresenta uma divisão implícita, e a outra, apresenta subtarefas de tamanhos fixos. As duas versões foram executadas nos dois *clusters* disponíveis (seção 5.1), apresentando resultados interessantes.

A terceira e última etapa do desenvolvimento consiste na interpretação dos resultados da fase anterior. É com base nestes resultados que o algoritmo de interpretação faz a escolha do subconjunto ótimo de variáveis para o modelo em questão.

Além de detalhar melhor estes ítens já mencionados, este capítulo ainda descreve o ambiente computacional e o conjunto de dados utilizados no experimento. Também apresenta e discute os resultados obtidos pela computação, serial e paralela, sobre a aplicação da metodologia de seleção.

5.1 Ambiente Computacional

Neste experimento foram utilizados dois *clusters*, um homogêneo e um heterogêneo, ambos compostos de 9 máquinas, sendo 1 mestre (*master*) e 8 escravos

(*slaves*) com as seguintes configurações de hardware e software:

- homogêneo - *cluster* UTF (CLUTF)
  - Processador Intel® Pentium4 2.8GHz, 16KB de cache L1, 1MB de cache L2; 512MB Memória DDR400; (*master/slave1/slave2/slave3/ slave4/slave5/slave6/slave7/slave8*)
  - Rede *Ethernet* 100Mbps (*Fast Ethernet*);
  - Sistema Operacional Linux Fedora core 3 (*kernel* 2.6.9-1.667); LAM-MPI versão 7.0.6-3.
- heterogêneo - *cluster* UEM (CLUEM)
  - Processador AMD® Athlon 1.0 GHz, 16KB de cache L1, 256KB de cache L2; 512MB Memória DDR333; (*master*)
  - Processador Intel® Pentium4 3.0GHz, 16KB de cache L1, 1MB de cache L2; 512MB Memória DDR333; (*slave1/slave2/slave3*)
  - Processador Intel® Pentium4 3.0GHz, 16KB de cache L1, 512KB de cache L2; 512MB Memória DDR333; (*slave4*)
  - Processador Intel® Pentium4 1.8GHz, 16KB de cache L1, 256KB de cache L2; 512MB Memória DDR333; (*slave5/slave6/slave7/ slave8*)
  - Rede *Ethernet* 100Mbps (*Fast Ethernet*);
  - Sistema Operacional Linux RedHat 9 (*kernel* 2.4.29-1); LAM-MPI versão 7.0.6-3;

As combinações (configurações) do modelo mestre/escravo foram na ordem de 1 mestre para 1, 2, 4 e 8 máquinas escravas. Os algoritmos seriais foram executados no nó "*slave1*" de ambos os *clusters*.

5.2 Conjunto de Dados

O conjunto de dados experimental é constituído de séries temporais meteorológicas, cedidas pelo Instituto Tecnológico SIMEPAR (Sistema Meteorológico do Paraná), provenientes da estação situada na Cidade de Londrina. O SIMEPAR é, desde 1993, o instituto responsável pela geração de dados e informações de natureza hidrológica, meteorológica e ambiental para o Estado do Paraná.

Os dados históricos correspondem a 9 anos, totalizando originalmente 78.888 registros horários, datados de janeiro de 1997 a dezembro de 2005, de 11 tipos diferentes de variáveis. Estas variáveis são listadas a seguir e estão no formato "Nome(quantidade): descrição":

1. temperatura máxima(1): maior temperatura medida na hora (^{o}C);
2. temperatura média(1): média horária das leituras de um termohigrômetro[1] (^{o}C);
3. temperatura mínima(1): menor temperatura medida na hora (^{o}C);
4. umidade relativa(1): média horária das leituras de um termohigrômetro (%);
5. vento[2](2): valor instantâneo da leitura de um anemômetro[3];
    - velocidade (m/s);
    - direção (^{o}): é indicada em graus pelo ângulo formado entre norte e a direção de onde o vento sopra;

[1] Termohigrômetro é um instrumento que permite a leitura de temperatura e umidade ambiente.
[2] A variável Vento possui dois vetores, um para velocidade (chamado Vento1) e outro para direção (chamado Vento2).
[3] Anemômetro é um instrumento para medição de velocidade do ar.

6. rajada[4](2): valor instantâneo da leitura de um anemômetro;
    - velocidade (m/s);
    - direção (^{o}): é indicada em graus pelo ângulo formado entre norte e a direção de onde o vento sopra;
7. precipitação(1): ocorrência de chuva na última hora (mm);
8. pressão(1): média horária das leituras de um barômetro[5] (hPa);
9. radiação(1): média horária das leituras de um piranômetro[6] (W/m2).

Os meteorologistas afirmam que para nominar o clima de uma região (Normal Climatológico) são necessários 30 anos de dados [Pereira et al. 2002], e isto representa nada menos que, aproximadamente, 262.800 registros para cada característica.

5.3 Implementação

5.3.1 Etapa 1 - Preparação

Visando estudar as propriedades da metodologia descrita no capítulo 4, os dados coletados do sistema SIMEPAR passaram por um processo de preparação para serem incluídos ao experimento.

Inicialmente os dados foram importados e tabulados. Baseado em uma nota de esclarecimento a respeito da pré-consistência dos dados meteorológicos, a qual indicava a possibilidade de alguma ausência ou anomalia nas leituras, os dados precisavam ser validados pelo usuário. Sendo assim, quando identificadas essas ocorrências, as tuplas[7] correspondentes de cada uma das variáveis foram

[4] A variável Rajada possui dois vetores, um para velocidade (chamado Rajada1) e outro para direção (chamado Rajada2).

[5] Barômetro é um instrumento para medir a pressão atmosférica.

[6] Piranômetro é um instrumento usado para medir radiação solar.

[7] Tupla é similar ao conceito de uma linha de uma tabela.

suprimidas, ou seja, se por algum motivo não existir a leitura da umidade relativa do dia 21/10/2002 às 20:00 horas, como no exemplo da Figura 5.1, todas as outras variáveis terão este mesmo dia e hora eliminados.

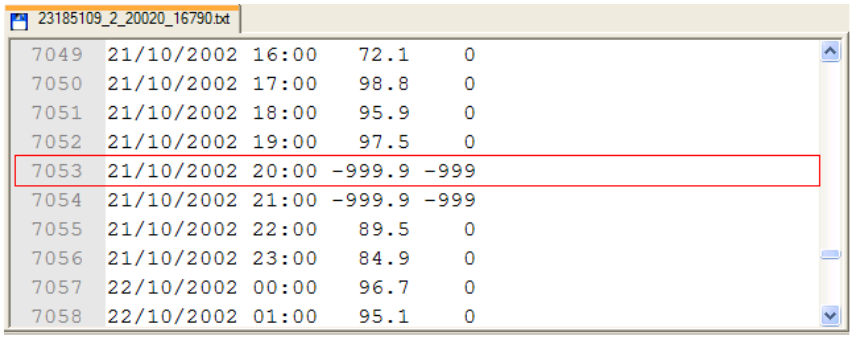

Figura 5.1: Arquivo original mostra a referência "−999", indicando a ausência da leitura.

Após corrigidas as irregularidades descritas anteriormente, a coluna de interesse (valores de umidade) foi exportada para um arquivo no formato texto (ASCII). Este arquivo totalizou 70.900 registros separados por espaços (ver Figura 5.2). Este procedimento foi aplicado a cada uma das outras tabelas e os arquivos gerados foram nomeados com base no seu conteúdo.

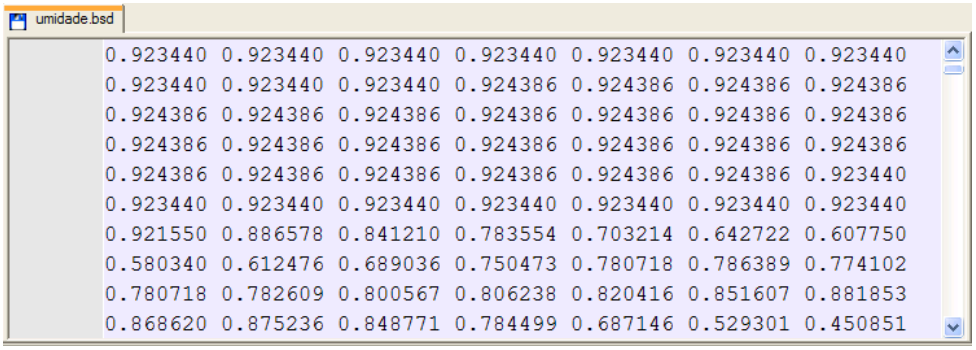

Figura 5.2: Arquivo normalizado e pronto para ser usado.

No último passo, os arquivos foram normalizados, isto é, cada valor foi dividido pelo maior da sua série, para que todos ficassem em uma escala entre 0 e 1. Conforme a necessidade, os vetores foram dimensionados em tamanhos correspondentes a anos completos (ver Tabela 5.1), e para os testes iniciais foram usados vetores de 4.380 registros, representando um histórico de 6 meses.

Tabela 5.1: Equivalência entre ano e quantidade de registros.

Qtde de anos	Qtde de registros
1	8.760
2	17.520
3	26.280
4	35.040
5	43.800
6	52.560
7	61.320
8	70.080

Outro procedimento executado nesta etapa foi a criação de um arquivo-texto, contendo a combinação entre todas as variáveis do conjunto. Sua necessidade será discutida a seguir.

5.3.2 Etapa 2 - Processamento

As referências [Haykin 1999] e [Blum and Langley 1997] indicam um alto custo computacional para o processo de treinamento de uma RNA, mas Yu e Liu em [Yu and Liu 2004] afirmam que para alguns domínios, a seleção de variáveis e características também apresenta um alto custo. A justificativa é a diversidade de variáveis e/ou o grande volume de dados, necessários para caracterizar o problema. Este custo pode ser comprovado observando os tempos de execução dos algoritmos descritos neste experimento.

Vale ressaltar que é nesta subseção que estão concentrados todos os esforços contextualizados nos capítulos anteriores e, passo a passo, é descrito e discutido todo o processo de desenvolvimento dos algoritmos. Quando se deseja obter alto desempenho, cada detalhe do algoritmo precisa ser minuciosamente observado, e as técnicas de otimização e paralelização devem ser empregadas para aumentar o desempenho do programa sem perder o seu propósito original.

Baseando-se no conjunto de dados descrito na subseção 5.2, para um total de 11 vetores tem-se igual número de Entropia, e combinando esses vetores dois a dois, realizam-se 55 cálculos de Informação Mútua.

A Entropia de uma variável é o resultado de uma seqüência de outros cálcu-

los dependentes. Como observado no diagrama da Figura 5.3, para calcular a Entropia(x), onde x representa uma variável qualquer do experimento, precisa-se do valor da Energia(x), que internamente usa o valor da Variância(x), que usa o valor da Média(x). Essa relação de dependência é em parte um complicador para a computação, pois inibe o reaproveitamento de recurso.

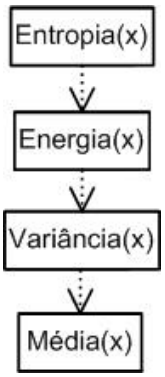

Figura 5.3: Seqüência e dependência para o cálculo da Entropia.

O segundo cálculo exigido pela metodologia é a Informação Mútua, que leva em conta a manipulação de duas variáveis simultâneas, ou InformaçãoMútua(x, y). Inicialmente, calcula-se para a variável x o ParcialMarginal(x) e a Energia(x), depois, os mesmos cálculos são realizados para a variável y, ParcialMarginal(y) e Energia(y), e finalmente, concatenam-se as duas variáveis e é calculada a Energia(x, y). De posse de todos esses valores, pode-se encontrar a Correlação (Informação Mútua) entre as duas variáveis. O diagrama esquemático do cálculo e suas dependências são mostrados na Figura 5.4.

Construído de uma forma didática, o experimento foi implementado no software Octave[8], e manualmente os dados foram carregados em vetores e os cálculos executados pelo hardware descrito na seção 5.1. Com um desempenho ruim e um volume pequeno de dados, os cálculos parciais foram sendo executados um a um, e o resultado final obtido. A metodologia estava validada, pois encontrou os mesmos resultados descritos na bibliografia. Assim, consolidou-se a possibilidade de adicionar mais variáveis e aumentar o acervo de dados.

[8]GNU/Octave é uma linguagem de alto nível voltada para análise numérica, similar ao Matlab®.

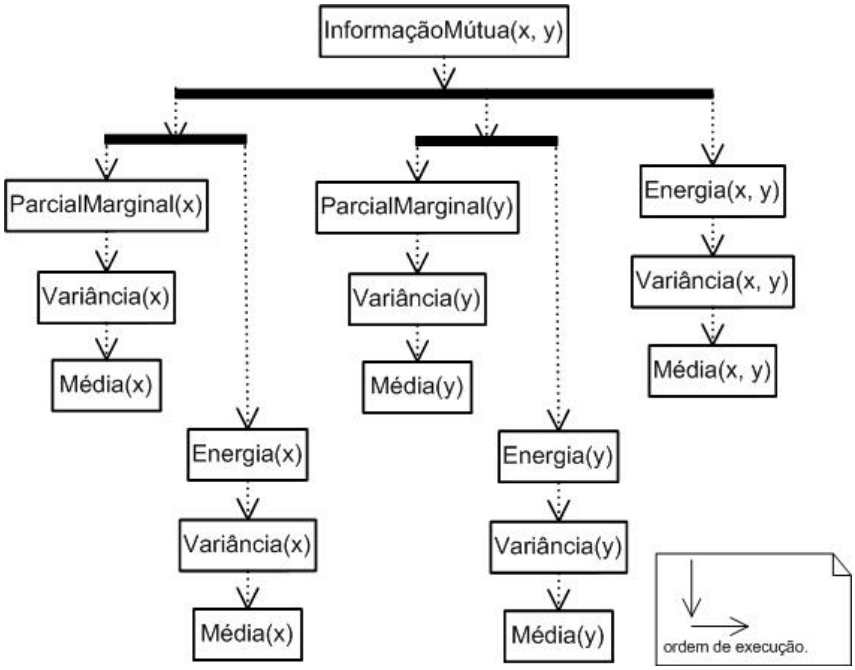

Figura 5.4: Seqüência e dependência para o cálculo da Informação Mútua.

Versão Serial Original (VS1)

A primeira versão serial do programa teve basicamente a intenção de transcrever o código do Octave para a Linguagem C, que já nos primeiros testes propiciou uma maior flexibilidade no volume de dados e uma pequena diminuição do tempo total de execução.

A obtenção do perfil de tempos de execução das subrotinas que compõem o programa fornece subsídios para identificar os gargalos de tempo de utilização de CPU. Assim, os resultados obtidos pelo *profiler*[9], apresentados na Figura 5.5, indicam as subrotinas mais problemáticas e uma grande incoerência na quantidade de cálculos executados.

A interpretação do relatório de saída do *profiler* "gprof", Figuras 5.5 e 5.6, dá-se da seguinte forma: na coluna "% tempo", tem-se a percentagem usada por uma determinada subrotina sobre o tempo total de execução do programa. A próxima coluna, "cumulativo", apresenta a soma cumulativa, em segundos, do

[9]Profilers são ferramentas de predição e análise de desempenho que identificam, dentre outras características, quais as funções que consomem maior tempo de processamento e quantas vezes são chamadas.

tempo já gasto. A coluna "si mesmo" mostra em segundos o tempo gasto somente nesta subrotina, e a coluna "chamadas" apresenta o número de vezes que ela foi invocada. As próximas colunas, "si mesmo s/chama" e "total s/chama" mostram, respectivamente, o tempo médio gasto na subrotina e o tempo médio gasto por ela e seus descendentes. Finalmente, a coluna "nome" apresenta o nome das subrotinas que foram analisadas.

Assim, observa-se por exemplo que, a função "calc_variancia" embora não exija muito tempo de processamento, fora chamada 275 vezes, sendo que, baseado em 11 vetores, deveriam existir 66 chamadas.

```
Cada amostra conta como 0.01 seconds.
  %   cumulativo si mesmo          si mesmo   total
tempo   seconds   seconds chamadas  s/chama  s/chama  nome
 78.25    142.20   142.20      165     0.86     0.86  calc_energia
 21.65    181.55    39.35      110     0.36     0.36  calc_parcialMarginal
  0.04    181.62     0.07      275     0.00     0.00  calc_media
  0.03    181.67     0.05       55     0.00     0.00  concatenaVetor
  0.01    181.69     0.02      275     0.00     0.00  calc_variancia
  0.01    181.71     0.02      110     0.00     0.00  carregaVetor
  0.01    181.73     0.02       55     0.00     3.30  calc_iMutua
```

Figura 5.5: Resultado do *profiler* da primeira versão do algoritmo (versão original).

Versão Serial Organizada (VS2)

Para solucionar o problema do excesso de chamadas aos cálculos, na segunda versão do programa foi realizada a seguinte alteração: conforme as operações sobre os vetores eram realizadas, seus resultados eram armazenados em variáveis indexadas e também em arquivos textos, para um posterior reaproveitamento. Assim, foram eliminadas as chamadas desnecessárias e uma maior coerência na quantidade de invocações pôde ser confirmada pelo segundo relatório do *profiler*, mostrado na Figura 5.6.

Na Tabela 5.2 comprova-se a eficiência das melhorias ao observar os tempos de execução dos algoritmos seriais. A coluna "Dado" mostra o tamanho dos

```
Cada amostra conta como 0.01 seconds.
  %   cumulativo si mesmo          si mesmo   total
tempo   seconds   seconds chamadas  s/chama  s/chama  nome
 73.17     94.90    94.90       66     1.44     1.44  calc_energia
 26.80    129.68    34.78       66     0.32     0.32  calc_parcialMarginal
  0.01    129.69     0.01       66     0.00     0.00  calc_media
  0.00    129.69     0.00      110     0.00     0.00  carregaVetor
  0.00    129.69     0.00      110     0.00     0.00  testa_calculados
  0.00    129.69     0.00       66     0.00     0.00  calc_variancia
  0.00    129.69     0.00       55     0.00     0.00  concatenaVetor
```

Figura 5.6: Resultado do *profiler* da segunda versão do algoritmo (versão organizada).

vetores: 4.380, 8.760, 17.520 e 26.280 registros. As colunas "VS1", "VS2" e "VS3", correspondem aos nomes das implementações e mostram, em segundos, o tempo total de suas execuções. Por fim, as colunas "Ganho 1-2", "Ganho 2-3" e "Ganho Total", trazem em percentagem, a economia de tempo entre as versões 1 e 2, versões 2 e 3 e versões 1 e 3, respectivamente.

Tabela 5.2: Valores comparativos entre as versões seriais dos algoritmos.

Dado	**VS1 T(seg)**	**Ganho 1-2**	**VS2 T(seg)**	**Ganho 2-3**	**VS3 T(seg)**	**Ganho Total**
4.380	1.733	15,98%	1.456	48,30%	753	56,56%
8.760	7.001	16,77%	5.827	48,31%	3.012	56,98%
17.520	28.000	16,87%	23.277	48,25%	12.046	56,98%
26.280	60.243	13,09%	52.357	48,27%	27.086	55,04%

Analisando o vetor de 17.520 registros, observa-se que o "Ganho 1-2" foi de 16,87%, o que representa neste exemplo 4.723 segundos, ou seja, uma economia de processamento de 01h 18m 43s, para uma aplicação que demandaria 28.000 segundos ou 07h 46m 40s. As demais execuções apresentaram ganhos percentuais próximos, 15,98%, 16,77% e 13,09%, respectivamente para vetores de 4.380, 8.760 e 26.280 registros.

Voltando a analisar a Figura 5.6, pode-se visualizar os resultados percentuais de tempo consumido pelas funções, apresentado na primeira coluna do *profiler*. Observa-se que, as subrotinas que mais consomem tempo da execução da aplicação também são as responsáveis pelos cálculos propostos pela metodologia

de seleção. Esses cálculos são procedimentos matemáticos embutidos em laços que se repetem centenas ou milhares de vezes. Após uma minuciosa avaliação dessas subrotinas, a terceira versão do programa foi implementada.

Versão Serial Otimizada (VS3)

Depois de corrigidas as discrepâncias de quantidades de chamadas, o enfoque passou a ser o percentual de tempo gasto pelas subrotinas, como mencionado anteriormente. A priori, atacou-se as de maiores índices, pois elas representam o gargalo para a aplicação.

A estratégia adotada nesta terceira versão foi a utilização das técnicas de redução de condicionais (ver subseção 3.1.2) e reestruturação de laços (ver subseção 3.1.3). Terminada esta etapa de otimização, procurou-se ainda a possibilidade de trocar expressões por variáveis contendo o seu resultado e também a substituição de funções matemáticas definidas na biblioteca da linguagem por operações aritméticas correspondentes, conforme visto na subseção 3.1.1.

Como se trata de uma aplicação real, de características particulares, nem todas as técnicas descritas na seção 3.1 puderam ser implantadas. Também é sabido que aplicações distintas podem apresentar performances diferentes para cada uma delas. Mas, apesar de serem muitos os fatores que influenciam no desempenho de um algoritmo, o uso das técnicas tradicionais de otimização na transição da "VS2" para a "VS3", apresentou resultados satisfatórios, como visto na coluna "Ganho 2-3" da Tabela 5.2.

Ainda observando o vetor de 17.520 registros, nota-se que os 48,25% de economia de tempo representam 9.149 segundos (03h 07m 11s). Para as demais avaliações, os resultados mostraram também valores superiores a 48%.

Num comparativo entre a "VS1" e a "VS3", os ganhos são superiores a 55%, como pode ser visto na coluna "Ganho Total". Isto representa mais da metade

do tempo total de execução, para quaisquer acervos analisados. Em vetores de 26.280 registros, reduz-se de 60.243 segundos (16h 44m 03s) para 27.086 segundos (07h 31m 26s), ou seja, 55,04% de economia, que em segundos daria 33.157 (09h 12m 37s). Reforçando a tendência, na Tabela 5.3 é apresentado um resumo dos percentuais e dos tempos economizados na execução na aplicação.

Tabela 5.3: Redução tempo de execução entre as versões VS1 e VS3.

Dado	**VS1**	**VS3**	**Ganho Total**		
	T(seg)	**T(seg)**	**%**	**T(seg)**	**T(h m s)**
4.380	1.733	753	56,56%	980	(00h 16m 20s)
8.760	7.001	3.012	56,98%	3.989	(01h 06m 29s)
17.520	28.000	12.046	56,98%	15.954	(04h 25m 54s)
26.280	60.243	27.086	55,04%	33.157	(09h 12m 37s)

Até então, todos os esforços de melhoria foram executados sobre as versões seriais e os marcos alcançados são expressivos no quesito desempenho. Atingir economias de tempo superiores a 55%, somente aperfeiçoando o código serial, reforça ainda mais a teoria de Cunha "[...] não havendo qualquer sentido em usar um conjunto de processadores antes de executar a aplicação tão rápido quanto possível em um único processador [...]" [Cunha 2004]. Vale ressaltar também que, diminuindo em mais da metade o tempo total de execução de uma aplicação, pode-se até, em alguns casos, ser descartada a necessidade da utilização das técnicas de computação paralela.

Embora cumprido parte do objetivo proposto, que é diminuir o tempo total de execução da aplicação, os acervos de dados ainda estariam aquém do esperado para o domínio em questão. Analisando o gráfico apresentado na Figura 5.7, observa-se que o crescimento do fator tempo está diretamente ligado ao crescimento do vetor, o que neste caso já era esperado, mas a progressão do crescimento é muito significativa. Quando se dobra o tamanho do vetor (4.380 para 8.760), quadruplica-se o tempo (753 segundo para 3.012 segundos), ou seja, o tempo cresce 300%. Isto implica dizer que, projetando a utilização de um vetor de 8 anos, que contém 70.080 registros, haveria demanda de aproxima-

damente 190.000 segundos ou mais de 52h de processamento. Já para a Normal Climatológica são necessários 262.800 registros, o que resultaria em cerca de 2.850.000 segundos, ou 791h, ou ainda 33 dias para finalizar a execução.

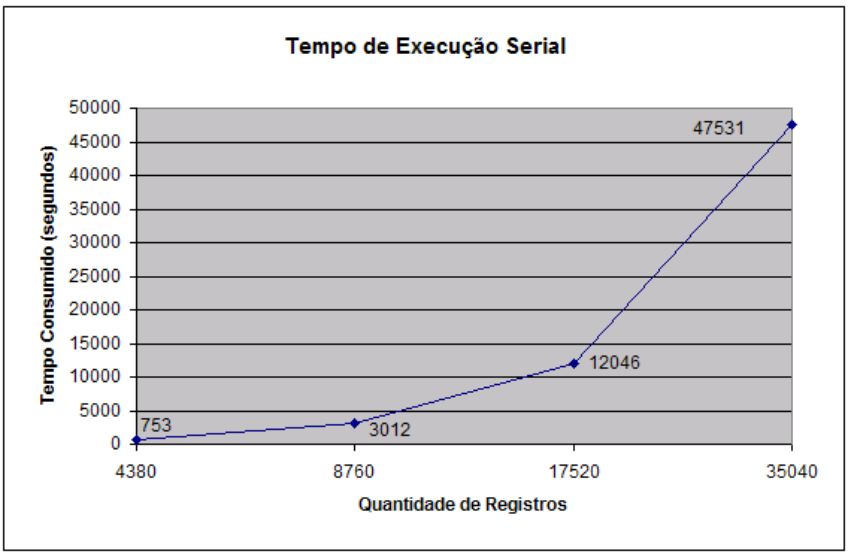

Figura 5.7: Crescimento do tempo de execução do algoritmo serial.

Sendo assim, no passo seguinte, as rotinas da biblioteca MPI foram incorporadas ao algoritmo para possibilitar sua execução em paralelo.

Algoritmo Paralelo Divisão Implícita (APDI)

Após aplicadas as melhorias ao algoritmo seqüencial, passa-se a buscar outras alternativas para reduzir ainda mais o tempo total de execução deste algoritmo, e a encontrada e implementada neste trabalho diz respeito à computação paralela baseada em *cluster* MPI. Várias são as vantagens apresentadas por esta arquitetura, dentre elas o excelente custo/benefício. Mesmo não sendo de implementação trivial, a programação paralela baseada em MPI pode ser usada eficientemente, com certa dose de atenção e bom senso.

Em uma das abordagens de programação com MPI sabe-se que todos os processos executam uma instância[10] do mesmo algoritmo, embora, cada processo,

[10]Instância é cada uma das execuções de um programa, realizadas durante uma mesma seção do sistema operacional.

pela sua identificação própria (*rank*), execute ou não alguns trechos. Um exemplo desta abordagem, escolhida para este trabalho, pode ser visto na Figura 5.8, em que apenas o nó com identificador igual a 0 ($nIdTarefa == 0$) executará as linhas de 202 até 207.

paralelo.c

```
199         }
200         MPI_Bcast( &nOK, 1, MPI_INT, TAREFA_MESTRE, MPI_COMM_WORLD );
201
202         if ( nIdTarefa == TAREFA_MESTRE )
203         {
204             swap_tempo();
205             carregaVetor( vetorY, nCarac, tam );
206             swap_tempo();
207         }
208         MPI_Bcast( vetorY, tam, MPI_DOUBLE, TAREFA_MESTRE, MPI_COMM_WORLD );
209         if( vFeitos[nOK] == 0 )
```

Figura 5.8: Algoritmo mostrando as linhas executadas no MPI baseado no *rank*.

Conforme foi visto na subseção 3.2.1, a biblioteca MPI implementa a comunicação entre os processos através da troca explícita de mensagens. Além das primitivas básicas de envio e recebimento, MPI_Send() e MPI_Recv(), algumas outras funções mais avançadas facilitam principalmente a comunicação global, como: MPI_Bcast(), MPI_Gather() e MPI_Reduce(). Cabe mencionar que esta versão paralela (APDI) foi escrita utilizando estas funções.

Segundo [Amdahl 1967], todo programa paralelo tem uma porção serial, ou seja, uma parte não paralelizável. Com base nisto, analisam-se os tempos de execução do algoritmo, no intuito de avaliar o custo desta porção serial. Seus valores estão resumidos na Tabela 5.4. Observa-se que, proporcionalmente, este tempo é irrisório na grande maioria dos casos, porém é estratégico para o balanceamento do conjunto hardware/software.

Tabela 5.4: Tempos da porção serial vs porção paralela.

	4.380 registros		8.760 registros		17.520 registros		26.280 registros	
Nós	**Serial T(seg)**	**Paralelo T(seg)**	**Serial T(seg)**	**Paralelo T(seg)**	**Serial T(seg)**	**Paralelo T(seg)**	**Serial T(seg)**	**Paralelo T(seg)**
M + 1	0,78	886,90	1,54	3.549,95	2,97	14.191,98	5,87	31.915,31
M + 2	0,78	445,98	1,54	1.782,76	2,94	7.124,57	4,13	16.020,40
M + 4	0,88	225,51	1,61	895,95	2,81	3.572,67	4,10	8.026,75
M + 8	0,84	118,88	1,97	458,10	3,31	1.805,37	4,80	4.041,71

A Tabela 5.4 mostra nas suas colunas o tempo gasto pelas porções serial e paralela dos vetores de 4.380, 8.760, 17.520 e 26.280 registros. Cada linha representa uma configuração mestre/escravo para os ambientes com 1, 2, 4 e 8 nós escravos. Com um tempo de 0,78 segundos contra 445,98 segundos da porção paralela, para vetores de 4.380 registros, em um *cluster* com 2 nós escravos, esta porção serial não chega a representar 0,2% do custo total de execução. Esta disparidade é ainda intensificada conforme se aumenta o tamanho do conjunto. Mas nem mesmo com a adição de mais nós ao *cluster* este valor torna-se representativo.

Para monitorar a execução da aplicação, foi utilizado o software "Ganglia". Com o monitoramento, observa-se que o nó mestre fica sobrecarregado se executar a porção serial e uma parte da paralela. A quantidade de trabalho do nó mestre chega a 263%, enquanto os nós escravos oscilam ininterruptamente, com pouca e muita carga (ver Figura 5.9). Retirando do nó mestre esta porção de processamento paralelo, embora seja subtraída uma unidade computacional do *cluster*, os nós escravos passam a trabalhar mais uniformemente, como pode ser visto na Figura 5.10.

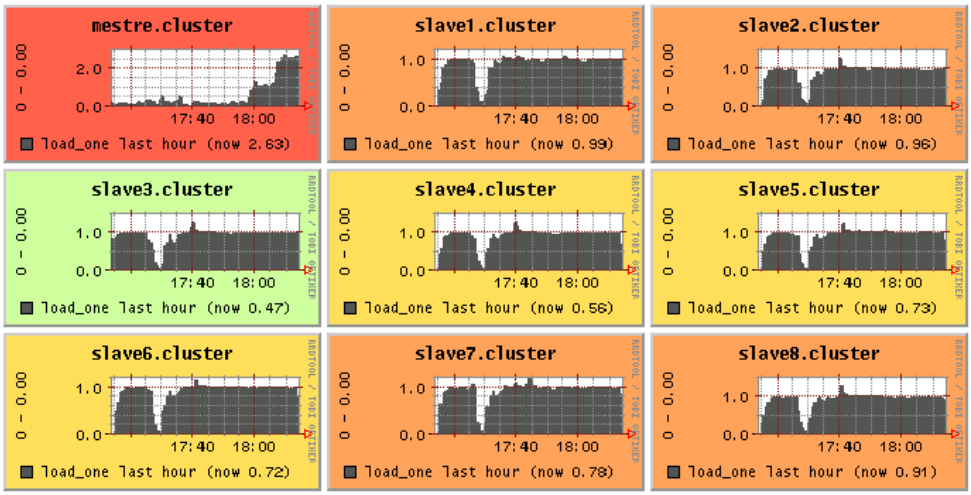

Figura 5.9: Nó mestre sobrecarregado e aplicação desbalanceada (relatório Ganglia).

Sendo assim, seguindo esta estratégia, o nó mestre não é computado no ambiente paralelo. Mas o tempo consumido por ele para fazer o carregamento

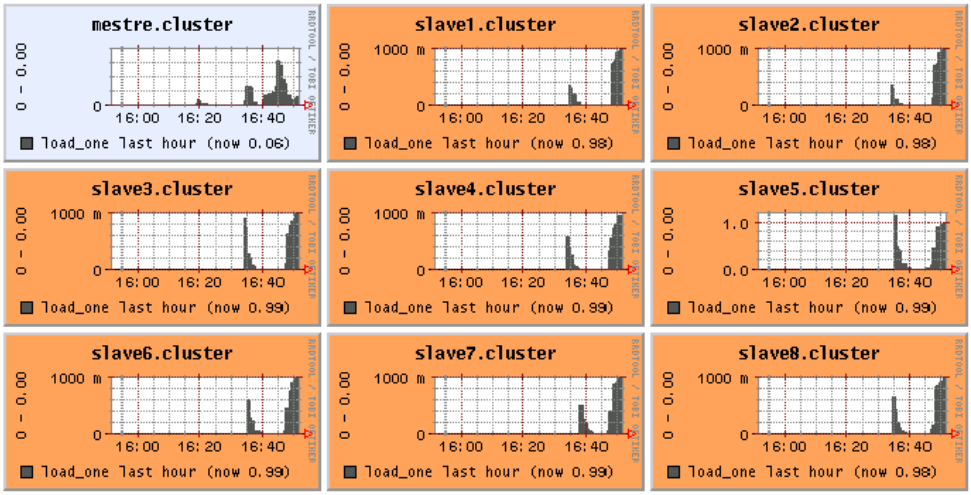

Figura 5.10: Uniformidade de carga de trabalho nos nós escravos (relatório Ganglia).

dos vetores (baseado em arquivos), o envio e o recebimento de dados, e a divisão e o sincronismo das tarefas, foi acrescentado ao tempo total.

A primeira versão paralela do experimento é caracterizada por uma divisão implícita dos dados, doravante chamada Algoritmo Paralelo Divisão Implícita (APDI), ou seja, a divisão da tarefa é feita pelo próprio nó executor (escravo), conforme observado na Listagem 5.1. Tem-se aqui um exemplo de granularidade grossa, em que os nós farão pouca comunicação e serão responsáveis por manipular, de uma só vez, toda sua porção de dados. Desta forma, cada nó escravo opera sobre a mesma quantidade de dados, podendo variar em apenas 1 unidade.

```
1  if( SOU MESTRE )
2  {
3          envia vetor para os escravos;
4          recebe resultado dos escravos;
5  }
6  else    // NAO SOU MESTRE
7  {
8          recebe vetor do mestre;
9          for( id = rank; id < TAM_VETOR; id += QTD_NOS_ESCRAVOS )
10         {
11                 executa calculos;       // sobre os id
12         }
13         envia resultado para o mestre;
14 }
```

Listagem 5.1: Tarefas Iguais.

Na Listagem 5.1, somente o nó mestre executa as linhas de 2 a 5, enquanto os demais nós executam as linhas de 7 a 14. O nó mestre envia o vetor de dados (inteiro) a todos os escravos e fica aguardando a resposta de cada um deles. Após receber o vetor de dados, cada nó escravo inicia a varredura do vetor com índice (id), de valor idêntico ao seu próprio *rank*. A cada iteração do laço o índice é incrementado em quantidade correspondente a de escravos do *cluster* (QTD_NOS_ESCRAVOS). Com isto, consegue-se percorrer todo o vetor e cada nó escravo manipula uma determinada quantidade não seqüencial dos dados.

A Tabela 5.5 mostra as informações das execuções do algoritmo APDI no *cluster* homogêneo (CLUTF). A coluna "Dado" apresenta a quantidade de registros no vetor, a coluna "Nós" indica a quantidade de nós escravos da configuração mestre/escravo. A coluna "Tempos" traz em horas, minutos e segundos (e também somente em segundos) o tempo total de cada execução. A coluna Real mostra em números o *speedup* Real, que usa o tempo do melhor programa serial executado em uma máquina, pelo tempo do programa paralelo executado em *n* máquinas, e em percentagem a eficiência, que é o nível de aproveitamento de cada nó. A coluna Relativo apresenta a mesma estrutura da coluna Real, mas está baseada no *speedup* Relativo, que é a razão entre o tempo do programa paralelo executado em uma máquina pelo seu tempo executado em *n* máquinas.

Na Tabela 5.5, observa-se que a progressão de crescimento do tempo total de execução em relação ao tamanho do vetor, manteve-se próxima a 300%, mesmo índice alcançado nas versões seriais. Dobrando-se o tamanho do vetor, de 4.380 registros para 8.760 registros, o tempo é aproximadamente quadruplicado. Por exemplo, para *cluster* com 1 nó escravo, passa-se de 888 segundos para 3.551 segundos; com 4 nós escravos, de 226 segundos para 898 segundos. Mesmo com acervos maiores, os índices se mantiveram. Na transição de 8.760 registros para 17.520 registros, para *cluster* com 4 nós escravos, passa-se de

Tabela 5.5: Dados das execuções do algoritmo APDI no *cluster* CLUTF.

		Tempos				Real		Relativo	
Dado	**Nós**	**H**	**M**	**S**	**Segundos**	**Speedup**	**Eficiência**	**Speedup**	**Eficiência**
4.380	M + 1	0	14	48	888	0,85	84,80%	1,00	100,00%
	M + 2	0	7	27	447	1,68	84,24%	1,99	99,35%
	M + 4	0	3	46	226	3,32	83,12%	3,92	98,02%
	M + 8	0	1	60	120	6,29	78,60%	7,41	92,69%
8.760	M + 1	0	59	11	3.551	0,85	84,80%	1,00	100,00%
	M + 2	0	29	44	1.784	1,69	84,40%	1,99	99,52%
	M + 4	0	14	58	898	3,36	83,89%	3,96	98,92%
	M + 8	0	7	40	460	6,55	81,83%	7,72	96,49%
17.520	M + 1	3	56	35	14.195	0,85	84,86%	1,00	100,00%
	M + 2	1	58	48	7.128	1,69	84,50%	1,99	99,58%
	M + 4	0	59	35	3.575	3,37	84,23%	3,97	99,25%
	M + 8	0	30	9	1.809	6,66	83,25%	7,85	98,10%
26.280	M + 1	8	52	1	31.921	0,85	84,85%	1,00	100,00%
	M + 2	4	27	5	16.025	1,69	84,51%	1,99	99,60%
	M + 4	2	13	51	8.031	3,37	84,32%	3,97	99,37%
	M + 8	1	7	27	4.047	6,69	83,67%	7,89	98,61%

Tabela 5.6: Tempos seriais executados no *slave1* do *cluster* CLUTF.

	Tempos			
Dado	**H**	**M**	**S**	**Segundos**
4.380	0	12	33	753
8.760	0	50	12	3.012
17.520	3	20	46	12.046
26.280	7	31	26	27.086

898 segundos para 3.575 segundos; com 8 nós escravos, de 460 segundos para 1.809 segundos.

Ao confrontar-se o tempo de execução da versão paralela (Tabela 5.5) com a versão serial (Tabela 5.6), observa-se que o algoritmo APDI mostra-se mais lento que o algoritmo VS3, quando o APDI é executado no *cluster* com apenas 1 nó. No entanto, o APDI ao contrário do serial, pode ser executado em outras configurações do *cluster*, com mais nós escravos, apresentando assim benefícios. A essência do *speedup* Real é justamente mostrar a aceleração obtida na transição dos algoritmos, destacando-se o fato de que esta comparação se dá entre algoritmos diferentes.

Ressalta-se que, para um vetor de 4.380 registros, o valor do *speedup* Real é igual a 0,85, o que representa a desaceleração discutida anteriormente. Isto implica dizer que, enquanto VS3 realiza 100 partes de trabalho, o APDI realizaria apenas 85 partes. Tal fato pode ser evidenciado pelo valor da eficiência (Real), que é de 84,80%. Para um vetor de 26.280 registros, executado em um *cluster*

com 4 nós escravos, o *speedup* Real é 3,37 com uma eficiência de 84,32%, ou seja, o percentual não aproveitado para processamento (15,68%), representado pela diferença faltante a 100% da eficiência, é o que impossibilita o algoritmo paralelo ser executado 4 vezes mais rápido, como teoricamente é esperado com o uso de 4 máquinas.

A coluna Relativo expressa bem o nível de escalabilidade da aplicação. Sua importância é destacada, pois mostra o comportamento do mesmo algoritmo sendo executado em diferentes configurações do *cluster*. Observa-se uma uniformidade entre os valores, e um leve ganho de eficiência com o crescimento do vetor. Por exemplo, em um *cluster* de 8 máquinas, a transição de um vetor de 8.760 registros para 17.520 registros, aumenta o speedup de 7,72 para 7,85 e a eficiência de 96,46% para 98,10%.

Em *clusters* com 1 nó apenas, o *speedup* Relativo sempre será 1 e a eficiência 100%, isto porque, a fórmula do *speedup* Relativo, como já foi vista, divide o tempo do algoritmo paralelo executado em 1 máquina pelo seu tempo executado em n máquinas, e neste caso n é igual a 1, ou seja, divide-se ele por ele mesmo.

O *speedup* Ideal ou Linear é quando se tem um valor igual à quantidade de máquinas usadas no *cluster*. Então, entende-se que os valores do *speedup* distanciando do ideal e a eficiência diminuindo conforme são adicionados nós ao *cluster*, representam o prejuízo obtido com o crescimento do ambiente paralelo. Isto já era esperado, mesmo porque as bibliografias afirmam que o *speedup* Ideal é difícil de ser alcançado. O algoritmo APDI no *cluster* CLUTF apresentou valores satisfatórios, pois no pior caso, observa-se um *speedup* Relativo de 7,41 com eficiência de 92,69%, para vetores de 4.380 registros.

A Tabela 5.7 mostra as informações das execuções do algoritmo APDI no *cluster* heterogêneo (CLUEM). A descrição do seu cabeçalho e a interpretação dos resultados são semelhantes às da Tabela 5.5.

Tabela 5.7: Dados das execuções do algoritmo APDI no *cluster* CLUEM.

		Tempos				Real		Relativo	
Dado	**Nós**	**H**	**M**	**S**	**Segundos**	**Speedup**	**Eficiência**	**Speedup**	**Eficiência**
4.380	M + 1	0	15	5	905	0,83	83,14%	1,00	100,00%
	M + 2	0	7	35	455	1,65	82,68%	1,99	99,44%
	M + 4	0	3	50	230	3,27	81,78%	3,93	98,36%
	M + 8	0	3	11	191	3,94	49,27%	4,74	59,26%
8.760	M + 1	1	0	17	3.617	0,83	83,14%	1,00	100,00%
	M + 2	0	30	13	1.213	1,66	82,93%	1,99	99,74%
	M + 4	0	15	12	912	3,30	82,42%	3,97	99,13%
	M + 8	0	12	32	752	4,00	50,00%	4,81	60,14%
17.520	M + 1	4	1	10	14.470	0,83	83,47%	1,00	100,00%
	M + 2	2	0	47	7.247	1,67	83,34%	2,00	99,84%
	M + 4	1	0	36	3.636	3,32	83,04%	3,98	99,49%
	M + 8	0	49	46	2.986	4,04	50,56%	4,85	60,57%
26.280	M + 1	9	2	30	32.550	0,83	83,43%	1,00	100,00%
	M + 2	4	31	33	16.293	1,67	83,33%	2,00	99,89%
	M + 4	2	16	9	8.169	3,32	83,11%	3,98	99,61%
	M + 8	1	57	27	7.047	3,85	48,17%	4,62	57,74%

Com um comportamento similar ao apresentado pelo algoritmo APDI no *cluster* CLUTF, os dados mais significativos da Tabela 5.7 são os resultados díspares para as configurações de *cluster* com 8 máquinas. O *speedup*, tanto Real quanto Relativo, afasta-se e muito do ideal. Inclusive, fica distante até dos valores apresentados pela execução no *cluster* CLUTF. As respectivas eficiências também estão bem aquém das idealizadas.

Uma amostra de que o *speedup* sozinho pode camuflar um resultado, o valor 3,94 seria muito bom se fosse obtido por uma configuração com 4 nós, entretanto, é o resultado da execução de vetores de 4.380 registros em um *cluster* com 8 máquinas. Ainda na Tabela 5.7, outros exemplos como este podem ser observados.

Combinar o *speedup* com a eficiência facilita a interpretação e evita desacertos. Para o exemplo citado anteriormente, a eficiência é de 49,27%, ou seja, as 8 máquinas dispostas no *cluster* realizaram uma média de processamento útil menor que a metade do tempo consumido. Sucintamente, a Tabela 5.7 mostra que o objetivo de diminuir o tempo total de execução da aplicação, para os vetores analisados está acontecendo, porém com recursos subutilizados. Isto se dá porque o *cluster* CLUEM é heterogêneo, e para esta configuração, são adicionadas 4 máquinas (slave5/slave6/slave7/slave8) de desempenho individual

inferior, conforme se pode conferir na seção 5.1. Lembrando que a estratégia adotada pelo algoritmo APDI é distribuir para todos os nós a mesma quantidade e tamanho de tarefa, então a mesma carga de trabalho que é submetida ao nó rápido é também submetida ao nó mais lento. Portanto, todo o ambiente paralelo fica limitado pelo nó mais lento.

A Tabela 5.8 mostra um comparativo da execução do algoritmo APDI entre os *clusters* CLUTF e CLUEM. Novamente, a coluna "Dado" apresenta a quantidade de registros no vetor e a coluna "Nós" a quantidade de nós escravos da configuração mestre/escravo. A coluna "CLUTF" e a coluna "CLUEM" mostram em segundos o tempo total de execução do algoritmo para os respectivos *clusters*. A coluna "DD" corresponde, em percentagem, à diferença de desempenho entre os *clusters*. Cabe ressaltar que o intuito não é classificar o melhor *cluster*, mas enfatizar o problema apresentado pela configuração com 8 nós escravos do *cluster* CLUEM.

Tabela 5.8: Diferença de desempenho entre os *clusters* CLUTF e CLUEM (Algoritmo APDI).

Dado	Nós	CLUTF T(Seg)	CLUEM T(Seg)	DD %
4.380	M + 1	888	905	1,94%
	M + 2	447	455	1,84%
	M + 4	226	230	1,59%
	M + 8	120	191	59,44%
8.760	M + 1	3.551	3.617	1,85%
	M + 2	1.784	1.813	1,62%
	M + 4	898	912	1,63%
	M + 8	460	752	63,41%
17.520	M + 1	14.195	14.470	1,94%
	M + 2	7.128	7.247	1,67%
	M + 4	3.575	3.636	1,69%
	M + 8	1.809	2.986	65,11%
26.280	M + 1	31.921	32.550	1,97%
	M + 2	16.025	16.293	1,68%
	M + 4	8.031	8.169	1,72%
	M + 8	4.047	7.047	74,14%

A limitação imposta pelos nós mais lentos apresentou índices de até 74,14%. Ainda observa-se na Tabela 5.8 uma tendência de crescimento desta diferença com o aumento do tamanho do vetor. Os demais valores da coluna "DD", que variam entre 1,59% e 1,97%, são aceitáveis.

Algoritmo Paralelo Divisão Fixa (APDF)

Com o intuito de diminuir a diferença de desempenho entre as arquiteturas homogênea e heterogênea, e melhor aproveitar o potencial das máquinas pertencentes ao *cluster* CLUEM, foi implementada uma segunda versão paralela para o experimento, nomeada Algoritmo Paralelo Divisão Fixa (APDF).

Contrapondo-se, em partes, à distribuição paritária vista anteriormente, o algoritmo APDF tem como princípio básico o trabalho sob demanda. A técnica usada neste algoritmo visa maximizar os recursos pela submissão, por parte do nó mestre aos nós escravos, de subtarefas menores, de tamanho fixo e arbitrário, em uma analogia às aplicações do tipo "*bag of tasks*" das grades computacionais. Portanto, ao final da execução, cada processo manipulará de zero a muitas subtarefas, dependendo da combinação entre o tamanho do vetor e o tamanho da subtarefa. Por exemplo, para vetores de 17.520 registros e subtarefas de 876 registros, forma-se um conjunto contendo 20 subtarefas. Se a análise for para um *cluster* com 4 máquinas, tem-se uma média de 5,00 subtarefas para cada uma; para 8 máquinas, uma média de 2,50 subtarefas e assim sucessivamente.

A Tabela 5.9 mostra todas estas possíveis combinações. A coluna "Dado" apresenta a quantidade de registros no vetor, a coluna "Subtarefa", representa o tamanho da subtarefa e a coluna "Qtde de Subtarefas" mostra a quantidade total de subtarefas, que é a razão entre as colunas anteriores. Já a coluna "Média de Subtarefas" apresenta, em números, a quantidade média teórica de subtarefas para cada nó, respectivamente para *clusters* com 1, 2, 4 e 8 nós escravos. Na seqüência, a Listagem 5.2 descreve o pseudocódigo da abordagem utilizada no algoritmo APDF.

Observa-se no pseudocódigo que entre as linhas 1 e 11, o nó mestre identifica a quantidade de tarefas (tamanho do vetor dividido pelo tamanho da subtarefa) e entra em um ciclo até que se zere esse valor. Se existir nó ocioso, o

Tabela 5.9: Média de Subtarefas.

Dado	Subtarefa	Qtde de Subtarefas	Média de Subtarefas			
			1	2	4	8
4.380	219	20	20,00	10,00	5,00	2,50
	438	10	10,00	5,00	2,50	1,25
	876	5	5,00	2,50	1,25	0,63
	1.095	4	4,00	2,00	1,00	0,50
8.760	219	40	40,00	20,00	10,00	5,00
	438	20	20,00	10,00	5,00	2,50
	876	10	10,00	5,00	2,50	1,25
	1.095	8	8,00	4,00	2,00	1,00
17.520	219	80	80,00	40,00	20,00	10,00
	438	40	40,00	20,00	10,00	5,00
	876	20	20,00	10,00	5,00	2,50
	1.095	16	16,00	8,00	4,00	2,00
26.280	219	120	120,00	60,00	30,00	15,00
	438	60	60,00	30,00	15,00	7,50
	876	30	30,00	15,00	7,50	3,75
	1.095	24	24,00	12,00	6,00	3,00

```
1  if( SOU MESTRE )
2  {
3          calcula quantidade de tarefas;
4          while( TEM TAREFA )
5          {
6                  if( ALGUEM OCIOSO )
7                          envia tarefa para o nó ocioso;
8                  else
9                          aguarda resposta de qualquer nó;
10         }
11 }
12 else    // NÃO SOU MESTRE
13 {
14         while( ORDEM DE TRABALHO DO MESTRE )
15         {
16                 recebe tarefa do mestre;
17                 executa calculos;
18                 envia resultado para o mestre;
19         }
20 }
```

Listagem 5.2: Distribuição de Tarefas.

mestre envia uma das tarefas para ser executada, caso contrário, fica aguardando uma resposta do escravo, que pode ser proveniente de qualquer um. Os nós escravos executam as linhas de 13 a 20 e trabalham em função das ordens vindas do mestre. Sendo assim, ficam sempre em modo de espera, e tão logo recebam as ordens, executam os procedimentos e devolvem o resultado correspondente.

Tomando como exemplo um vetor de 8.760 registros e subtarefas de 438 registros, verifica-se que há 20 tarefas a serem cumpridas, como pode ser confirmado na Tabela 5.9. Isto representa para um *cluster* com 4 nós escravos, teoricamente, 5 tarefas para cada nó. Como o algoritmo APDF envia as tarefas

sob demanda, na prática esta informação pode ser verdadeira para o *cluster* homogêneo (CLUTF), mas talvez não para o heterogêneo (CLUEM). É justamente este o benefício que o APDF apresenta sobre o APDI, pois quando da existência de várias tarefas, conforme elas vão sendo finalizadas, os nós ociosos recebem outras tarefas. Isto, para um *cluster* heterogêneo, possibilita a amortização do tempo total de execução.

As Tabelas 5.10 e 5.11 reúnem as informações das execuções do algoritmo APDF nos *clusters* CLUTF e CLUEM, respectivamente. A descrição dos cabeçalhos e a interpretação dos resultados são semelhantes às tabelas anteriores, com a inclusão da coluna "Subtarefa", que representa o tamanho da subtarefa, um valor fixo de uma parte do vetor de dados.

Em uma visão geral, nas Tabelas 5.10 e 5.11, observa-se que o algoritmo APDF teve um comportamento inicial melhor que o algoritmo APDI. Isto pode ser confirmado pelos valores dos *speedups* Reais para a configuração de *cluster* com 1 nó de processamento. Com números sempre superiores a 0,95, para quaisquer vetores analisados, em oposição aos 0,85 do algoritmo APDI no *cluster* CLUTF (ver Tabela 5.5) e os 0,83 no *cluster* CLUEM (ver Tabela 5.7). Ainda observa-se que, com subtarefas maiores, este índice é mais evidente.

A Tabela 5.12 mostra novamente um comparativo entre os *clusters* CLUTF e CLUEM, mas, neste caso, pela execução do algoritmo APDF em configurações com 8 nós. Outra vez, a coluna "Dado" apresenta a quantidade de registros no vetor e a coluna "Nós" a quantidade de nós escravos da configuração mestre/escravo. As colunas "CLUTF" e "CLUEM" mostram, em segundos, o tempo total de execução do algoritmo para os respectivos *clusters*. Os valores da coluna "DD" representam em percentagem, a diferença de desempenho entre os *clusters*.

O maior benefício desta abordagem foi apresentado nas configurações com 8 nós, justamente porque ali se caracteriza a heterogeneidade do *cluster* CLUEM.

Tabela 5.10: Dados das execuções do algoritmo APDF no *cluster* CLUTF.

			Tempos				Real		Relativo	
Dado	**Subtarefa**	**Nós**	**H**	**M**	**S**	**Segundos**	**Speedup**	**Eficiência**	**Speedup**	**Eficiência**
4.380	219	M + 1	0	13	14	794	0,95	94,83%	1,00	100,00%
		M + 2	0	6	53	413	1,82	91,04%	1,92	96,00%
		M + 4	0	3	44	224	3,36	83,92%	3,54	88,50%
		M + 8	0	2	13	133	5,67	70,93%	5,98	74,80%
	438	M + 1	0	12	49	769	0,98	97,82%	1,00	100,00%
		M + 2	0	6	34	394	1,91	95,59%	1,95	97,71%
		M + 4	0	3	31	211	3,58	89,38%	3,65	91,37%
		M + 8	0	2	15	135	5,57	69,58%	5,69	71,13%
	876	M + 1	0	12	38	758	0,99	99,35%	1,00	100,00%
		M + 2	0	6	32	392	1,92	96,02%	1,93	96,65%
		M + 4	0	4	1	241	3,12	77,95%	3,14	78,46%
		M + 8	0	2	43	163	4,63	57,89%	4,66	58,27%
	1.095	M + 1	0	12	35	755	1,00	99,67%	1,00	100,00%
		M + 2	0	6	22	382	1,97	98,42%	1,97	98,75%
		M + 4	0	3	16	196	3,84	95,88%	3,85	96,19%
		M + 8	0	3	18	198	3,80	47,47%	3,81	47,63%
8.760	219	M + 1	0	52	49	3.169	0,95	95,03%	1,00	100,00%
		M + 2	0	27	26	1.646	1,83	91,46%	1,92	96,24%
		M + 4	0	14	48	888	3,39	84,82%	3,57	89,25%
		M + 8	0	8	30	510	5,90	73,81%	6,21	77,66%
	438	M + 1	0	51	17	3.077	0,98	97,88%	1,00	100,00%
		M + 2	0	26	12	1.572	1,92	95,79%	1,96	97,87%
		M + 4	0	13	40	820	3,67	91,79%	3,75	93,78%
		M + 8	0	7	34	454	6,64	82,99%	6,78	84,78%
	876	M + 1	0	50	30	3.030	0,99	99,39%	1,00	100,00%
		M + 2	0	25	34	1.534	1,96	98,18%	1,98	98,78%
		M + 4	0	13	22	802	3,76	93,91%	3,78	94,49%
		M + 8	0	8	20	500	6,02	75,29%	6,06	75,75%
	1.095	M + 1	0	50	20	3.020	1,00	99,72%	1,00	100,00%
		M + 2	0	25	27	1.527	1,97	98,64%	1,98	98,92%
		M + 4	0	13	0	780	3,86	96,52%	3,87	96,79%
		M + 8	0	6	49	409	7,37	92,12%	7,39	92,38%
17.520	219	M + 1	3	30	57	12.657	0,95	95,18%	1,00	100,00%
		M + 2	1	49	33	6.573	1,83	91,64%	1,93	96,28%
		M + 4	0	58	51	3.531	3,41	85,29%	3,58	89,62%
		M + 8	0	33	34	2.014	5,98	74,76%	6,28	78,55%
	438	M + 1	3	24	59	12.299	0,98	97,94%	1,00	100,00%
		M + 2	1	44	35	6.275	1,92	95,98%	1,96	98,00%
		M + 4	0	54	26	3.266	3,69	92,22%	3,77	94,15%
		M + 8	0	29	25	1.765	6,83	85,33%	6,97	87,12%
	876	M + 1	3	21	56	12.116	0,99	99,42%	1,00	100,00%
		M + 2	1	42	8	6.128	1,97	98,29%	1,98	98,86%
		M + 4	0	52	15	3.135	3,84	96,05%	3,86	96,61%
		M + 8	0	27	49	1.669	7,22	90,20%	7,26	90,72%
	1.095	M + 1	3	21	21	12.081	1,00	99,71%	1,00	100,00%
		M + 2	1	41	38	6.098	1,98	98,78%	1,98	99,06%
		M + 4	0	51	47	3.107	3,88	96,94%	3,89	97,22%
		M + 8	0	26	55	1.615	7,46	93,21%	7,48	93,48%
26.280	219	M + 1	7	48	5	28.085	0,96	96,44%	1,00	100,00%
		M + 2	4	3	7	14.587	1,86	92,85%	1,93	96,27%
		M + 4	2	10	35	7.835	3,46	86,43%	3,58	89,61%
		M + 8	1	14	33	4.473	6,06	75,70%	6,28	78,49%
	438	M + 1	7	34	38	27.278	0,99	99,30%	1,00	100,00%
		M + 2	3	52	4	13.924	1,95	97,26%	1,96	97,95%
		M + 4	2	0	40	7.240	3,74	93,53%	3,77	94,19%
		M + 8	1	5	30	3.930	6,89	86,16%	6,94	86,76%
	876	M + 1	7	27	58	26.878	1,01	100,77%	1,00	100,00%
		M + 2	3	46	30	13.590	1,99	99,65%	1,98	98,89%
		M + 4	1	56	28	6.988	3,88	96,90%	3,85	96,16%
		M + 8	1	3	54	3.834	7,07	88,31%	7,01	87,64%
	1.095	M + 1	7	26	36	26.796	1,01	101,08%	1,00	100,00%
		M + 2	3	45	25	13.525	2,00	100,13%	1,98	99,06%
		M + 4	1	54	46	6.886	3,93	98,34%	3,89	97,29%
		M + 8	0	59	35	3.575	7,58	94,72%	7,50	93,71%

Tabela 5.11: Dados das execuções do algoritmo APDF no *cluster* CLUEM.

			Tempos				Real		Relativo	
Dado	**Subtarefa**	**Nós**	**H**	**M**	**S**	**Segundos**	**Speedup**	**Eficiência**	**Speedup**	**Eficiência**
4.380	219	M + 1	0	13	14	794	0,95	94,72%	1,00	100,00%
		M + 2	0	6	53	413	1,82	91,13%	1,92	96,21%
		M + 4	0	3	42	222	3,39	84,63%	3,57	89,35%
		M + 8	0	2	37	157	4,78	59,75%	5,05	63,08%
	438	M + 1	0	12	50	770	0,98	97,67%	1,00	100,00%
		M + 2	0	6	34	394	1,91	95,49%	1,96	97,77%
		M + 4	0	3	30	210	3,58	89,49%	3,67	91,63%
		M + 8	0	2	24	144	5,22	65,27%	5,35	66,83%
	876	M + 1	0	12	39	759	0,99	99,18%	1,00	100,00%
		M + 2	0	6	33	393	1,92	95,83%	1,93	96,62%
		M + 4	0	3	58	238	3,16	78,96%	3,18	79,61%
		M + 8	0	4	11	251	3,00	37,49%	3,02	37,80%
	1.095	M + 1	0	12	35	755	1,00	99,61%	1,00	100,00%
		M + 2	0	6	24	384	1,96	98,04%	1,97	98,43%
		M + 4	0	3	17	197	3,83	95,63%	3,84	96,01%
		M + 8	0	3	17	197	3,82	47,78%	3,84	47,97%
8.760	219	M + 1	0	52	43	3.163	0,95	95,07%	1,00	100,00%
		M + 2	0	27	22	1.642	1,83	91,55%	1,93	96,30%
		M + 4	0	14	37	877	3,43	85,69%	3,61	90,14%
		M + 8	0	10	12	612	4,91	61,40%	5,17	64,58%
	438	M + 1	0	51	13	3.073	0,98	97,86%	1,00	100,00%
		M + 2	0	26	11	1.571	1,91	95,72%	1,96	97,81%
		M + 4	0	13	36	816	3,69	92,14%	3,77	94,16%
		M + 8	0	9	18	558	5,39	67,36%	5,51	68,84%
	876	M + 1	0	50	29	3.029	0,99	99,30%	1,00	100,00%
		M + 2	0	25	35	1.535	1,96	97,95%	1,97	98,64%
		M + 4	0	13	19	799	3,76	94,11%	3,79	94,77%
		M + 8	0	8	53	533	5,64	70,51%	5,68	71,00%
	1.095	M + 1	0	50	19	3.019	1,00	99,63%	1,00	100,00%
		M + 2	0	25	28	1.528	1,97	98,40%	1,98	98,77%
		M + 4	0	12	60	780	3,86	96,45%	3,87	96,81%
		M + 8	0	10	42	642	4,69	58,59%	4,70	58,80%
17.520	219	M + 1	3	31	14	12.674	0,95	95,30%	1,00	100,00%
		M + 2	1	49	35	6.575	1,84	91,85%	1,93	96,37%
		M + 4	0	57	28	3.448	3,50	87,58%	3,68	91,90%
		M + 8	0	39	27	2.367	5,10	63,79%	5,35	66,93%
	438	M + 1	3	25	31	12.331	0,98	97,95%	1,00	100,00%
		M + 2	1	44	54	6.294	1,92	95,96%	1,96	97,97%
		M + 4	0	54	12	3.252	3,71	92,86%	3,79	94,80%
		M + 8	0	36	37	2.197	5,50	68,72%	5,61	70,16%
	876	M + 1	3	22	35	12.155	0,99	99,37%	1,00	100,00%
		M + 2	1	42	33	6.153	1,96	98,15%	1,98	98,78%
		M + 4	0	52	12	3.132	3,86	96,42%	3,88	97,03%
		M + 8	0	34	46	2.086	5,79	72,39%	5,83	72,85%
	1.095	M + 1	3	22	2	12.122	1,00	99,64%	1,00	100,00%
		M + 2	1	42	4	6.124	1,97	98,62%	1,98	98,98%
		M + 4	0	51	51	3.111	3,88	97,05%	3,90	97,41%
		M + 8	0	34	10	2.050	5,89	73,63%	5,91	73,90%
26.280	219	M + 1	0	0	0	-	-	-	-	-
		M + 2	0	0	0	-	-	-	-	-
		M + 4	0	0	0	-	-	-	-	-
		M + 8	1	28	11	5.291	5,13	64,16%	-	-
	438	M + 1	0	0	0	-	-	-	-	-
		M + 2	0	0	0	-	-	-	-	-
		M + 4	0	0	0	-	-	-	-	-
		M + 8	1	21	47	4.907	5,53	69,17%	-	-
	876	M + 1	0	0	0	-	-	-	-	-
		M + 2	0	0	0	-	-	-	-	-
		M + 4	0	0	0	-	-	-	-	-
		M + 8	1	25	35	5.135	5,29	66,10%	-	-
	1.095	M + 1	0	0	0	-	-	-	-	-
		M + 2	0	0	0	-	-	-	-	-
		M + 4	0	0	0	-	-	-	-	-
		M + 8	1	19	46	4.786	5,67	70,93%	-	-

- valores indisponíveis. O tempo elevado e a necessidade de exclusividade no uso do *cluster* inviabilizaram estas execuções.

Tabela 5.12: Diferença de desempenho entre os *clusters* CLUTF e CLUEM com 8 nós (Algoritmo APDF).

Dado	Subtarefa	Nós	CLUTF T(Seg)	CLUEM T(Seg)	DD %
4.380	219	M + 8	133	157	18,65%
	438	M + 8	135	144	6,54%
	876	M + 8	163	251	54,35%
	1.095	M + 8	198	197	-0,70%
8.760	219	M + 8	510	612	20,03%
	438	M + 8	454	558	23,02%
	876	M + 8	500	533	6,63%
	1.095	M + 8	409	642	57,00%
17.520	219	M + 8	2.014	2367	17,51%
	438	M + 8	1.765	2.197	24,50%
	876	M + 8	1.669	2.086	24,94%
	1.095	M + 8	1.615	2.050	26,93%
26.280	219	M + 8	4.473	5.291	18,29%
	438	M + 8	3.930	4.907	24,87%
	876	M + 8	3.834	5.135	33,94%
	1.095	M + 8	3.575	4.786	33,88%

Para o vetor de 8.760 registros com subtarefas de 438 registros, exemplificado anteriormente, observa-se uma diferença de 23,02% contra 63,41% do algoritmo APDI (ver Tabela 5.8). A economia é ainda mais acentuada (24,87% contra os 74,14%) ao se considerar as mesmas configurações do *cluster* e subtarefas, mas para vetores de 26.280 registros.

Alguns valores discrepantes precisam ser analisados com ressalva. Por exemplo, a percentagem de 57% da diferença de performance do conjunto de 8.760 registros com subtarefa de 1.095 registros apresenta apenas 8 subtarefas, ou seja, uma para cada nó. Isto representa uma combinação idêntica à encontrada no algoritmo APDI em 8 nós, na qual cada escravo manipulava também 1.095 registros. O conjunto de 4.380 registros e subtarefa de 1.095 registros é incoerente com a configuração de *cluster* com 8 nós, porque a combinação demandaria apenas 4 subtarefas, ou seja, a utilização de apenas 4 dos seus nós.

Pela diversidade de percentuais encontrados, não foi possível apontar a melhor combinação entre o conjunto de dados e a subtarefa. Mas, é inegável que todos os percentuais de diferença de desempenho diminuíram. Os gráficos a), b), c) e d), da Figura 5.11 mostram nas colunas destacadas a percentagem desta recuperação.

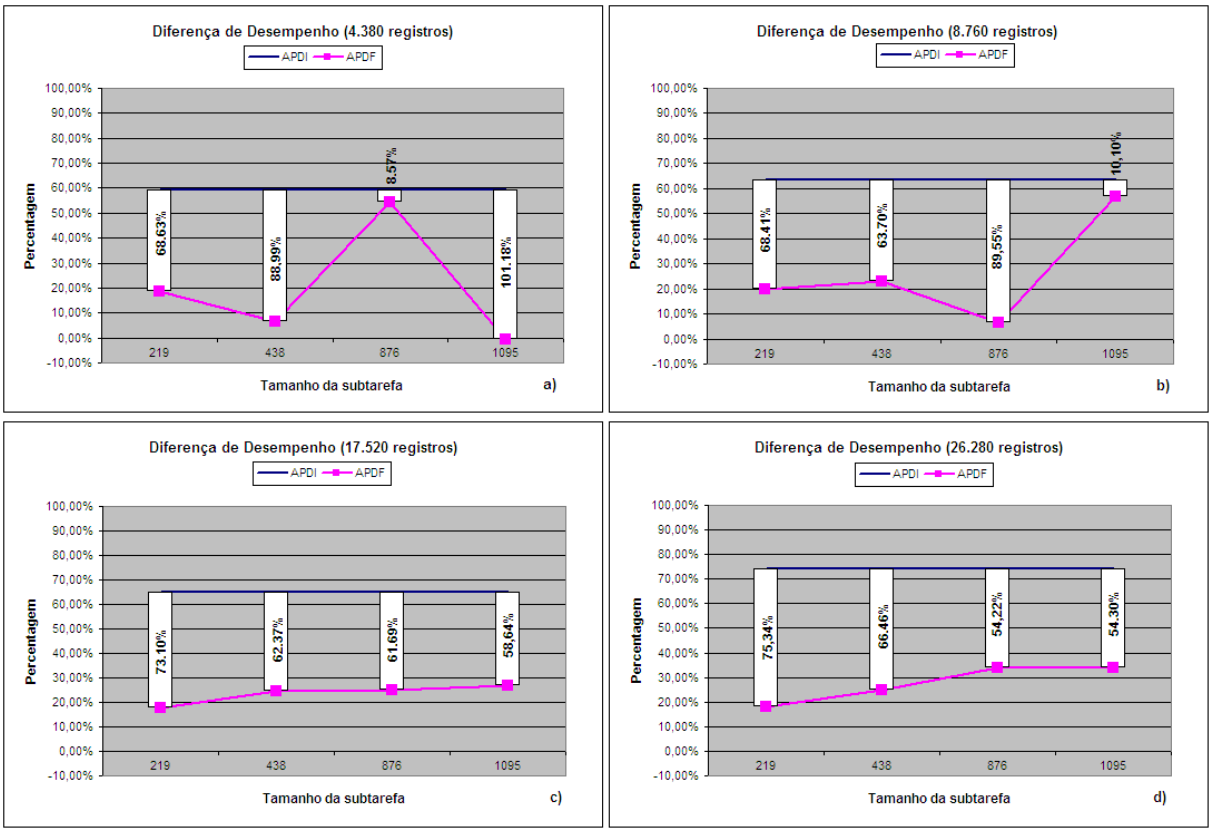

Figura 5.11: Diferenças de desempenho entre os *clusters* (algoritmos APDI e APDF).

Interpretando-se os gráficos, tem-se no eixo das coordenadas o tamanho das subtarefas usadas pelo algoritmo APDF, e nas abscissas, as percentagens de diferença de desempenho entre os *clusters*. A linha lisa representa o algoritmo APDI e a linha demarcada o algoritmo APDF.

Mesmo nos piores casos, por exemplo, com vetores de 8.760 registros e subtarefas de 1.095 registros, obteve-se diminuição da diferença de desempenho entre os *clusters*, neste caso, 10,10%. Já com subtarefas de 438 registros, tem-se uma redução de 63,70%. Os vetores de 17.520 e 26.280 registros apresentaram uma dedução mais uniforme, com médias de 63,95% e 62,58%, respectivamente.

Pelos resultados citados e observados nos gráficos, comprova-se a eficiência do algoritmo APDF. Embora, por uma nova análise, contrapondo-se esses valores aos tempos de execução dos algoritmos APDI (ver Tabelas 5.5 e 5.7)

e APDF (ver Tabelas 5.10 e 5.11), observa-se que em alguns casos, a redução desta diferença foi influenciada pela degradação do desempenho do algoritmo APDF no *cluster* CLUTF. Com subtarefas pequenas e em maior quantidade, menor será a granularidade e, conseqüentemente, maiores os custos de particionamento, comunicação e sincronização, *overheads* adicionais que não existiam no algoritmo APDI no *cluster* CLUTF. Isto não inviabiliza a utilização deste algoritmo, apenas demonstra a influência do tamanho da subtarefa no desempenho do algoritmo. Além disso, o algoritmo APDF permite que novos nós sejam agregados ao *cluster*, mesmo que nós heterogêneos e de pouco recurso computacional, elevando verdadeiramente seu poder de processamento.

5.3.3 Etapa 3 - Interpretação

Terminada a etapa de processamento, na qual foram gerados os valores dos cálculos de Entropia e Informação Mútua, entra em ação o algoritmo de interpretação. Este algoritmo é responsável por filtrar dentre todas as variáveis de entrada, o subconjunto ótimo, ou seja, as variáveis representativas para o modelo.

Baseado nas séries temporais meteorológicas de medidas horárias, foi realizado um estudo com o objetivo de delimitar o subconjunto ótimo de variáveis que interferem na previsão da umidade do ar, influenciador direto na disseminação da Ferrugem Asiática. Para isso, os dados da seção 5.2 foram aplicados em fórmulas da literatura e no algoritmo de interpretação, discutidos na seção 4.3.

A Tabela 5.13(a) foi preenchida com os valores da Entropia, possibilitando a observação do potencial de informação das variáveis. Junto com a análise dos resultados da Tabela 5.13(b), do ganho de informação normalizado (Informação Mútua), possibilitou a seleção de um modelo parcimonioso de qualidade, mostrado na Tabela 5.14, ou seja, um subconjunto ótimo de variáveis e características, contendo apenas o vento2.bsd, a rajada2.bsd, a pressão.bsd e a classe

de estudo (umidade.bsd).

Tabela 5.13: Resultados dos cálculos propostos pela metodologia.

(a) Valores de Entropia.

Ordem	Vetor analisado	Entropia
1	umidade.bsd	0.552466
2	vento2.bsd	0.559906
3	vento1.bsd	0.485590
4	tminima.bsd	0.547568
5	tmedia.bsd	0.550347
6	tmaxima.bsd	0.553079
7	rajada2.bsd	0.552613
8	rajada1.bsd	0.507515
9	radiacao.bsd	0.437186
10	pressao.bsd	0.074871
11	precipitacao.bsd	0.015744

(b) Valores de Informação Mútua.

Ordem	Vetores analisados		Informação Mútua
1	umidade.bsd	vento2.bsd	0.520326
2	umidade.bsd	vento1.bsd	0.462167
3	umidade.bsd	tminima.bsd	0.492729
4	umidade.bsd	tmedia.bsd	0.495272
5	umidade.bsd	tmaxima.bsd	0.498778
6	umidade.bsd	rajada2.bsd	0.512085
7	umidade.bsd	rajada1.bsd	0.482829
8	umidade.bsd	radiacao.bsd	0.376285
9	umidade.bsd	pressao.bsd	0.230494
10	umidade.bsd	precipitacao.bsd	0.003387
11	vento2.bsd	vento1.bsd	0.583548
12	vento2.bsd	tminima.bsd	0.523088
13	vento2.bsd	tmedia.bsd	0.525329
14	vento2.bsd	tmaxima.bsd	0.527499
15	vento2.bsd	rajada2.bsd	0.484707
16	vento2.bsd	rajada1.bsd	0.607048
17	vento2.bsd	radiacao.bsd	0.503243
18	vento2.bsd	pressao.bsd	0.104998
19	vento2.bsd	precipitacao.bsd	0.083252
20	vento1.bsd	tminima.bsd	0.459662
21	vento1.bsd	tmedia.bsd	0.462072
22	vento1.bsd	tmaxima.bsd	0.463642
23	vento1.bsd	rajada2.bsd	0.579531
24	vento1.bsd	rajada1.bsd	0.426641
25	vento1.bsd	radiacao.bsd	0.406633
26	vento1.bsd	pressao.bsd	0.008859
27	vento1.bsd	precipitacao.bsd	0.029900
28	tminima.bsd	tmedia.bsd	0.428677
29	tminima.bsd	tmaxima.bsd	0.433889
30	tminima.bsd	rajada2.bsd	0.512125
31	tminima.bsd	rajada1.bsd	0.480344
32	tminima.bsd	radiacao.bsd	0.383975
33	tminima.bsd	pressao.bsd	0.077252
34	tminima.bsd	precipitacao.bsd	0.008753
35	tmedia.bsd	tmaxima.bsd	0.435257
36	tmedia.bsd	rajada2.bsd	0.514310
37	tmedia.bsd	rajada1.bsd	0.482320
38	tmedia.bsd	radiacao.bsd	0.387477
39	tmedia.bsd	pressao.bsd	0.081441
40	tmedia.bsd	precipitacao.bsd	0.006156
41	tmaxima.bsd	rajada2.bsd	0.516404
42	tmaxima.bsd	rajada1.bsd	0.482774
43	tmaxima.bsd	radiacao.bsd	0.390454
44	tmaxima.bsd	pressao.bsd	0.091170
45	tmaxima.bsd	precipitacao.bsd	0.004401
46	rajada2.bsd	rajada1.bsd	0.603508
47	rajada2.bsd	radiacao.bsd	0.505428
48	rajada2.bsd	pressao.bsd	0.095103
49	rajada2.bsd	precipitacao.bsd	0.076872
50	rajada1.bsd	radiacao.bsd	0.416135
51	rajada1.bsd	pressao.bsd	0.010731
52	rajada1.bsd	precipitacao.bsd	0.012099
53	radiacao.bsd	pressao.bsd	0.055019
54	radiacao.bsd	precipitacao.bsd	0.172258
55	pressao.bsd	precipitacao.bsd	0.475741

A estratégia apresentada pela metodologia de seleção de variáveis e características reduziu a quantidade de vetores de 11 para 4. Embora os 7 vetores

Tabela 5.14: Vetores relevantes (Subconjunto ótimo).

Subconjunto ótimo
vento2.bsd
rajada2.bsd
pressao.bsd
umidade.bsd (classe de estudo)

eliminados contenham informações sobre o domínio, estes não são representativos para o objetivo em questão. Sua utilização, além de aumentar o custo computacional, poderia distorcer o resultado final, confundindo o aprendizado da RNA. O subconjunto resultante, apesar de ser representado por uma quantidade menor de variáveis, caracteriza informações de maior qualidade.

6 *Considerações Finais*

Este capítulo representa a síntese das atividades realizadas ao longo deste trabalho de mestrado. Aqui são apresentadas as conclusões extraídas, as principais contribuições e os temas a serem desenvolvidos em trabalhos futuros.

6.1 Conclusões

Neste trabalho foram usadas técnicas de otimização de algoritmos e de computação paralela, para diminuir o tempo de processamento de uma aplicação real. A aplicação em estudo foi uma metodologia de seleção de variáveis e características, cujo objetivo é reduzir a quantidade de variáveis utilizadas por uma aplicação qualquer. Pautada nos conceitos de redundância e relevância, a metodologia pode ser aplicada em diversas áreas do conhecimento, como um filtro, que escolhe as variáveis mais representativas para o modelo em estudo, mostrando-se essencial para qualificar os dados de entrada para uma Rede Neural de previsão de valores.

Em grande parte, os problemas científicos não são tratados de forma adequada pelas arquiteturas convencionais, por restrições físicas, pela inerente grandiosidade ou pela complexidade do problema. Embora as máquinas seqüenciais, baseadas na arquitetura de von Neumann, tenham apresentando uma evolução no poder de processamento, sozinhas elas não conseguiram acompanhar a demanda exigida pelos softwares científicos.

Logo, para amenizar esta deficiência de processamento, o foco foi direcionado para a computação paralela. Diante das diversas arquiteturas paralelas, os *clusters* de computadores destacam-se pelo excelente custo/benefício apresentado. Unindo computadores comuns em um trabalho cooperativo, mediante uma rede de interconexão e uma biblioteca de troca de mensagens, é possível obter poder de processamento compatível ao dos supercomputadores, por uma fração do seu preço. Entretanto, a arquitetura paralela exige um paradigma de programação diferente. Como há diversos processos sendo executados simultaneamente – que são as várias instâncias do algoritmo –, isto não é tão explícito para os programadores.

No espectro dos diversos problemas computacionais de grande escala, inclui-se a seleção de variáveis e características que, dependendo do domínio avaliado, exige muito tempo de processamento para resolver o modelo matemático proposto. A metodologia compreende cálculos complexos, repetitivos e dependentes, e é isto que avaliza o elevado custo da aplicação.

Acredita-se que o caráter evolutivo utilizado para implementar a metodologia facilitou a obtenção dos bons resultados no quesito tempo de processamento. Os índices desta redução totalizaram 93%. A primeira implementação serial, VS1, mesmo não apresentando desempenho adequado, foi muito importante, pois foi a base para o início da análise e predição de desempenho. A transição para a VS2 contemplou uma reorganização do algoritmo, diminuindo o excesso de chamadas às funções definidas pelo usuário, mostrando economia mínima superior a 13%.

Na versão VS3, o uso das técnicas tradicionais de otimização de algoritmos apresentou reduções de tempo maiores que 48%. Tais técnicas, apesar de se apresentarem em grande número, nem sempre podem ser usadas na sua totalidade, principalmente quando se trata de um aplicação real, pois ficam limitadas por suas características particulares.

Observa-se a grande valia dos *profilers*, que apoiaram, através de seus relatórios, todo o processo de desenvolvimento dos algoritmos seriais, elencando os pontos problemáticos.

Este primeiro grande ciclo evolutivo dos algoritmos mostrou que boas práticas de programação, atreladas ao conhecimento da arquitetura e das técnicas tradicionais de otimização, de laços e de condicionais, garantem uma boa diminuição no tempo de processamento da aplicação. A redução total de mais de 55% do tempo de execução evidencia a otimização serial como uma fase bastante importante no processo de paralelização.

Depois de ter optado pela melhor combinação de técnicas de otimização, o melhor algoritmo serial, aplicado às séries temporais meteorológicas (variáveis), ainda demandaria 33 dias ininterruptos de processamento. Assim, iniciou-se um novo ciclo de evolução dos algoritmos, com a incorporação das primitivas MPI, possibilitando sua execução em paralelo.

A arquitetura dos *clusters* MPI, escolhida para executar as implementações paralelas, apresenta a escalabilidade como outra grande vantagem, em relação aos multicomputadores de memória compartilhada. Isto implica dizer que novas unidades processadoras podem ser agregadas ao ambiente do *cluster*, incrementando assim o seu poder computacional. Embora esta seja uma afirmação verdadeira, nem todas as aplicações se beneficiarão deste recurso.

A implementação do algoritmo APDI, discutida na subseção 5.3.2, mostra que a abordagem implícita e igualitária das tarefas facilita a implementação, mas, em contrapartida, subutiliza algumas máquinas do *cluster* heterogêneo. Sendo assim, a granularidade grossa apresentada pelo APDI impõe uma diminuição da concorrência, e como todos os nós recebem a mesma carga de trabalho, o tempo de processamento da aplicação fica limitado pelo nó mais lento do *cluster*. As comparações feitas entre os *clusters* homogêneo (CLUTF) e heterogêneo (CLUEM), utilizando o algoritmo APDI, mostraram diferenças de

desempenho em favor do CLUTF maiores que 70%, e ainda com tendência de crescimento, acompanhando o crescimento do vetor de dados.

Mesmo não encarado como um estudo de balanceamento de carga, a abordagem de se fixar tamanhos para uma subtarefa impactou positivamente no equilíbrio dos *clusters*. Podendo enviar trabalho sob demanda aos nós escravos, aproveitando naturalmente o poder de processamento de cada máquina, o algoritmo APDF maximizou o uso dos recursos computacionais outrora desperdiçados pelo algoritmo APDI e apresentou uma melhora de desempenho, principalmente no *cluster* heterogêneo CLUEM. Esta recuperação computacional alcançada pelo algoritmo APDF amenizou a diferença de desempenho entre os *clusters* em valores médios de 62,83%. Assim, pode-se eleger o algoritmo APDF como o melhor dentre as abordagens paralela, pois, além dos benefícios citados, possibilita o crescimento e o aproveitamento computacional do *cluster* com a adição de novos nós, não importando o seu poder individual de processamento.

De uma forma geral, o objetivo inicial de diminuir o tempo total de execução da aplicação foi alcançado. Em valores, obteve-se, aproximadamente, 55% de economia nos algoritmos seriais; até 86% na transição VS3 (melhor algoritmo serial) para o APDF (segundo algoritmo paralelo); e de 93% entre a VS1 (versão serial original) e a versão paralela APDF. A seleção das variáveis da Normal Climatológica poderia então, ser executada por um *cluster* com 8 nós de processamento, em aproximadamente $4\ \frac{1}{2}$ dias. Além disso, o bom comportamento escalável apresentado pelos algoritmos paralelos indica que, com a adição de novas unidades processadoras, o tempo computacional da seleção seria reduzido proporcionalmente.

É preciso salientar que a abordagem aqui implementada visa a solução de todas as fases da metodologia de seleção de variáveis e características, desde a importação dos dados até a interpretação dos resultados, passando pela execu-

ção paralela dos cálculos.

Os resultados obtidos neste trabalho mostram que é altamente compensador o esforço despendido no procedimento de seleção e os mesmos comprovam a eficácia do proposto. No domínio meteorológico avaliado, reduziu-se de 11 para 4 o número de variáveis representativas. Uma das grandes vantagens da utilização desta metodologia é que com a diminuição da quantidade de variáveis de um modelo, pode-se até inviabilizar a necessidade posterior de uso da computação de alto desempenho (vislumbrando-se até um rebaixamento da categoria de "grande" problema).

A implementação da metodologia com os dados meteorológicos, cedidos pelo SIMEPAR, propiciaram a veracidade para a aplicação. Como discutido no capítulo 1 e no apêndice A, a escolha prévia destas variáveis é o passo anterior e indispensável ao desenvolvimento do módulo de previsão micro-climática.

6.2 Publicações

Um artigo com os resultados finais obtidos com a implementação da metodologia de seleção de variáveis e características está sendo elaborado e será submetido a um periódico internacional. Até o momento foram publicados os seguintes trabalhos:

- Pereira Junior, F., Santos, M. H. D., Martini, J. A., and Freitas, A. A. C. D. (2006). Seleção de variáveis e características como aplicação paralela em cluster MPI. *XIII Escola Regional de Informática da SBC - Paraná* (ERI2006).

- Pereira Junior, F., Santos, M. H. D., and Martini, J. A. (2006). Reduzindo o tempo de execução de uma aplicação com técnicas de otimização e clusters MPI. *VII Fórum de Informática e Tecnologia de Maringá* (FITEM2006).

6.3 Trabalhos Futuros

Finalmente, pode-se observar ao longo do desenvolvimento deste trabalho alguns pontos que poderiam ser alvo de futuros desenvolvimentos, objetivando melhorias do mesmo. Uma extensão natural seria a aplicação desta metodologia de seleção em outras massas e volumes de dados. Uma área promissora, e já vislumbrada, é a econometria, pois apresenta uma grande quantidade de variáveis e um enorme interesse entre suas correlações.

Como o algoritmo APDF trabalha com tamanhos fixos de subtarefas, um próximo estudo poderia verificar o poder de processamento e o status de carga dos nós escravos, além de dimensionar dinamicamente, em tempo de execução, a carga de trabalho. Pode-se também estudar outra política de escalonamento para distribuição das subtarefas.

Uma outra possibilidade de desenvolvimento, seria a criação de um interfaceamento para que membros de outras comunidades possam, através de um sistema web, carregar os vetores e executar a metodologia para seus próprios dados, colhendo apenas o resultado final, sem a necessidade de conhecimento da computação paralela (intermediária), funcionando assim como um serviço a outros grupos de pesquisa.

Referências Bibliográficas

[Amdahl 1967] Amdahl, G. M. (1967). Validity of the single-processor approach to achieving large scale capabilities. *AFIPS Conference Proceedings*, 30:483–485.

[Aveleda 2003] Aveleda, A. D. A. (2003). *Utilização de Sistemas de Alto Desempenho no Processamento de Sinais na Análise de Problemas de Vibrações Induzidas por Desprendimento de Vórtices em Estruturas Offshore*. PhD thesis, Universidade Federal do Rio de Janeiro (COPPE-UFRJ), Rio de Janeiro, RJ, Brasil.

[Bailey and Thompson 1990] Bailey, D. and Thompson, D. (1990). How to develop neural-network applications. *AI Expert*, 5(6):38–47.

[Baker 2000] Baker, M. (2000). Cluster computing white. *University of Portsmouth*.

[Bal et al. 1989] Bal, H. E., Steiner, J. G., and Tanenbaum, A. S. (1989). Programming languages for distributed computing systems. *ACM Computing Surveys*, 21(3):261–322.

[Bell and Gray 2002] Bell, G. and Gray, J. (2002). What's next in high-performance computing? *Communications of the ACM*, 45(2):91–95.

[Blum and Langley 1997] Blum, A. L. and Langley, P. (1997). Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97:245–271.

[Castelo 1999] Castelo, K. R. L. J. (1999). Extensão da ferramenta de apoio a programação paralela (f.a.p.p.) para ambientes paralelos virtuais. Master's thesis, Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo (ICMC-USP), São Carlos, SP, Brasil.

[CENAPAD-NE 1998] CENAPAD-NE (1998). Programação paralela com mpi: Manual do curso. Centro Nacional de Processamento de Alto Desempenho do Nordeste, Fortaleza, CE, Brasil.

[CENAPAD-SP 2006] CENAPAD-SP (2006). Guia do usuário. Centro Nacional de Processamento de Alto Desempenho em São Paulo, Campinas, SP, Brasil.

[CONAB 2005] CONAB (2005). 4º levantamento safra 2004/2005. Disponível na Internet via WWW. URL: http://www.conab.gov.br/. Último acesso 10 de junho de 2005.

[Corrêa 2001] Corrêa, R. V. (2001). Otimização de desempenho utilizando contadores de hardware. Master's thesis, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, SP, Brasil.

[Cunha 2004] Cunha, M. T. F. (2004). *Uma Metodologia Portável para a Paralelização de Programas de Elementos de Contorno*. PhD thesis, Universidade Federal do Rio de Janeiro (COPPE-UFRJ), Rio de Janeiro, RJ, Brasil.

[Cunha et al. 2001] Cunha, M. T. F., de Faria Telles, J. C., and Coutinho, A. L. G. D. A. (2001). High perfomance techniques applied to boundary elements: Potential problems. *In 22nd Iberian Latin American Congress on Computational Methods in Engineering*.

[Filho and Richetti 2004] Filho, G. A. D. M. and Richetti, A. (2004). Custo do controle químico da ferrugem asiática da soja. Technical Report 93, Embrapa Agropecuária Oeste, Dourados, MS.

[Flynn 1972] Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE Transactions on Computers*, 21(9):948–960.

[Foster 1995] Foster, I. (1995). *Designing and Building Parallel Programs*. Addison-Wesley, 1 edition.

[Franco 2004] Franco, L. D. (2004). *Implementação Computacional em Ambiente Paralelo de Memória Distribuída para Análise Acoplada de Sistemas Offshore*. PhD thesis, Universidade Federal do Rio de Janeiro (COPPE-UFRJ), Rio de Janeiro, RJ, Brasil.

[Freitas et al. 2006] Freitas, A. A. C., Securato, J. R., and Neto, M. L. D. A. (2006). Brazilian exchange rates modeling: Macro and microstructure variable selection by means of the analysis of relevance and redundancy. *Global Finance Conference Web-Proceedings*.

[Goedecker and Hoisie 2001] Goedecker, S. and Hoisie, A. (2001). *Performance Optimization of Numerically Intensive Codes*. Soc for Industrial & Applied Math, Philadelphia, PA.

[Gonçalves et al. 2006] Gonçalves, R. A. D. L., Martini, J. A., and Junior, A. D. S. (2006). *Computação de Alto Desempenho*. Escola Regional de Informática da SBC Sul. XIII Escola Regional de Informática da SBC - Paraná, FFALM - Faculdades Luis Meneghel, Bandeirantes, PR.

[Grama et al. 2003] Grama, A., Gupta, A., Karypis, G., and Kumar, V. (2003). *Introduction to Parallel Computing*. Addison Wesley, 2 edition.

[Gustafson 1988] Gustafson, J. L. (1988). Reevaluating amdahl's law. *Communications of the ACM*, 31(5):532–533.

[Guyon and Elisseeff 2003] Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182.

[Haykin 1999] Haykin, S. (1999). *Neural Networks: A Comprehensive Foundation*. Prentice Hall, Upper Saddle River, NJ, USA.

[Hwang 1993] Hwang, K. (1993). *Advance Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, New York, 2 edition.

[INMET 2005] INMET (2005). Instituto nacional de meteorologia. Disponível na Internet via WWW. URL: http://www.inmet.gov.br/. Último acesso 15 de junho de 2005.

[Jalote 1994] Jalote, P. (1994). *Fault Tolerance in Distributed Systems*. Prentice Hall, New Jersey.

[Júnior 1997] Júnior, A. M. (1997). *Predição do Desempenho de Programas Paralelos por Simulação do Grafo de Execução*. PhD thesis, Faculdade de Engenharia Elétrica e Computação - Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil.

[Junior 2002] Junior, E. M. (2002). Classificação e comparação de ferramentas para análise de desempenho de sistemas paralelos. Master's thesis, Faculdade de Engenharia de Ilha Solteira - Universidade Estadual Paulista Júlio de Mesquita Filho - UNESP, Ilha Solteira, SP, Brasil.

[Kohavi and John 1997] Kohavi, R. and John, G. H. (1997). Wrappers for feature selection, artificial inteligence. 97(1-2):273–324.

[Lu and Chung 1998] Lu, N. P. and Chung, C. P. (1998). Performance metrics: Keeping the focus on runtime. *IEEE Proceedings - Computers and Digital Techniques*, 145(4):255–264.

[MAPA 2005] MAPA (2005). Ministério da agricultura, pecuária e abastecimento. Disponível na Internet via WWW. URL: http://www.agricultura.gov.br/. Último acesso 10 de junho de 2005.

[Miles et al. 2003] Miles, M. R., Frederick, R. D., and Hartman, G. L. (2003). Soybean rust: Is the u.s. soybean crop at risk? Disponível na Internet via WWW. URL: http://www.apsnet.org/. APS - The American Phytopathological Society.

[Morettin and Toloi 1985] Morettin, P. A. and Toloi, C. M. D. C. (1985). *Previsão de Séries Temporais*. Atual, São Paulo, SP.

[Oliveira 2003] Oliveira, H. M. D. (2003). Modelagem e predição de desempenho de primitivas de comunicação mpi. Master's thesis, Escola Politécnica - Universidade de São Paulo (POLI-USP), São Paulo, SP, Brasil.

[Pacheco 1995] Pacheco, P. S. (1995). A user guide to mpi. Technical report, San Francisco, CA, USA.

[Pacheco 1997] Pacheco, P. S. (1997). *Parallel Programming with MPI*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Parzen 1962] Parzen, E. (1962). On the estimation of a probability density function and the mode. *Annals of Mathematical Statistics*, 33:1065–1076.

[Patterson and Hennessy 2005] Patterson, D. A. and Hennessy, J. L. (2005). *Organização e Projeto de Computadores: A Interface Hardware/Software*. Campus, Rio de Janeiro, 3 edition.

[Pereira et al. 2002] Pereira, A. R., Angelocci, L. R., and Sentelhas, P. C. (2002). *Agrometeorologia: Fundamentos e Aplicações Práticas*. Agropecuária, Guaíba.

[Principe et al. 1999] Principe, J. C., Xu, D., and III, J. F. (1999). Information theoretic learning. *Unsupervised Adaptive Filtering*, pages 265–319.

[Renyi 1976] Renyi, A. (1976). On measures of entropy and information. 2:565–580.

[Sahni and Thanvantri 1996] Sahni, S. and Thanvantri, V. (1996). Performance metrics: Keeping the focus on runtime. *IEEE Parallel and Distributed Technology*, 4:43–56.

[Severance and Dowd 1998] Severance, C. and Dowd, K. (1998). *High Performance Computing*. O'Reilly & Associates, Sebastopol, CA, USA, 2 edition.

[Smith 1975] Smith, L. P. (1975). *Methods in agricultural meteorology*. Elsevier Scientific Pub, New York, USA.

[Soja 2004] Soja, E. (2004). *Tecnologia de Produção de Soja - Paraná 2005*. Embrapa Soja, Londrina, PR.

[Stallings 2002] Stallings, W. (2002). *Arquitetura e Organização de Computadores*. Prentice Hall, São Paulo, 5 edition.

[Sterling et al. 2003] Sterling, T., Gropp, W., and Lusk, E. (2003). *Beowulf Cluster Computing with LINUX*. The MIT Press, 2 edition.

[Tanenbaum 2001] Tanenbaum, A. S. (2001). *Organização Estruturada de Computadores*. LTC, Rio de Janeiro, 4 edition.

[Vazquez 2002] Vazquez, P. A. M. (2002). Técnicas para análise de desempenho de computadores. In *Química Nova*, volume 25, pages 117–122. Instituto de Química - Universidade Estadual de Campinas (UNICAMP), Campinas, SP, Brasil.

[Xu et al. 1998] Xu, D., Principe, J. C., III, J. F., and Wu, H.-C. (1998). A novel measure for independent component analysis (ica). *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2:1145–1148.

[Yorinori et al. 2004] Yorinori, J. T., Junior, J. N., and Lazzarotto, J. J. (2004). *Ferrugem "asiática"da Soja no Brasil: evolução, importância econômica e controle*. Number 247. Embrapa Soja, Londrina, PR.

[Yorinori and Lazzarotto 2004] Yorinori, J. T. and Lazzarotto, J. J. (2004). *Situação da Ferrugem Asiática da Soja no Brasil e na América do Sul*. Number 236. Embrapa Soja, Londrina, PR.

[Yorinori et al. 2003] Yorinori, J. T., Paiva, W. M., Costamilan, L. M., and Bertagnolli, P. F. (2003). *Ferrugem da Soja: Identificação e Controle*. Embrapa Soja, Londrina, PR.

[Yu and Liu 2004] Yu, L. and Liu, H. (2004). Efficient feature selection via analysis of relevance and redundancy. *Journal of Machine Learning Research*, 5:1205–1224.

[Zuben 2003] Zuben, F. J. V. (2003). Uma caricatura funcional de redes neurais artificiais. Technical report.

APÊNDICE A – O Agronegócio Brasileiro

A.1 Introdução

Com forte influência na balança comercial, o agronegócio no Brasil vem sendo o maior gerador de divisas para o país, e o mercado da Soja se destaca no setor agroindustrial pela expressividade do seu porte econômico [Soja 2004]. Hoje o Brasil ocupa lugar de destaque no cenário do agronegócio mundial do complexo Soja, como grande produtor e exportador. O faturamento brasileiro com a Soja, no ano de 2003, foi da ordem de 12 bilhões de dólares, aproximadamente 50 milhões de toneladas, cerca de 1/4 da produção mundial, e o Paraná contribuiu com esse montante com uma parcela de 20% [CONAB 2005] [MAPA 2005] [Soja 2004]. Para se firmar nesta posição, universidades, pesquisadores, empresas públicas e privadas, lideradas pela EMBRAPA Soja, divisão do Ministério da Agricultura, Pecuária e Abastecimento (MAPA) sediada em Londrina, norte do Paraná, investem em pesquisas multidisciplinares que apóiam as tecnologias para este cultivar. As iniciativas vão desde melhoramento genético da semente a procedimento de estocagem, passando por estudo do solo, plantio, clima, patologias entre outras.

Há alguns anos, mais precisamente desde 2001, a cultura da Soja vem perdendo produtividade decorrente da disseminação de um fungo chamado Phakopsora Pachyrhizi, causador da Ferrugem Asiática. Esta fitopatologia causa um rápido amarelecimento e prematura queda das folhas, diminuindo ou até inibindo a formação dos grãos. Segundo [Miles et al. 2003] [Yorinori et al. 2003]

[Yorinori et al. 2004] [Yorinori and Lazzarotto 2004] [Soja 2004], as perdas na produção variam entre 10% a 80%, dependentes também de alguns outros fatores, mas podem atingir índices que inviabilizam até a própria colheita. O monitoramento constante da lavoura é de suma importância para que se identifique logo no início a ocorrência da ferrugem. Produtores e técnicos devem estar atentos às informações de sua região. Se a ferrugem for identificada em alguma propriedade próxima, o ideal é fazer a aplicação de fungicida, antes que o fungo se espalhe pelo vento. Atualmente, única fonte de controle, os fungicidas devem ser aplicados na dose e na hora correta [Yorinori et al. 2004] [Soja 2004]. Um outro fator influenciador tanto na infecção, disseminação e produção da leguminosa é o clima. Dependente de uma combinação dos elementos climatológicos para uma boa safra, a agricultura por si só torna-se uma atividade de risco, assim, a Agrometeorologia vem auxiliar com previsões e estudos do zoneamento agroclimático.

Por iniciativa do MAPA foi criado o Consórcio Anti-ferrugem, reunindo representantes de todos os segmentos da cadeia produtiva da Soja. Como ações iniciais, o consórcio padronizou as informações sobre o manejo da doença e centralizou-as no Sistema de Alerta. O sistema que funciona via Internet (ver: http://www.cnpso.embrapa.br/alerta), tem finalidade difusora e está atualmente agregando novos serviços para auxiliar técnicos, produtores, empresários do setor e interessados em geral.

A.2 A Soja

A Soja é conhecida no mundo há mais de cinco mil anos. No Brasil, chegou em 1882, pelo estado da Bahia. A partir de 1940, começou a ganhar importância na agricultura do país. Passados quase 64 anos, transformou-se no maior destaque do agronegócio brasileiro. No ano de 2003, o Brasil assumiu a liderança

no mercado internacional do complexo Soja (grãos, farelo e óleo), com exportações de US$ 8,1 bilhões, 31% acima do valor alcançado em 2002 [MAPA 2005] [Soja 2004].

No cenário mundial a Soja aparece como o quarto grão mais produzido, com cerca de 200 milhões de toneladas, perdendo apenas para milho, arroz e trigo respectivamente, com montantes na ordem dos 600 milhões de toneladas cada. Os Estados Unidos da América (EUA) são os maiores produtores de Soja com uma fatia global de 38%, ou seja, cerca de 75 milhões de toneladas [Soja 2004].

O potencial e a vocação agrícola brasileira tornam-se indiscutíveis mediante dados da expansão do plantio da Soja [MAPA 2005] [CONAB 2005]. As lavouras da oleaginosa, até a década de 80, se concentravam nos estados do Sul. Graças ao desenvolvimento de cultivares adaptados ao solo e ao clima das diferentes regiões brasileiras, a Soja se espalhou também pelo Centro-Oeste e Distrito Federal, além de parte do Nordeste (no oeste da Bahia e no sul do Maranhão e do Piauí).

O crescimento da Soja no Brasil foi fantástico. Baseado em informações da Companhia Nacional de Abastecimento (CONAB) e MAPA, em pouco mais de uma década, a colheita saltou de 15,3 milhões de toneladas, em uma área plantada de 9,7 milhões de hectares para 52 milhões de toneladas, em uma área de 18,4 milhões de hectares na safra 2002/03, ou seja, uma produção três vezes maior. Com isso, a Soja, um grão originária da China, tornou-se nesta safra, o principal produto do agronegócio brasileiro [MAPA 2005]. O país, ocupa o posto de segundo maior produtor mundial da oleaginosa, com cerca de 25% da produção global [Soja 2004].

Mesmo com forte crescimento na produção e plantio, a produtividade (produção / hectares plantados) teve e ainda tem uma oscilação considerável (ver mapa na Figura A.1). A justificativa apresentada pelos órgãos controladores do agronegócio é desdobrada em vários fatores, e muitos deles apresentados tam-

bém em outros cultivares, como: fitopatologias (doenças do vegetal), variação climática (excesso de chuva ou sol), preço de venda, etc. Esses elementos que não podem ser totalmente descartados, devem ser atentamente monitorados.

A.3 Fitopatologias (patologia vegetal)

Fitopatologia é uma ciência interdisciplinar que visa estudar as doenças das plantas. Doenças em plantas podem ser causadas tanto por organismos vivos (elementos patogênicos) como fungos, bactérias, vírus, nematóides, fitoplasmas, protozoários e plantas parasitárias; quanto por agentes não-vivos como poluição do ar, desequilíbrios nutricionais e vários outros fatores ambientais relativos ao clima, como temperatura, pluviosidade, umidade do ar entre outros [Soja 2004].

Agravantes como novas doenças e mutações ou variações dos patogênicos tornam-se constantes ameaças para os cultivares. Os fitopatologistas e pesquisadores, em geral, usam técnicas tradicionais e inovadoras nesta constante batalha para o controle dessas ameaças. Fitopatologias podem ser controladas através de alterações genéticas na planta originária (matriz), criando-se variedades de plantas resistentes a certos organismos patogênicos, resistentes ou adaptadas a condições ambientais; ou por produtos químicos, dentre eles os fungicidas [Yorinori et al. 2003] [Filho and Richetti 2004].

Como em outros cultivares, a cultura da Soja está, durante todo seu ciclo, sujeita a essas ameaças. Entre outros fatores que limitam a obtenção de rendimentos desta oleaginosa, foram identificadas, no Brasil, cerca de 40 doenças [Soja 2004]. Com um número crescente, essas doenças refletem economicamente de forma variada de ano para ano e de região para região, dependendo da condição climática apresentada em cada safra. Decorrente a isso, as perdas anuais são estimadas entre 15% e 20%, mas podendo chegar a 100% da produ-

Tabela A.1: Principais doenças que atacam cultivares da Soja.

Doença	Tipo	Agentes	Safra Surgimento	Controle	Variação de Perdas de Rendimento
Ferrugem "Americana"	fungo	*Phakopsora Meibomiae*	1979	Fungicida	raramente
Ferrugem "Asiática"	fungo	*Phakopsora Pachyrhizi*	2000/01	Fungicida	até 100%
Oídio	fungo	*Erysiphe Diffusa*	1996/97	Cultivares resistentes/ Fungicida	até 40%
Crestamento Bacteriano	bactéria	*Pseudomonas Savastanoi*	s/ referência	Cultivares resistentes	bem controlada
Mosaico Comum da Soja	vírus	*VMCS*	s/ referência	Cultivares resistentes	bem controlada
Nematóides de Cisto da Soja	nematóides	*Heterodera Glycines*	1991/92	Rotação de culturas/ Manejo do solo/ Cultivares resistentes	bem controlada

ção [Soja 2004]. A Tabela A.1 mostra um resumo das principais doenças que agridem a sojicultura.

A cultura da Soja é atacada por duas espécies do fungo do gênero *Phakopsora*: *Phakopsora Meibomiae* (Ferrugem "Americana") e *Phakopsora Pachyrhizi* (Ferrugem "Asiática") [Yorinori and Lazzarotto 2004]. A primeira, infecta diversos tipos de leguminosas, mas aparece freqüentemente na Soja. Tem sua ocorrência mais comum no final da safra, estando restrita a um clima mais ameno. A Ferrugem "Americana" raramente causa prejuízos econômicos. A segunda, necessita de uma combinação de fatores climáticos, atinge todo período produtivo da planta e causa enormes prejuízos à agricultura.

A.4 Ferrugem Asiática

A Ferrugem Asiática, uma das piores doenças que afetam as plantações de Soja, conforme afirmam os pesquisadores da EMBRAPA Soja em seus artigos e relatórios técnicos [Yorinori et al. 2003] [Yorinori and Lazzarotto 2004] [Yorinori et al. 2004], foi identificada pela primeira vez no Continente Americano em 5 de março de 2001, no Paraguai. Na safra 2001/2002 já se alastrava por lavouras no Brasil, mais especificamente nos Estados do Rio Grande do Sul, Paraná, São Paulo, Mato Grosso do Sul, Goiás e Mato Grosso, acarretando pequenas perdas econômicas, minimizadas pela forte estiagem do período. A

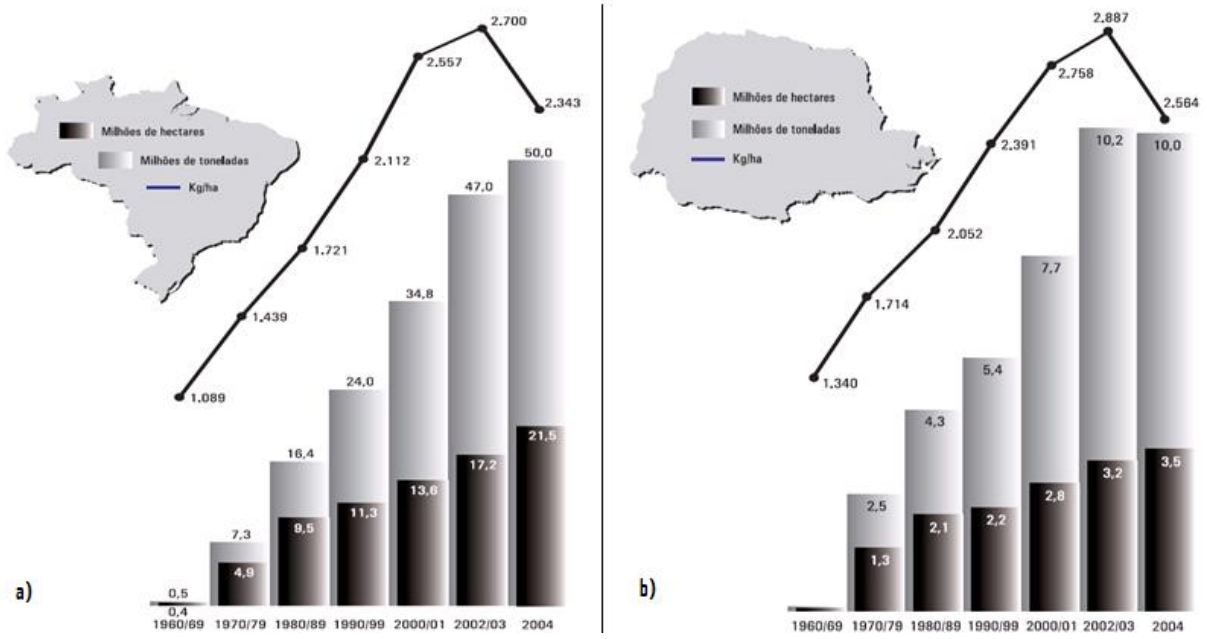

Figura A.1: Evolução da área, produção e produtividade da Soja: a)Brasil b)Paraná. (Fonte: [CONAB 2005])

região de Londrina (norte do Paraná) apresentou os primeiros indícios de ferrugem entre os dias 26 a 28 de maio 2001.

Uma doença imprevisível, causada pelo fungo *Phakopsora Pachyrhizi*, a Ferrugem Asiática, nativa do oriente, mais especificamente da China (ver mapa da disseminação Figura A.2) é uma fitopatologia que causa rápido amarelecimento e uma queda prematura das folhas, impedindo a plena formação dos grãos [Yorinori and Lazzarotto 2004] [Miles et al. 2003]. Quanto mais precoce sua desfolha, menor o grão e conseqüentemente, baixa qualidade e perda de rendimentos. Quando a doença atinge a Soja na fase de formação das vagens ou no início da granação pode causar o aborto e a queda das vagens.

Os primeiros sintomas da ferrugem apresentam pontos de coloração cinza-esverdeada de no máximo 1mm a 2mm de diâmetro. Neste local, inicia-se uma minúscula protuberância, semelhante a uma ferida, chamada urédia ou pústulas, caracterizando o começo da formação da estrutura da frutificação do fungo. Posteriormente, abre-se em um minúsculo poro expelindo daí os uredosporos. Esses uredosporos se acumulam ao redor dos poros ou são carregados pelo

vento, que combinado a outros fatores, aumentam a possibilidade de contaminação. Conforme prossegue a esporulação, as folhas adquirem coloração de castanho-claro a castanho-avermelhado, agravando a fitopatologia e justificando a escolha do nome ferrugem [Soja 2004].

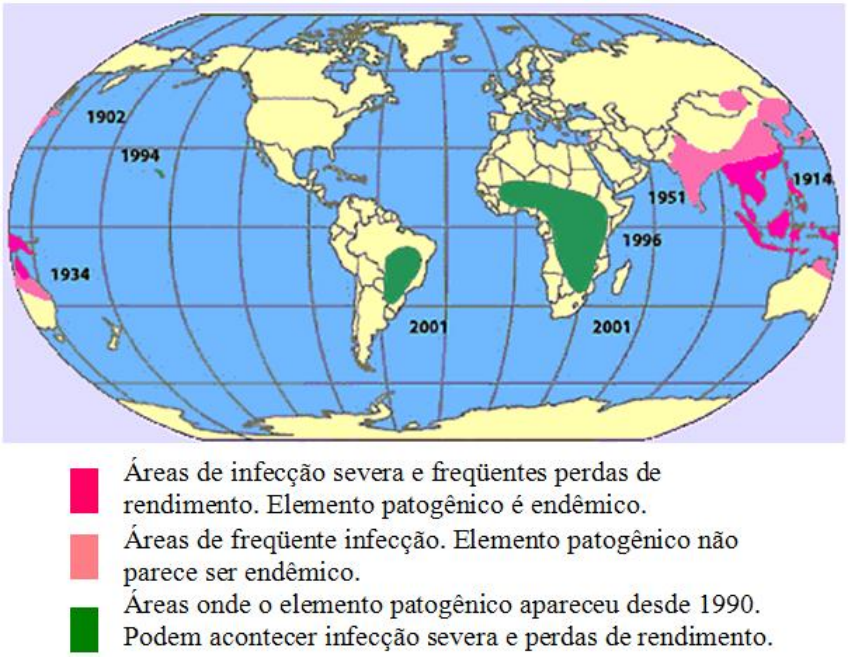

Figura A.2: Disseminação mundial da ferrugem da Soja causada pelo fungo *Phakopsora Pachyrhizi*. (Fonte: [Miles et al. 2003])

Os uredosporos são disseminados para outras lavouras, próximas ou distantes, não pelas sementes ou processados da Soja, mas através dos ventos, e a infecção é altamente dependente da umidade. Entre as condições predisponentes ao aparecimento da Ferrugem Asiática, com severas perdas de produtividade, estão os prolongados períodos de molhamento foliar, por chuva ou orvalho, e temperaturas médias abaixo de 28^{o}C, que não é um fator limitante. Mas, o fungo *Phakopsora Pachyrhizi* está adaptado a temperaturas que variam entre 15^{o}C e mais de 30^{o}C, com molhamento da folha acima de seis horas. Os uredosporos, à temperatura ambiente (25^{o}C e 27^{o}C), germinam em uma hora [Yorinori et al. 2003] e em condições favoráveis, as primeiras lesões aparecem em quatro ou cinco dias. As urédias e as esporulações em seis ou sete dias, após a inoculação.

Tabela A.2: Amostragem aproximada das perdas de grão pela Ferrugem Asiática. (Fonte: [Soja 2004])

Safra	**Área atingida**	**Produção (mil toneladas)**	**Perdas (mil toneladas)**	**Preço na Safra (tonelada)**	**Prejuízo (milhões)**
2001/02	60%	41.916,9	569,2	U$ 220,50	U$ 125.500
2002/03	90%	45.017,5	3.300,0	U$ 223,40	U$ 737.400
2003/04	+90%	49.770,1	4.600,0	U$ 260,80	U$ 1.200.000
			8.469,2		**U$ 2.062.900**

É por isso que nas regiões mais quentes o aparecimento da doença é mais difícil, e quando aparece, não se desenvolve de forma explosiva. Regiões com altitude superior a 700 metros são mais favoráveis à ocorrência da doença devido às temperaturas noturnas mais amenas, associadas a um maior número de horas de orvalho. Regiões mais baixas, porém com chuvas bem distribuídas, também são favoráveis para o rápido desenvolvimento.

Os grãos, sementes e alimentos, apresentam características higroscópicas, assim, quando em contato com a umidade do ar, são estabelecidas trocas de água, para se atingir o equilíbrio higroscópico. Dessa maneira, é criado um micro ambiente sobre a superfície, que é favorável à germinação de esporos e proliferação dos fungos.

A maior ou menor severidade da doença, depende então das condições climáticas e da proximidade da fonte de inóculo. Segundo [Yorinori et al. 2003] [Yorinori et al. 2004], algumas localidades apresentaram perdas de rendimento ao nível de lavoura, variando de 30% a 75% podendo até inviabilizar a colheita. A Tabela A.2 mostra que os prejuízos acumulados nas três safras descritas chegam a 8,5 milhões de toneladas, ou U$ 2,06 bilhões, isto em grãos, não contabilizando os gastos com produtos químicos (fungicidas e despesas com aplicações) [Filho and Richetti 2004] e perdas na arrecadação de impostos pelo governo. Somando-se esses gastos os prejuízos alcançam U$ 3,7 bilhões, no mesmo período [Yorinori et al. 2004].

A única forma de controle da Ferrugem Asiática é a aplicação de fungicida, mas é preciso saber reconhecer o momento certo [Filho and Richetti 2004]. Es-

tudos da EMBRAPA Soja [Yorinori et al. 2004] [Soja 2004] mostram que os fungicidas protegem em média cerca de 25 dias. Se aplicado no momento errado, o agricultor poderá ter que repetir as aplicações, o que aumenta sensivelmente os custos de produção. Por outro lado, se o produtor não estiver monitorando a lavoura, só vai perceber os sintomas quando for tarde demais, comprometendo assim a eficiência do produto.

A.5 Agrometeorologia

Dentre todas as atividades econômicas, a agricultura é, sem dúvida, a que sofre maior influência das condições do tempo e do clima. Com total dependência dos fatores atmosféricos em todo seu ciclo, da preparação do solo até o armazenamento do produto, quaisquer adversidades provocarão enormes prejuízos econômicos e graves impactos sociais. Sendo assim, a imprevisibilidade das condições do tempo a médio e longo prazo, promove a agricultura como uma atividade de alto risco [Pereira et al. 2002]. Deve-se ressaltar também, que no processo produtivo rural, os fatores meteorológicos influenciam no surgimento e na proliferação de doenças nos cultivares.

Um dos desafios da Meteorologia, ciência que se ocupa dos fenômenos físicos da atmosfera, é prever, com razoável antecedência, os resultados das movimentações desses fenômenos e suas possíveis conseqüências. Com um objetivo bem focado e uma visão especializada, a Agrometeorologia (ou Meteorologia Agrícola), se volta para as condições atmosféricas e suas conseqüências para o meio rural.

> Meteorologia Agrícola tem por objetivo colocar a ciência da Meteorologia a serviço da Agricultura em todas suas formas e facetas, para melhorar o uso da terra, para ajudar a produzir o máximo de alimentos, e a evitar o abuso irreversível dos recursos da terra [Smith 1975].

O programa de uso da terra baseado nos aspectos do clima procura fornecer elementos para o desenvolvimento agrícola sustentável, minimizando os efeitos

desfavoráveis da exploração agrícola sobre o ambiente, cumprindo restrições dos consumidores e satisfazendo condições de produção de alimentos. O Zoneamento Agroclimático resulta da delimitação da capacidade das regiões aos cultivos, relacionados ao fator clima [Pereira et al. 2002].

Acompanhamento in loco e a utilização da previsão do tempo constituem condições indispensáveis na tomada de decisão e no planejamento de operações cotidianas, ou seja, tornam-se ferramentas fundamentais para a operacionalização das atividades agrícolas.

Os estados da atmosfera podem ser descritos em termos instantâneos e estáticos [Pereira et al. 2002] [INMET 2005], o primeiro, chamado tempo, são condições atuais em um local e instante, caracterizado pela temperatura, pressão, concentração de vapor, velocidade e direção do vento e precipitação. O segundo, chamado clima, expressa as condições médias, valor mais provável, geralmente um intervalo de 30 anos do sequenciamento do tempo neste mesmo local. Este período, 30 anos, chamado Normal Climatológica, foi escolhido pela Organização Meteorológica Mundial (OMM) com base em princípios estatísticos de tendências do valor médio. Desse modo, inclui-se anos de desvios para mais e para menos em todos os elementos do clima. Usando períodos menores, invariavelmente, conduzia-se a conclusões inconsistentes [Pereira et al. 2002].

Mudanças climáticas, são tendências que ocorrem nas condições regionais, num período razoavelmente longo de tempo (década, século), para uma grande região. Os causadores dessas mudanças são fenômenos naturais (vulcões, atividades solar), sem qualquer influência humana, e mais aqueles desencadeados realmente pelas atividades humanas (desmatamento, poluição, urbanização). Por exemplo, a necessidade de incorporar novas áreas na produção de alimentos pressiona o desmatamento e sua substituição por plantas de ciclo menor, com impacto sobre o clima local e regional.

Fatores importantes para a previsão do tempo e manejo agrícola, os fenôme-

nos atmosféricos e os efeitos da ação humana são desdobrados em três escalas: macro-escala, meso-escala e micro-escala.

i) Macro-escala: fenômeno em escala regional ou geográfica. O (Macro)clima condiciona o clima de grandes regiões pelos fatores geográficos como latitude, altitude, etc.

ii) Meso-escala: fenômenos em escala local. O (Topo)clima ou (Meso)clima é caracterizado pela topografia regional ou relevo local. Faces do terreno voltadas ao sol são mais quentes e mais secas, terrenos côncavos acumulam o ar frio, agravando o efeito das geadas, etc.

iii) Micro-escala: fenômenos em escala local, em uma área de proporções reduzidas. O (Micro)clima é influenciado principalmente pela cobertura do terreno como solo nu, gramado, floresta, cultura rasteira, represa, etc, que determina o balanço local de energia.

As escalas meteorológicas são orientadas segundo os elementos e fatores climáticos. Os elementos, também chamados variáveis, são grandezas que caracterizam o estado da atmosfera, são eles: radiação solar, temperatura, umidade relativa, pressão, velocidade e direção do vento, precipitação. Os fatores são agentes causais que condicionam os elementos climáticos, sendo eles: latitude, altitude, oceanidade, continentalidade, tipo de corrente oceânica. Para exemplificar esta influência, quanto maior a altitude, menores a temperatura e a pressão.

Livros Grátis

(http://www.livrosgratis.com.br)

Milhares de Livros para Download:

Baixar livros de Literatura
Baixar livros de Literatura de Cordel
Baixar livros de Literatura Infantil
Baixar livros de Matemática
Baixar livros de Medicina
Baixar livros de Medicina Veterinária
Baixar livros de Meio Ambiente
Baixar livros de Meteorologia
Baixar Monografias e TCC
Baixar livros Multidisciplinar
Baixar livros de Música
Baixar livros de Psicologia
Baixar livros de Química
Baixar livros de Saúde Coletiva
Baixar livros de Serviço Social
Baixar livros de Sociologia
Baixar livros de Teologia
Baixar livros de Trabalho
Baixar livros de Turismo