JEAN MARCOS LAINE

Uma Metodologia para Desenvolvimento de Programas Paralelos Eficientes em Ambientes Homogêneos e Heterogêneos

São Paulo
2008

JEAN MARCOS LAINE

Uma Metodologia para Desenvolvimento de Programas Paralelos Eficientes em Ambientes Homogêneos e Heterogêneos

Tese apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Doutor em Engenharia Elétrica

Área de Concentração:
Sistemas Digitais

Orientador:
Prof. Dr. Edson T. Midorikawa

São Paulo
2008

AGRADECIMENTOS

À minha mãe Maria, meu pai Jacir, meus irmãos Ivan e Alan pelo apoio em todos os momentos da minha vida. Em palavras eu não conseguiria expressar o que sinto e penso sobre cada um de vocês. Sou grato pela educação, pelo carinho, pela amizade e por toda a dedicação nestes 30 anos. Vocês são tudo pra mim.

À minha namorada Patricia, pela compreensão, paciência, apoio e constante incentivo. Seu amor e entrega me fizeram mais forte para superar os problemas e os desafios. Sua alegria e seu espírito de criança iluminam minha vida e me contagiam. Na sua presença eu me sinto mais feliz. T.A.P.S.

Ao meu orientador, Professor Dr. Edson Toshimi Midorikawa, pela oportunidade de desenvolver este trabalho. Suas orientações, muitas vezes, não foram só para o desenvolvimento desta tese e sim para a vida. Nestes sete anos (Mestrado e Doutorado) tive a oportunidade de reconhecer e admirar seu conhecimento e a pessoa que é.

Aos meus amigos, Hélio e José Roberto, pela amizade, compreensão e paciência durante estes longos anos de trabalho. Agradeço cada momento de reflexão, pois, muitas vezes, me fizeram enxergar além das minhas limitações. Serei sempre grato a vocês, "meus irmãos". "Talvez as melhores amizades sejam aquelas em que haja muita discussão, muita disputa e mesmo assim muito afeto"(George Eliot).

Aos amigos e companheiros do LAHPC, em especial à Professora Dra. Liria M. Sato, pelas sugestões sempre admiráveis. A todos que de uma forma ou de outra colaboraram no desenvolvimento desta pesquisa.

E sobretudo, a Deus, por estar sempre ao meu lado, guiando meus passos e dando força para prosseguir mesmo diante de tantas dificuldades. Se este trabalho está se concretizando é graças a Ele, tenho certeza disso.

"Take the first step in faith. You don't have to see the whole staircase, just take the first step."
(Martin Luther King Jr.)

"A alegria está na luta, na tentativa, no sofrimento envolvido. Não na vitória propriamente dita."
(Mahatma Gandhi)

RESUMO

Uma metodologia para desenvolvimento de programas paralelos eficientes deve especificar mecanismos capazes de caracterizar o comportamento das aplicações e permitir estudos sobre o desempenho de diferentes modelos de soluções. Nos ambientes distribuídos, em particular, a eficiência da solução também está relacionada à estratégia utilizada na divisão e distribuição do trabalho entre os processos que cooperam na solução do problema. Para abordar esses aspectos, uma metodologia, denominada PEMPIs-Het (*Performance Estimation of MPI Programs in Heterogeneous Systems*), é especificada e apresentada nesta tese. A metodologia permite a modelagem, avaliação e predição de desempenho de programas paralelos em ambientes homogêneos e heterogêneos. Técnicas de modelagem analítica são utilizadas para representar o comportamento das aplicações no ambiente distribuído. Um modelo gráfico, denominado DP*Graph^{++}, é proposto para ilustrar as principais estruturas do código da aplicação e facilitar análises sobre a complexidade algorítmica do programa. Algumas aplicações são modeladas e a precisão das predições é verificada através de testes experimentais. Os modelos de desempenho permitem uma estimativa pontual para o tempo de execução da aplicação. Entretanto, uma estratégia alternativa, baseada em intervalos de predição, também é discutida e avaliada. Algumas estratégias para balanceamento de carga de aplicações paralelas distribuídas são implementadas e avaliadas. Essas estratégias utilizam informações de um vetor com índices de desempenho (*Vector of Relative Performances – VRP*), gerados pelos modelos analíticos, para especificar a divisão e distribuição do trabalho. Esses índices caracterizam a capacidade computacional das máquinas. Uma formalização matemática é apresentada para explicar como os índices são determinados. Testes experimentais são realizados para verificar a aplicabilidade das estratégias e a eficiência no balanceamento das cargas.

ABSTRACT

A methodology for developing efficient parallel programs must specify mechanisms capable of characterizing the behavior of applications and allow studies on the performance of different solution models. In distributed environments, in particular, the solution efficiency is also related to strategy adopted in the workload division and distribution among the processes which cooperate in the solution of the problem. To address these issues, a methodology, called PEMPIs-Het (*Performance Estimation of MPI Programs in Heterogeneous Systems*), is specified and presented in this thesis. The methodology allows performance modeling, evaluation and prediction of parallel programs in homogeneous and heterogeneous environments. Analytical modeling techniques are used to represent the applications behavior in the distributed environment. A graph model, called DP*Graph^{++}, is proposed to illustrate the main structures of the application code and facilitate some analyses about the program algorithmic complexity. Some applications are modeled and the accuracy of predictions is verified by experimental tests. The models allow estimate a punctual performance to the application execution time. Meanwhile, an alternative strategy, based on prediction intervals, is also discussed and evaluated. Some strategies for load balancing of distributed parallel applications are implemented and evaluated. These strategies use information from a vector with performance indexes (*Vector of Relative Performances - VRP*), generated by analytical models to specify the division and distribution of work. These indexes characterize the machines computational capacity. A mathematical formalization is presented to explain how the rates are determined. Experimental tests are conducted to verify the applicability and effectiveness of the strategies in load balancing.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

A constante evolução dos sistemas computacionais tem provocado mudanças significativas em diferentes áreas da computação (DONGARRA; LASTOVETSKY, 2006). A restrição quanto à capacidade de processamento das máquinas limitava, indubitavelmente, a quantidade e a classe de problemas que podiam ser resolvidos computacionalmente. Isso porque a capacidade de processamento exigida por grande parte das aplicações de larga escala ultrapassava o que as máquinas ou sistemas computacionais do passado podiam oferecer. Com os avanços na área da computação e o desenvolvimento de sistemas computacionais de alta capacidade de processamento, diversas áreas do conhecimento humano têm se beneficiado. Os recentes resultados alcançados pela física, química e biologia computacional comprovam o impacto desta crescente e significativa evolução.

Graças ao aumento da capacidade de processamento dos computadores pessoais e a velocidade de transmissão de dados alcançada pelas redes de interconexão, ambientes de processamento distribuído foram criados e desenvolvidos, passando a oferecer uma alternativa viável para quem deseja processar grandes volumes de dados. Neste contexto, podemos citar as redes de estações de trabalho, os *clusters*, os *grids* computacionais e, mais recentemente, os *clouds* (NÉMETH; SUNDERAM, 2003, 2002; FOSTER; KESSELMAN, 2003; FOSTER et al., 2002; FOSTER; KESSELMAN; TUECKE, 2001). Por causa das características destes sistemas, como custo/benefício, por exemplo, eles se tornaram atraentes para a execução de aplicações de larga escala e/ou de alto desempenho. Dessa forma, os tradicionais computadores paralelos centralizados estão perdendo espaço para estes novos ambientes de computação (CLARK et al., 1995).

Concomitantemente a essa evolução, surgiram algumas abordagens para o desenvolvimento de aplicações paralelas em sistemas distribuídos. Bibliotecas com funções para comunicação remota foram criadas, como o PVM (GEIST et al., 1994) e a padronização MPI (SNIR; OTTO, 1998). Além disso, diferentes modelos de programação foram definidos e novas estruturas de soluções foram criadas para explorar as características relacionadas à organização de cada sistema, visando melhorar o desempenho das aplicações. Hoje em dia, os modelos mestre-escravo, *bag of tasks*, *peer-to-peer* e cliente-servidor (BUYYA,

1999b) são os mais utilizados para organização do processamento paralelo e distribuído.

Mesmo com essa evolução e a inevitável mudança de paradigma, os programadores, em geral, ainda têm pouco conhecimento e encontram muita dificuldade ao tentar implementar uma solução paralela distribuída eficiente. Essa deficiência pode ser explicada, em parte, pela ausência de disciplinas específicas sobre o tema na grade curricular da maioria dos cursos de graduação na área de computação. Tradicionalmente, os cursos se limitam a ensinar os modelos convencionais de programação seqüencial e não discutem aspectos relacionados ao paralelismo em nenhuma arquitetura.

Se a programação paralela em sistemas multiprocessados ou multinúcleos, baseada em variáveis de memória compartilhada, já é uma tarefa complicada, ainda mais difícil é o desenvolvimento de aplicações paralelas eficientes em arquiteturas distribuídas heterogêneas, como discutido em (BADIA et al., 2007). Isso porque é necessário considerar diversos aspectos, relacionados ao desempenho do programa, que não estão presentes na implementação das aplicações que utilizam variáveis de memória compartilhada. Alguns desses aspectos dizem respeito à comunicação entre os processos remotos, às sincronizações estabelecidas entre as tarefas e às transferências de dados necessárias na execução do programa. Além de explorar o paralelismo intrínseco às aplicações, o programador também deve, na maioria das vezes, especificar como o trabalho será dividido e distribuído entre os nós de processamento.

Um programa paralelo eficiente deve utilizar os recursos computacionais do sistema de modo otimizado. Portanto, o intuito do desenvolvimento de aplicações eficientes é maximizar o uso dos elementos da arquitetura, principalmente do processador. Neste sentido, todas as estratégias propostas neste trabalho, integradas ao escopo da metodologia especificada, buscam melhorar a eficácia das aplicações paralelas distribuídas, principalmente em relação ao desempenho computacional. A eficiência das aplicações pode ser determinada pela seguinte equação:

$$E_p = \frac{S_p}{p} \tag{1.1}$$

Esta fórmula investiga se os processadores do sistema computacional estão sendo utilizados de forma útil pelo programa. Na equação, S_p representa a aceleração (*speedup*) (WILKINSON; ALLEN, 1999) da aplicação paralela usando p processadores. Nos casos onde a aceleração é linear, a eficiência da solução é de 100%. Outra possível métrica, definida no desenvolvimento deste trabalho, para a análise da eficiência das aplicações é a DFAT (*Distance From Average Time*) (LAINE; MIDORIKAWA, 2008a), apresentada na Seção 5.5.1. Essa métrica define o grau de variação dos tempos de execução de cada um dos processos em relação ao tempo médio calculado. Um programa eficiente apresenta uma variação pequena em relação aos valores calculados para esta métrica. Este fato é alcançado se o balanceamento de carga do sistema for preciso.

Para realizar um balanceamento de carga adequado, em sistemas distribuídos heterogêneos, é preciso levar em consideração diversos aspectos, como a capacidade de processamento individual de cada uma das máquinas e o estado computacional dos nós no momento da distribuição. Por estes e outros motivos é que muitos trabalhos vêm sendo publicados apresentando estratégias alternativas para tentar resolver a questão (CASAVANT; KUHL, 1988), (MAHESHWARI, 1996), (IBRAHIM; XINDA, 2002), (LIU; GOLDSMITH, 2005), (SHIH; YANG; TSENG, 2006) e (CIORBA et al., 2006).

Em geral, a implementação de aplicações paralelas distribuídas fazem uso do modelo de programação denominado mestre-escravo. Nessa organização, o processo mestre é o responsável por decompor o problema em pequenas tarefas e distribuí-las aos processos escravos. Os escravos recebem o trabalho, processam e devolvem o resultado para o mestre que reorganiza a solução final. Toda a organização da divisão e distribuição do trabalho aos escravos pode fazer uso de estratégias de balanceamento de carga.

As aplicações *bag of tasks* (CIRNE et al., 2003) representam um tipo específico do modelo mestre-escravo. As tarefas são independentes, não se comunicam e podem ser executadas em qualquer ordem. Aplicações como mineração de dados, simulações Monte Carlo e processamento de imagens utilizam esse modelo de programação (CIRNE et al., 2003).

Diante dessas inúmeras alternativas para o desenvolvimento de aplicações paralelas distribuídas, a existência de metodologias para análise, avaliação e predição de desempenho passa a ser essencial no desenvolvimento de aplicações eficientes. Na literatura, é possível encontrar alguns trabalhos elaborados para auxiliar atividades relacionadas a estudos de desempenho de aplicações paralelas e distribuídas (DIAS, 2006; GROVE, 2003; CULLER et al., 1993; SCHOPF, 1998). Entretanto, nenhum deles aborda todos os aspectos de interesse desta tese. Por este motivo, resolvemos formalizar uma nova metodologia capaz de guiar atividades relacionadas ao estudo de desempenho de aplicações paralelas distribuídas. Um dos aspectos não mencionados nos trabalhos avaliados diz respeito às comparações de diferentes modelos de soluções. Por isso, um dos objetivos de nossa metodologia é especificar mecanismos que possam ser utilizados para determinar o modelo de organização de solução mais adequado a um determinado problema.

A descrição da metodologia PEMPIs-Het (LAINE; MIDORIKAWA, 2007b, 2007a) é realizada no Capítulo 4. A metodologia formaliza alguns processos e atividades para realizar avaliações e análises de desempenho de aplicações paralelas distribuídas. Atividades voltadas ao planejamento da divisão e distribuição das cargas computacionais em sistemas distribuídos também são contempladas.

Para a modelagem das aplicações, especificamos um conjunto de técnicas analíticas. Estas técnicas utilizam o método de ajuste de curvas dos mínimos quadrados para modelar o comportamento das aplicações. Modelos de desempenho individuais são elaborados para caracterizar a aplicação em cada tipo de máquina do ambiente. Depois, estes modelos específicos podem ser combinados para estimar o tempo de execução da aplicação sobre

um conjunto de máquinas diferentes.

Implementamos algumas estratégias para balanceamento de carga em sistemas distribuídos homogêneos e também heterogêneos. As estratégias levam em consideração a capacidade computacional das máquinas para ajustar a divisão e distribuição do trabalho. Para isso, definimos um vetor de relações de desempenho que contém índices que expressam a capacidade relativa de cada uma das máquinas do ambiente. Baseado nestes índices, as estratégias definem um plano de execução inicial para as aplicações, considerando a distribuição das cargas computacionais. Abordagens estáticas, dinâmicas e adaptativas foram especificadas, implementadas e avaliadas.

No modelo de distribuição estático, todo o trabalho é dividido e distribuído no início da execução do programa. Dependendo das características da máquina, uma fatia maior ou menor de trabalho é enviado ao processo que lá estará sendo executado. O intuito das estratégias é ajustar a distribuição do trabalho para que todos os processos terminem a execução do trabalho de modo sincronizado. O sincronismo perfeito é ideal mas impossível de ser atingido. No entanto, quanto mais ajustado estiver o tempo de execução de cada processo, melhor o desempenho final da solução.

Na estratégia dinâmica, o trabalho é dividido e distribuído por fases de processamento. Os processos que são executados em máquinas de maior capacidade de processamento irão realizar a maior parte do trabalho. Se durante a execução da aplicação, uma máquina tiver sua carga computacional aumentada, o processo que lá estiver sendo executado vai contribuir menos com a solução do problema. Esta estratégia não modifica o tamanho do trabalho que deve ser enviado aos processos escravos durante a execução do programa.

Já na versão adaptativa de balanceamento, a estratégia consegue reagir às modificações de carga computacional do ambiente e, em tempo de execução, alterar o plano de distribuição inicial. Dessa forma, estaremos evitando que algumas máquinas apresentem sobrecargas de processamento enquanto outras fiquem ociosas ao longo da execução das aplicações. Para isso, o processo mestre avalia o tempo de resposta de cada escravo e decide se aumenta ou diminui a quantidade de trabalho que será enviado ao mesmo na próxima fase do processamento.

Demonstramos neste trabalho como avaliar e comparar o desempenho de diferentes modelos de soluções. Essas atividades podem indicar qual modelo é mais adequado para uma determinada aplicação, prevendo o comportamento de cada versão do programa.

1.1 Justificativa

Cada vez mais, os multicomputadores estão sendo utilizados em projetos na área de computação de alto desempenho, tentando atender à demanda das aplicações por capacidade de processamento. A lista dos maiores sistemas computacionais, divulgada no TOP500[1],

[1] www.top500.org

deixa clara a importância desses ambientes de processamento distribuído em diferentes segmentos de pesquisas. Portanto, é necessário elaborar metodologias e ferramentas de análise e avaliação de desempenho capazes de contribuir para a qualidade da solução ou da arquitetura do sistema.

Vários trabalhos disponíveis na literatura abordam aspectos relacionados ao tema. No entanto, as avaliações e estudos, muitas vezes, estão restritos aos sistemas distribuídos homogêneos, como acontece em (CHING; GAUDIOT; SATO, 2002; JUHÁSZ, 2006; GEMUND, 1993). Talvez isso possa ser explicado se considerarmos que, até pouco tempo, os *clusters* homogêneos predominavam nessa área da computação. Logo, é necessário criar metodologias e estratégias mais amplas para auxiliar não só o desenvolvimento de aplicações mas também avaliações de desempenho de aplicações paralelas em ambientes heterogêneos de computação distribuída.

Atualmente, as máquinas têm sua capacidade de processamento limitada por propriedades físicas, o que restringe, principalmente, o aumento da freqüência de operação dos processadores. Conseqüentemente, uma alternativa para melhorar o desempenho das soluções é abordar o problema na própria camada de aplicação do sistema. Com esse enfoque, explorar o paralelismo existente nas aplicações é uma tendência natural ao tentar diminuir o tempo de execução dos programas. Até mesmo as máquinas convencionais do tipo *von Neumann* (STALLINGS, 1999) evoluíram nesse sentido e hoje utilizam processadores multinúcleos em sua arquitetura. Esse fato nos permite utilizar linguagens que dão suporte ao uso de *threads* na implementação dos programas. Além disso, é possível desenvolver aplicações com linguagens paralelas, como o CPAR-Cluster (CRAVEIRO; SATO, 2004). Esse novo cenário, dentro da computação, vem justificar a importância de uma metodologia com os propósitos já citados neste trabalho.

1.2 Objetivos

O objetivo principal deste trabalho é especificar e formalizar uma metodologia e um conjunto de técnicas que possam auxiliar os programadores a desenvolver aplicações paralelas eficientes em sistemas distribuídos homogêneos e heterogêneos. Nesse intuito, a metodologia deve oferecer meios para que se possa avaliar e comparar, não só quantitativamente, diferentes arquiteturas de *software* ou modelos de soluções e determinar qual estratégia é a mais adequada para o ambiente que será utilizado.

Para alcançar esse objetivo, um conjunto de técnicas e estratégias de modelagem, avaliação e predição de desempenho foi especificado e implementado. Além de permitir o desenvolvimento de modelos matemáticos para representar o comportamento das aplicações é possível estimar o tempo de execução dos programas e comparar diferentes soluções variando parâmetros de entrada, como a quantidade de processos e o tamanho do problema. Nosso propósito é utilizar essas estimativas não só em estudos e análises de

desempenho, mas também em atividades relacionadas a divisão e distribuição de carga. Dessa forma, criamos a possibilidade de usar essas análises para comparar e escolher a melhor estratégia de paralelismo para uma aplicação distribuída.

Assim, desejamos que a metodologia ofereça condições para que os programadores possam seguir um protocolo que os auxilie na implementação de programas paralelos distribuídos eficientes. A idéia é que a organização das soluções paralelas possa ser melhorada, que a sincronização entre os processos seja mais ajustada e que, com isso, a aplicação apresente bons índices de desempenho no ambiente de execução.

Dessa forma, ao atingir os objetivos citados, este trabalho deve apresentar as seguintes contribuições:

- definição de uma metodologia para avaliação, análise e predição de desempenho de programas paralelos em ambientes distribuídos, homogêneos ou heterogêneos;
- especificação de mecanismos que permitem a avaliação e comparação de diferentes modelos de soluções paralelas distribuídas;
- avaliação dos fatores que podem afetar a precisão dos modelos analíticos de desempenho;
- especificação e implementação de estratégias alternativas para balanceamento de cargas utilizando índices de desempenho gerados por modelos analíticos de predição;
- integração de atividades de modelagem de desempenho e distribuição de cargas computacionais com objetivos voltados a otimizações de recursos computacionais e redução do tempo de execução das aplicações;
- comparação e análise de algumas estratégias de paralelização em aplicações distribuídas;
- revisão atualizada das referências bibliográficas sobre os temas: avaliação, análise, predição de desempenho e balanceamento de carga em sistemas distribuídos.

1.3 Metodologia

Inicialmente, trabalhamos na especificação da metodologia PEMPIs (*Performance Prediction of MPI Programs*) (MIDORIKAWA; OLIVEIRA; LAINE, 2005). Essa metodologia foi criada para formalizar um conjunto de estratégias de modelagem e predição de desempenho de programas paralelos MPI em *clusters* homogêneos (LAINE, 2003).

Pelo fato da metodologia PEMPIs restringir as análises e avaliações a sistemas distribuídos homogêneos, resolvemos estender o trabalho para ambientes heterogêneos. Investigamos os problemas, as dificuldades, as estratégias e as metodologias existentes nessa

área para abordar o tema em sistemas heterogêneos. Nessa pesquisa descobrimos que a maioria das metodologias faz restrições quanto à heterogeneidade do ambiente e não se preocupam em auxiliar os programadores a avaliar possíveis modelos ou paradigmas de programação paralela e distribuída.

Na extensão, alteramos o nome da metodologia para PEMPIs-Het (*Performance Estimation of MPI Programs in Heterogeneous Systems*) (LAINE; MIDORIKAWA, 2007a) e realizamos várias modificações nas estratégias de modelagem propostas anteriormente. Essas modificações incluíram tanto simples alterações na simbologia criada para o modelo gráfico da aplicação quanto na forma de avaliar e modelar as aplicações. Uma modificação importante nas estratégias de modelagem foi a inclusão de metamodelos capazes de estimar o desempenho da aplicação em um conjunto distinto de máquinas heterogêneas. Esses metamodelos são criados a partir dos modelos individuais gerados para a aplicação em cada uma das máquinas do sistema. Para refinar as estratégias de modelagem foi necessário testar experimentalmente algumas aplicações e, exaustivamente, avaliar as conseqüências das alterações realizadas.

Com os modelos analíticos ajustados para ambientes heterogêneos, resolvemos aplicá-los em atividades de balanceamento de carga. Nesse intuito, elaboramos algumas estratégias de divisão e distribuição de cargas computacionais. Inicialmente, definimos um vetor de desempenho relativo, chamado VRP, para caracterizar a capacidade relativa das máquinas do sistema. Esse vetor é elaborado através dos modelos analíticos da aplicação, gerados para cada uma das diferentes máquinas do ambiente. Os índices especificados nesse vetor indicam o quanto cada uma das máquinas é mais rápida ou mais lenta em relação às demais, durante o processamento da aplicação avaliada.

Logo, esses índices podem ser utilizados para direcionar a divisão e distribuição das cargas computacionais, de modo que o tamanho da tarefa seja ajustado à capacidade individual de cada uma das máquinas. As técnicas propostas foram testadas e ajustadas para otimizar a utilização dos recursos e reduzir o tempo de execução das aplicações. Durante o desenvolvimento das estratégias, implementamos um monitor de desempenho capaz de capturar informações pontuais sobre a carga computacional de cada um dos nós do sistema e ajudar nas tomadas de decisões. Esse monitor foi utilizado para implementar uma estratégia dinâmica e adaptativa de distribuição de carga.

1.4 Organização do Texto

Além deste capítulo de introdução, este texto possui um capítulo sobre os tipos de sistemas para computação paralela; um capítulo sobre análise, avaliação e predição de desempenho; um capítulo apresentando e formalizando a metodologia PEMPIs-Het; um capítulo com a apresentação e análise de resultados experimentais; e um último capítulo que apresenta as conclusões e discussões finais.

O Capítulo 2 possui algumas considerações sobre computação paralela. Inicialmente, é apresentado um conjunto de terminologias relacionadas à área da computação paralela. Depois, discutimos duas taxonomias para arquiteturas de computadores e descrevemos, resumidamente, os modos de acesso à memória e os possíveis modelos de programação paralela. O capítulo termina com uma comparação entre *clusters* e *grids*.

O Capítulo 3 é iniciado com uma descrição das principais etapas envolvidas em uma metodologia de análise, avaliação e predição de desempenho. Posteriormente, técnicas para avaliação de desempenho, como medições, redes de Petri, redes de filas e simulações são apresentadas. No final do capítulo, alguns trabalhos relacionados a essa linha de pesquisa são discutidos.

O Capítulo 4 descreve a metodologia PEMPIs-Het, especificada para modelar e estimar o desempenho de programas paralelos em sistemas distribuídos. Cada um dos módulos que formam a metodologia é detalhado e as técnicas de modelagem elaboradas, visual e analítica, são explicadas. Nesse capítulo também apresentamos as estratégias de distribuição de carga, elaboradas para fazer uso dos modelos de desempenho e otimizar a execução das aplicações, e algumas estratégias de paralelismo para soluções distribuídas.

O Capítulo 5 destaca o desenvolvimento e a aplicação dos modelos analíticos em atividades de predição de desempenho em ambientes distribuídos homogêneos e heterogêneos. É feito uma avaliação das estratégias criadas para distribuição de carga e uma comparação entre modelos de soluções paralelas. Para as avaliações, modelamos e analisamos três aplicações paralelas.

O último capítulo desta tese apresenta as conclusões e considerações sobre os resultados obtidos no desenvolvimento deste trabalho. Além disso, as principais contribuições geradas são destacadas, os artigos publicados são citados e as sugestões de trabalhos futuros são comentadas.

Capítulo 2

Computação Paralela

Até pouco tempo atrás, pensava-se que para melhorar o desempenho dos computadores e atender às necessidades das aplicações seria suficiente aumentar a capacidade de processamento e armazenamento das máquinas. No entanto, ao descobrir e esbarrar nos limites físicos e tecnológicos, inerentes à arquitetura tradicional de von Neumann, verificou-se que essa estratégia precisava ser superada. Desde então, a computação paralela passou a ser avaliada como uma alternativa capaz de melhorar cada vez mais o desempenho das aplicações e maximizar o uso dos recursos computacionais.

2.1 Introdução

Atualmente, diversos modelos de arquiteturas ou sistemas computacionais dão suporte ao desenvolvimento e execução de programas paralelos. Esses ambientes, em geral, são classificados de acordo com o tipo de organização da memória. As máquinas paralelas centralizadas (*SMP – Symmetric Multiprocessing*) são caracterizadas por possuírem várias unidades de processamento mas uma única memória global compartilhada e igualmente acessível pelos processadores. O desenvolvimento de programas paralelos para essa plataforma costuma utilizar *threads* ou diretivas OpenMP (CHANDRA et al., 2001). Normalmente, a comunicação entre os processos acontece através de variáveis de memória compartilhada.

Já os sistemas formados por multicomputadores (*MPP – Massively Parallel Processing*) não possuem uma memória global única. Conseqüentemente, a comunicação entre os processos deve ser modificada. Toda a troca de dados entre os processos distribuídos acontece através de mensagens que são transmitidas pela rede de interconexão, em geral proprietária e de alta velocidade. Essa plataforma oferece maior escalabilidade que a anterior e, possivelmente, maior capacidade computacional. Nesse ambiente, é necessário fazer uso de bibliotecas para trocas de mensagens entre os processos distribuídos. Além disso, a latência da rede de interconexão tem grande influência sobre o desempenho das aplicações.

Com a convergência de duas áreas em constante evolução tecnológica, as redes de interconexão e a arquitetura de computadores, surgiu uma nova infra-estrutura de computação distribuída, conhecida como NOWs (*Networks of Workstation*). A idéia é agrupar recursos computacionais simples, como PCs (*Personal Computers*), e formar um ambiente computacional integrado através de uma rede de interconexão. A possibilidade de construir um ambiente com alta capacidade de processamento a baixo custo despertou o interesse da comunidade que trabalha com programação paralela. Assim, surgiram ferramentas e bibliotecas de programação distribuída, como, por exemplo, o MPI (*Message Passing Interface*) (SNIR; OTTO, 1998; GROPP et al., 1998).

A relação custo/benefício e a escalabilidade desses ambientes quanto à quantidade de recursos computacionais, fizeram desses sistemas uma alternativa viável para o processamento paralelo e distribuído. Por esse motivo, aplicações de larga escala, que demandam grande capacidade de processamento, estão sendo implementadas e executadas em plataformas como *clusters* e *grids* computacionais[1]. Portanto, a computação paralela distribuída pode ser vista como uma união de duas áreas distintas da computação: a Computação Paralela e os Sistemas Distribuídos.

Nas próximas seções serão abordados diversos temas ligados à computação paralela. Alguns termos-chave da área serão definidos, e mais detalhes sobre a organização das arquiteturas serão apresentados.

2.2 Termos-chave

Na computação paralela existem alguns termos que definem conceitos importantes e necessários para a compreensão dos assuntos relacionados à área. Dentre os conceitos apresentados a seguir, alguns estão relacionados ao tipo de paralelismo existente nas aplicações e outros fazem referência a algumas medidas de desempenho. Portanto, antes de prosseguir neste capítulo, é necessário apresentar e definir esses termos. De acordo com (OLIVEIRA, 2006; BUYYA, 1999b), os principais conceitos são:

- tarefa: pode ser definida como uma sessão discreta de trabalho computacional;
- tarefas paralelas: são tarefas computacionalmente independentes e que podem ser executadas simultaneamente;
- problema paralelizável: um problema é dito paralelizável se a aplicação pode ser implementada através de tarefas paralelas;
- concorrência: disputa entre os processos por recursos computacionais do ambiente;

[1]Várias aplicações para *grid*, que utilizam recursos espalhados por todo o mundo, podem ser encontradas no site http://www.worldcommunitygrid.org

- paralelismo: representa a execução simultânea de processos independentes ou de tarefas de uma mesma aplicação;
- paralelismo de dados: cada tarefa executa um mesmo conjunto de instruções, mas sobre dados diferentes;
- paralelismo funcional: cada tarefa executa um conjunto distinto de instruções, podendo ser sobre um mesmo conjunto de dados ou não;
- aceleração (*speedup*): em geral representa a relação entre o tempo de execução de uma solução seqüencial (T_1) e o tempo de execução de uma solução paralela (T_p) com p processadores: $S_p = \frac{T_1}{T_p}$;
- eficiência: é uma medida do grau de utilização dos processadores e pode ser definida pela seguinte fórmula: $E_p = \frac{Sp}{p}$. A eficiência pode ser determinada e apresentar valores entre 0 e 1. Algoritmos com aceleração linear e soluções executadas em um único processador apresentam uma eficiência igual a 1;
- sincronização: é uma coordenação temporal de tarefas paralelas. Geralmente, a sincronização envolve a espera de duas ou mais tarefas, em um determinado ponto da computação, para que se possa prosseguir na execução das instruções;
- custo adicional do paralelismo: tempo gasto com as atividades envolvidas na coordenação das tarefas paralelas, envolvendo o tempo para iniciar uma tarefa paralela, o tempo para terminar uma tarefa, o tempo gasto com atividades de sincronização e replicação, entre outras;
- granularidade: também conhecida como nível de paralelismo, a granularidade diz respeito ao tamanho da unidade de trabalho atribuída a um processador. Essa grandeza pode ser classificada em três níveis: fina, média e grossa. Um paralelismo de granularidade grossa representa o paralelismo de execução entre aplicações distintas. Já um paralelismo de granularidade fina indica a execução simultânea de diferentes instruções de um mesmo programa.

2.3 Taxonomias

As arquiteturas paralelas evoluíram ao longo dos anos e, ao mesmo tempo, novos paradigmas de programação paralela foram criados para atender às necessidades de cada ambiente. Em geral, os sistemas computacionais influenciam diretamente no modo de organização das soluções. Nesse contexto, o modo de acesso à memória e o modelo de comunicação entre os processos é o que, normalmente, direciona a programação.

Tabela 2.1: Taxonomia de Flynn.

	Fluxo de Instrução Único	**Múltiplos Fluxos de Instrução**
Fluxo de Dados Único	SISD	MISD
Múltiplos Fluxos de Dados	SIMD	MIMD

Assim, algumas classificações (taxonomias) foram elaboradas para tentar representar e organizar os diferentes sistemas computacionais, seja ele seqüencial ou paralelo. Em geral, essas classificações tentam agrupar elementos computacionais através de características associadas ao modo de processamento ou do acesso aos dados. Baseado nessa classificação podemos mapear os diferentes ambientes computacionais e identificar aqueles que permitem um mesmo modelo de programação.

2.3.1 Taxonomia de Flynn

A taxonomia de Flynn (FLYNN, 1972; FLYNN; RUDD, 1996), apresentada na Tabela 2.1, é uma das classificações mais utilizadas na área da computação paralela. Basicamente, o autor se baseia em dois conceitos para criar sua organização: fluxos de instruções e fluxos de dados. Um fluxo de instruções nada mais é do que a seqüencia de instruções seguida na execução do programa, com registradores individuais que gerenciam a execução de cada fluxo. Já o fluxo de dados representa a instância dos dados que serão manipulados pelas instruções dos programas. Combinando essas informações têm-se as seguintes categorias para as arquiteturas: SISD, SIMD, MISD e MIMD.

- Single Instruction, Single Data (SISD): modelo de arquitetura que utiliza um único fluxo de instruções e um único fluxo de dados. Os computadores convencionais seqüenciais, como o modelo proposto por von Neumann, seguem essa abordagem no processamento das instruções que compõem um programa. A cada instante, somente uma instrução está sendo executada, manipulando uma única instância dos dados referentes aos operandos. Nesse modelo, Figura 2.1, há somente uma unidade de controle e uma unidade de processamento, que acessa a memória para buscar as informações que serão processadas;

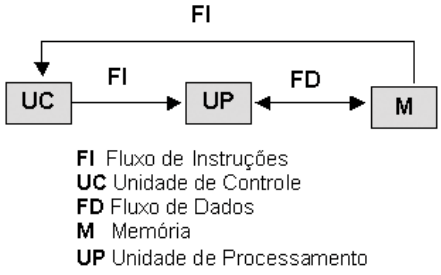

Figura 2.1: SISD – Fluxo de instruções e dados único.

- Single Instruction, Multiple Data (SIMD): embora exista, nessa categoria, somente uma unidade de controle para gerenciar a execução das instruções, há várias unidades de processamento que manipulam conjuntos distintos de dados. Portanto, cada unidade de processamento pode fazer uso de posições distintas de memória ao manipular os operandos envolvidos na instrução. Os computadores matriciais utilizam esse modelo, e estão organizados conforme a ilustração da Figura 2.2;

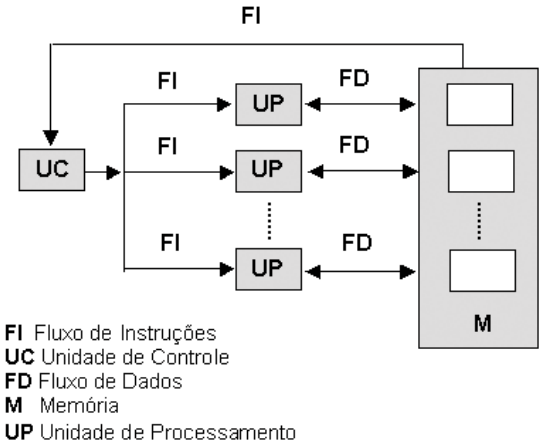

Figura 2.2: SIMD – Único fluxo de instruções e múltiplos fluxos de dados.

- Multiple Instruction, Single Data (MISD): nesse modelo, diferentes instruções de um programa atuam, simultaneamente, sobre uma mesma posição de memória. Portanto, existe uma concorrência na disputa pelo acesso aos dados compartilhados que prejudica, sensivelmente, a execução do programa. Por esse motivo, o modelo ilustrado na Figura 2.3, geralmente, não é implementado (STALLINGS, 2002);

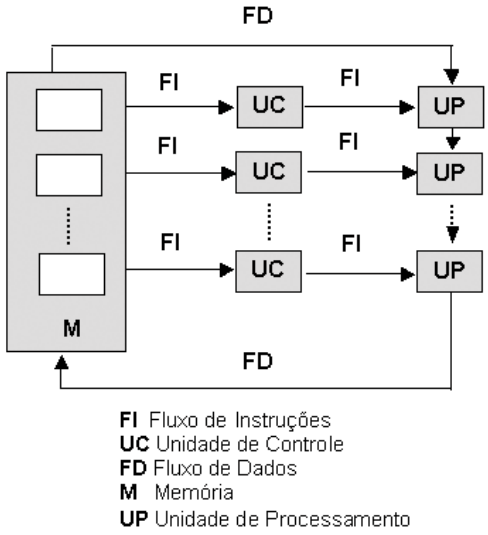

Figura 2.3: MISD – Múltiplo fluxo de instruções e único fluxo de dados.

- Multiple Instruction, Multiple Data (MIMD): é um modelo de computação composto por várias unidades de controle e processamento, acessando um conjunto de

dados distinto na memória. Nessa categoria é possível incluir grande parte dos computadores ou sistemas de processamento paralelo, seja centralizado ou distribuído. Uma representação desse modelo é apresentada na Figura 2.4.

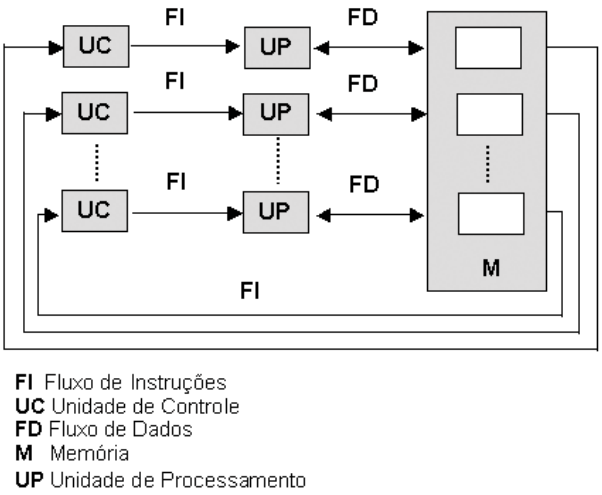

Figura 2.4: MIMD – Múltiplo fluxo de instruções e dados.

Embora muito utilizada, a classificação proposta por Flynn é considerada incompleta por não representar algumas arquiteturas recentes nem definir uma hierarquia entre arquiteturas similares. Por esse motivo surgiram outras classificações como a de Tanenbaum (TANENBAUM; GOODMAN, 1998) e a de Duncan (DUNCAN, 1990).

2.3.2 Taxonomia de Tanenbaum

A classificação para arquitetura de computadores proposta por Tanenbaum mantém os elementos da classificação anterior, mas adiciona modelos de arquiteturas paralelas mais recentes. As principais mudanças dizem respeito à inserção de algumas ramificações dos modelos arquiteturais SISD e MIMD. Neste último, há uma apresentação diferenciada das máquinas multiprocessadas e dos multicomputadores, bem como as alternativas para cada um desses modelos. Essas modificações na classificação proposta originalmente por Flynn podem ser visualizadas na Figura 2.5.

Na taxonomia proposta por Tanenbaum são apresentados três tipos de arquiteturas multiprocessadas:

- UMA (*Uniform Memory Access*): nas máquinas do tipo UMA o tempo de acesso a cada um dos módulos de memória e a cada um dos endereços é idêntico para qualquer processador. Essa uniformidade torna o desempenho do sistema mais fácil de ser estimado;

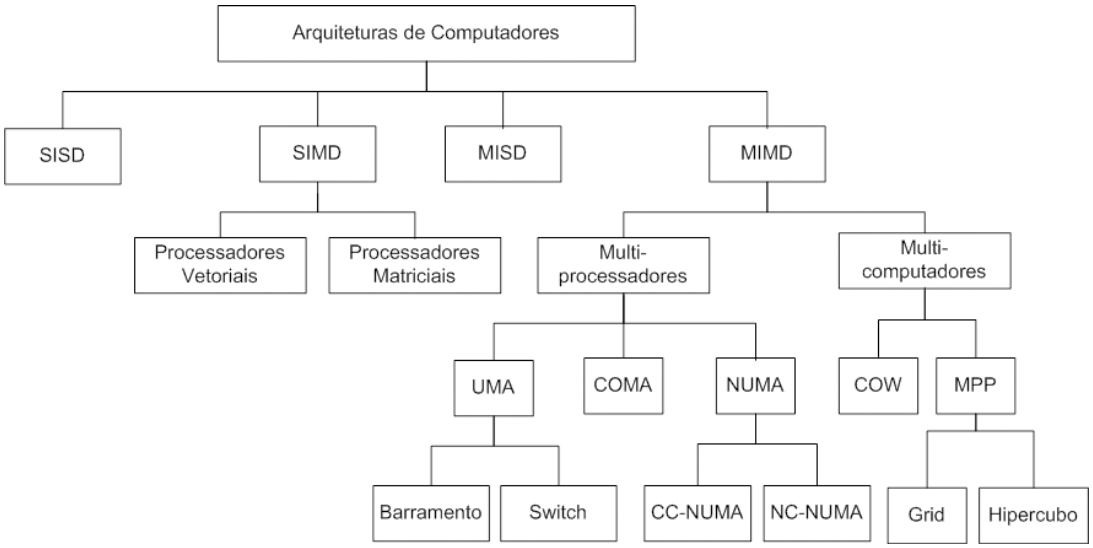

Figura 2.5: Taxonomia de Tanenbaum.

- NUMA (*NonUniform Memory Access*): nesse tipo de arquitetura, cada processador tem um módulo de memória local, e o tempo de acesso a esse módulo é menor, se comparado com o tempo de acesso a um módulo associado a outro processador. Portanto, mesmo existindo um espaço de endereçamento comum a todos os processadores, a latência de acesso aos módulos locais é menor. Esse modelo é dividido em: CC-NUMA (*Cache Coherent NUMA*) e NC-NUMA (*Non-Cache Coherent NUMA*). Nas máquinas do tipo CC-NUMA a coerência do *cache* é mantida por mecanismos implementados no próprio *hardware*. Já nas arquiteturas NC-NUMA não há garantia de coerência dos dados;

- COMA (*Cache Only Memory*): nessa arquitetura, a memória principal, associada a cada um dos processadores, é utilizada como um *cache* e a coerência dos dados é garantida por *hardware*, através de atualizações simultâneas nas várias cópias dos dados.

Enquanto as máquinas multiprocessadas utilizam uma memória compartilhada para implementar a comunicação entre os processos, as arquiteturas baseadas em multicomputadores fazem uso de passagem de mensagem por redes de interconexão. Esses sistemas são divididos em:

- MPP (*Massively Parallel Processors*): nessa arquitetura todas as unidades de processamento estão fortemente acopladas em *hardware* e conectadas através de uma rede de alta velocidade. Hoje em dia, supercomputadores como o *Earth Simulator*[2] e *Blue Gene*[3] são exemplos de sistemas MP (*Massively Parallel*);

[2]http://www.jamstec.go.jp/esc/index.en.html
[3]http://domino.research.ibm.com/comm/research_projects.nsf/pages/bluegene.index.html

- COW (*Cluster of Workstations*): diferentemente do modelo anterior, essas plataformas são construídas com computadores convencionais, interconectados por redes tradicionais.

2.4 Modos de Acesso à Memória

Todo programa paralelo implementa troca de informações entre os processos que colaboram na solução do problema. Basicamente, essa comunicação pode ocorrer através de variáveis de memória compartilhada ou via passagem de mensagem por redes de interconexão. Em máquinas multiprocessadas, a troca de dados é feita através de variáveis de memória compartilhada, pois existe um espaço de endereçamento único no ambiente. Já em multicomputadores, como os *clusters*, por exemplo, toda comunicação acontece através da troca de mensagens via rede. Isso porque os processos só têm acesso ao espaço de endereçamento local da máquina onde estão sendo executados.

2.4.1 Memória compartilhada

No modelo de memória compartilhada existe um único espaço de endereçamento para a memória, acessível igualmente por todos os processadores da máquina. Os processos em execução utilizam esse espaço comum para compartilhar informações entre si. Através desse compartilhamento de endereços de memória é que as tarefas paralelas colaboram na solução dos problemas. No entanto, é necessário gerenciar o acesso concorrente às posições de memória compartilhada para que a consistência das informações seja mantida. Nos casos em que a concorrência por variáveis de memória é baixa, esse modelo de acesso à memória é capaz de proporcionar um bom desempenho às aplicações. Isso porque a latência apresentada por uma estrutura de barramento ou *switch* é muito menor que o tempo gasto em comunicações via rede. A Figura 2.6 ilustra a organização desse modelo. Esse modo de acesso à memória é utilizado em todas as arquiteturas classificadas como multiprocessadores.

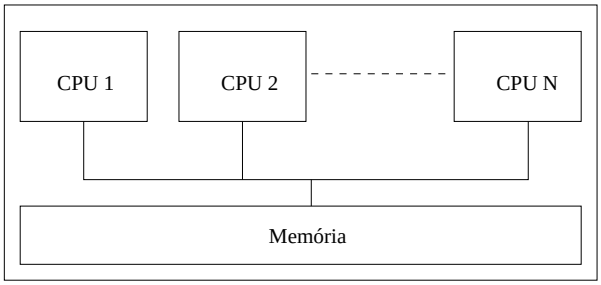

Figura 2.6: Modelo de memória compartilhada.

2.4.2 Memória distribuída

Nesse modelo de acesso à memória, cada máquina do sistema tem sua própria memória e seu próprio espaço de endereçamento. Portanto, a memória total do sistema se encontra lógica e fisicamente distribuída entre os nós que compõem o ambiente computacional. Toda comunicação entre os processos é feita através de mensagens trocadas pela rede de interconexão. Portanto, é necessário fazer uso de primitivas de comunicação, ponto-a-ponto ou coletivas, para permitir a comunicação distribuída.

Nos sistemas com memória distribuída, o tempo gasto com as comunicações entre processos é muito maior que no modelo anterior. O custo adicional da comunicação é proporcional à quantidade de dados transmitidos e à latência da rede utilizada. Dependendo das características da rede utilizada, a latência de comunicação pode ser maior ou menor. Geralmente, o tempo gasto com as comunicações em um *grid* computacional é maior que o tempo gasto em *clusters*, devido à maior amplitude de distribuição dos recursos de processamento e o fato de, em geral, se utilizar a internet. Uma ilustração desse modelo pode ser visualizada na Figura 2.7.

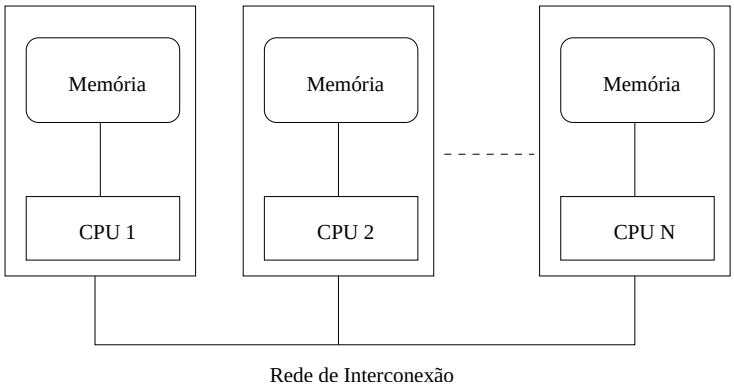

Figura 2.7: Modelo de memória distribuída.

2.4.3 Modelo Híbrido

Um modelo híbrido e conhecido de acesso à memória é o DSM (*Distributed Shared Memory*) (TAM; SMITH; FARBER, 1990). Embora nessa abordagem as memórias estejam fisicamente distribuídas, a gerência desses módulos de armazenamento é realizada por um *middleware*, que permite simular a existência de uma única memória global compartilhada. Esse mecanismo foi criado para facilitar a programação e evitar o uso de passagem de mensagens.

2.5 Modelos de Programação Paralela

Basicamente, o processamento paralelo consiste em utilizar múltiplos processadores para executar diferentes programas ou trechos de um mesmo programa simultaneamente. Nesse sentido, o principal objetivo da programação paralela é dividir o trabalho total em pequenas tarefas e melhorar o desempenho da aplicação permitindo a execução simultânea dos processos criados. Para alcançar esse objetivo, diferentes arquiteturas paralelas podem ser utilizadas, conforme apresentado na taxonomia de Tanenbaum (Figura 2.5).

Como já discutido no início deste capítulo, a programação paralela surgiu como uma alternativa viável para melhorar o desempenho das aplicações frente as restrições físicas para aumento da capacidade de processamento dos computadores atuais. Estas restrições estão relacionadas não só aos limites relacionados à miniaturização dos componentes, mas à densidade possível para os transistores e ao controle de temperatura dos componentes.

A escolha do modelo de programação paralela depende, entre outras coisas, do tipo de arquitetura alvo que será utilizada no processamento da aplicação. Portanto, os modelos de programação constituem uma abstração sobre o tipo de arquitetura do *hardware* e do modo de acesso à memória da máquina (BARNEY, 2007). Nesse contexto, os modelos de programação paralela mais utilizados são (SILVA; BUYYA, 1999):

- Memória Compartilhada;
- *Multithreading*;
- Trocas de Mensagem.

2.5.1 Memória Compartilhada

Esse modelo de programação utiliza o espaço de endereçamento único e compartilhado para implementar a troca de dados entre os processos. Nesse modelo, as tarefas podem ler e escrever em qualquer endereço de memória. Por isso, é necessário tratar o controle de acesso aos recursos compartilhados pelas tarefas, a fim de manter a consistência e a coerência das informações armazenadas. Logo, mecanismos de sincronização (BENARI, 1990; ADVE; GHARACHORLOO, 1996), como, por exemplo, semáforos, devem ser utilizados para controlar o acesso às posições de memória compartilhada. A programação em ambientes de memória compartilhada pode ser feita através de *threads* ou diretivas OpenMP, por exemplo. Além disso, o gerenciamento da localidade e acesso aos dados é complicado e o excesso de concorrência pode prejudicar o desempenho das aplicações.

2.5.2 Multithreading

Nesse modelo, o programa pode ter vários fluxos de execução simultâneo, cada um gerenciado por uma *thread* exclusiva. Portanto, é possível relacionar e associar cada *thread* a

uma sub-rotina diferente do programa. O modelo de programação baseado em *threads* é, em geral, utilizado em máquinas multiprocessadas ou multinúcleos de memória compartilhada. Também é necessário utilizar mecanismos de sincronização para evitar que duas ou mais *threads* escrevam, simultaneamente, sobre uma mesma posição de memória. Na tentativa de estabelecer um padrão para esse modelo de programação, surgiram muitas implementações de *threads*, dentre as quais estão o *POSIX Threads* e o *OpenMP*.

O *POSIX Threads* (MUELLER, 1993) se baseia no uso de bibliotecas e exige do programador uma codificação capaz de explorar, explicitamente, o paralelismo da aplicação. Na linguagem C este padrão é comumente denominado *Pthreads*.

Por outro lado, a programação em *OpenMP* (QUINN, 2003) é baseada em diretivas para o compilador. Essas diretivas devem ser inseridas no código para indicar as estruturas que serão paralelizáveis. Essa versão de implementação está disponível para as linguagens C, C^{++}, entre outras.

2.5.3 Trocas de Mensagem

Nesse modelo de programação, é necessário utilizar primitivas de comunicação para transmitir dados entre processos remotos. Além das primitivas de comunicação, essas bibliotecas possuem rotinas para sincronização de tarefas. Hoje em dia, o PVM *(Parallel Virtual Machine)* (BEGUELIN et al., 1991; GEIST et al., 1994) e o MPI (*Message Passing Interface*) (SNIR; OTTO, 1998; GROPP et al., 1998) são as duas principais "bibliotecas"de passagem de mensagem. A rigor, o MPI é uma especificação padronizada para transmissão de dados em computação paralela e distribuída. Essa padronização evoluiu da biblioteca PVM.

No PVM, o problema a ser resolvido é dividido e executado em programas separados. Os programas podem ser escritos na linguagem C ou Fortran, mas devem ser compilados para cada tipo específico de máquina. No entanto, se o sistema distribuído for homogêneo, é necessário compilar apenas uma vez. Uma das considerações é que o código executável do programa deve estar em todas as máquinas, portanto, é interessante manter um sistema de arquivo único no sistema.

Antes de executar o programa é preciso criar um arquivo de máquinas para definir quais nós computacionais serão utilizados para constituir a máquina paralela virtual. É possível iniciar a execução do programa em uma única máquina e, depois, adicionar com comandos do PVM novos nós de processamento ao sistema. O PVM permite criar qualquer quantidade de processos, independente do número de processadores ou nós de processamento utilizados.

Geralmente, as aplicações PVM utilizam o modelo mestre-escravo para organizar as soluções dos problemas. O processo mestre é o responsável por distribuir o trabalho e organizar a resposta final da solução. Para a criação dos processos, o PVM oferece um conjunto de rotinas e argumentos que podem ser passados na inicialização. Os programas se co-

municam através de passagem de mensagem pela rede de interconexão, usando primitivas do tipo *pvm_send()* e *pvm_recv()*. É interessante comentar que todas as rotinas de envio de dados do tipo *send* são primitivas não-bloqueantes, enquanto as recepções (*receive*) podem ser bloqueantes ou não (WILKINSON; ALLEN, 1999). O PVM também disponibiliza primitivas de comunicação coletiva, como *pvm_bcast(), pvm_scatter(), pvm_gather()* e *pvm_reduce()*; que podem ser utilizadas para comunicação em grupo.

O MPI, assim como o PVM, oferece uma biblioteca de rotinas para comunicação com passagem de mensagem em sistemas distribuídos. Uma das vantagens do MPI sobre o PVM é que ele tem se tornado um padrão para esse tipo de programação e, com isso, um número muito grande de rotinas foi definido e implementado ao longo do tempo. Algumas modificações foram realizadas para corrigir deficiências técnicas apresentadas pelo PVM. Assim como o PVM, as primitivas do MPI estão disponíveis para a linguagem C, C^{++} e Fortran. No entanto, há versões MPI também para a linguagem Java (GETOV; GRAY; SUNDERAM, 1999).

Desde a definição do padrão, várias implementações surgiram e se tornaram publicamente disponíveis, sendo o LAM-MPI (BURNS; DAOUD; VAIGL, 1994; SQUYRES; LUMSDAINE, 2003), o MPICH (WILLIAM; LUSK, 2007, 1996) e o Open MPI (GRAHAM et al., 2006; GRAHAM; WOODALL; SQUYRES, 2005) as versões mais conhecidas.

Assim como acontece no PVM, no MPI o trabalho é subdividido em tarefas menores que são associadas a processos distintos. Muitas primitivas encontradas no PVM são implementadas no MPI. A definição do arquivo de máquinas também é necessária no MPI. Esta biblioteca também implementa o conceito de grupo. Geralmente, esse paradigma de programação é utilizado em arquiteturas do tipo COW (*Cluster of Workstations*), NOW (*Network of Workstations*), MPP e Grid.

Dentre as razões para usar o padrão MPI, nas implementações de programas paralelos, estão (MPI-FORUM, 1994):

- padronização: o MPI é o único padrão para biblioteca de passagem de mensagem;
- portabilidade: um programa MPI é facilmente portável para outras plataformas;
- desempenho: fabricantes de *hardware* podem explorar as características do próprio equipamento para melhorar o desempenho de programas MPI;
- funcionalidade: existem mais de 120 rotinas definidas no padrão;
- disponibilidade: muitas implementações do padrão podem ser encontradas, sejam elas de domínio público ou particular.

2.6 Clusters e Grids Computacionais

De acordo com (BUYYA, 1999a) um *cluster* pode ser definido como um conjunto de nós computacionais, interconectados por uma rede local (LAN), que cooperam na solução de problemas simulando uma única máquina paralela. Em geral, os *clusters* se apresentam como um sistema único, fortemente acoplado e homogêneo. Contudo, nada impede que esse sistema seja constituído de diferentes máquinas e sistemas operacionais; formando um sistema distribuído heterogêneo. Os nós computacionais podem ser formados por máquinas mono ou multiprocessadas e a rede que conecta os nós de um *cluster* é dedicada à integração dos elementos de processamento e separada do ambiente externo ao qual o sistema pode estar conectado. A flexibilidade de configurações desses sistemas favorece modificações em relação à quantidade de nós de processamento, à capacidade de memória dos nós, à topologia da rede e a outros parâmetros relacionados à infra-estrutura do ambiente.

Em computação paralela é comum existir uma troca intensa de dados entre as tarefas paralelas que estão sendo executadas por diferentes nós do sistema. Por esse motivo, a largura de banda e a latência da rede de interconexão são fundamentais para o desempenho das aplicações.

Várias aplicações utilizam os *clusters* como infra-estrutura para o processamento das tarefas, dentre as quais: simulações financeiras; simulações de fenômenos naturais; processamento de imagens; e previsão meteorológica.

Diferentemente dos *grids* computacionais (NÉMETH; SUNDERAM, 2003, 2002; FOSTER et al., 2002; FOSTER; KESSELMAN; TUECKE, 2001), os nós de processamento dos *clusters* pertencem a um único domínio administrativo. Assim, toda a gerência de usuários e do escalonamento das aplicações é feita de modo centralizado. Com isso, o sistema é visto como um único dispositivo de processamento.

Devido à estrutura descentralizada dos *grids*, os recursos computacionais (*hardware* e *software*) são muito mais heterogêneos e dinâmicos do que os de um *cluster*. Essa heterogeneidade aumenta a complexidade do sistema e o torna mais difícil de ser gerenciado. Entretanto, essa característica permite maior flexibilidade ao sistema computacional.

Hoje em dia, os *clusters* e os *grids* computacionais têm sido muito utilizados para a execução de aplicações que demandam alta capacidade de processamento. A utilização desses sistemas é motivada pela semelhança com os MPPs, que fazem uso de trocas de mensagens para implementar a comunicação entre os processos. O MPICH-G2[4] (KARONIS; TOONEN; FOSTER, 2003), é uma distribuição MPI que utiliza serviços do *Globus Toolkit* (SOTOMAYOR; CHILDERS, 2005) para trocas de mensagem entre processos distribuídos. As principais características dos *grids* computacionais são:

- heterogeneidade: tanto em elementos de *hardware* quanto de *software*;

[4]http://www3.niu.edu/mpi/

- dinamicidade: a quantidade e localização dos recursos podem ser alteradas constantemente;
- alta dispersão geográfica: o sistema pode atingir uma dimensão mundial;
- múltiplos domínios administrativos: o sistema pode ter elementos de processamento de diferentes domínios administrativos;
- controle distribuído do ambiente e suas funções;
- grande compartilhamento de recursos;
- sistemas fracamente acoplados;
- o gerenciamento dos processos e do escalonamento é feito de modo distribuído.

Hoje em dia, diversas aplicações estão sendo implementadas e executadas em *grid*, tais como: seqüenciamento de DNA; simulação de semicondutores; simulação de reações nucleares; processamento de imagem; reconhecimento de padrões; entre outras.

2.7 Considerações Finais

Este capítulo apresentou algumas taxonomias para arquiteturas de computadores, dando ênfase às arquiteturas paralelas voltadas para o processamento distribuído. Assim, alguns conceitos relacionados à computação paralela e distribuída foram explicados, bem como as possíveis alternativas de acesso às memórias. Além disso, comentou-se sobre os modelos de programação paralela, principalmente a estratégia de trocas de mensagem. Algumas diferenças foram apontadas entre os modelos mais conhecidos, que são o PVM e o MPI.

Foram apresentadas definições e características de duas plataformas distribuídas amplamente utilizadas em programação paralela: os *clusters* e os *grids* computacionais. Destacaram-se algumas diferenças e similaridades, bem como os tipos de aplicações que utilizam essas estruturas como plataforma de processamento.

Capítulo 3

Avaliação, Análise e Predição de Desempenho

Neste capítulo discutimos a importância de uma metodologia para avaliação, análise e predição de desempenho no desenvolvimento de aplicações paralelas eficientes. Uma descrição sobre as atividades envolvidas no processo de modelagem e predição é realizada. As principais técnicas de modelagem são apresentadas e detalhadas. Além disso, diversas áreas de aplicação para a metodologia são citadas e alguns trabalhos relacionados a esta área de pesquisa são comentados.

3.1 Introdução

As limitações físicas encontradas no desenvolvimento das arquiteturas tradicionais (FRANK, 2005), geralmente associadas ao consumo e dissipação de energia por operação, têm incentivado os pesquisadores a buscar alternativas para melhorar a capacidade de processamento e eficiência dos sistemas computacionais. Assim, seria possível atender a crescente demanda por recursos computacionais exigida pelas complexas aplicações reais.

Nesse contexto, agregar recursos computacionais distribuídos tem sido uma opção viável e cada vez mais utilizada em computação de alto desempenho. Arquiteturas distribuídas, como *clusters* e *grids* computacionais, são capazes de oferecer até centenas de *teraflops* de capacidade de processamento e dezenas de *terabytes* de armazenamento para as aplicações de larga escala. Esse é o caso, por exemplo, do supercomputador da IBM denominado *BlueGene/L – eServer Blue Gene Solution*[1], que atualmente consegue ultrapassar os 500 *teraflops* de processamento.

O aumento crescente e contínuo da complexidade desses sistemas distribuídos tem feito das atividades de avaliação, análise e predição de desempenho instrumentos essenciais para melhorar ainda mais o desempenho e a eficiência das soluções elaboradas. Essa tendência

[1] http://domino.research.ibm.com

pode ser verificada através da quantidade de trabalhos que vêm sendo publicados, da existência de grande quantidade de revistas e jornais específicos da área de computação de alto desempenho, dos livros lançados recentemente e das conferências anuais que reúnem pesquisadores de todo o mundo para debater sobre temas relacionados a essa área.

O desempenho computacional de uma aplicação distribuída pode estar associado a vários parâmetros, dentre os quais é possível citar, por exemplo, o tempo de execução de uma tarefa e o tempo de transmissão de uma mensagem pela estrutura de interconexão. Dependendo dos objetivos criados para a avaliação de desempenho, algumas métricas podem ser mais adequadas do que outras para o desenvolvimento das análises. Portanto, o conceito de desempenho está diretamente relacionado ao parâmetro do sistema que será avaliado.

Hoje em dia, as técnicas de avaliação, análise e predição de desempenho ainda são pouco utilizadas por empresas que trabalham com desenvolvimento de sistemas não críticos. Geralmente, requisitos não funcionais são ignorados durante o desenvolvimento de *software*. Talvez essa prática não seja comum por uma deficiência de formação das pessoas que trabalham na área. A falta de conhecimento especializado no assunto, seja dos desenvolvedores de sistemas ou dos gerentes de projetos, pode ser explicada pela carência de disciplinas técnicas que tratam o tema em cursos de graduação. Normalmente, pouco é ensinado e discutido sobre o assunto, favorecendo a ausência de práticas associadas ao tema no ambiente comercial. Portanto, é importante discutir não só as possíveis aplicações para as técnicas de avaliação e análise de desempenho, mas também os benefícios e os valores que tais práticas podem agregar aos produtos ou sistemas comerciais.

Em (JAIN, 1991) todo o processo relacionado à atividade de avaliação de desempenho é considerado uma arte. Isso porque o processo em si não pode ser mecanizado e envolve, sobretudo, a sensibilidade do analista que o realiza. Para o desenvolvimento de uma boa análise e avaliação de desempenho é necessário um conhecimento total do sistema que será modelado. Além disso, a escolha de metodologias e ferramentas adequadas é fundamental para o sucesso das atividades.

3.2 Considerações sobre Metodologias de Avaliação e Análise de Desempenho

Atualmente, o desempenho é um requisito não funcional que deveria ser considerado e avaliado desde o início do ciclo de vida do *software* com o objetivo de melhorar a qualidade final do mesmo. No entanto, é importante observar que, dependendo da fase que se encontra o projeto, o analista ou desenvolvedor tem à disposição diferentes técnicas de avaliação de desempenho que podem ser aplicadas a fim de estimar se um requisito de desempenho do sistema será ou não alcançado no final da implementação. Nesse

contexto, uma metodologia para avaliação de desempenho é importante e capaz de auxiliar o desenvolvimento de programas paralelos eficientes. Cabe a metodologia recomendar um conjunto de práticas para orientar aqueles que realizarão atividades dessa natureza.

Assim, uma metodologia de avaliação, análise e predição de desempenho deve ter como objetivo a especificação de um conjunto de técnicas e métodos que permita modelar o sistema estudado e avaliar com clareza e precisão os aspectos relacionados ao desempenho das aplicações. Para isso, é necessário definir corretamente os objetivos da análise, as métricas que serão utilizadas na avaliação, a(s) técnica(s) escolhida(s) para realizar as atividades planejadas, os parâmetros relevantes para a análise, o planejamento experimental, o modo de desenvolvimento dos modelos e a apresentação e análise dos resultados (JAIN, 1991). Portanto, é importante descrever individualmente e de forma detalhada cada uma dessas fases a fim de facilitar a compreensão dos objetivos de cada atividade envolvida nas práticas propostas por uma metodologia de avaliação de desempenho.

3.2.1 Enumerar Objetivos e Compreender o Sistema

Nessa fase é essencial que sejam enumerados com clareza não só os objetivos das atividades de análise e avaliação de desempenho, mas também o objetivo do sistema que será modelado. Para que se possa desenvolver uma atividade com qualidade é fundamental que o sistema seja compreendido por completo. As sugestões de alterações para melhorar o desempenho da aplicação só poderão ocorrer, plenamente, se o analista compreender em detalhes a aplicação que está sendo analisada. A visualização de possíveis gargalos no sistema exige, necessariamente, completo domínio sobre o código fonte do programa e sobre o modelo organizacional da aplicação. Além disso, as configurações do ambiente e do sistema que será analisado também são importantes para realizar um dimensionamento correto das variáveis de desempenho.

No estudo do sistema, o analista deve definir claramente o escopo do que será avaliado, de modo a identificar exatamente quais serão os limites da análise a ser desenvolvida. A qualidade da documentação do sistema e as informações relatadas são fundamentais para essa atividade inicial, que pode, inclusive, ocorrer ao longo de todo o ciclo de vida do *software*.

3.2.2 Escolher Métricas de Desempenho

As métricas representam as grandezas que podem ser utilizadas para mensurar e avaliar o desempenho do sistema. Dependendo das características do sistema e dos objetivos do analista, pode-se definir uma métrica específica para avaliar o que precisa ser caracterizado como desempenho. A escolha adequada das métricas é importante para dar confiabilidade aos valores especificados no projeto do sistema ou da aplicação. Em geral, os aspectos de desempenho mais comumente avaliados são:

- tempo de execução de uma tarefa;
- tempo de resposta de um servidor;
- quantidade de requisições tratadas pela aplicação por unidade de tempo;
- tempo de transmissão de uma mensagem pela estrutura de interconexão;
- disponibilidade do sistema.

3.2.3 Definir a Carga de Trabalho Computacional

O desempenho de um sistema depende fortemente da demanda por *hardware* e por recursos de *software* que as aplicações solicitam. A carga de trabalho computacional determina o volume de processamento a que o sistema está submetido (BRANCO, 2004) e pode ser medida através do(a):

- número de processos aptos na fila da CPU;
- quantidade de memória livre;
- porcentagem de CPU livre;
- *throughput* – número de tarefas processadas por unidade de tempo;
- número de pacotes que chegam pela interface de rede;
- número de pacotes que saem pela interface de rede;
- número de leituras por segundo realizadas no disco rígido;
- número de escritas por segundo realizadas no disco rígido.

No sistema Linux, por exemplo, esses índices de carga são obtidos facilmente através de comandos do próprio sistema. A quantidade de memória livre na máquina é determinada através da leitura do arquivo */proc/meminfo*; a porcentagem de CPU livre ou ocupada é conhecida com o comando *top*; e a quantidade de processos aptos na fila da CPU, através da função *getindexload*( ).

A carga de trabalho computacional, em cada máquina do sistema, é classificada como real ou sintética (artificial) (LO; MACHE; WINDISCH, 1998; PFNEISZL; KOTSIS, 1996). Uma carga real é conseqüência direta da execução de tarefas por parte dos usuários logados no ambiente. Esse tipo de carga computacional não é adequado para ser controlado e reproduzido durante testes experimentais. Já uma carga sintética ou artificial é produzida e controlada através da execução de um programa que simula a existência de usuários no ambiente, compartilhando e disputando os recursos de processamento com suas aplicações.

Uma das vantagens das cargas sintéticas é a possibilidade de parametrização e controle, dando maior flexibilidade para a representação ideal da carga desejada. Isso é fundamental para criar ambientes idênticos e calibrar parâmetros na repetição de experimentos.

3.2.4 Elaborar os Modelos de Desempenho

Após a conclusão das etapas anteriores, é possível determinar o modelo de desempenho conceitual da aplicação no sistema alvo. Esse modelo é uma representação teórica do comportamento esperado ou previsto para a aplicação no ambiente avaliado. Em geral, as técnicas de modelagem analítica exigem um estudo anterior sobre a complexidade algorítmica do código fonte da aplicação para facilitar a construção de uma equação matemática que seja adequada e capaz de representar o desempenho do programa no ambiente de execução. Assim, os modelos reais são elaborados para representar esse comportamento teórico predeterminado. Para isso, algumas técnicas de modelagem podem ser empregadas nessa atividade, conforme descrito na próxima seção.

3.2.5 Escolher as Técnicas de Modelagem

As técnicas de modelagem permitem representar sistemas (*software* e/ou *hardware*) na forma de modelos. Esses modelos são abstrações de uma aplicação ou sistema computacional e expressam as principais características avaliadas e observadas durante a análise.

A representatividade e complexidade do modelo está diretamente relacionada à quantidade de informações considerada na modelagem e à precisão obtida nos resultados. No entanto, é importante levar em consideração, durante a elaboração dos modelos, não só o grau de representação do mesmo, mas sua aplicabilidade. Em geral, modelos que prezam por uma precisão exagerada são demasiadamente complexos e de difícil aplicação, inviabilizando seu uso.

Como já comentado anteriormente, a escolha da técnica de modelagem depende não só do tipo de sistema que será analisado, mas também da fase do ciclo de desenvolvimento em que o mesmo se encontra. Por isso, algumas técnicas só podem ser aplicadas no final da implementação do sistema; enquanto outras podem ser aplicadas já no início do desenvolvimento. Nesse contexto, há na literatura diversas técnicas disponíveis, entre elas: Redes de Petri, Redes de Filas, Cadeias de Markov e *Statecharts* (ORLANDI, 1995).

3.2.6 Validar os Modelos

A precisão dos modelos deve ser validada através de testes experimentais, de modo que seja possível comparar resultados reais de execução do sistema com resultados teóricos apresentados pelos modelos. É importante que durante os testes experimentais a carga

computacional seja bem dimensionada para que os testes possam atingir os objetivos planejados inicialmente.

3.2.7 Analisar e Apresentar os Resultados

Após os experimentos, se o resultado apresentado pelos modelos não for o esperado, é preciso refazer a modelagem para melhorar a representatividade dos modelos e, com isso, aumentar a precisão. Esse processo pode ser repetido até que os resultados sejam satisfatórios.

A análise dos resultados gerados envolve estudos estatísticos que aferem a confiabilidade dos dados obtidos. Durante a apresentação dos resultados, geralmente na forma de gráficos, é preciso que esteja clara qual a informação mais importante a ser destacada, de modo que a apresentação possa expressar com clareza os resultados encontrados pelo analista.

3.2.8 Erros Comuns

Na atividade de análise e avaliação de desempenho é comum cometer erros que acabam prejudicando a qualidade dos resultados gerados. A falta de experiência do analista é a causa mais provável da ocorrência de certos equívocos durante o processo. Assim, os erros mais comuns são (JAIN, 1991):

- falta de objetivos bem definidos para a atividade de análise e avaliação de desempenho;
- ausência de uma abordagem sistemática para o processo;
- falta de compreensão do problema ou sistema analisado;
- escolha incorreta das métricas de desempenho;
- escolha incorreta da técnica de avaliação de desempenho;
- simplicidade excessiva dos modelos ignorando parâmetros ou fatores importantes;
- dimensionamento equivocado da carga computacional durante os testes experimentais;
- omissão de limitações e suposições;
- tratamento inapropriado às anomalias percebidas durante os testes experimentais;
- análise equivocada dos resultados obtidos;
- apresentação imprópria dos resultados;
- repetição de testes em situações diferentes do ambiente.

3.3 Técnicas para Avaliação de Desempenho

Em geral, as técnicas para avaliação de desempenho estão relacionadas aos meios utilizados pelo analista para obter informações a respeito das métricas de desempenho escolhidas para a análise. Normalmente, uma técnica é suficiente para realizar o estudo. Contudo, é possível refinar as medidas através de diferentes abordagens (JAIN, 1991), almejando informações mais precisas.

As técnicas para avaliação e análise de desempenho podem ser classificadas em três grupos (BREWER et al., 1991; JAIN, 1991), que são:

- modelagem analítica;
- medições reais de um sistema ou aplicação;
- simulação de sistemas.

Existe pesquisador que considera somente duas categorias na classificação; colocando a modelagem analítica e as simulações em um único grupo (FERRARI, 1983). Uma explicação dada pelo autor é que a característica fundamental dessas técnicas é que elas podem ser aplicadas sem que o sistema esteja disponível e implementado.

Para cada uma das técnicas aplicáveis no processo de avaliação de desempenho existem diversos mecanismos que podem ser utilizados no desenvolvimento das atividades. Apenas como exemplo, é possível citar Redes de Filas e Redes de Petri na modelagem analítica; Evento Discreto, Execução Dirigida por *Traces* e Monte Carlo em simulação. Já em relação às medições, elas podem ser feitas por *software*, por *hardware* ou por artefatos híbridos. É justamente pela grande quantidade de técnicas disponíveis que a escolha adequada se torna uma tarefa difícil e, ao mesmo tempo, importante para o resultado final do estudo. As características principais das técnicas citadas são descritas nas próximas seções.

3.3.1 Medições

As medições fazem uso de monitores (NUTT, 1976) para acompanhar o comportamento do sistema e é uma das técnicas de avaliação e análise de desempenho que apresenta maior precisão. Isso porque o sistema em execução é monitorado, e parâmetros relevantes ao seu desempenho, como o tempo de execução de uma determinada tarefa, por exemplo, é medido e caracterizado. Um dos objetivos dessa técnica é auxiliar o analista a descobrir problemas no sistema computacional que possam ser eliminados para que o desempenho seja melhorado. Além disso, as medições são freqüentemente utilizadas para gerar dados de desempenho que possam ser utilizados na elaboração de modelos analíticos e então caracterizar o comportamento da aplicação. De forma equivalente, os dados obtidos com

as medições também podem ser utilizados para validar resultados gerados pelos modelos analíticos elaborados.

Os monitores de *software* podem ser compostos por instruções de programa que, inseridos na aplicação, coletam informações específicas de alguns trechos do código ou de serviços oferecidos pelo sistema. Já os monitores de *hardware* são dispositivos que, acoplados ao sistema, conseguem detectar sinais que caracterizam a ocorrência de fenômenos físicos observáveis (FERRARI, 1983). Uma comparação detalhada entre esses dois tipos de monitores pode ser obtida em (PETERSON, 1974). Quando combinados, os monitores formam um modelo híbrido para monitoração (JAIN, 1991).

3.3.2 Redes de Petri

As Redes de Petri (PNs) (PETERSON, 1977) constituem outro formalismo para modelagem de sistemas, com a vantagem de utilizar uma representação gráfica que facilita o entendimento dos aspectos modelados. Com essa técnica é possível modelar características, como a sincronização de processos paralelos, que não podem ser tratadas usando as redes de filas. A modelagem que as PNs oferecem é adequada para a representação de aspectos relacionados à concorrência, sincronismos, comunicação e cooperação entre processos. Por esses motivos, as Redes de Petri constituem uma ferramenta de alto grau de representação para modelagem de sistemas computacionais.

Contudo, quando a complexidade do programa é alta, especialmente em aplicações paralelas, os modelos gerados pelo formalismo das PNs também aumentam de forma significativa; o que pode dificultar seu entendimento. Basicamente, as PNs utilizam estados e transições para representar situações do sistema e dos eventos, respectivamente. Logo, a complexidade do sistema é diretamente proporcional ao número de estados e transições do modelo.

No entanto, um fato positivo das PNs é que elas podem ser utilizadas nos estágios iniciais do desenvolvimento dos sistemas, corrigindo problemas, poupando modificações futuras e diminuindo custos de manutenção e adequação do projeto. Freqüentemente, as PNs têm sido utilizadas para descobrir a presença de propriedades desejáveis ou não, como por exemplo *deadlocks* ou falhas de segurança em sistemas tolerantes a falhas, como citado por (LEVESON; STOLZY, 1987) e (MURATA; SHATZ; SHENKER, 1989).

Outro aspecto interessante para ser destacado, em relação às Redes de Petri, é a possibilidade de utilização de seu formalismo para atividades de predição de desempenho, conforme destacam (FERSCHA, 1992; BUCHHOLZ, 1999).

3.3.3 Redes de Filas

A teoria de filas (COOPER, 1981) foi formulada para modelar e analisar sistemas de diferentes tipos. Já as redes de filas, definidas como um subconjunto da teoria de filas, são

aplicadas na modelagem de sistemas computacionais (LAZOWSKA et al., 1984). As técnicas que constituem essas teorias acabam formando uma ferramenta matemática importante para avaliações quantitativas de sistemas.

Pelo fato de as redes de filas constituírem uma técnica de modelagem analítica, também é comum o desenvolvimento de modelos complexos ao tentar representar o comportamento de sistemas. No entanto, esse problema é menor quando o domínio de aplicação é o computacional. Segundo descrito em (LAZOWSKA et al., 1984), algoritmos precisos para avaliação de desempenho podem ser elaborados ponderando custo e complexidade.

Nos casos onde ocorre a concorrência ou disputa por recursos computacionais entre os processos ativos no sistema, é comum a formação de filas para acesso ou uso desses elementos. Essas situações são facilmente modeladas pelas técnicas definidas nas teorias das redes de filas.

3.3.4 Simulação

Outra técnica utilizada em avaliação e predição de desempenho é a simulação. As técnicas de simulação permitem estimar o comportamento de sistemas que ainda não estão completamente implementados, bem como validar modelos analíticos (AGARWAL, 1992; MENASCÉ; BARROSO, 1992).

Existem vários tipos de simulações disponíveis, dentre eles: Monte Carlo, Simulação Dirigida por *Traces*, Simulação Dirigida por Execução, Simulação por Evento Discreto e Emulação.

O Monte Carlo (KLEIJNEN, 1974) é um método de simulação ideal para sistemas estáticos, cujas características não se alteram com o passar do tempo. Portanto, para sistemas computacionais, geralmente dinâmicos, este método não é adequado.

A simulação dirigida por *traces* (HSU; BANERJEE, 1992) utiliza dois componentes básicos: um gerador de eventos (*trace*) e um simulador. O simulador terá como entrada os *traces* gerados e os aplicará durante a simulação do que foi planejado. Esses eventos podem ser gerados com a execução de programas reais, habilitando, por exemplo, simulações e avaliações de desempenho de aplicações em arquiteturas hipotéticas.

A simulação por evento discreto (XU; CHUNG, 2004) está restrita aos sistemas ou recursos que apresentam esse tipo de comportamento. A transmissão de pacotes de dados em uma rede de interconexão é um bom exemplo para ilustrar onde essa técnica pode ser aplicada, uma vez que enviar ou receber mensagens são eventos discretos no tempo.

Contudo, o grande problema dessas simulações é a quantidade de recursos computacionais consumidos e o tempo que se gasta para executar os testes planejados. Por isso, alguns pesquisadores têm modificado os simuladores para trabalhar em plataformas multiprocessadas e, com isso, melhorar o desempenho das ferramentas (FUJIMOTO, 1993).

3.3.5 Aplicações para as Técnicas

Geralmente, o objetivo das técnicas para análise e avaliação de desempenho é estudar o comportamento das aplicações e otimizar o desempenho dos sistemas. No entanto, existem outros objetivos associados ao uso dessas estratégias, entre os quais é possível citar (KANT, 1992):

- **projeto do sistema**: no projeto do sistema se inicia a especificação de alguns parâmetros relacionados ao desempenho da arquitetura que será escolhida para dar suporte à execução do mesmo. Portanto, já nesta fase, é possível elaborar modelos para representar o comportamento esperado do sistema, com o nível de detalhe desejado, e avaliar a escolha dos parâmetros especificados através de simulações;

- **seleção do sistema**: as técnicas de avaliação de desempenho podem ser utilizadas no processo de avaliação e escolha de sistemas, quando se tem opções para aquisição ou desenvolvimento. Assim, é possível escolher um sistema avaliando aspectos de interesse, como custo, desempenho, confiabilidade, disponibilidade, entre outros;

- **atualização do sistema**: atualizações do sistema requerem análises sobre o comportamento do mesmo após possíveis modificações. Nesse ponto, modelos analíticos são interessantes para estimar o comportamento do sistema diante das modificações pretendidas;

- **análise do sistema**: esta atividade envolve monitoração do sistema, análise de cada recurso de *hardware*, estudo do comportamento e reação do *software* sobre diferentes cargas de processamento. O objetivo, neste caso, é encontrar possíveis problemas relacionados ao desempenho da solução que poderiam ser eliminados;

- **otimização do sistema**: a representação do sistema através de modelos analíticos permite variar parâmetros e simular alterações no comportamento do mesmo. Assim, as modificações podem direcionar otimizações em relação ao uso dos recursos computacionais.

3.4 Trabalhos Relacionados

Alguns trabalhos de autores que pesquisam e desenvolvem projetos na área de análise, avaliação e predição de desempenho são discutidos nesta seção. A maioria dos trabalhos encontrados na literatura utiliza como infra-estrutura sistemas distribuídos homogêneos para computação paralela de alto desempenho. Por isso, é possível perceber que existe um número reduzido de estratégias ou metodologias elaboradas para auxiliar atividades de análise e predição de desempenho de aplicações paralelas em sistemas heterogêneos. Em

geral, a complexidade e o comportamento dinâmico desses sistemas dificultam o estudo de seu comportamento e, conseqüentemente, sua previsibilidade.

3.4.1 PEVPM

Um sistema para modelar o desempenho de programas paralelos que utilizam o modelo de programação baseado em passagem de mensagem foi proposto em (GROVE, 2003). Esse sistema, denominado PEVPM (*Performance Evaluating Virtual Parallel Machine*), faz uso de máquinas paralelas virtuais para realizar as atividades relacionadas à análise e avaliação de desempenho das aplicações, em uma abordagem *bottom-up* (GROVE; CODDINGTON, 2005b).

Primeiramente, o programa é instrumentado com diretivas PEVPM e depois é simulado sobre uma máquina paralela virtual. Em vez de executar o próprio programa em uma situação real, o sistema simula a execução das diretivas que foram inseridas e constrói, a partir dos resultados, os modelos de predição para a aplicação. As medidas e avaliações de desempenho são obtidas através da técnica de simulação baseada em modelos estocásticos, denominada Monte Carlo (KLEIJNEN, 1974). Essa técnica é utilizada em diversas áreas, inclusive em aplicações da física e da matemática computacional, ajudando a resolver, de modo eficiente, complexos sistemas de equações diferenciais integrais. Uma das diferenças em relação ao PEMPIs-Het está no fato do PEVPM fazer uso de simulações e não elaborar os modelos de predição a partir da execução real do programa. Embora nas simulações seja possível repetir com mais precisão testes experimentais, o fato de não utilizar a aplicação real pode diminuir a precisão das representações dos modelos de predição. Além disso, os modelos gerados não são aplicados em atividades de balanceamento de carga computacional.

A estrutura do programa é dividida em segmentos de código seqüenciais e de comunicação, sendo cada trecho modelado separadamente. Essa estratégia também é possível de ser adotada e desenvolvida no PEMPIs-Het, o que possibilita um estudo separado de elementos do programa.

Pelo fato de o estudo ser realizado sobre uma máquina paralela virtual, muitos fatores relacionados à complexidade da arquitetura de uma máquina paralela real podem ser abstraídos. Com isso, a construção dos modelos torna-se mais fácil. De acordo com os comentários do autor, o sistema proposto tem condições de elaborar modelos de predição de desempenho precisos a um custo relativamente baixo.

3.4.2 Dias Jr., E.

Em (DIAS, 2006) o autor investiga como reduzir o tempo de execução de aplicações paralelas que utilizam o modelo de programação mestre-escravo. As avaliações são sobre aplicações que originalmente foram implementadas para executar em um único *cluster*,

mas que estão sendo testadas em ambientes que fazem uso de vários *clusters*. Este enfoque é diferente do que tem sido proposto pela metodologia PEMPIs-Het. Enquanto Dias, em seu trabalho, preocupa-se com aspectos de desempenho e eficiência de aplicações que podem ser portáveis para sistemas com múltiplos *clusters*, a metodologia PEMPIs-Het estabelece formas de se modelar o comportamento das aplicações MPI e, com isso, poder estimar seu desempenho e avaliar a escalabilidade do sistema e o escalonamento de tarefas para otimizar o uso dos recursos computacionais de diferentes ambientes, heterogêneos ou não.

O autor utiliza técnicas de modelagem analítica para modelar as aplicações e estimar seu comportamento. Uma metodologia foi criada para orientar as atividades de análise e predição de desempenho, bem como a seleção de recursos do ambiente, de modo a reduzir o tempo de execução das aplicações e garantir eficiência.

3.4.3 Schopf, J. M.

Schopf (SCHOPF, 1998) utiliza uma modelagem estrutural para representar o comportamento das aplicações paralelas e distribuídas. Nesta modelagem, alguns aspectos isolados da aplicação são avaliados, como a estrutura funcional do programa e a interação entre as tarefas. Os modelos de desempenho são parametrizados através de valores estocásticos e os resultados das predições são aplicados em atividades de predição de desempenho e escalonamento de tarefas em *clusters* compartilhados de estações de trabalho.

Embora esse trabalho apresente atividades semelhantes ao que está sendo proposto por nossa metodologia PEMPIs-Het, a forma e a técnica utilizada para elaborar os modelos de predição de desempenho são diferentes. Além disso, o trabalho de Schopf analisa aplicações em *clusters* compartilhados de estações de trabalho, enquanto o PEMPIs-Het descreve um conjunto de técnicas que podem ser aplicáveis em sistemas mais amplos e dinâmicos, como os *grids* computacionais. Uma das semelhanças entre os dois trabalhos está na classe de aplicações estudada: programas que usam o modelo mestre-escravo para computação distribuída. Muitas vezes, aplicações MPI fazem uso desse modelo na organização das soluções distribuídas. No entanto, nada impede o uso da metodologia PEMPIs-Het em análises de outros modelos de programação, como cliente-servidor ou *peer-to-peer*.

3.4.4 BSP

Segundo Culler (CULLER et al., 1993), o modelo de computação paralela mais utilizado, chamado PRAM, não é um modelo realístico, pois considera que todos os processadores trabalham do modo síncrono e que a comunicação entre eles utiliza uma largura de banda infinita, sem latência e nenhum atraso na comunicação. Para Harris (HARRIS, 1994) o PRAM é um modelo abstrato que pode ser útil para o estudo de alguns aspectos da com-

putação paralela, mas não é possível implementá-lo fisicamente em *hardware*. Algumas variações do modelo PRAM foram criadas na tentativa de eliminar restrições e torná-lo mais prático, preservando sua simplicidade. O BSP (Bulk-Synchronous Parallel model) (VALIANT, 1990) é um modelo que foi elaborado com esse intuito.

No BSP os programas são escritos para processadores virtuais e executados em programas reais. Assim, seu objetivo principal é fornecer um ambiente para a elaboração de algoritmos eficientes e portáveis. No BSP é possível modelar latência e largura de banda limitada de rede, bem como permitir que os processadores trabalhem de modo síncrono.

O modelo gerado abstrai estruturas de baixo nível de uma aplicação através de um conjunto de "superpassos"envolvidos na computação do programa. Os processos executam uma seqüência de "superpassos", que inicia com alguma computação local, passa por comunicações e termina com uma barreira de sincronização.

O modelo de programação BSP pode ser utilizado em várias linguagens e bibliotecas, como, por exemplo, o PVM e MPI. Os modelos gerados são aplicados em atividades de predição de desempenho, permitindo análises individuais de cada um dos "superpassos"identificados no programa. Uma análise total do desempenho da aplicação envolveria a somatória dos custos computacionais de cada um desses passos.

3.4.5 LogP

O modelo LogP (CULLER et al., 1993) foi elaborado com o propósito de caracterizar os principais fatores associados ao desempenho de máquinas paralelas com memória distribuída. O modelo considera um conjunto finito de processadores e a sincronização entre eles é feita via passagem de mensagem, caracterizando o desempenho da rede de intercomunicação, mas não a topologia da mesma. Assim como o PEMPIs-Het, o LogP é utilizado para avaliar desempenho de aplicações que utilizam o paradigma de passagem de mensagem. Os modelos gerados pelo LogP dão ênfase aos fatores de desempenho relacionados à rede de interconexão. Esses modelos são descritos a partir dos seguintes parâmetros:

- L: caracteriza a latência na transmissão de mensagens pela rede;
- o: modela atrasos (*overheads*) gerados pelas transmissões com o uso de primitivas do tipo *send* e *receive*, durante o período em que o processador fica dedicado para a transmissão dos dados;
- g: define o tempo mínimo entre transmissões ou recepções consecutivas de dados pela rede;
- P: representa a quantidade de processadores/módulos de memória.

Tanto o BSP quanto o LogP podem ser utilizados para auxiliar no desenvolvimento de algoritmos paralelos eficientes. O BSP modifica o modelo PRAM, tratando algumas

de suas restrições de programação, e o modelo LogP conduz a uma convergência de arquiteturas paralelas já existentes.

3.4.6 PAPS

O PAPS (WABNIG; HARING, 1995) é um conjunto de ferramentas que permite especificar sistemas paralelos orientados a desempenho, descrevendo aspectos relacionados à carga de trabalho, recursos de *hardware* e o mapeamento do programa com os elementos de processamento. O objetivo é permitir que se realize avaliação de desempenho logo no início do desenvolvimento do sistema. Nesse ambiente, existe um gerador automático dos modelos de predição de desempenho e um simulador para obter resultados das predições. Assim como o PEVPM, esta metodologia faz uso de simulações para a realização das análises de desempenho.

A ferramenta que permite estimar o desempenho é composta por três camadas, conforme explicado a seguir:

- 1^a camada: denominada camada de especificação. Informações sobre a carga de trabalho, as características do ambiente (*hardware*) e alguns mapeamentos dos elementos de processamento e tarefas são especificados nesta primeira camada. Esses parâmetros podem ser variados, permitindo analisar o comportamento do programa sobre diferentes condições;

- 2^a camada: chamada de camada de transformação. Essa camada obtém informações definidas na camada de especificação e gera o modelo de desempenho do programa paralelo. Redes de Petri temporizadas são utilizadas para analisar as especificações;

- 3^a camada: utiliza o modelo gerado pela camada anterior para realizar a análise e predição do desempenho do programa modelado. Os resultados das predições são obtidos através de um simulador de Redes de Petri.

PAPS também utiliza um modelo gráfico, baseado em grafos acíclicos direcionados, para representar a estrutura do programa. Nessa representação, os nós dos grafos caracterizam as tarefas, e os arcos ilustram as comunicações ou alguma relação de dependência entre os nós. O PEMPIs-Het também faz uso de modelagem gráfica, através do DP*Graph^{++}, mas com uma representação que oferece mais detalhes para o entendimento do programa.

3.4.7 PAMELA

O autor van Gemund (GEMUND, 1993, 1994, 2003) também apresenta uma metodologia para predição de desempenho de sistemas paralelos. A metodologia elaborada pode ser aplicada tanto em ambientes de memória compartilhada, quanto em ambientes de memória distribuída.

Nessa metodologia, é gerado um modelo simbólico do programa e outro da máquina, através de uma linguagem imperativa de simulação orientada a processo, denominada PAMELA. Depois de elaborados esses modelos individuais são combinados em um único modelo, que une as características do programa e da arquitetura. Portanto, é possível realizar estudos de desempenho variando parâmetros do programa e da arquitetura, como o tamanho do problema e o número de processadores. A metodologia PEMPIs-Het também permite essas análises, considerando os parâmetros citados. No entanto, a maior parte das variáveis que podem ser modeladas diz respeito a aspectos da aplicação.

A linguagem PAMELA serve como uma linguagem de simulação de desempenho, permitindo que o modelo seja elaborado e analisado em tempo de execução. Além disso, ela também permite a especificação de modelos de predição em tempo de compilação. Já os modelos criados com o PEMPIs-Het utilizam resultados de execução para o seu desenvolvimento.

Durante a fase de análise, uma técnica denominada análise de serialização, que se baseia em técnicas tradicionais de análise de programas paralelos, é utilizada, e uma análise probabilística sobre os problemas de contenção envolvidos no sistema é incluída.

3.4.8 Gubitoso, M. D.

Em (GUBITOSO, 1996), modelos analíticos, combinados com simulações, são utilizados para estimar o desempenho de aplicações sobre sistemas de memória compartilhada virtual. Nesse ambiente, um sistema de memória distribuída faz uso do paradigma de passagem de mensagens, mas para o programador é oferecido um ambiente de memória compartilhada, tornando mais fácil o desenvolvimento das aplicações. Até o momento, os métodos definidos pelo PEMPIs-Het ainda não foram testados sobre sistemas do tipo DSM (*Distributed Shared Memory*) (TAM; SMITH; FARBER, 1990).

Um estudo aprofundado é conduzido por Gubitoso sobre as estratégias de alocação de processadores na execução paralela de estruturas de repetições com interações independentes (chamados de *DOALL*). Além disso, uma combinação de modelos analíticos com simulações é utilizada para predizer o desempenho de aplicações baseadas em decomposição de domínio, levando em consideração máquinas que utilizam uma comunicação assíncrona. Com os modelos analíticos gerados pelo PEMPIs-Het é possível analisar e predizer o comportamento de aplicações que utilizam tanto comunicação síncrona, quanto assíncrona na troca de dados.

Um novo método para elaboração de modelos de desempenho, baseado na implementação de um sistema chamado *VOTE*, utilizando uma máquina conhecida como *MANNA*, é proposto. O objetivo principal do método é modelar o custo das sobrecargas. Para isso, é necessário identificar todos os pontos do programa que podem gerar uma sobrecarga. A idéia do método é permitir que os modelos gerados possam ser aplicados em outras

arquiteturas.

3.4.9 Fahringer, T.

Uma estratégia para predição de desempenho de programas paralelos é apresentada em (FAHRINGER, 1994). A atividade de predição de desempenho de programas paralelos é realizada com o auxílio de uma ferramenta denominada P^3T (*Parameter based Performance Prediction Tool*). Essa ferramenta modela três dos mais importantes aspectos de desempenho de um programa paralelo: balanceamento de carga, atrasos de comunicação e localidade dos dados. A ferramenta P^3T permite saber o número de mensagens transferidas, a quantidade de dados transferidos, o tempo gasto nessas transferências, o número de faltas em cache, a forma de distribuição do trabalho, entre outros fatores. Esses elementos ajudam a realizar análises de desempenho de versões diferentes de um mesmo programa. De forma semelhante, na metodologia PEMPIs-Het, é possível avaliar, separadamente, aspectos relacionados não só ao desempenho computacional, mas também ao balanceamento de carga do sistema. Uma das vantagens do PEMPIs-Het é que o balanceamento, em conjunto com os modelos, pode ser recalculado dinamicamente, ao longo da execução das aplicações. Esse fato pode melhorar, sensivelmente, o desempenho dos programas em sistemas heterogêneos dinâmicos, como *grids* computacionais.

Os valores dos parâmetros, citados anteriormente, podem ser determinados para conjuntos específicos de instruções, estruturas de repetição (laços), procedimentos específicos ou para todo o programa paralelo. Com base nos resultados obtidos pela P^3T, modificações no código do programa são realizadas através de um compilador denominado VFCS (*Vienna Fortran Compilation System*). O VFCS é um compilador integrado a P^3T que automaticamente transforma programas Fortran em programas paralelos. A ferramenta P^3T simplesmente avalia versões diferentes de programas paralelos gerados pelo compilador VFCS, determinando qual delas oferece melhor desempenho.

3.4.10 Li, K. C.

Li propõe uma metodologia de análise e predição de desempenho de programas paralelos implementados com MPI (*Message Passing Interface*) sobre redes de estações de trabalho (NOWs) (LI, 2001). A metodologia faz uso de uma abordagem híbrida, aplicando técnica de modelagem analítica em conjunto com medições para realizar o estudo de desempenho dos programas MPI, estratégia semelhante à adotada pelo PEMPIs-Het.

Uma representação gráfica e outra matemática são utilizadas para avaliar o desempenho dos programas MPI. Para a representação gráfica, foram desenvolvidas duas novas classes de grafos de tempo, uma denominada T-Graph* (alto nível) e a outra DP-Graph* (baixo nível). No T-Graph* apenas a estrutura geral do programa é representada, enquanto no DP-Graph* os trechos de códigos seqüenciais, as comunicações e o fluxo de

execução do programa são ilustrados.

Cada trecho de código do programa paralelo representado no DP-Graph* possui características individuais (tempo de execução, quantidade de processamento, quantidade de comunicação, etc.). A idéia da metodologia é, após desenvolvido o grafo de baixo nível, submeter o programa a testes experimentais sobre o ambiente de estudo (*cluster* de PCs). Durante os testes alguns parâmetros são modificados, como, por exemplo, o número de nós ou o tamanho do problema. Através de medições, o tempo de execução de cada um dos trechos representados no DP-Graph* é coletado. Depois, os resultados experimentais são utilizados pela metodologia na elaboração do modelo analítico de desempenho. O modelo analítico permite a realização de predição de desempenho do programa paralelo em condições não consideradas nos experimentos.

3.4.11 Carnival

Carnival (MEIRA; LEBLANC; POULOS, 1996) é uma ferramenta criada para realizar atividades de medição, análise e visualização de desempenho de programas que utilizam passagem de mensagem em máquinas do tipo SIMD (FLYNN, 1972). A ferramenta automatiza o processo de inferência utilizado para descobrir pontos de espera na aplicação. Para isso, um detalhado rastreamento (*trace*) de eventos é realizado e depois utilizado para obter informações sobre o desempenho dos programas. Informações estatísticas, tanto sobre o desempenho global da aplicação quanto de pontos específicos do processamento, são apresentadas. A *interface* da ferramenta para o usuário apresenta informações sobre o desempenho junto com o código fonte da aplicação. Além disso, algumas ligações são criadas entre o fenômeno detectado na análise realizada pela ferramenta Carnival e as instruções do código responsáveis pelo evento. Além de apontar onde os eventos relacionados aos tempos de esperas ocorrem, a ferramenta também explica a origem de cada um destes tempos e os seus efeitos, facilitando a análise da aplicação e sua otimização.

Toda instrumentação do programa com os monitores de tempo e as chamadas para as funções da biblioteca de rastreamento de eventos são realizadas pela própria ferramenta. As informações coletadas são utilizadas em um pós-processamento, momento em que as análises são realizadas. A ferramenta mede e apresenta resultados de desempenho sobre várias categorias diferentes do tempo de execução, tais como: computação local, computação paralela, chamadas de sistema, operações de envio e recepção de dados, barreiras de sincronização, tempos de espera, entre outros. Os resultados são apresentados na *interface* da ferramenta.

3.5 A Metodologia PEMPIs

No início do nosso trabalho de pesquisa, a metodologia PEMPIs (**P***erformance* **E***valuation of* **MPI** *program***s**) (MIDORIKAWA; OLIVEIRA; LAINE, 2004, 2005) foi elaborada com o propósito de permitir atividades de análises, avaliação e predição de desempenho de programas paralelos MPI em sistemas distribuídos homogêneos. Combinando várias técnicas de modelagem (incluindo modelagem gráfica baseada em grafos, métodos de ajustes de curvas, modelagem analítica, etc.), a metodologia PEMPIs oferece uma abordagem híbrida para o desenvolvimento de modelos de predição de desempenho. Os modelos analíticos podem ser utilizados com diferentes propósitos, como a avaliação da escalabilidade dos sistemas ou a predição do comportamento da aplicação em situações não modeladas. Uma deficiência da metodologia é que as técnicas elaboradas só podem ser aplicadas em sistemas homogêneos de computação paralela. Portanto, aplicações para *grids* computacionais e *clusters* heterogêneos não podem ser modeladas com o PEMPIs.

Um sumário das principais características do PEMPIs é listado a seguir:

- modelagem estrutural: usando os símbolos do modelo gráfico DP*Graph$^+$, é possível representar, através de um modelo baseado em grafos, a estrutura estática do programa paralelo MPI;
- modelagem analítica: o PEMPIs utiliza técnicas de modelagem analítica para gerar modelos de estimativas de desempenho da aplicação;
- medições de desempenho: os coeficientes dos modelos de predição são obtidos através de medições de desempenho realizadas em testes experimentais;
- comunicação e processamento são analisados separadamente: diferentes estratégias são utilizadas para modelar as primitivas de comunicação MPI e os trechos de computação local do programa;
- análise do código fonte: o modelo teórico da aplicação é obtido através da análise da complexidade do código fonte do programa;
- abordagem prática: a metodologia PEMPIs foi elaborada para avaliar e estimar o desempenho de sistemas paralelos através de estudos práticos da aplicação real. Por fazer uso de medições, as técnicas elaboradas no PEMPIs permitem uma avaliação de desempenho consistente e precisa para o desenvolvimento dos modelos de predição. A Figura 3.1 apresenta a organização da metodologia.

As etapas para a aplicação da metodologia são:

1. A partir do código fonte do programa MPI, é elaborado o modelo gráfico DP*Graph$^+$ da aplicação, cujos principais símbolos são apresentados na Figura 3.2. Esse modelo

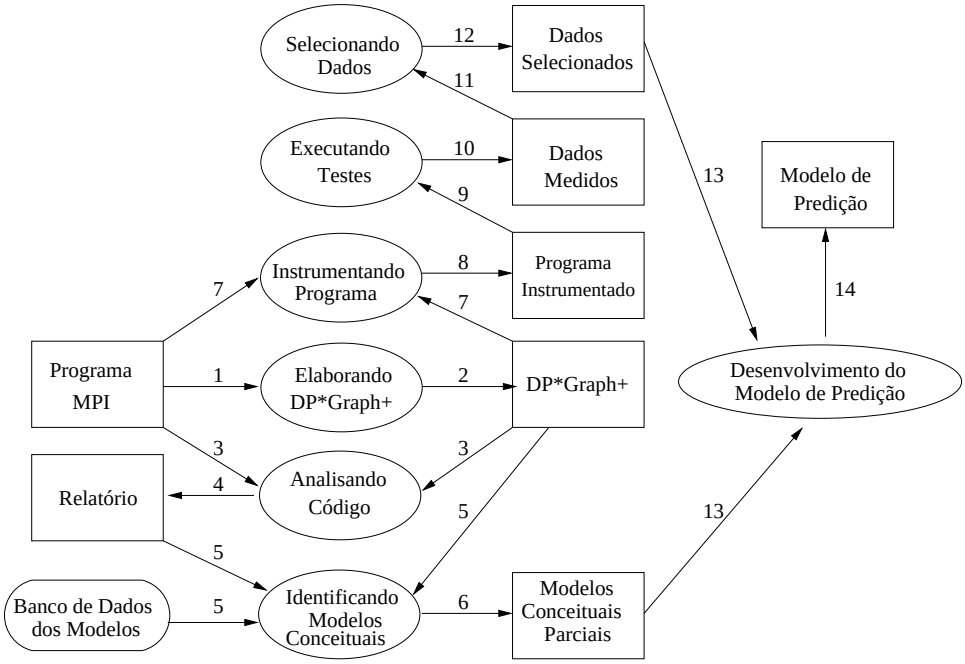

Figura 3.1: Esquema utilizado pela metodologia PEMPIs.

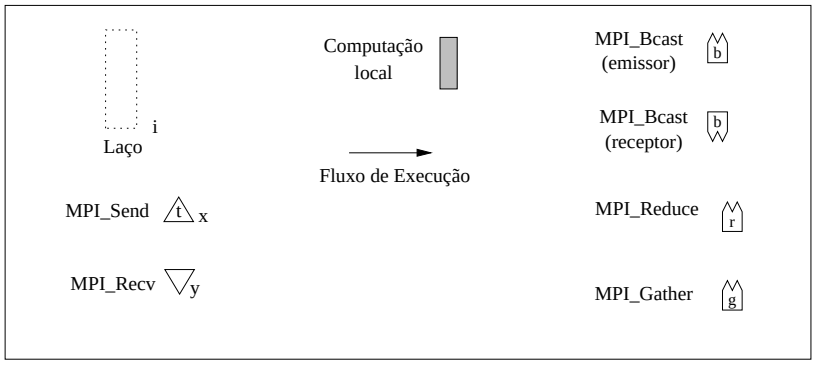

Figura 3.2: Alguns símbolos do modelo gráfico DP*Graph$^+$.

consegue representar as estruturas do programa através de um conjunto de símbolos individuais. Facilitar o entendimento do programa, encontrar os possíveis gargalos para o desempenho, determinar os pontos de sincronismo entre as tarefas e separar as primitivas de comunicação das instruções de processamento são os objetivos do modelo gráfico DP*Graph$^+$;

2. Após elaborar o modelo gráfico DP*Graph$^+$, é possível utilizá-lo para ajudar a determinar a ordem de complexidade do programa e o modelo teórico da aplicação. Os modelos conceituais podem ser desenvolvidos para modelar trechos individuais que se deseja analisar no programa MPI ou um modelo completo para representar o comportamento do programa como um todo. Os modelos são armazenados em um repositório de dados para que possam ser utilizados no estudo de outras aplicações;

3. Depois de identificar o que será modelado, é necessário inserir os monitores de

software para coletar os tempos de execução, durante os testes experimentais, necessários para o desenvolvimento dos modelos analíticos;

4. O passo seguinte é projetar os testes experimentais para o programa. Para cada configuração de teste, várias execuções são realizadas a fim de aumentar a confiabilidade dos tempos medidos e melhorar a precisão dos modelos de predição. Os tempos coletados são armazenados em arquivos e selecionados com uma ferramenta elaborada para descartar eventuais anomalias;

5. Com os tempos selecionados e o modelo teórico do programa, outra função é aplicada para gerar os modelos de predição, através de técnicas de ajustes de curvas. Esses modelos são desenvolvidos em função de dois parâmetros: tamanho do problema e o número de processos utilizado na execução do programa. Portanto, variando esses parâmetros, é possível estimar o desempenho em situações ainda não analisadas.

A metodologia PEMPIs surgiu com a junção e extensão dos trabalhos (LAINE, 2003) e (OLIVEIRA, 2003). As técnicas e ferramentas elaboradas para o desenvolvimento das atividades, descritas nesta seção, representam uma versão inicial do PEMPIs-Het, só que restrita a sistemas homogêneos e aplicadas somente às atividades de análise, avaliação e predição de desempenho.

3.6 Características dos Trabalhos Relacionados

Na Tabela 3.1, apresentamos uma comparação entre as principais características de trabalhos relacionados aos temas tratados nesta tese. Todos os trabalhos estão relacionados na lista de referências. O objetivo desta tabela é comparar os aspectos tratados em nosso trabalho e os considerados pelos demais autores.

Até 2003, os trabalhos, basicamente, avaliavam sistemas homogêneos de computação. Recentemente, as análises e avaliações passaram a considerar sistemas heterogêneos, devido ao avanço dos *grids* computacionais, principalmente. A maioria dos trabalhos citados permite avaliações de aplicações completas, aspectos relacionados ao desempenho da comunicação, estruturas específicas de computação, como laços de repetição, e apresentam algum mecanismo para balanceamento de carga. No entanto, nenhum deles permite a análise e comparação de diferentes modelos de soluções paralelas. Essa análise é uma atividade que pode ser realizada com as estratégias definidas na metodologia PEMPIs-Het.

A seguinte legenda deve ser utilizada na leitura da tabela: HOM (Homogêneo); HET (Heterogêneo); AC (Aplicação Completa); COM (Comunicação); EP (Estruturas do Programa); ER (Estruturas de Repetição); BC (Balanceamento de Carga) e MSP (Modelos de Soluções Paralelas).

Tabela 3.1: Resumo das principais características dos trabalhos relacionados.

	Ano	HOM	HET	AC	COM	EP	ER	BC	MSP
LogP	1993	X		X	X				
Fahringer	1994	X		X	X	X	X	X	
Gubitoso	1996	X		X	X	X	X		
Schopf	1998	X		X	X	X		X	
PEVPM	2003	X		X	X				
Banino	2004		X					X	
Lastovetsky	2006		X		X				
Shih	2007		X			X	X	X	
Yang	2007		X			X	X	X	
PEMPIs	2004	X		X	X	X	X		
PEMPIs-Het	2007	X	X	X	X	X	X	X	X

3.7 Considerações Finais

Neste capítulo destacou-se a importância das técnicas para análise, avaliação e predição de desempenho e suas aplicações. Embora várias técnicas possam ser utilizadas com esse propósito, a modelagem analítica é uma das mais comuns e permite estimar e otimizar o desempenho das aplicações ao longo de várias etapas do ciclo de vida de desenvolvimento dos sistemas. Os modelos analíticos podem ser aplicados com outros propósitos, como auxiliar a distribuição de cargas computacionais no sistema. Portanto, é possível avaliar formas de divisão do trabalho e estratégias de distribuição de tarefas, a fim de diminuir o tempo de execução dos programas.

Apresentamos alguns trabalhos relacionados ao tema desta pesquisa e discutimos as particularidades de cada um. Comentamos que o intuito deste trabalho é definir uma metodologia que permita estudar aplicações paralelas em sistemas homogêneos e heterogêneos de computação distribuída.

Capítulo 4

A Medodologia PEMPIs-Het

Neste capítulo, descrevemos as técnicas e os métodos de análise, avaliação e predição de desempenho contemplados na metodologia PEMPIs-Het. Além disso, apresentamos a seqüência das etapas propostas na metodologia para a orientação das atividades especificadas. As atividades implementadas durante o desenvolvimento deste trabalho também são apresentadas. Discutimos a importância de balanceamento de carga em sistemas distribuídos e alguns modelos de soluções para aplicações paralelas distribuídas.

4.1 Introdução

Inicialmente, nosso objetivo foi especificar uma metodologia capaz de orientar atividades relacionadas à avaliação, análise e predição de desempenho de programas paralelos em sistemas distribuídos heterogêneos. Para isso, estendemos as atividades propostas na metodologia PEMPIs (MIDORIKAWA; OLIVEIRA; LAINE, 2004), originalmente criada para orientar estudos sobre o desempenho de aplicações paralelas em ambientes homogêneos. Com a evolução do trabalho, criamos a metodologia PEMPIs-Het.

A esta nova metodologia, adicionamos atividades relacionadas ao planejamento da divisão e distribuição de trabalho entre as máquinas do sistema. Para isso, algumas estratégias de balanceamento de carga foram criadas com o objetivo de melhorar o desempenho dos programas, ajustando a sincronização dos processos e maximizando o uso dos recursos computacionais.

Além disso, decidimos especificar um conjunto de atividades na metodologia PEMPIs-Het para guiar avaliações comparativas de modelos de soluções distribuídas. O objetivo destas análises é construir modelos analíticos para representar o comportamento de diferentes possíveis soluções de um problema. Conseqüentemente, estimativas de desempenho poderão ser realizadas e a estrutura de solução mais adequada a uma determinada aplicação poderá ser determinada com as avaliações.

Pelo fato do desenvolvimento de aplicações paralelas eficientes não ser uma tarefa trivial, explorar o paralelismo das aplicações é tão complicado quanto estruturar a solução

para otimizar o uso dos recursos computacionais e alcançar o desempenho desejado. Essa atividade é ainda mais complexa se o sistema computacional for distribuído e heterogêneo. Soluções paralelas eficientes para ambientes distribuídos devem levar em consideração não só a correta computação do problema mas também a forma como o trabalho é dividido e distribuído entre os recursos computacionais, a fim de maximizar o uso da capacidade individual de cada uma das máquinas do ambiente. Uma discussão sobre o conceito de eficiência para programas paralelos foi feita no Capítulo 1.

Na tentativa de melhorar o desempenho das aplicações, vários problemas devem ser considerados e tratados, tais como o modelo de divisão e distribuição de carga, a insuficiência de paralelismo no programa, o sincronismo estabelecido entre as tarefas, a transmissão de dados pela rede e a contenção de recursos (GROVE; CODDINGTON, 2005a).

Nesse contexto, a avaliação de diferentes modelos de soluções para um programa distribuído é uma tarefa imprescindível no desenvolvimento de aplicações de alto desempenho. Decidir qual modelo é o mais eficiente requer análises sobre o desempenho de cada possível abordagem bem como predições sobre o comportamento de cada uma das soluções implementadas.

Em geral, uma solução paralela pode ser organizada através dos seguintes modelos: mestre-escravo, paralelismo de dados, *workpool* e divisão e conquista. No entanto, o modelo mestre-escravo é um dos mais difundidos (SILVA; BUYYA, 1999; QUINN, 2003; WILKINSON; ALLEN, 1999). Nesse modelo, o processo mestre é o responsável por organizar a divisão e distribuição do trabalho e agrupar as respostas parciais para obter a solução do problema. Cabe aos escravos processar a tarefa recebida e devolver o resultado da execução para o mestre. Quando todos os escravos enviam os resultados parciais, a solução final do problema é obtida (BEAUMONT; LEGRAND; ROBERT, 2003).

Pelo nosso conhecimento, a área da computação paralela e distribuída é carente de metodologias que auxiliem estudos sobre estruturas de programas paralelos. Alguns trabalhos têm avaliado elementos específicos de um programa paralelo, como as primitivas de comunicação (LASTOVETSKY; MKWAWA; O'FLYNN, 2006), as estruturas de repetição (SHIH; YANG; TSENG, 2007; YANG; CHENG; SHIH, 2007), o escalonamento das tarefas (BANINO et al., 2004) ou o desempenho de aplicações (BADIA; RODRÍGUEZ; LABARTA, 2003; GROVE; CODDINGTON, 2005a). No entanto, desconhecemos trabalhos que avaliem especificamente estruturas de programas paralelos com o intuito de gerar soluções otimizadas para um ambiente computacional conhecido. Para auxiliar nos estudos relacionados às atividades citadas neste capítulo criamos a metodologia PEMPIs-Het (*Performance Estimation of MPI Programs in Heterogeneous Systems*) (LAINE; MIDORIKAWA, 2007b).

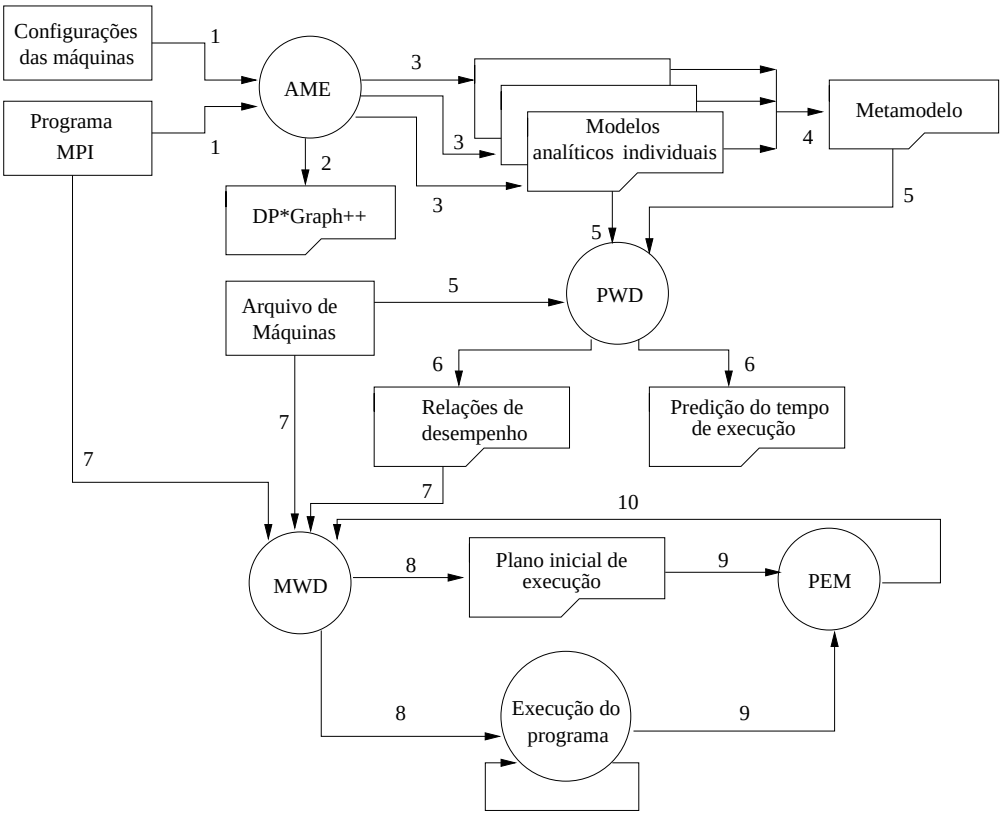

Figura 4.1: Metodologia PEMPIs-Het.

4.2 A Metodologia PEMPIs-Het

Além de estender o processo de modelagem e avaliação de desempenho, especificado na metodologia PEMPIs, atividades associadas ao planejamento da distribuição das cargas computacionais agora são contempladas pela metodologia PEMPIs-Het. As restrições sobre o ambiente computacional foram eliminadas, permitindo que aplicações paralelas sejam modeladas matemática e visualmente em ambientes de computação distribuída, homogêneos ou não. O esquema da metodologia PEMPIs-Het está ilustrado na Figura 4.1. Os seguintes processos foram especificados na metodologia PEMPIs-Het:

- AME - ***A****pplication* ***M****odeling* ***E****nvironment*;
- PWD - ***P****erformance Estimation and* ***W****orkload* ***D****istribution*;
- MWD - ***M****iddleware for* ***W****orkload* ***D****istribution*;
- PEM - ***PE****rformance* ***M****onitor*.

Nas próximas seções, esses processos são explicados e os produtos gerados em cada um deles são especificados.

4.2.1 AME

O processo AME (***A**pplication **M**odeling **E**nvironment*) especifica um conjunto de atividades relacionadas à modelagem das aplicações MPI na metodologia PEMPIs-Het. Nesta etapa, analisamos a estrutura e organização do código fonte do programa e construímos o DP*Graph^{++} da aplicação. Tanto a construção do modelo gráfico quanto a análise da complexidade do algoritmo são atividades ainda não automatizadas em *software*. Criamos um repositório para os símbolos gráficos usados pelo modelo, e através de um programa de edição de figuras montamos o DP*Graph^{++}. Como trabalho futuro desejamos implementar uma ferramenta que analise o código da aplicação e gere o DP*Graph^{++} de modo automático, e realize algumas análises sobre o desempenho, semelhante à ferramenta Carnival (MEIRA; LEBLANC; POULOS, 1996). A partir da representação visual, determinamos a complexidade algorítmica da aplicação e o modelo de desempenho teórico do programa.

A próxima etapa do processo de modelagem consiste em instrumentar o código do programa e preparar a aplicação para os testes experimentais. Para isso, inserimos os monitores de tempo no código fonte do programa. Atualmente, esta atividade também é realizada manualmente, mas pode ser uma funcionalidade da ferramenta proposta anteriormente. Esses monitores são responsáveis por medir o tempo de execução da aplicação ou de algumas estruturas específicas do programa, como instruções de processamento ou primitivas de comunicação, por exemplo.

O planejamento dos testes experimentais é uma atividade importante e está relacionada a precisão dos modelos analíticos de desempenho. É neste planejamento que definimos quais as combinações de p (número de processos) e n (tamanho do problema) deverão ser analisadas, quais máquinas serão utilizadas na execução do programa e a quantidade de vezes que cada teste deve ser repetido no ambiente.

Após o término dos testes experimentais, iniciamos o desenvolvimento dos modelos analíticos. Para auxiliar a elaboração dos modelos, implementamos algumas funções com o auxílio do programa Scilab[1]. O Scilab oferece um ambiente computacional aberto para computação numérica, semelhante ao Matlab[2].

Uma das funções implementadas realiza a leitura dos arquivos que contêm os tempos coletados com os monitores e aplica uma política de seleção de dados para descartar eventuais anomalias. A política implementada utiliza o tempo médio (tm) e desvio padrão (dp) da amostra dos tempos para selecionar os dados. Dessa forma, o intervalo de seleção é definido da seguinte maneira: $[tm - dp, tm + dp]$. A partir desta seleção é calculado um tempo médio para representar o comportamento padrão de cada um dos trechos analisados.

Com os tempos selecionados, temos outra função implementada com o auxílio do Scilab que permite aplicar o método de ajuste de curva dos mínimos quadrados e gerar os

[1] http://www.scilab.org/

[2] *MAT*rix *LAB*oratory: http://www.mathworks.com/products/matlab/

coeficientes para o modelo teórico da aplicação. Nesta etapa, a aplicação é caracterizada por modelos analíticos individuais, elaborados para representar o comportamento da aplicação em cada tipo de máquina do sistema.

A partir desses modelos individuais, geramos um metamodelo para caracterizar a aplicação em um conjunto de máquinas do ambiente. Assim, um único modelo de desempenho pode ser gerado, em função de quantidade de processos e/ou do tamanho do problema, para estimar o tempo de execução do programa em um ambiente heterogêneo.

4.2.2 PWD

PWD (***P**erformance Estimation and **W**orkload **D**istribution*) é o processo da metodologia que especifica as atividades associadas à predição de desempenho da aplicação e a forma como as relações de desempenho são determinadas para as atividades de balanceamento de carga.

A partir dos modelos analíticos, elaborados anteriormente, é possível estimar o tempo de execução da aplicação para diferentes configurações de p e n, e criar o gráfico de tendência. Com as estratégias de modelagem da metodologia PEMPIs-Het, o comportamento do programa pode ser previsto para qualquer conjunto de máquinas do ambiente modelado.

Outra atividade especificada no processo PWD é a geração das relações de desempenho entre as máquinas do sistema. Essas relações representam a capacidade de processamento relativa entre cada par de máquinas do ambiente, considerando uma aplicação específica. O intuito dessas relações é determinar a quantidade de trabalho adequada a cada uma das máquinas para evitar sobrecargas ou ociosidades no sistema. Este ajuste deve ser feito no instante em que o trabalho total será dividido pelo processo mestre e distribuído aos escravos. Uma distribuição adequada das cargas favorece a utilização dos recursos computacionais e, conseqüentemente, pode ajudar a diminuir o tempo total de execução da aplicação. Para determinar as relações de desempenho utilizamos os modelos analíticos individuais da aplicação, conforme detalhado na Seção 4.7.

Assim, os produtos deste processo, denominado PWD, são as estimativas para o tempo de execução da aplicação e as relações de desempenho entre as máquinas. As relações serão utilizadas no processo MWD para gerar um plano inicial de execução da aplicação, conforme descrito na próxima seção.

4.2.3 MWD

O objetivo das atividades especificadas neste processo da metodologia, denominado MWD (***M**iddleware for **W**orkload **D**istribution*) é gerar um plano de execução inicial para a aplicação. Este plano busca otimizar o desempenho do programa, maximizando o uso da capacidade de processamento dos recursos do sistema. Para a criação e execução

do plano, especificamos algumas políticas de balanceamento de carga que utilizam as relações de desempenho, determinadas anteriormente. Estas políticas são implementadas no programa que implementa a aplicação.

As estratégias de balanceamento buscam minimizar as diferenças entre os tempos de execução dos processos distribuídos e sincronizar o término das tarefas. Para isso, a quantidade de trabalho atribuído a cada processo é definido de acordo com a capacidade de processamento da máquina onde o mesmo será executado. A indicação da capacidade é extraída através dos índices de desempenhos relativos.

Algumas estratégias realizam a divisão e distribuição do trabalho por fases de processamento. Isso permite que o planejamento inicial seja modificado ao longo da execução do programa e ajustado às alterações que o ambiente apresentar. Essas estratégias alteram a distribuição das cargas utilizando informações sobre a carga computacional média ou instantânea das máquinas, medidas através de um monitor de desempenho, especificado no processo PEM da metodologia.

4.2.4 PEM

O PEM (***PE****rformance* ***M****onitor*) é um processo da metodologia que especifica o uso de um monitor de desempenho para acompanhar o comportamento das aplicações e da carga computacional das máquinas do sistema.

No código do programa implementado inserimos chamadas para o uso das funções deste monitor. Basicamente, o monitor pode coletar informações sobre o uso do processador, da memória, do disco e da rede de cada uma das máquinas. Além disso, baseado no tempo de resposta do processo escravo e no tamanho do trabalho processado em cada fase da execução é possível determinar a capacidade atual de processamento das máquinas. Assim, a cada nova distribuição de carga, o processo mestre avalia as informações do monitor e decide se altera ou não o plano inicial de balanceamento.

Informações como o tamanho da fila de processos e a taxa de paginação do sistema também podem ser analisadas. Se durante o processamento da aplicação uma máquina tem sua carga computacional alterada, a estratégia de balanceamento tenta compensar e equilibrar as cargas do sistema nas próximas fases da distribuição. Assim, é possível realizar um ajuste dinâmico e adaptativo em relação à distribuição das cargas computacionais, reagindo, em tempo de execução, às modificações do ambiente. Portanto, as informações do monitor podem ser utilizadas por estratégias de balanceamento de carga.

4.3 O Modelo Gráfico DP*Graph^{++}

A metodologia PEMPIs-Het utiliza uma modelagem gráfica para representar a estrutura estática dos programas MPI e facilitar a análise do código fonte das aplicações. Ao

estender as atividades para ambientes heterogêneos, foi necessário modificar a simbologia original do modelo gráfico DP*Graph^{+} (MIDORIKAWA; OLIVEIRA; LAINE, 2004, 2005), e adicionar elementos que pudessem representar aspectos relacionados à heterogeneidade do ambiente. Com isso, o conjunto original de símbolos do modelo gráfico DP*Graph^{+} foi alterado e outros elementos foram criados, conforme pode ser visto na Figura 4.2.

O novo conjunto de símbolos aumentou a representatividade do modelo gráfico, diferenciando as estruturas do código que podem ser influenciadas pela heterogeneidade do ambiente de teste. A transferência de dados por uma rede e o processamento intensivo de instruções são exemplos de operações que podem ter o desempenho influenciado por características do *hardware*. Entretanto, algumas estruturas não sofrem influência das características do sistema e, portanto, não exigem dupla representação gráfica, como é o caso dos condicionais. Basicamente, diferenciamos os símbolos usando ou não o preenchimento de cor.

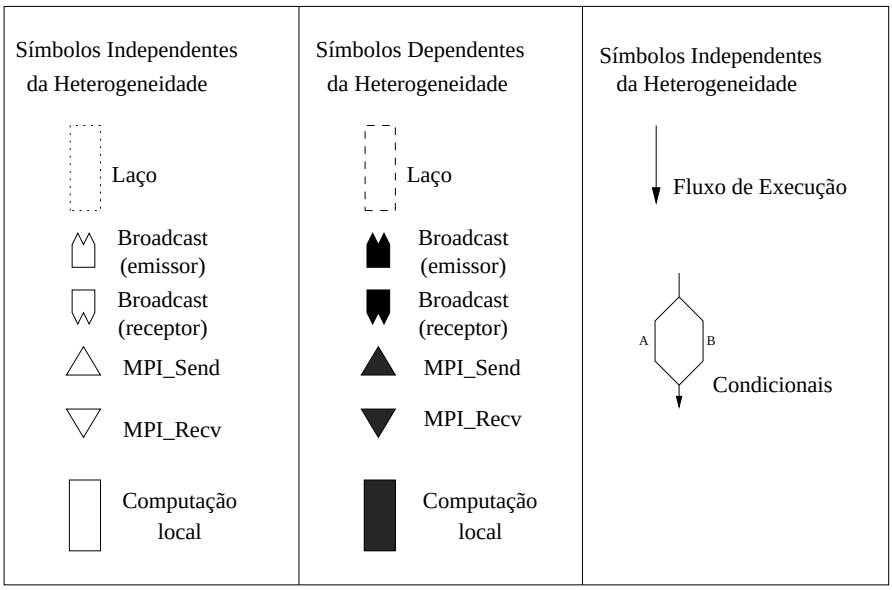

Figura 4.2: Simbologia do modelo gráfico DP*Graph^{++}.

A Figura 4.3 mostra um exemplo do modelo gráfico DP*Graph^{++} em um ambiente distribuído homogêneo. A figura representa o código fonte de uma aplicação, denominada *Heat*, que utiliza o modelo mestre-escravo na solução do problema. Essa aplicação simula a propagação de calor em uma superfície metálica. Na figura é possível identificar os trechos de computação seqüencial (representados pelos retângulos preenchidos na cor cinza), as estruturas de repetição (retângulos com bordas pontilhadas) e as primitivas de comunicação (*MPI_Send* e *MPI_Recv*) representadas por triângulos com a base para baixo e para cima, respectivamente. As etiquetas associadas à representação das primitivas de comunicação indicam a quantidade de *bytes* transferidos na mensagem. Já as etiquetas das estruturas de repetição mostram o número de iterações dos laços.

Com uma simples análise da Figura 4.3 é possível compreender o modelo da estrutura

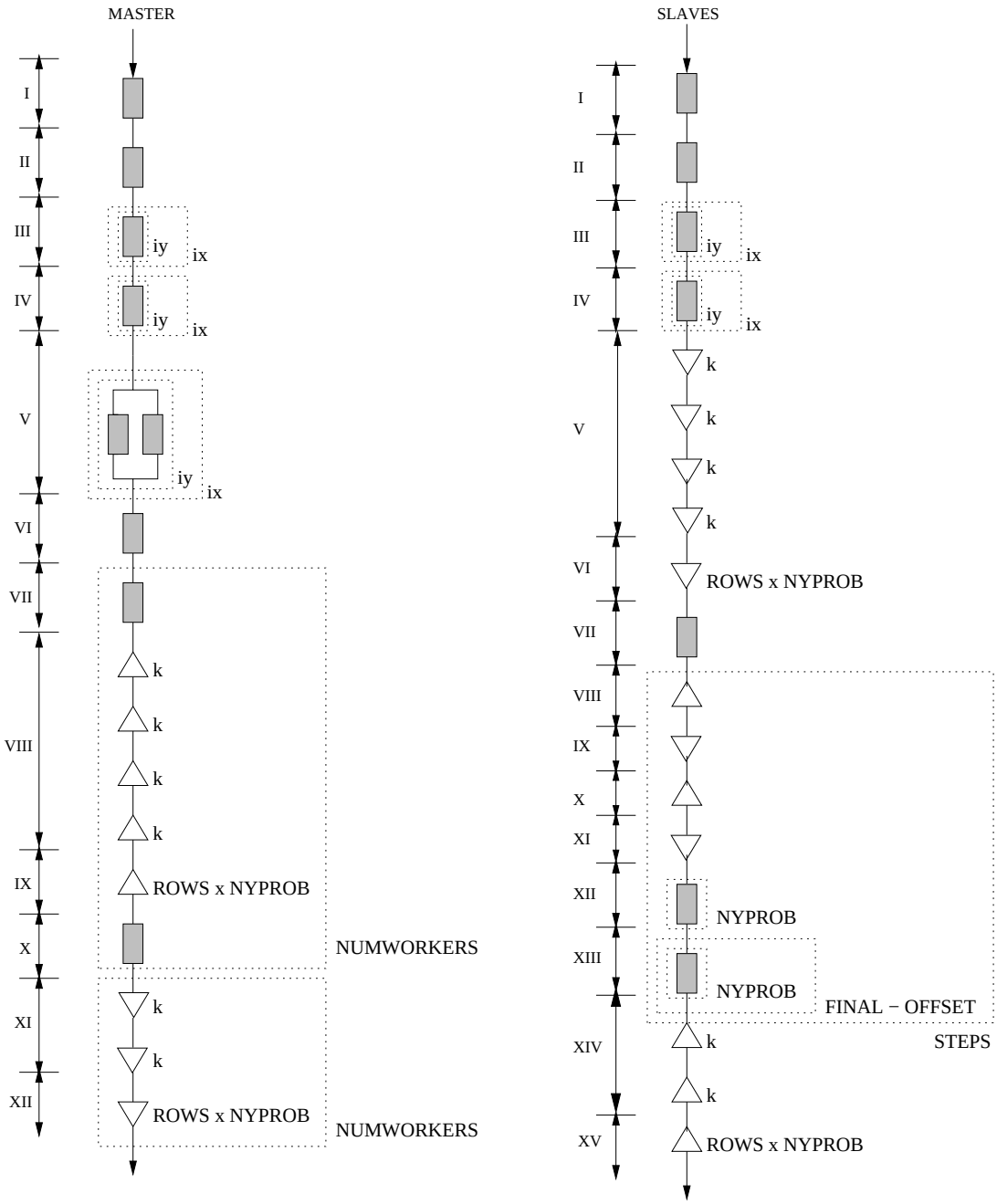

Figura 4.3: O modelo gráfico do programa Heat.

de solução do programa, identificar as trocas de mensagens existentes, eventuais sincronismos de execução e os trechos de processamento, sem mesmo conhecer o código fonte do programa. Entretanto, para que seja possível comparar o modelo gráfico e o código fonte do programa, resolvemos apresentar, a seguir, o código da solução do problema.

```
1  #include <stdio.h>
2  #include <mpi.h>
3  #include <math.h>
4  #define EPSILON 0.00001
5  #define N 180
6  #define time_steps 100
7
8  int main (int argc, char *argv[])
9   {
10   int i,j,k,p;
11   int myid, numprocs;
12   int from,to;
13   int step;
14   int nlinhas;
15   double time;
16   double eps, enew, global_eps;
17   double time_max = 3.0;
18   double alpha = 0.06;
19   double dx = 1.0/N;
20   double dy = 1.0/time_steps;
21   double dt = time_max/time_steps;
22   double dxinv = 1.0/dx;
23   double dyinv = 1.0/dy;
24   double dtinv = 1.0/dt;
25   double divinv = 1.0/(dtinv + 2 * alpha * (dxinv * dxinv + dyinv * dyinv));
26   double **t,*tS;
27   double **told,*toldS;
28   double **tbuff,*tbuffS;
29   double **tstep,*tstepS;
30   double *sendR1,*sendR2;
31   double *recvR1,*recvR2;
32   double minval, maxval;
33   long clock(),cputime;
34   char fname[40];
35   FILE *out;
36   MPI_Status status;
37
38   MPI_Init(&argc, &argv);
39   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
40   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
41
42   if (((N % numprocs) == 0) && (numprocs > 1))
43    {
44     nlinhas = N/numprocs;
45
46     if (myid == 0)
47      {
48       clock();
49       tbuffS = (double *)  malloc(((nlinhas)+2) * N * sizeof(double));
50       tstepS = (double *)  malloc(N * N * sizeof(double));
51       tbuff =  (double **) malloc(((nlinhas)+2) * sizeof(double *));
52       tstep =  (double **) malloc(N * sizeof(double *));
53
```

```
54          for (i=0; i<((nlinhas)+2); i++)
55           tbuff[i] = &tbuffS[i*N];
56          for (i=0; i<N; i++)
57           tstep[i] = &tstepS[i*N];
58         }
59        //allocate enough memory for each fraction of the mesh
60        tS =     (double *)  malloc(((nlinhas)+2) * N * sizeof(double));
61        toldS = (double *)  malloc(((nlinhas)+2) * N * sizeof(double));
62        t =      (double **) malloc(((nlinhas)+2) * sizeof(double *));
63        told =  (double **) malloc(((nlinhas)+2) * sizeof(double *));
64        printf("rows = %d\n", (nlinhas)+2);
65        for (i=0; i<(nlinhas)+2; i++)
66         {
67          t[i] = &tS[i*N];
68          told[i] = &toldS[i*N];
69         }
70        // set initial boundary conditions
71        for (i=0; i<(nlinhas)+2; i++)
72         for (j=0; j<N; j++)
73          told[i][j] = 0.0;
74        // for all time steps
75        for (step = 1; step <= time_steps; step++)
76         {
77          time = step * (time_max/time_steps);
78          // reset top boundary condition each timestep
79          if (myid == 0)
80           for (j=0; j<N; j++)
81             told[0][j] = 2.0 * sin(time);
82          do
83           {
84            // exchange the rows you need to share with other processes
85            if (myid == 0)
86             {
87              //allocate memory for message arrays
88              sendR1 = (double *) malloc(N * sizeof(double));
89              recvR1 = (double *) malloc(N * sizeof(double));
90              //send to my last computed row to myid+1
91              for(j=0; j<N; j++) sendR1[j]=told[(nlinhas)][j];
92              MPI_Send(sendR1,N,MPI_DOUBLE,myid+1,0,MPI_COMM_WORLD);
93              //receive my last row from myid+1
94              MPI_Recv(recvR1,N,MPI_DOUBLE,myid+1,0,MPI_COMM_WORLD,&status);
95              for(j=0; j<N; j++) told[(nlinhas)+1][j]=recvR1[j];
96              free(sendR1);
97              free(recvR1);
98             }
99            else if ((myid > 0) && (myid < (numprocs-1)))
100            {
101             //allocate memory for message arrays
102             sendR1 = (double *) malloc(N * sizeof(double));
103             recvR1 = (double *) malloc(N * sizeof(double));
104             sendR2 = (double *) malloc(N * sizeof(double));
105             recvR2 = (double *) malloc(N * sizeof(double));
106             //send my first computed row to myid-1
107             for(j=0; j<N; j++) sendR1[j]=told[1][j];
108             MPI_Send(sendR1,N,MPI_DOUBLE,myid-1,0,MPI_COMM_WORLD);
109             //send my last computed row to myid+1
110             for(j=0; j<N; j++) sendR2[j]=told[(nlinhas)][j];
111             MPI_Send(sendR2,N,MPI_DOUBLE,myid+1,0,MPI_COMM_WORLD);
112             //receive my first row from myid-1
```

```
113             MPI_Recv(recvR1,N,MPI_DOUBLE,myid-1,0,MPI_COMM_WORLD,&status);
114             for(j=0; j<N; j++) told[0][j]=recvR1[j];
115             //receive my last row from myid+1
116             MPI_Recv(recvR2,N,MPI_DOUBLE,myid+1,0,MPI_COMM_WORLD,&status);
117             for(j=0; j<N; j++) told[(nlinhas)+1][j]=recvR2[j];
118             free(sendR1);
119             free(sendR2);
120             free(recvR1);
121             free(recvR2);
122            }
123           if (myid == (numprocs-1))
124            {
125             //allocate memory for message arrays
126             sendR1 = (double *) malloc(N * sizeof(double));
127             recvR1 = (double *) malloc(N * sizeof(double));
128             //send my first computed row myid-1
129             for(j=0; j<N; j++) sendR1[j]=told[1][j];
130             MPI_Send(sendR1,N,MPI_DOUBLE,myid-1,0,MPI_COMM_WORLD);
131             //receive my first row myid-1
132             MPI_Recv(recvR1,N,MPI_DOUBLE,myid-1,0,MPI_COMM_WORLD,&status);
133             for(j=0; j<N; j++) told[0][j]=recvR1[j];
134             free(sendR1);
135             free(recvR1);
136            }
137           eps = 0.0;
138           for (i=1; i<=(nlinhas); i++)
139             for (j=1; j<(N-1); j++)
140             t[i][j]=((told[i][j+1]+told[i][j-1])*alpha*dyinv*dyinv+(told[i+1][j]+told[i-1][
                    j])*alpha*dxinv*dxinv+(told[i][j]*dtinv))*divinv;
141           for (i=1; i<=(nlinhas); i++)
142            {
143             for (j=1; j<(N-1); j++)
144              {
145               enew = fabs(t[i][j] - told[i][j]);
146               if (enew > eps) { eps = enew; }
147              }
148            }
149           for (i=0; i<((nlinhas)+2); i++)
150            for (j=0; j<N; j++)
151             told[i][j] = t[i][j];
152           MPI_Allreduce(&eps,&global_eps,1,MPI_DOUBLE,MPI_MAX,MPI_COMM_WORLD);
153          }
154         while(global_eps > EPSILON);
155         // Dump raster data to a file
156         // collect told segments from each process
157         if (myid == 0)
158          {
159           k=0;
160           for (i=1; i<=(nlinhas); i++)
161            {
162             for (j=0; j<N; j++)
163              tstep[k][j] = told[i][j];
164             k++;
165            }
166           for (p=1; p<numprocs; p++)
167            {
168             MPI_Recv(tbuff[0],N*((nlinhas)+2),MPI_DOUBLE,p,0,MPI_COMM_WORLD,&status);
169             for (i=1; i<=(nlinhas); i++)
170              {
```

```
171               for (j=0; j<N; j++)
172                tstep[k][j] = tbuff[i][j];
173               k++;
174              }
175            }
176           minval = 0.0;
177           maxval = 0.0;
178           for (i=0; i<N; i++)
179            {
180             for (j=0; j<N; j++)
181              {
182               if (tstep[i][j] < minval) { minval = tstep[i][j]; }
183               if (tstep[i][j] > maxval) { maxval = tstep[i][j]; }
184              }
185            }
186           sprintf(fname,"Output\\heat%03d.raw",step);
187           out = fopen(fname,"w+b");
188           for (i=0; i<N; i++)
189             for (j=0; j<N; j++)
190               fprintf(out,"%c",(int)(((tstep[i][j]-minval)*255.0)/(maxval - minval)));
191           fclose(out);
192           printf("Time step: %d\r",step);
193          }
194         else
195          {
196           // send my told segment back to the master
197           MPI_Send(told[0],N*((nlinhas)+2),MPI_DOUBLE,0,0,MPI_COMM_WORLD);
198          }
199         MPI_Barrier(MPI_COMM_WORLD);
200        } // for all time steps
201       free(t);
202       free(tS);
203       free(told);
204       free(toldS);
205       if (myid == 0)
206        {
207         free(tbuff);
208         free(tbuffS);
209         free(tstep);
210         free(tstepS);
211         cputime = clock();
212         printf("%d time steps in %.2f seconds\n",step-1,cputime/1.0e+3);
213        }
214       } // if ((N % numprocs) == 0))
215      else
216       if (myid == 0)
217        {
218         if (numprocs < 2)
219          printf("ERROR: Must have at least 2 processes\n");
220         if ((N % numprocs) != 0)
221          printf("ERROR: Grid must be evenly divisible by the number of processes\n");
222        }
223      MPI_Finalize();
224    }
```

O principal objetivo do DP*Graph^{++} é facilitar a análise da complexidade do programa. Entretanto, o modelo gráfico também pode ser utilizado com os seguintes propósitos:

- ilustrar a organização estática e o modelo arquitetural do código fonte do programa

modelado;

- ajudar a determinar o modelo de desempenho teórico da aplicação;
- identificar pontos de sincronismo entre processos;
- permitir comparações qualitativas entre diferentes soluções de um problema;
- auxiliar estudos e análises específicas para paralelização de soluções distribuídas.

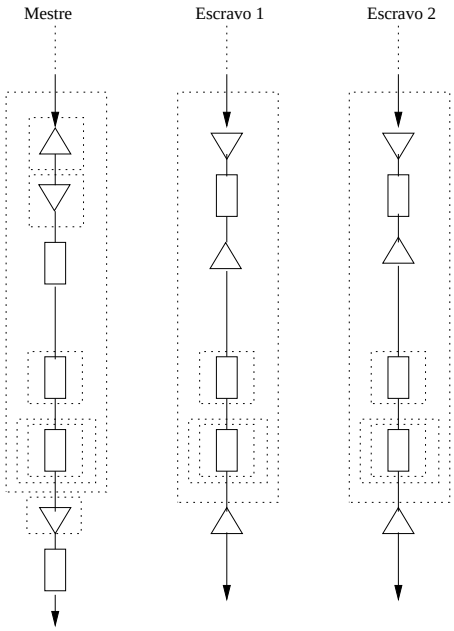

Figura 4.4: Sistema homogêneo.

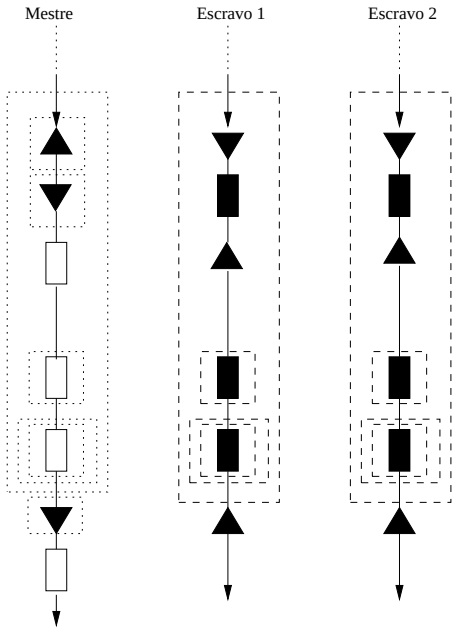

Figura 4.5: Sistema heterogêneo.

Para ilustrar as diferenças na representação do DP*Graph^{++} em ambientes homogêneos e heterogêneos, apresentamos as Figuras 4.4 e 4.5. Esse exemplo mostra trechos de um simples programa que adota o modelo mestre-escravo na organização da solução. Inicialmente, o processo mestre envia uma parte do trabalho para cada um dos escravos. Esses processos recebem, realizam o processamento e devolvem o resultado parcial ao mestre. As figuras têm a intenção de ilustrar os trechos do programa que tem o comportamento influenciado pelas características da máquina onde o processo está sendo executado. Em um ambiente heterogêneo, o comportamento individual de cada processo está relacionado à capacidade computacional da máquina utilizada na execução do programa.

4.4 Um Passo-a-Passo da Metodologia

As atividades contempladas pelo processo definido na metodologia PEMPIs-Het podem ser resumidas e compreendidas através do conjunto de passos destacado a seguir. Este passo-a-passo descreve, ordenadamente, não só a seqüência das atividades envolvidas no

processo de modelagem e predição de desempenho, mas também as ações envolvidas nas atividades relacionadas ao balanceamento de carga do sistema.

1. pode ser criado um arquivo com a descrição de cada uma das máquinas do sistema ou somente daquelas utilizadas na execução da aplicação. Nesta descrição, informações como a quantidade de processadores ou núcleos, a freqüência do(s) processador (es), a quantidade de memória cache e principal, o tipo de *interface* de rede utilizada em cada máquina e o sistema operacional são listadas;

2. o programa MPI é instrumentado com monitores de tempo. A inserção dos monitores de *software* acontece para que se possa avaliar e medir o desempenho de alguns trechos do programa. Os monitores podem coletar informações sobre o tempo de execução de instruções de processamento e o tempo gasto em comunicação entre os processos. Essas informações são utilizadas durante a elaboração dos modelos analíticos de desempenho;

3. a partir do código fonte do programa, é elaborado o modelo gráfico DP*Graph^{++}. A simbologia desse modelo gráfico auxilia a modelagem da aplicação e facilita o entendimento estrutural do código do programa, conforme explicado na Seção 4.3;

4. com a ajuda do modelo gráfico DP*Graph^{++} estudamos a complexidade algorítmica da aplicação, e o modelo teórico de desempenho pode ser especificado;

5. as configurações do ambiente de teste são definidas. Neste momento criamos um arquivo com o nome das máquinas que serão utilizadas pelo LAM (SQUYRES; LUMSDAINE, 2003; BURNS; DAOUD; VAIGL, 1994) na execução do programa MPI. As configurações de p e n são planejadas para que o estudo de desempenho seja realizado. Normalmente, escrevemos um programa *shell script* para controlar a execução do programa nos casos de testes planejados;

6. os testes experimentais planejados são realizados. Cada caso de teste é avaliado no mínimo 30 vezes e os tempos medidos são armazenados em arquivos de *log*;

7. os tempos medidos são selecionados, através de uma função de avaliação estatística, e as anomalias observadas são descartadas. Esta atividade é fundamental para a qualidade dos modelos de predição, pois permite aumentar a confiabilidade dos valores selecionados. A partir desses dados é calculado um valor médio para representar as grandezas avaliadas;

8. um método de ajuste de curvas é aplicado sobre os tempos médios calculados, e os modelos analíticos de predição de desempenho são elaborados. Esses modelos, geralmente, estão em função da quantidade de processos (p) e do tamanho do problema (n). Para cada tipo de máquina do sistema é elaborado um modelo individual,

capaz de caracterizar o comportamento da aplicação naquele nó de processamento específico;

9. a partir dos modelos analíticos individuais, é criado um metamodelo para a aplicação. Esse metamodelo caracteriza o comportamento do programa MPI no ambiente formado pelas máquinas listadas no arquivo de máquinas;

10. estimativas para o desempenho da aplicação podem ser calculadas, variando p e n. Portanto, além de prever o comportamento da aplicação em situações não analisadas, o metamodelo permite realizar estudos sobre o comportamento da aplicação no ambiente;

11. relações de desempenho são obtidas através dos modelos analíticos individuais. Atribuindo valores para p e n, é possível determinar quantas vezes uma máquina é mais rápida ou mais lenta que a outra na execução da aplicação modelada. Esses índices são utilizados no planejamento das atividades de distribuição de carga do sistema;

12. um plano inicial de execução é elaborado para a aplicação. Esse plano utiliza as relações de desempenho para determinar a quantidade de trabalho ideal que cada uma das máquinas deve receber durante a execução do programa. A execução da aplicação é dividida em fases de processamento e, a cada fase, uma quantidade de trabalho diferente pode ser enviada aos processos que colaboram na solução do problema;

13. o desempenho dos processos distribuídos é monitorado em tempo real, através das funções implementadas no PEM. O objetivo é avaliar se a distribuição de carga planejada está adequada à situação de momento do sistema. Utilizando as informações coletadas pelo monitor, é possível, dependendo da estratégia de balanceamento de carga, reajustar a distribuição do trabalho para melhorar o desempenho da aplicação.

4.5 Modelagem de Programas Paralelos

Um modelo analítico de desempenho, como já foi comentado, pode ser definido através de uma equação matemática capaz de representar, em função de alguns parâmetros, o comportamento de uma aplicação em um sistema qualquer. Na metodologia PEMPIs-Het, os modelos analíticos são elaborados para modelar o desempenho de programas MPI em sistemas distribuídos, homogêneos ou heterogêneos. A estimativa do tempo de execução dada pelos modelos também é aplicada em atividades relacionadas à distribuição da carga computacional.

A precisão dos modelos analíticos está diretamente relacionada à quantidade de informações consideradas e ao comportamento da aplicação modelada. Entretanto, esse comportamento pode ser modificado em função do tamanho do problema, por exemplo. Assim, utilizar um método de ajuste de curvas para elaborar um único modelo de predição, considerando qualquer intervalo para p e n, pode não ser a melhor estratégia.

Uma alternativa possível, que pode ser utilizada para melhorar a precisão dos modelos, é modelar o problema por intervalos. Nessa estratégia, a aplicação não é representada mais por um único modelo, e sim por algumas funções que permitem prever o tempo de execução do programa em função de p e n. Dependendo do valor de n, usa-se um modelo ou outro.

Outra abordagem utilizada em atividades de predição de desempenho consiste em definir estimativas por intervalos de mínimo e máximo (SCHOPF; BERMAN, 1997). Nesse caso, a previsão do tempo não é um valor pontual, e sim um intervalo aceitável.

Na metodologia PEMPIs-Het, todo modelo de predição é elaborado com base na complexidade do código fonte do programa. Por exemplo, se um programa paralelo Px apresentar uma complexidade algorítmica $O(\frac{n^3}{p})$, o modelo analítico teórico para estimar o desempenho dessa aplicação pode ser escrito como:

$$\delta(p,n) = \frac{an^3}{p} + \frac{bn^2}{p} + \frac{cn}{p} + d \tag{4.1}$$

Nesse modelo, $\delta(p,n)$ é elaborado para expressar não só o tempo gasto com processamento de instruções (T_{CPU}), mas também com a transmissão de mensagens pelas estruturas de interconexão (T_{COM}). Dessa forma, a equação que determina o tempo total de execução (T_{total}) pode ser reescrita da seguinte forma:

$$T_{total} = \delta(p,n) \tag{4.2}$$

$$T_{total} = \sum T_{CPU} + \sum T_{COM} + \alpha \tag{4.3}$$

Nessa equação, a variável α representa a influência, no tempo de execução, de eventuais elementos não tratados pelo modelo de desempenho mas que podem comprometer a precisão das estimativas. Sobrecargas na estrutura de interconexão e contenção no tráfego da rede são exemplos de situações que podem alterar o comportamento das aplicações e dos modelos. Nesses casos, a estimativa gerada pelo modelo fica prejudicada.

4.6 Aplicações para os Modelos de Predição

Na metodologia PEMPIs-Het, os modelos analíticos são aplicados em duas atividades distintas: predição de desempenho e balanceamento de carga. Nas atividades relacionadas à predição de desempenho, é possível estimar o tempo de execução individual de cada um dos processos ou o tempo total de execução da aplicação no sistema.

Essas análises podem ser utilizadas para tomada de decisão de escalonadores de processos de modo a otimizar o uso dos recursos computacionais disponíveis. Além disso, a modelagem também pode ser realizada para comparar modelos de soluções paralelas, como mestre-escravo, cliente-servidor, *peer-to-peer*, entre outros. O objetivo dessa comparação é avaliar e decidir qual estrutura de solução é capaz de maximizar o desempenho da aplicação.

Já o balanceamento de carga está relacionado ao planejamento da divisão e distribuição da carga computacional entre as máquinas. A divisão do trabalho é feita em função das estimativas de tempo que o modelo analítico calcula para a execução do programa em cada máquina do ambiente. A partir dessas estimativas, é possível determinar a capacidade de processamento relativa entre as máquinas e, conseqüentemente, a quantidade de trabalho adequada que cada uma deve receber durante a execução dos programas.

4.7 Relações de Desempenho

Em sistemas distribuídos, a distribuição adequada da carga computacional é fundamental para o desempenho das aplicações paralelas. A existência, simplesmente, de máquinas com grande capacidade de processamento não é capaz de garantir o bom desempenho dos programas e a eficiência dos sistemas.

Todas as estratégias de balanceamento de carga, especificadas neste trabalho e avaliadas durante os testes experimentais, fazem uso de relações de desempenho para o planejamento da divisão e distribuição do trabalho. Essas relações são obtidas a partir dos modelos analíticos, elaborados para representar o comportamento das aplicações e prever o tempo de execução dos programas. Esses modelos, gerados de acordo com o processo especificado na metodologia PEMPIs-Het, são fundamentais também para a caracterização da capacidade de processamento individual das máquinas. Portanto, antes de apresentarmos as estratégias é necessário definir as relações de desempenho e compreender como elas são obtidas.

Definição 1: Um sistema distribuído heterogêneo S é composto por um conjunto de máquinas de diferentes tipos. Seja $M = \{M_1, M_2, ..., M_m\}$ o conjunto dos m tipos de máquinas do sistema S. Definem-se, então, p_i a quantidade de máquinas do tipo $M_i \in M$ e f_i a fração do trabalho total atribuída a cada uma das máquinas do tipo M_i. Assim, a

relação que define a distribuição de carga entre as máquinas de S pode ser dada por:

$$\sum_{i=1}^{m} p_i \times f_i = 1 \tag{4.4}$$

A partir dessa expressão, é possível derivar a fração do trabalho total que cada uma das máquinas do conjunto P deve receber em função do trabalho atribuído às demais máquinas, como mostrado a seguir:

$$f_1 = \frac{1 - p_2 \times f_2 - p_3 \times f_3 - ... - p_m \times f_m}{p_1} \tag{4.5}$$

$$f_2 = \frac{1 - p_1 \times f_1 - p_3 \times f_3 - ... - p_m \times f_m}{p_2} \tag{4.6}$$

$$\vdots$$

$$f_m = \frac{1 - p_1 \times f_1 - p_2 \times f_2 - ... - p_{m-1} \times f_{m-1}}{p_m} \tag{4.7}$$

Para simplificar as análises, consideramos uma aceleração (*speedup*) teórica linear durante o desenvolvimento das relações. Dessa forma, o tempo total de execução de uma aplicação A para um problema de tamanho n, utilizando p máquinas idênticas do tipo q, pode ser expresso da seguinte forma:

$$\delta_q^A(p, n) = \frac{1}{p} \times \delta_q^A(1, n) \tag{4.8}$$

Essa equação define que o tempo total de execução de uma aplicação distribuída, em um sistema homogêneo constituído de p máquinas do tipo q, é $\frac{1}{p}$ do tempo necessário para executar a mesma aplicação A em uma única máquina q.

Uma vez definida a fração de trabalho que será atribuída a cada uma das máquinas e conhecendo o modelo analítico que expressa o tempo de execução seqüencial da aplicação, é possível estimar quanto tempo cada tipo de máquina $M_i \in M$ levará para executar a fração de trabalho que lhe foi enviada, da seguinte forma:

$$T_{p_i}^A(f_i, n) = f_i \times \delta_{p_i}^A(1, n) \tag{4.9}$$

O intuito dessa distribuição é fazer com que a seguinte relação seja satisfeita:

$$f_i \times \delta_{p_i}^A(1, n) = f_j \times \delta_{p_j}^A(1, n), \quad \forall \quad p_i, p_j \tag{4.10}$$

Essa relação de igualdade representa o caso ideal, pois independente da capacidade das máquinas nosso objetivo é fazer com que todos os processos gastem o mesmo tempo na execução do trabalho recebido. Dessa forma, nenhum atraso é gerado em eventuais sincronizações e a solução final do problema é alcançada de maneira eficiente e otimizada.

Na prática, nossa intenção é minimizar ao máximo as diferenças entre os tempos de execução de cada um dos processos, pois a situação ideal dificilmente é alcançada. A busca por este sincronismo perfeito evita que os processos que estão sendo executados em máquinas de menor capacidade computacional atrasem a finalização do trabalho.

Definição 2: Considere $\delta_i^A(p,n)$ e $\delta_j^A(p,n)$ os modelos analíticos de desempenho capazes de estimar o tempo de execução de uma aplicação A em p máquinas do tipo M_i e M_j, respectivamente. Assim, a capacidade de desempenho relativa g, entre os tipos de máquinas M_i e M_j, é expressa como:

$$g_{i,j}(p,n) \triangleq \frac{\delta_i^A(p,n)}{\delta_j^A(p,n)} \tag{4.11}$$

A relação $g_{i,j}(p,n)$ determina quantas vezes a máquina i é mais rápida ou mais lenta em relação à máquina j ao executar a aplicação A com um trabalho de tamanho n.

Usando as equações de 4.5 a 4.7, 4.10 e 4.11, é possível determinar a fração de trabalho adequada a cada tipo de máquina $M_i \in M$, conforme apresentado a seguir:

$$f_1 = \frac{1}{p_1 + p_2 g_{1,2}(p,n) + ... + p_m g_{1,m}(p,n)} \tag{4.12}$$

$$f_2 = \frac{g_{1,2}(p,n)}{p_1 + p_2 g_{1,2}(p,n) + ... + p_m g_{1,m}(p,n)} \tag{4.13}$$

$$\vdots$$

$$f_m = \frac{g_{1,m}(p,n)}{p_1 + p_2 g_{1,2}(p,n) + ... + p_m g_{1,m}(p,n)} \tag{4.14}$$

Já o tempo de execução de um processo p_i é estimado ao aplicar as equações numeradas de 4.12 a 4.14 na equação 4.9, da seguinte forma:

$$T_i^A(f_i,n) = \frac{g_{1,i}(p,n)}{p_1 + p_2 g_{1,2}(p,n) + ... + p_m g_{1,m}(p,n)} \times \delta_i^A(1,n) \tag{4.15}$$

Considerando a igualdade apresentada na fórmula 4.10, esse modelo analítico de desempenho pode ser generalizado para estimar o tempo de execução de um processo sendo executado em qualquer tipo de máquina do conjunto M.

4.8 Exemplo Teórico

Nessa seção, apresentamos um exemplo teórico para ilustrar o desenvolvimento das equações e a geração das relações de desempenho. Para simplificar, consideramos um ambiente distribuído S com somente três tipos diferentes de máquinas: $M = \{t, b, i\}$. A capacidade de processamento das máquinas não precisa ser definida, pois os modelos estarão representados algebricamente. Assim, o conjunto formado pelos processos que são executados

em cada tipo de máquina do conjunto M é definido como $P = \{p_t, p_b, p_i\}$. Nesse sistema, uma aplicação A terá seu processamento dividido entre cada um desses processos. Assim, a equação 4.4, que define a distribuição de trabalho entre as máquinas, pode ser expressa da seguinte forma:

$$p_t \times f_t + p_b \times f_b + p_i \times f_i = 1 \tag{4.16}$$

A partir dessa equação, é possível determinar a porcentagem de trabalho que cada processo deverá receber durante a execução do programa. Para isso, é preciso isolar as variáveis que permitem caracterizar essas frações, do seguinte modo:

$$f_t = \frac{1 - p_b \times f_b - p_i \times f_i}{p_t} \tag{4.17}$$

$$f_b = \frac{1 - p_t \times f_t - p_i \times f_i}{p_b} \tag{4.18}$$

$$f_i = \frac{1 - p_t \times f_t - p_b \times f_b}{p_i} \tag{4.19}$$

O objetivo teórico da divisão do trabalho é igualar os tempos de execução de cada um dos processos do conjunto P. Portanto, com as equações f_t, f_b e f_i e a expressão de igualdade 4.10, é possível derivar as seguintes relações entre as máquinas t, b e i:

$$f_t \times \delta_t(p, n) = f_b \times \delta_b(p, n) \tag{4.20}$$

$$f_t \times \delta_t(p, n) = f_i \times \delta_i(p, n) \tag{4.21}$$

$$f_b \times \delta_b(p, n) = f_i \times \delta_i(p, n) \tag{4.22}$$

Agora, para determinar os respectivos valores de f_t, f_b e f_i é preciso desenvolver cada um dos casos destacados a seguir.

Caso 1: $\mathbf{f_t \times \delta_t(p, n) = f_b \times \delta_b(p, n)}$

Desenvolvendo essa expressão e substituindo f_t pela relação 4.17, temos:

$$\begin{aligned}
\frac{(1 - p_b f_b - p_i f_i)}{p_t} \times \delta_t(p, n) &= f_b \delta_b(p, n) \\
\frac{(\delta_t(p, n) - p_b f_b \delta_t(p, n) - p_i f_i \delta_t(p, n))}{p_t} &= f_b \delta_b(p, n) \\
\delta_t(p, n) - p_b f_b \delta_t(p, n) - p_i f_i \delta_t(p, n) &= p_t f_b \delta_b(p, n) \\
\delta_t(p, n) - p_b f_b \delta_t(p, n) - p_t f_b \delta_b(p, n) &= p_i f_i \delta_t(p, n)
\end{aligned}$$

$$f_i = \frac{\delta_t(p,n) - p_b f_b \delta_t(p,n) - p_t f_b \delta_b(p,n)}{p_i \delta_t(p,n)} \tag{4.23}$$

Caso 2: $\mathbf{f_t \times \delta_t(p,n) = f_i \times \delta_i(p,n)}$

Novamente, desenvolvemos a expressão e substituímos f_t pela relação 4.17, conforme segue:

$$\frac{(1 - p_b f_b - p_i f_i)}{p_t} \times \delta_t(p,n) = f_i \delta_i(p,n)$$

$$\frac{(\delta_t(p,n) - p_b f_b \delta_t(p,n) - p_i f_i \delta_t(p,n))}{p_t} = f_i \delta_i(p,n)$$

$$\delta_t(p,n) - p_b f_b \delta_t(p,n) - p_i f_i \delta_t(p,n) = p_t f_i \delta_i(p,n)$$

$$\delta_t(p,n) - p_b f_b \delta_t(p,n) = (p_i \delta_t(p,n) + p_t \delta_i(p,n)) f_i$$

$$f_i = \frac{\delta_t(p,n) - p_b f_b \delta_t(p,n)}{p_i \delta_t(p,n) + p_t \delta_i(p,n)} \tag{4.24}$$

Agora, igualamos f_i do Caso 1 e f_i do Caso 2, obtendo a seguinte relação para f_b:

$$\frac{\delta_t(p,n) - p_b f_b \delta_t(p,n) - p_t f_b \delta_b(p,n)}{p_i \delta_t(p,n)} = \frac{\delta_t(p,n) - p_b f_b \delta_t(p,n)}{p_i \delta_t(p,n) + p_t \delta_i(p,n)}$$

$$(\delta_t(p,n) - p_b f_b \delta_t(p,n) - p_t f_b \delta_b(p,n))(p_i \delta_t(p,n) + p_t \delta_i(p,n)) = p_i \delta_t(p,n)^2 - p_i p_b f_b \delta_t(p,n)^2$$

$$f_b = \frac{\delta_i(p,n)\delta_t(p,n)}{p_i \delta_t(p,n)\delta_b(p,n) + p_b \delta_i(p,n)\delta_t(p,n) + p_t \delta_i(p,n)\delta_b(p,n)} \tag{4.25}$$

Depois de encontrar as expressões para as frações de trabalho, é preciso também determinar as relações que definem a capacidade de processamento relativa entre cada uma das máquinas do sistema. Essas relações são geradas através dos modelos analíticos elaborados para representar o comportamento da aplicação A em cada uma das máquinas disponíveis. Assim, é preciso desenvolver a equação 4.11 de modo a obter as seguintes

expressões:

$$g_{t,b}(p,n) = \frac{\delta_t^A(p,n)}{\delta_b^A(p,n)} \tag{4.26}$$

$$g_{t,i}(p,n) = \frac{\delta_t^A(p,n)}{\delta_i^A(p,n)} \tag{4.27}$$

$$g_{b,i}(p,n) = \frac{\delta_b^A(p,n)}{\delta_i^A(p,n)} \tag{4.28}$$

Com essas relações, é possível determinar o tempo de execução de cada um dos processos ao executar uma fração do trabalho total de tamanho n. Por exemplo, avaliando o processo que vai ser executado na máquina do tipo b, a fração de trabalho e o modelo analítico que representa o tempo de execução de uma parte da aplicação A, podem ser escritos da seguinte forma:

$$f_b = \frac{g_{t,b}(p,n)}{p_t + p_b g_{t,b}(p,n) + p_i g_{t,i}(p,n)} \tag{4.29}$$

Substituindo essa equação na fórmula 4.9, geramos o seguinte modelo para o tempo de execução:

$$T_b^A(f_b,n) = \frac{g_{t,b}(p,n)}{p_t + p_b g_{t,b}(p,n) + p_i g_{t,i}(p,n)} \times \delta_b^A(1,n) \tag{4.30}$$

Derivações similares são feitas para f_t e f_i, a fim de determinar o tempo de execução dos processos que estão sendo executados nas máquinas t e i. As equações para a porcentagem de trabalho e para o tempo de execução são apresentadas a seguir.

$$f_t = \frac{1}{p_t + p_b g_{t,b}(p,n) + p_i g_{t,i}(p,n)} \tag{4.31}$$

$$T_t^A(f_t,n) = \frac{1}{p_t + p_b g_{t,b}(p,n) + p_i g_{t,i}(p,n)} \times \delta_t^A(1,n) \tag{4.32}$$

$$f_i = \frac{g_{t,i}(p,n)}{p_t + p_b g_{t,b}(p,n) + p_i g_{t,i}(p,n)} \tag{4.33}$$

$$T_i^A(f_i,n) = \frac{g_{t,i}(p,n)}{p_t + p_b g_{t,b}(p,n) + p_i g_{t,i}(p,n)} \times \delta_i^A(1,n) \tag{4.34}$$

4.9 Balanceamento de Carga

Em sistemas distribuídos existem vários fatores que podem influenciar o desempenho das aplicações e afetar o tempo total de execução dos programas. Entre esses fatores estão a capacidade de processamento das CPUs, a carga de trabalho atribuída às CPUs, atrasos gerados em atividades do próprio sistema operacional, largura de banda da rede de interconexão, quantidade de memória disponível, entre outros. A capacidade de processamento de cada máquina está relacionada não apenas à freqüência do *clock* da CPU, mas também à quantidade de memória disponível, à latência da memória e do cache, ao tamanho do cache, à freqüência dos barramentos e à velocidade dos dispositivos de E/S. Entretanto, outros fatores podem influenciar o desempenho das aplicações distribuídas, como, por exemplo, o balanceamento de carga do sistema. Portanto, para utilizar eficientemente os recursos computacionais que o ambiente disponibiliza e melhorar o desempenho global do sistema, é essencial utilizar estratégias de balanceamento de carga.

Em um ambiente heterogêneo, uma estratégia de balanceamento de carga eficiente deve levar em consideração a capacidade computacional de cada uma das máquinas e o custo da comunicação associada às distribuições das tarefas. Na literatura é possível encontrar vários trabalhos que adotam uma estratégia de balanceamento conhecida como *self-scheduling* (SS). Essa estratégia utiliza o modelo de comunicação mestre-escravo para organizar a solução distribuída. Inicialmente, o processo mestre envia uma carga unitária de trabalho a cada um dos processos. Em uma multiplicação de matrizes, por exemplo, essa carga unitária pode ser os elementos necessários para determinar o valor de uma única posição da matriz resultado. À medida que os processos terminam o processamento, eles enviam requisições ao mestre solicitando mais trabalho. Então, o mestre envia mais uma carga unitária e essa dinâmica continua até que todo o trabalho tenha sido executado. Na literatura é possível encontrar algumas variações da estratégia SS, tais como o *chunk self-scheduling* (CSS), o *guided self-scheduling* (GSS) e o *trapezoidal self-scheduling* (TSS) (POLYCHRONOPOULOS; KUCK, 1987; TZEN; NI, 1993; SHIH; YANG; TSENG, 2006; YANG; SHIH; TSENG, 2006; YANG; CHANG, 2003).

Em sistemas homogêneos, a estratégia de balanceamento de carga mais simples consiste em dividir igualmente um trabalho de tamanho n entre os p processos envolvidos na solução do problema. Assim, cada processo receberá $\frac{n}{p}$ do trabalho total. As estratégias Block Scheduling, Cyclic Scheduling e Block-D Scheduling (LI et al., 1993) utilizam essa abordagem na distribuição.

Entretanto, em sistemas distribuídos heterogêneos, um balanceamento de carga adequado deve levar em consideração a capacidade individual das máquinas na distribuição do trabalho. O objetivo das estratégias é evitar que alguns nós de processamento fiquem sobrecarregados enquanto outros estão ociosos, aumentando a eficiência do sistema e melhorando a utilização dos recursos.

Embora vários trabalhos tenham sido publicados propondo ou avaliando estratégias para o balanceamento de carga em sistemas distribuídos (MAHESHWARI, 1996; SÁNCHEZ; MACÍAS; SUÁREZ, 2002; IBRAHIM; XINDA, 2002; BANICESCU et al., 2005; LIU; GOLDSMITH, 2005; CIORBA et al., 2006; PENMATSA; CHRONOPOULOS, 2006), a constante mudança na organização desses sistemas e nas características das aplicações exige novas abordagens para tratar o problema. Por essa razão, estratégias continuam sendo implementadas e testadas (DU; SUN; WU, 2007; DHAKAL et al., 2007; QIN; XIE, 2008).

Com esse objetivo, novas estratégias para o balanceamento de carga em sistemas distribuídos heterogêneos foram definidas e implementadas durante o desenvolvimento deste trabalho (LAINE; MIDORIKAWA, 2007b). A próxima seção descreve os principais modelos utilizados na implementação de aplicações paralelas e algumas estratégias especificadas e implementadas nesse trabalho para realizar um balanceamento de carga eficiente.

4.10 Modelos para Aplicações Paralelas

Essa seção descreve algumas maneiras de se estruturar programas paralelos para organizar a divisão e distribuição de tarefas entre processos distribuídos. Os programas paralelos podem ser organizados em função dos seguintes modelos: paralelismo de dados, *workpool*, divisão e conquista e mestre-escravo.

O paralelismo de dados é uma técnica de programação que divide uma grande quantidade de dados em partes menores que podem ser processados em paralelo pelo programa (BAL; HAINES, 1998; CULLER; SINGH; GUPTA, 1998; TRAN; HLUCHY; NGUYEN, 2000). Após o término do processamento, os resultados parciais são combinados e a solução final é obtida. No modelo *workpool* (KNOPP; REICH, 1996) as tarefas são disponibilizadas em uma estrutura de dados global e os processos acessam posições de memória compartilhada para adquirir os dados que serão utilizados no processamento. Já no modelo de divisão e conquista (KUMARAN; QUINN, 1995) um trabalho complexo é dividido em tarefas mais simples para facilitar a solução. Basicamente, o problema é dividido recursivamente em problemas menores até que ele possa ser resolvido diretamente. Dessa forma, a solução final do problema inicial é obtida através da combinação dos resultados parciais de todos os problemas menores.

Neste trabalho, em particular, estamos analisando e comparando variações do modelo mestre-escravo (BANINO et al., 2004; BEAUMONT; LEGRAND; ROBERT, 2003) em relação à divisão e distribuição das cargas computacionais. Normalmente, a distribuição do trabalho entre os escravos é feita de maneira estática, como no caso da estratégia VRP (Seção 4.10.2). Isso significa que no início da execução do programa todo o trabalho é dividido e distribuído pelo mestre. Essa estratégia é adequada quando o ambiente de execução é homogêneo e dedicado. Se a carga computacional das máquinas sofre variação ao longo do tempo, a distribuição estática não produz bons resultados. Assim, uma abordagem

dinâmica pode ser utilizada para melhorar o desempenho da solução. As versões SS, VRP-CSS e VRP-CSS-Adapt são estratégias dinâmicas para distribuição de cargas. Nessas estratégias a divisão e distribuição do trabalho ocorre em fases. À medida que os processos terminam uma execução parcial, um novo trabalho é enviado pelo mestre. Isso se repete até que não haja mais trabalho a ser processado. Dessa forma, os processos que estão sendo executados em máquinas de maior capacidade computacional ou com menor carga de trabalho, conseguem colaborar mais na execução do programa. As implementações *self-scheduling*, apresentadas na Seção 4.10.1, seguem essa abordagem de distribuição.

4.10.1 Self-Scheduling

Na estrutura *Self-Scheduling* (SS) o processo mestre divide o trabalho total em pequenas tarefas, denominadas cargas unitárias de trabalho, para, em tempo de execução, enviá-las aos processos escravos. A carga unitária representa a menor unidade de trabalho possível. No caso da multiplicação de matrizes ($A \times B = C$), por exemplo, a carga unitária representa uma linha da matriz A e uma coluna da matriz B. Esses são os dados necessários para determinar um único elemento $C_{i,j}$ da matriz produto C. Ao terminar o processamento do trabalho recebido, os processos escravos enviam uma resposta ao mestre com o resultado alcançado no processamento, e solicitam uma nova carga unitária de trabalho. Portanto, o próprio modelo de implementação da estratégia permite um balanceamento de carga dinâmico no sistema.

Um dos problemas dessa abordagem é a excessiva quantidade de mensagens trocadas entre o processo mestre e os escravos na execução de programas que manipulam grande quantidade de dados. Dependendo das configurações da rede de interconexão, o desempenho da aplicação pode ser prejudicado ou até comprometido.

Uma ilustração desse modelo é apresentada na Figura 4.6. O trabalho total (tarefa) é dividido pelo mestre em cargas unitárias que são distribuídas aos escravos dinamicamente durante a execução do programa. Na literatura é possível encontrar algumas variações para essa estratégia, tais como a *Chunk Self-Scheduling (CSS)*, *Guided Self-Scheduling (GSS)* e a *Trapezoidal Self-Scheduling (TSS)* (POLYCHRONOPOULOS; KUCK, 1987; SHIH; YANG; TSENG, 2006; TZEN; NI, 1993; YANG; CHANG, 2003; YANG; SHIH; TSENG, 2006).

4.10.2 VRP - Vetor de Desempenho Relativo

Essa estratégia (Figura 4.7) foi especificada e implementada para permitir que cada processo receba uma tarefa adequada à capacidade computacional da máquina onde o mesmo está sendo executado (LAINE; MIDORIKAWA, 2007b). Para isso, o algoritmo utiliza um vetor, chamado VRP (Vetor de Desempenho Relativo), contendo valores associados à capacidade de processamento relativa de cada tipo de máquina do ambiente. Os valores do VRP são determinados através da relação definida na equação 4.11, e são utilizados

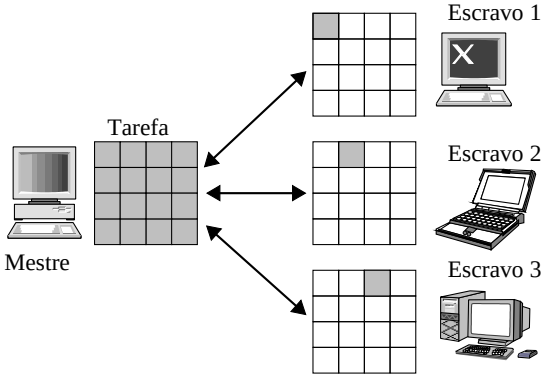

Figura 4.6: Ilustração do modelo SS.

pelo processo mestre para determinar a forma de divisão do trabalho. Diferentemente do *Self-Scheduling*, todo o trabalho é dividido e distribuído no início do processamento, caracterizando uma estratégia estática de balanceamento. Com isso, a quantidade de mensagens trocadas entre os processos diminui e o tempo gasto com as comunicações passa a ter menor influência sobre o tempo total de execução do programa. Entretanto, a eficiência desta distribuição pode ser comprometida se as cargas computacionais das máquinas apresentarem variações significativas ao longo do processamento da aplicação.

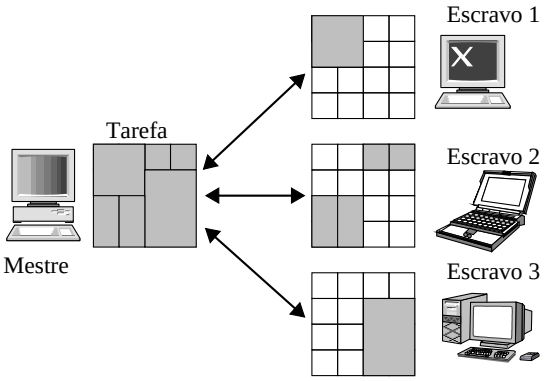

Figura 4.7: Ilustração do modelo VRP.

As relações de desempenho, definidas na Seção 4.7, são utilizadas para determinar os valores do VRP. O índice de desempenho relativo de cada máquina do conjunto M é calculado em função da máquina de menor capacidade computacional. No VRP, φ_i representa o índice de desempenho relativo da i-ésima máquina representada na vetor, enquanto φ_1 é o índice de desempenho da máquina mais lenta. Assim, o vetor de desempenho para m máquinas pode ser escrito da seguinte forma:

$$VRP = [\varphi_1, \varphi_2, \varphi_3, ..., \varphi_{m-1}, \varphi_m] \tag{4.35}$$

onde:

$$\varphi_m = g_{1,m}(p, n) = \frac{\delta_1(p, n)}{\delta_m(p, n)} \tag{4.36}$$

Os parâmetros $\delta_1(p,n)$ e $\delta_m(p,n)$ representam os modelos analíticos elaborados para as máquinas 1 e m, respectivamente. No VRP, o índice de desempenho da máquina de menor capacidade de processamento é igual a 1 ($\varphi_1 = 1$). Isto, porque $\frac{\delta_1(p,n)}{\delta_1(p,n)} = 1$.

Assim, o VRP é capaz de indicar, proporcionalmente, quanto de trabalho cada uma das máquinas deve receber para que o tamanho da tarefa esteja ajustado à capacidade de processamento de cada um dos nós. Assim como no *Self-Scheduling*, a distribuição do trabalho é realizada utilizando o conceito de carga unitária. Entretanto, cada máquina listada no VRP deve receber n cargas unitárias, onde n é dado pelo seu índice de desempenho relativo. A carga unitária representa a menor unidade de trabalho computável para a aplicação que está sendo modelada e testada, e pode ser determinada da seguinte maneira:

$$ul = \frac{\tau}{\sum_{i=1}^{m} VRP[i]} \tag{4.37}$$

$$\Delta_m = \lceil ul \times VRP[m] \rceil \tag{4.38}$$

onde ul representa a carga unitária, τ a quantidade de trabalho total a ser computada e Δ_m a quantidade de trabalho que deve ser enviada para a m-ésima máquina.

Quanto maior a precisão dos modelos analíticos, mais ajustados estarão os valores calculados para os índices do VRP. Por este motivo, criamos duas abordagens para a definição do VRP, uma estática e outra dinâmica. Na versão estática, o VRP permanece o mesmo em todas as configurações de testes, independente dos valores usados para n e p. Na abordagem dinâmica, os valores dos índices do VRP são determinados em função do tamanho do problema e da quantidade de processos (n e p). Essa estratégia obtém melhores resultados, pois está sintonizada às modificações de comportamento que as aplicações podem apresentar em função de n e p. A versão estática pode oferecer bons resultados em ambientes homogêneos, conforme será discutido no Capítulo 5.

4.10.3 VRP com Chunk-Self-Scheduling (VRP-CSS)

A eficiência da estratégia adotada pelo VRP pode ser comprometida por possíveis alterações na carga computacional das máquinas, conforme discutimos na seção anterior. Pensando nisso, propomos e implementamos a estratégia VRP-CSS (Figura 4.8). Essa nova estratégia utiliza um modelo dinâmico para a distribuição das cargas e adota uma variação da abordagem utilizada na política *Chunk Self-Scheduling (CSS)*.

Em tempo de execução, cada processo escravo recebe do mestre um trabalho adequado à capacidade de processamento local da máquina que está sendo utilizada para sua execução. O tamanho da tarefa é determinado através dos valores especificados no VRP, assim como é feito na estratégia anterior. Entretanto, as cargas computacionais vão sendo

distribuídas por parte, em fases de processamento. Ao terminar o processamento parcial da tarefa recebida, os escravos enviam a resposta ao mestre e solicitam uma nova fatia de trabalho. Essa dinâmica se repete até que todo o trabalho tenha sido processado.

Portanto, nosso objetivo é fazer com que os processos mais rápidos possam calcular uma parte maior da solução. Assim, se uma máquina apresentar sobrecarga de trabalho durante a execução do programa, a estratégia ajusta a distribuição para que a mesma não comprometa o desempenho final da solução. Dessa forma, esperamos que pouco atraso seja gerado por situações de espera no processamento final do trabalho. O sincronismo perfeito entre os processos escravos representa uma situação ideal, porém hipotética. Assim, desejamos maximizar a utilização dos recursos computacionais e minimizar o tempo de execução da aplicação melhorando a sincronização dos processos que colaboram na execução da aplicação.

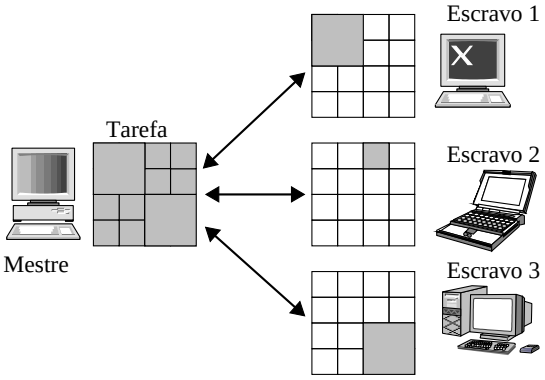

Figura 4.8: Ilustração do modelo VRP-CSS.

4.10.4 VRP Chunk-Self-Scheduling Adaptativo (VRP-CSS-Adapt)

Essa versão é uma abordagem adaptativa para distribuição e balanceamento de cargas computacionais. Em tempo de execução o programa calcula a capacidade de processamento efetiva (Cpe)[3] das máquinas que estão participando da solução. Baseado nesse valor, o processo mestre determina a quantidade ideal de trabalho que deve ser enviado ao escravo. Esta análise é realizada em cada iteração do processamento, no momento em que o mestre recebe uma resposta parcial dos escravos.

Para realizar o ajuste dinâmico da distribuição das cargas, o processo mestre deve atualizar, em tempo de execução, os índices do VRP. Para isso, o mestre calcula a Cpe de cada uma das máquinas onde os processos escravos estão sendo executados, no momento em que recebe uma resposta parcial do processamento. Assim, um VRP' é determinado em cada fase do processamento. O índice de cada máquina i do VRP é recalculado da seguinte forma:

[3] $Cpe = \frac{\tau}{t}$, sendo τ a quantidade de trabalho processada e t o tempo de execução gasto na computação de τ.

$$VRP'[i] = VRP[i] \times Cpe_i \tag{4.39}$$

Portanto, o ajuste proporcionado pela estratégia permite a aplicação reagir às alterações de carga do sistema, o que não é possível nas outras estratégias. Com isso, a estratégia tende a evitar ou a reajustar eventuais sobrecargas e/ou ociosidades detectadas nas máquinas do ambiente.

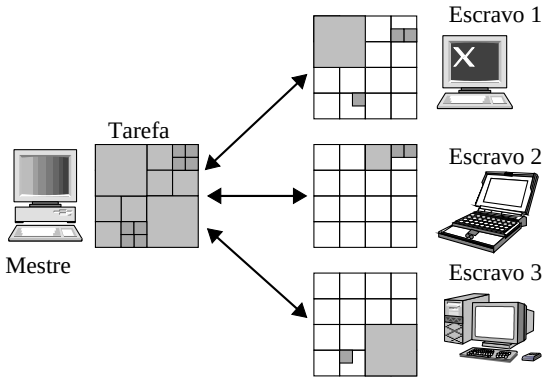

Figura 4.9: Ilustração do modelo VRP-CSS-Adapt.

4.11 Considerações Finais

Neste capítulo, apresentamos as etapas definidas pelo processo especificado na metodologia PEMPIs-Het. Além disso, explicamos, detalhadamente, cada uma das atividades envolvidas na modelagem e as possíveis aplicações para os modelos analíticos de desempenho. Destacamos a importância do modelo gráfico DP*Graph^{++} na compreensão do código fonte do programa e do modelo de solução adotado na implementação. Apresentamos as principais diferenças entre o modelo gráfico criado para sistemas homogêneos e o estendido para sistemas heterogêneos.

Especificamos e mostramos como as relações de desempenho são definidas. A partir das relações, explicamos como os modelos de desempenho são elaborados para predizer o comportamento das aplicações em cada uma das máquinas do sistema. Além disso, destacamos as estratégias implementadas para balanceamento de carga, e as vantagens e desvantagens de cada política. No final do capítulo, apresentamos alguns modelos de soluções para aplicações paralelas que fazem uso de estratégias de balanceamento de carga.

Capítulo 5

Resultados e Análises

Neste capítulo, apresentamos os resultados obtidos em avaliações experimentais que ilustram o uso das técnicas especificadas na metodologia PEMPIs-Het. Inicialmente, caracterizamos o comportamento de dois programas paralelos em ambientes distribuídos homogêneos e avaliamos a precisão dos modelos de desempenho elaborados. Depois, estendemos a análise para ambientes heterogêneos. Por fim, avaliamos algumas estratégias de balanceamento de carga e comparamos diferentes estruturas de programas paralelos como alternativas para soluções distribuídas.

5.1 Introdução

Ao longo do desenvolvimento da pesquisa, ajustes foram realizados no processo de modelagem dos programas paralelos, visando melhorar a qualidade e precisão das estimativas de desempenho. Como as estratégias de balaceamento de carga da metodologia PEMPIs-Het fazem uso dos modelos analíticos, essa precisão é fundamental para que os resultados esperados sejam alcançados.

Na modelagem analítica, vários fatores podem contribuir para a precisão dos modelos. O próprio método de ajuste de curva utilizado pode influenciar na precisão das estimativas. No caso do método dos mínimos quadrados, utilizado no processo de modelagem proposto pela metodologia PEMPIs-Het, o modelo pode se ajustar melhor a uma faixa de dados e comprometer outra. Particionar o espaço amostral da análise e elaborar modelos para intervalos menores pode aumentar a precisão das predições. No entanto, isso não evita a influência de possíveis anomalias.

Por causa disso, apresentamos, nesse capítulo, duas abordagens para modelagem analítica. A primeira estratégia consiste em elaborar modelos de predição para valores pontuais do tempo de execução. Já na segunda abordagem, definimos um intervalo para as estimativas. Esse intervalo estabelece um valor máximo e outro mínimo para os tempos gastos pelos processos ou aplicações. Modelagens e avaliações de desempenho são realizadas em ambientes homogêneos e heterogêneos.

Além desses estudos, caracterizamos o desempenho de algumas estratégias de balanceamento de carga e demonstramos a capacidade da metodologia em avaliar e prever o comportamento de modelos de soluções paralelas e distribuídas.

5.2 Modelagem, Avaliação e Predição de Desempenho em Sistemas Homogêneos

Para avaliar a eficiência das estratégias de modelagem, avaliação e predição de desempenho da metodologia PEMPIs-Het em sistemas distribuídos homogêneos, apresentamos, nas próximas seções, dois estudos de caso. O primeiro faz uso de um programa que simula a propagação de calor sobre uma superfície metálica. Esse programa é conhecido como *Heat*. No segundo estudo de caso, utilizamos o tradicional programa de multiplicação de matrizes para o desenvolvimento das atividades.

5.2.1 Descrição do Ambiente de Teste Homogêneo

As avaliações no ambiente homogêneo foram realizadas em um *cluster* de PCs, denominado *Bio*, do laboratório LAHPC[1]. Esse *cluster* possui 8 máquinas com 2 processadores AMD Athlon MP 2400+ cada, 1 GB de DDR SDRAM e uma placa de rede INTEL EtherExpress Pro. O sistema operacional instalado é o Fedora Core 5 e a distribuição MPI disponível no ambiente é o LAM-MPI 7.1.3.

5.2.2 Programa Heat

O programa *Heat* implementa um problema da termodinâmica que simula a transmissão de energia térmica entre as partículas de uma superfície metálica, mediante colisões e alterações das agitações térmicas. Na simulação, o valor da temperatura de cada um dos pontos da superfície é mapeado através de uma matriz de dados. Na simulação, um conjunto de passos é executado e, a cada iteração, um novo valor de temperatura é calculado para cada elemento da malha bidimensional, até que haja uma convergência final para a temperatura da placa metálica. A temperatura de cada um dos pontos, nomeados de $t_{i,j}$, é determinada com base nos valores dos pontos vizinhos, da seguinte forma:

$$t'_{i,j} = f(t_{i,j}, t_{i,j-1}, t_{i,j+1}, t_{i-1,j}, t_{i+1,j})$$

A paralelização utilizada na solução do problema faz uso do modelo mestre-escravo, e os cálculos da simulação são divididos entre todos os nós de processamento que participam da computação. Cada processo escravo recebe um subconjunto dos dados, representado

[1] http://regulus.pcs.usp.br/~lahpc/

por um bloco de linhas da matriz. A matriz caracteriza a malha bidimensional de pontos sobre a superfície metálica. No início de cada iteração, os processos trocam informações com seus vizinhos para atualizar o mapeamento da temperatura. Cada tarefa paralela é mapeada em um único processador.

A primeira etapa da metodologia consiste em analisar o código fonte do programa e mapear as instruções através dos símbolos do modelo gráfico DP*Graph^{++}. Pelo fato do ambiente de teste ser homogêneo, a representação ilustrada na Figura 5.1 utiliza a simbologia criada para o DP*Graph^{+}. Nesse modelo gráfico podemos identificar tanto os trechos de computação intensiva, como as comunicações estabelecidas entre os processos. O processamento da simulação é feito quase exclusivamente pelos processos escravos e está mapeado nos trechos de VIII a XIII do DP*Graph^{+}.

Nesse estudo de caso, resolvemos detalhar o processo de modelagem do programa Heat e elaborar equações parciais para representar o comportamento dos principais trechos do programa ilustrado na Figura 5.1. Com isso, esperamos deixar claro como as técnicas da metodologia são aplicadas.

Para o desenvolvimento dos modelos de predição foram realizados testes experimentais com 4, 5 e 6 processos, cada um sendo executando em uma máquina diferente do *cluster* Bio. Os valores utilizados para o tamanho da superfície metálica foram 500, 750, 1.000, 1.250, 1.500, 1.750 e 2.000. Para cada configuração de teste planejada, repetimos a execução do programa 50 vezes, de modo a dar mais confiabilidade aos dados coletados. Os tempos obtidos são selecionados por funções que descartam eventuais anomalias. Por fim, um valor médio é obtido como referência para o trecho ou o programa avaliado.

Em estudos preliminares, avaliamos a complexidade das principais primitivas do LAM-MPI (OLIVEIRA; LAINE; MIDORIKAWA, 2002). Nesse estudo, descobrimos a complexidade das operações e especificamos os modelos teóricos de algumas rotinas MPI. A instrução MPI_Init, por exemplo, representada no trecho I do modelo gráfico da Figura 5.1, pode ser modelada por uma função linear em p; onde p é o número de processos utilizados na execução do programa. Assim, após executar o programa, medir os tempos de cada um dos trechos de interesse e gerar os valores médios para os tempos coletados, aplicamos o método de ajuste de curvas dos mínimos quadrados para elaborar os modelos individuais para cada parte do programa. Em relação ao MPI_Init, o seguinte modelo analítico foi gerado para estimar o tempo de inicialização:

$$T_I(p) = -1,9232 \times 10^{-3} + 4,3877 \times 10^{-3} \times p \tag{5.1}$$

As variáveis p, n e m, que aparecem no desenvolvimento das equações, representam o número de processos, o tamanho do problema e o tamanho da mensagem transmitida (em bytes), respectivamente. As estimativas para os tempos, em cada uma das fórmulas apresentadas nesta seção, são dadas em segundos.

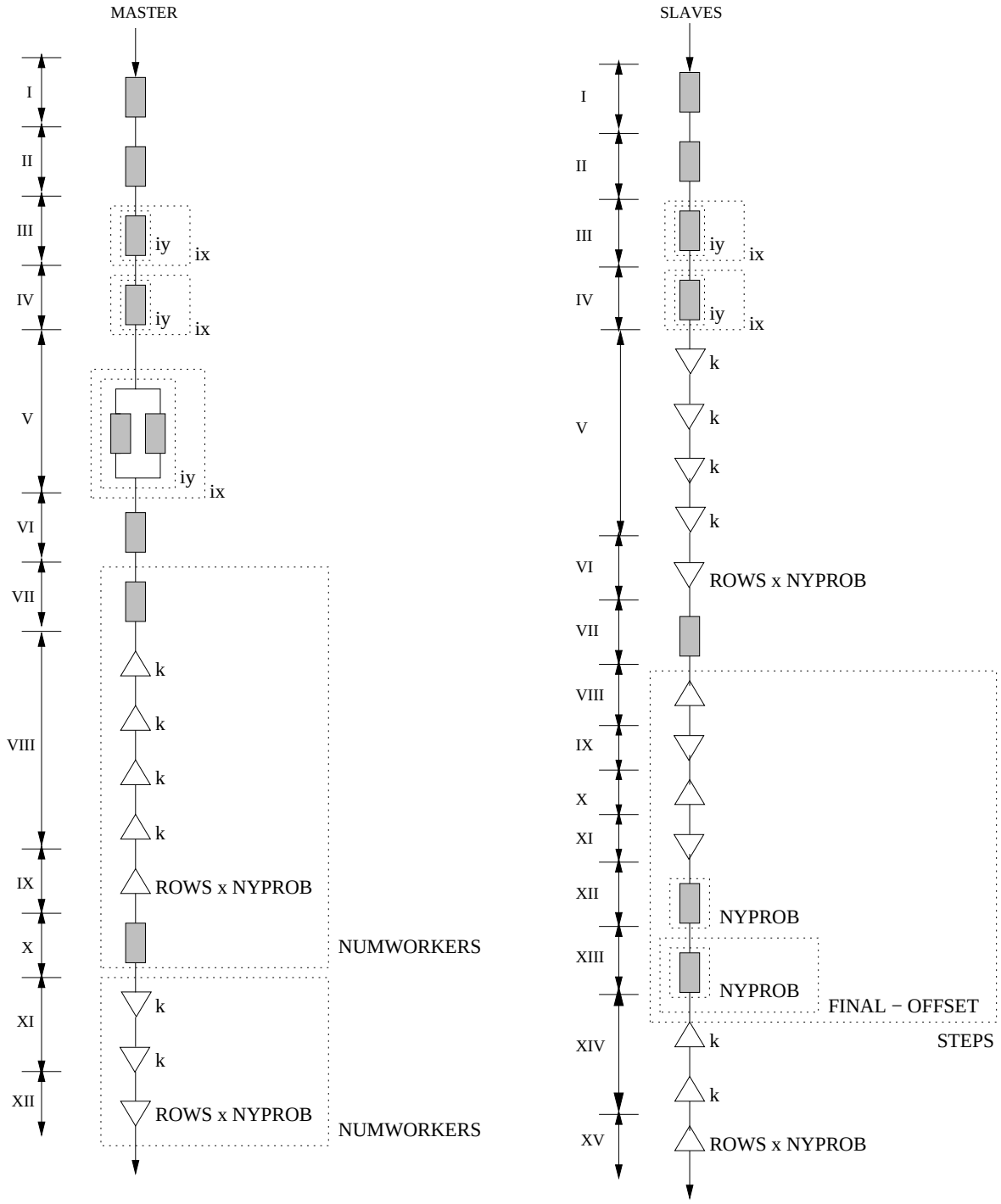

Figura 5.1: DP*Graph$^+$ do programa Heat.

O código dos trechos de II a VI representa o processamento de dois laços aninhados. Portanto, um polinômio de segundo grau pode ser utilizado para representá-lo (LAINE; OLIVEIRA; MIDORIKAWA, 2003). Assim, o modelo de predição elaborado para representar esse trecho é dado por:

$$T_{II-VI}(n) \quad = \quad 0,3425685 - 5,874 \times 10^{-4} \times n + 2,9 \times 10^{-6} \times n^2 \tag{5.2}$$

De VII a X, o processo mestre utiliza uma estrutura de repetição e a primitiva MPI_Send para distribuir a matriz entre os escravos. O comportamento da primitiva MPI_Send pode ser aproximado por um modelo linear em m (OLIVEIRA; LAINE; MIDORIKAWA, 2002). Assim, o modelo gerado para representar o tempo de execução desse trecho é:

$$T_{VII-X}(m) = (p-1) \times (6,8114 \times 10^{-3} + 5 \times 10^{-7} \times m) \tag{5.3}$$

O último trecho do processo mestre é composto somente por operações MPI_Recv. Nessa fase do programa o mestre recebe os resultados parciais dos processos escravos. A primitiva *receive*, assim como a *send*, é representada por uma função linear em m, conforme a seguinte expressão:

$$T_{XI-XII}(m) \quad = \quad (p-1) \times (-4,487 \times 10^{-4} + 4,899 \times 10^{-7} \times m) \tag{5.4}$$

Embora seja o processo mestre que determina o tempo final de execução do programa e seus modelos parciais já tenham sido elaborados, ainda é necessário incluir no modelo final de predição o tempo gasto pelos escravos nos cálculos envolvidos na simulação do problema. Esse processamento é representado pelos trechos de VIII a XIII do escravo. O mestre só pode finalizar o programa depois que os escravos enviarem o resultado parcial.

Para modelar o trecho da simulação, foram elaborados três modelos. O primeiro representa o tempo gasto em comunicações (T_{com}) do tipo *send* e *receive*. O segundo e o terceiro determinam o tempo gasto na execução de instruções dentro de estruturas de repetição simples (T_{sl}) e aninhada (T_{nl}). Repetindo o processo de modelagem, para cada um dos trechos, as seguintes expressões foram obtidas:

$$T_{com}(m) = 3,5 \times 10^{-4} + 1,3 \times 10^{-6} \times m \tag{5.5}$$

$$T_{sl}(n) = -5,25 \times 10^{-7} + 2,39 \times 10^{-8} \times n \tag{5.6}$$

$$T_{nl}(n) \quad = \quad -3,6341 \times 10^{-3} - 8,1 \times 10^{-6} \times n + 1,66 \times 10^{-7} \times n^2 \tag{5.7}$$

Para determinar o modelo analítico final da aplicação *Heat*, e prever o tempo total de execução do programa, é necessário compor todos os modelos parciais apresentados. Assim, é possível gerar estimativas para o tempo total de execução do programa em função dos parâmetros considerados como entrada para as expressões, como, por exemplo, p e

n. A Tabela 5.1 contém as configurações avaliadas nos testes experimentais e os tempos médios calculados a partir dos valores medidos e os estimados com o modelo final de predição. Podemos verificar a precisão das estimativas em todos os casos analisados. Isso demonstra a eficiência do método de modelagem e a qualidade dos testes experimentais.

Tabela 5.1: Tempo de execução (segundos) das configurações utilizadas na modelagem.

Dimensão da Placa (n)	4 processos		5 processos		6 processos	
	Estimado	Medido	Estimado	Medido	Estimado	Medido
500	1,022	0,844	1,019	0,856	1,016	0,858
750	2,108	2,231	2,105	2,275	2,102	1,921
1.000	3,668	3,840	3,685	3,883	3,682	3,184
1.250	5,762	5,589	5,759	5,611	5,756	5,074
1.500	8,329	7,972	8,326	8,378	8,323	7,987
1.750	11,389	10,849	11,386	10,996	11,384	11,110
2.000	14,943	14,763	14,941	14,673	14,938	14,614

Os dados apresentados na Tabela 5.2 são estimativas geradas pelo modelo para valores que extrapolam o intervalo de dados utilizado na modelagem da aplicação. Além de gerar a predição para os tempos de execução, executamos o programa para verificar a precisão das estimativas. O erro percentual foi calculado para cada configuração avaliada. Analisando os dados é possível verificar que a maioria dos valores estimados é menor que o tempo medido na execução do programa. Como o modelo não caracteriza nem expressa a influência de todos as possíveis alterações que o ambiente pode sofrer durante a execução real da aplicação, como contenções na rede e alterações pontuais na carga computacional das máquinas, a tendência é que os tempos medidos sejam realmente maiores. Embora os modelos analíticos tenham suas restrições em relação aos aspectos representados, o maior erro percentual das estimativas foi de -13,75% ($n = 3000$ e $p = 12$). O valor negativo na porcentagem indica que o tempo predito foi menor que o valor medido na execução do programa.

Uma representação gráfica para a variação dos erros percentuais em função da dimensão da placa e da quantidade de processos é apresentada na Figura 5.2. Mesmo para os valores mais distantes do intervalo utilizado na modelagem, os resultados obtidos são satisfatórios. Pelo gráfico é possível observar uma tendência na diminuição dos erros percentuais. Para os valores avaliados, os resultados obtidos comprovam a eficiência do processo de modelagem da metodologia PEMPIs-Het.

5.2.3 Multiplicação de Matrizes - MM

A multiplicação de matrizes é uma operação utilizada em muitas aplicações científicas (YANG et al., 2007), tais como a computação gráfica e o processamento de vídeos. Trabalhos realizados na área de predição de desempenho e balanceamento de carga usam,

Tabela 5.2: Comparação entre os tempos medidos e os estimados (seg.) – Heat.

Dimensão da Placa (n)	6 processos			8 processos		
	Estimado	Medido	Erro (%)	Estimado	Medido	Erro (%)
2.000	14,938	15,338	-2,61	14,93	14,745	1,27
2.500	23,527	24,653	-4,57	23,52	23,914	-1,64
3.000	34,090	39,021	-12,64	34,08	38,682	-11,89
3.500	46,627	53,137	-12,25	46,62	52,547	-11,28
4.000	61,139	68,420	-10,64	61,13	68,080	-10,20

Dimensão da Placa (n)	10 processos			12 processos		
	Estimado	Medido	Erro (%)	Estimado	Medido	Erro (%)
2.000	14,926	14,786	0,95	14,92	14,866	0,36
2.500	23,515	23,790	-1,16	23,51	23,953	-1,85
3.000	34,078	38,415	-11,29	34,07	39,505	-13,75
3.500	46,616	51,727	-9,88	46,61	53,014	-12,08
4.000	61,128	66,836	-8,54	61,12	67,253	-9,12

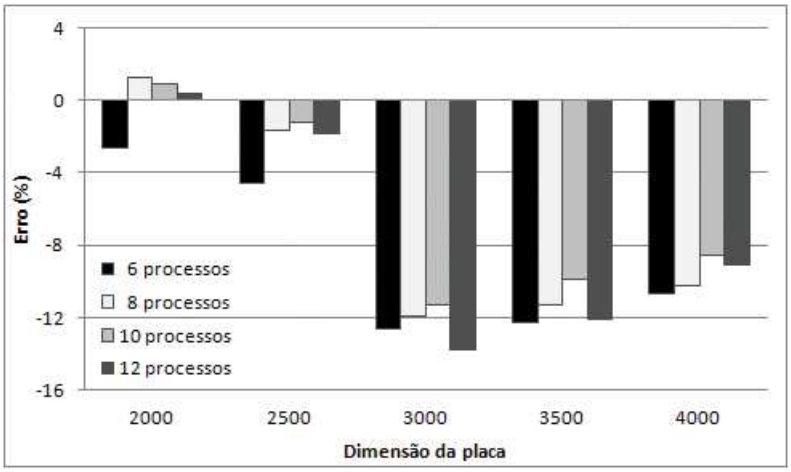

Figura 5.2: Variação dos erros percentuais nas estimativas do programa *Heat*.

com freqüência, essa aplicação em estudos de caso. A importância dessa aplicação é evidente, de modo que vários trabalhos têm sido desenvolvidos com o objetivo de criar algoritmos paralelos mais eficientes para a multiplicação de matrizes (LUO; DRAKE, 1995; HUSS-LEDERMAN et al., 1996; BEAUMONT et al., 2000; DAVE et al., 2007; CHAN et al., 2008).

Durante os testes experimentais foi utilizado o algoritmo convencional de multiplicação de matrizes, com complexidade $O(n^3)$. O modelo gráfico DP*Graph$^+$, elaborado para representar o código da aplicação, é ilustrado na Figura 5.3. Durante a execução do programa, 8 processos foram criados para serem utilizados no *cluster Bio*. Da mesma forma que fizemos na execução do programa *Heat*, cada processo foi associado a uma máquina distinta. Os valores utilizados para a dimensão das matrizes foram: 100, 300, 500, 700, 900, 1.200, 1.400 e 2.200. Após repetir o processo de modelagem usado na aplicação anterior, o modelo de desempenho da aplicação de multiplicação de matrizes

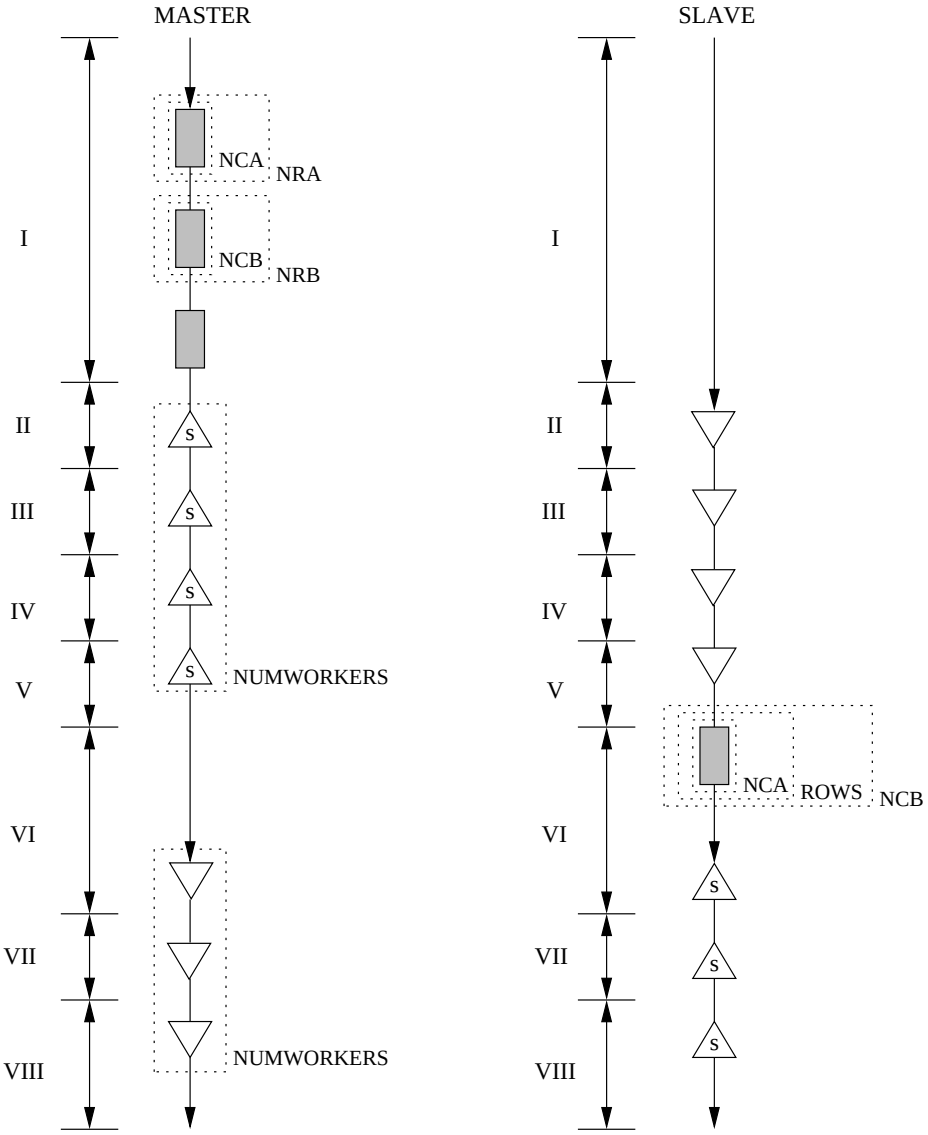

Figura 5.3: DP*Graph$^+$ do programa MM.

foi determinado. Nesse estudo, em particular, fixamos a modelagem para oito processos. Logo, as predições de desempenho foram realizadas somente em função da dimensão das matrizes (n). Como a complexidade do programa é $O(n^3)$, um polinômio de grau 3 é o mais indicado para representar o comportamento da aplicação. Nessa abordagem, o seguinte modelo foi elaborado:

$$T(n) = 0,014905 - 1,30 \times 10^{-4} \times n + 9 \times 10^{-7} \times n^2 + 3,348 \times 10^{-9} \times n^3 \qquad (5.8)$$

Os resultados apresentados na Tabela 5.3 mostram a precisão do modelo de predição em relação aos dados medidos nos testes experimentais. Para todas as configurações, o erro da estimativa foi inferior a 4%. O método de modelagem da metodologia PEMPIs-Het consegue representar adequadamente o comportamento dos programas que não apresentam anomalias durante a execução, como é o caso desta versão do programa de multi-

Tabela 5.3: Comparação entre os tempos medidos e os estimados (seg.) – MM.

Dimensão	Estimado	Medido	Erro (%)
500	0,593	0,617	3,92
700	1,513	1,562	3,20
900	3,067	3,095	0,91
1.200	6,939	6,808	-1,90
1.400	10,783	10,982	1,85
2.200	39,733	39,874	0,35

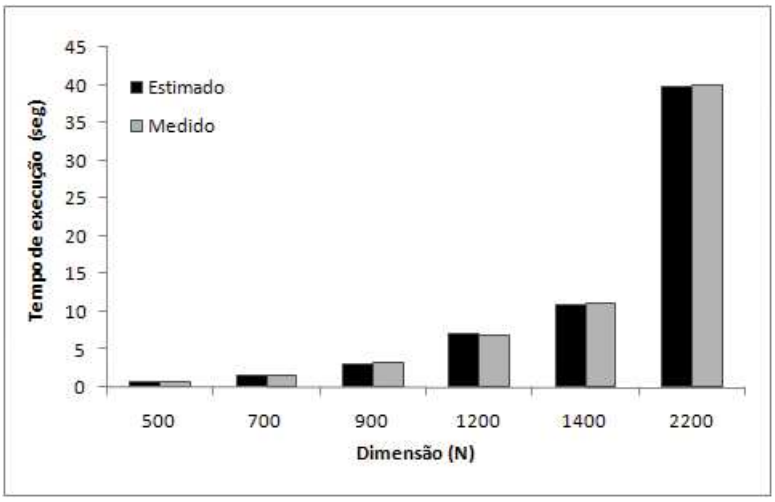

Figura 5.4: Comparação entre tempos medidos e estimados para o programa MM.

plicação de matrizes. A influência do método dos mínimos quadrados pode ser percebida nos resultados apresentados, pois os maiores erros percentuais se concentram nos menores da dimensão. O gráfico da Figura 5.4 apresenta uma comparação visual entre os tempos medidos e os estimados pelo modelo. A precisão das estimativas é aparente nessa ilustração. Portanto, nesses dois estudos de caso sobre aplicações paralelas em ambientes distribuídos homogêneos, ficou evidente a eficácia da metodologia no propósito de modelagem e predição de desempenho. Na próxima seção, repetimos as avaliações em ambientes heterogêneos.

5.3 Modelagem, Avaliação e Predição de Desempenho em Sistemas Heterogêneos

Para avaliar a aplicabilidade das estratégias de modelagem da metodologia PEMPIs-Het em ambientes heterogêneos, utilizamos um programa que simula a interação de partículas em um campo gravitacional. Esse programa é caracterizado por instruções complexas de processamento e várias primitivas de comunicação.

5.3.1 Descrição do Ambiente de Teste Heterogêneo

Os testes experimentais realizados no sistema distribuído heterogêneo utilizaram um ambiente com três tipos diferentes de máquinas: Intel, Bio e Taurus. As máquinas Intel possuem um processador dual-core Intel Pentium D 950, 2GB de DDR2 SDRAM, duas interface de rede gigabit Ethernet. As máquinas Bio possuem dois processadores AMD Athlon MP 2400$^+$, com 1GB de DDR SDRAM, duas interfaces de rede Intel Ether-Express Pro Fast Ethernet. As máquinas Taurus possuem um Intel Celeron 433MHz, 256 MB de SDRAM, interface de rede Fast Ethernet. As aplicações MPI usam a implementação LAM-MPI 7.1.3 e o sistema operacional instalado em todas as máquinas é o Fedora Core 6. Uma ilustração da organização do ambiente é dada pela Figura 5.5.

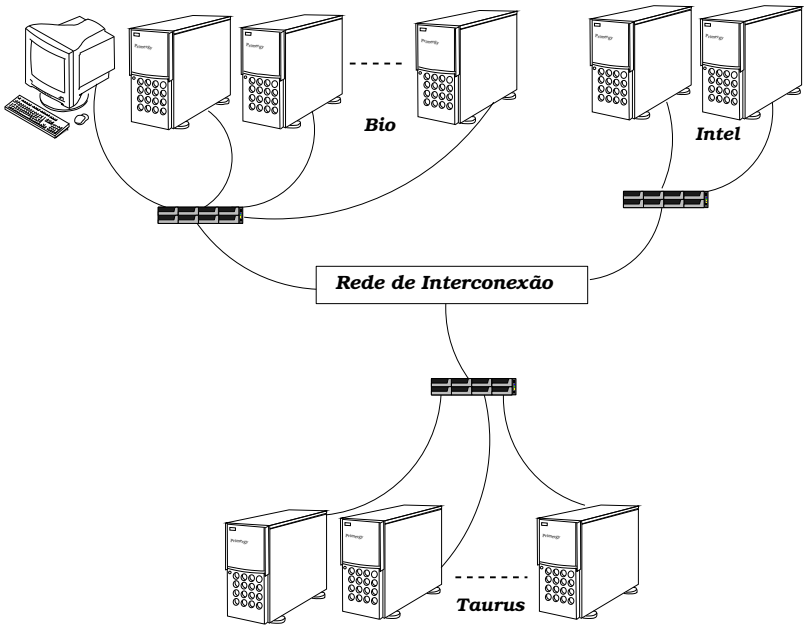

Figura 5.5: Ilustração do ambiente de teste.

5.3.2 Partículas sob Forças Gravitacionais - PGF

O programa de simulação de partículas sob forças gravitacionais implementa uma aplicação semelhante a um problema bem conhecido, denominado *problema dos n-corpos* (FRANKLIN; GOVINDAN, 1993). Cada uma das n partículas é caracterizada por sua massa, sua posição no campo gravitacional e pelo seu estado. O algoritmo usado adota um método chamado PP (*Particle-Particle*) para calcular as forças de interações entre cada par de partículas.

A paralelização da aplicação consiste em distribuir entre os nós de processamento um conjunto de partículas para que sejam calculadas as interações. O processo mestre mantém uma estrutura de dados que armazena o estado atual de cada uma das partículas. Essa estrutura é enviada aos demais nós, por *broadcast*, no início da simulação do problema.

Antes de começar o cálculo de cada uma das iterações envolvidas na solução do problema, cada máquina deve se comunicar com o mestre a fim de enviar o estado atual das partículas. Depois que o mestre adquire essas informações, ele retransmite a estrutura novamente a todas as tarefas. A memória utilizada para armazenar as estruturas de dados é alocada dinamicamente, o que otimiza o seu uso.

Na modelagem das aplicações em ambientes heterogêneos, o programa deve ser caracterizado em cada tipo de máquina do sistema. Os modelos analíticos individuais, então, são combinados para gerar um "metamodelo"da aplicação. Esse "metamodelo", segundo especificado na metodologia PEMPIs-Het, consegue estimar o desempenho do programa no sistema, e não em um tipo específico de máquina. Portanto, os aspectos modelados não estão associados somente às características da aplicação (*software*), mas consideram características do *hardware* utilizado. Dessa forma, o modelo contempla, implicitamente, a influência da heterogeneidade do ambiente.

As Tabelas 5.4, 5.5 e 5.6 apresentam os tempos médios calculados sobre os valores selecionados para os testes realizados nas máquinas *Taurus*, *Intel* e *Bio*. Além dos tempos, calculamos o valor para o desvio padrão sobre os dados da amostragem. Durante os testes experimentais, cada configuração avaliada foi executada 30 vezes. As anomalias foram descartadas em função do valor médio e do desvio padrão. Todos os tempos fora do intervalo definido pelo cálculo da média ± desvio padrão foram desconsiderados na seleção dos dados.

Tabela 5.4: Tempos médios (segundos) nas máquinas Taurus - PGF.

2	**3**	**4**	**Partículas**
1643,08	1097,00	836,74	**25.000**
0,10	0,11	0,23	**Desvio Padrão**
6569,50	4382,91	3341,37	**50.000**
0,35	0,26	0,45	**Desvio Padrão**
26276,31	17528,53	13356,30	**100.000**
12,45	13,32	2,73	**Desvio Padrão**
59115,68	39423,62	30046,01	**150.000**
14,24	15,36	14,71	**Desvio Padrão**

Tabela 5.5: Tempos médios (segundos) nas máquinas Intel - PGF.

2	**3**	**4**	**Partículas**
202,04	136,09	103,78	**25.000**
1,56	1,37	0,92	**Desvio Padrão**
802,37	540,64	408,59	**50.000**
8,23	6,11	2,30	**Desvio Padrão**
3213,70	2152,15	1623,19	**100.000**
15,75	13,87	10,45	**Desvio Padrão**
7223,75	4850,01	3663,72	**150.000**
39,28	36,51	21,10	**Desvio Padrão**

Tabela 5.6: Tempos médios (segundos) nas máquinas Bio - PGF.

2	**3**	**4**	**5**	**6**	**7**	**Partículas**
394,06	264,08	204,71	164,78	139,59	120,40	**25.000**
1567,07	1047,74	813,77	653,05	548,95	472,01	**50.000**
6244,54	4171,37	3245,36	2600,37	2176,10	1868,52	**100.000**
14041,62	9373,52	7293,47	5840,72	4881,06	4189,20	**150.000**

Nesse primeiro estudo de caso, os modelos analíticos são gerados em função de n (quantidade de partículas) e p (quantidade de processos). Portanto, as estimativas podem ser calculadas em função da variação desses dois parâmetros.

Nessa análise, resolvemos elaborar o modelo de desempenho da aplicação nas máquinas *Intel*. O modelo teórico da aplicação é dado pela seguinte expressão:

$$\delta^{PGF}(n,p) = \frac{C2}{p} \times n^2 + \frac{C1}{p} \times n + \frac{C0}{p} \tag{5.9}$$

As Tabelas 5.7 e 5.8 apresentam os coeficientes do modelo teórico, encontrados na modelagem da aplicação. Inicialmente, modelamos o programa em função de n e depois em função de p, segundo as técnicas descritas na metodologia PEMPIs-Het.

Tabela 5.7: Coeficientes em função de n (Intel).

C0	-1,125736	5,790233	8,578502
C1	0,000073	-0,000157	-0,000247
C2$\times 10^{-7}$	3,206325	2,163252	1,640802

Tabela 5.8: Coeficientes em função de p (Intel).

C0	18,569761- 39,199647$\times$(1/p)
C1	0,0005777+0,0012943$\times$(1/p)
C2	0,0756705$\times 10^{-7}$ + 6,26157$\times 10^{-7}$(1/p)

Compondo os coeficientes da Tabela 5.7 e 5.8 e substituindo na equação 5.9, o seguinte modelo de desempenho é obtido para estimar o tempo de execução da aplicação nas máquinas Intel:

$$\delta^{PGF}_{Intel}(p,n) = (7,57 \times 10^{-9} + \frac{6,26 \times 10^{-7}}{p})n^2 + (-5,78 \times 10^{-4} + \frac{1,3 \times 10^{-3}}{p})n + (-1,99 + \frac{8,74}{p}) \tag{5.10}$$

Para as máquinas do tipo Bio, repetimos a modelagem anterior e o seguinte modelo de desempenho foi gerado:

$$\delta^{PGF}_{Bio}(p,n) = (1,31 \times 10^{-8} + \frac{1,22 \times 10^{-6}}{p})n^2 + (2,22 \times 10^{-4} - \frac{2,55 \times 10^{-4}}{p})n + (-1,99 + \frac{8,74}{p}) \tag{5.11}$$

Por fim, o modelo de desempenho individual das máquinas Taurus foi gerado. A expressão que caracteriza o comportamento do programa nessas máquinas é:

$$\delta^{PGF}_{Taurus}(p,n) = (3,35 \times 10^{-8} + \frac{51,80 \times 10^{-7}}{p})n^2 + (1,45 \times 10^{-4} - \frac{4,97 \times 10^{-5}}{p})n + (-0,24 - \frac{4,67}{p}) \quad (5.12)$$

Esses modelos podem ser combinados para estimarmos o tempo de execução da aplicação em um sistema que utilize os três tipos de máquinas.

A Tabela 5.9 apresenta uma comparação entre os valores medidos e os estimados pelo modelo da equação 5.10 ao prever o comportamento da aplicação nas máquinas do tipo Intel. Já na Tabela 5.10, temos os resultados comparativos entre os tempos medidos e os valores estimados pela equação 5.11 referentes às máquinas do tipo Bio. Os tempos estão expressos em segundos. A precisão das estimativas é verificada em todos os casos apresentados.

Tabela 5.9: Precisão da predição de desempenho – PGF (Intel).

	2 processos		3 processos		4 processos	
Partículas	**Estimado**	**Medido**	**Estimado**	**Medido**	**Estimado**	**Medido**
25.000	201,10	202,04	137,08	136,09	104,95	103,78
50.000	804,12	802,37	538,78	540,64	406,41	408,59
100.000	3212,53	3213,70	2153,39	2152,15	1624,65	1623,19
150.000	7224,10	7223,75	4849,64	4850,01	3663,28	3663,72

Tabela 5.10: Precisão da predição de desempenho – PGF (Bio).

	2 processos		3 processos		4 processos	
Partículas	**Medido**	**Estimado**	**Medido**	**Estimado**	**Medido**	**Estimado**
25.000	394,06	394,73	264,08	264,29	204,71	204,63
50.000	1567,07	1565,83	1047,74	1047,36	813,77	813,94
100.000	6244,54	6245,37	4171,37	4171,63	3245,36	3245,25
150.000	14041,62	14041,37	9373,52	9373,44	7293,47	7293,50

Na Tabela 5.11, apresentamos algumas predições obtidas com a modelagem do programa PGF implementado com a estratégia VRP estática no ambiente formado por 4 máquinas Intel, 6 Bios e 8 Taurus. Nessa tabela destacamos os tempos médios de execução dos processos escravos, os valores estimados pelo modelo analítico, o desvio padrão da amostra e os erros percentuais dos tempos médios em relação aos valores preditos.

Em qualquer ambiente distribuído compartilhado, a execução da aplicação está sujeita à interferências não previstas na modelagem. Essas alterações podem resultar das mudanças na carga computacional do sistema, do tráfego excessivo de dados pela rede ou de eventuais problemas no *hardware*. Essas situações modificam, dinamicamente, o comportamento do programa MPI e prejudicam a precisão dos modelos. Além disso, possíveis

Tabela 5.11: Comparação entre os tempos medidos e os estimados – PGF.

	20.000	**30.000**	**40.000**	**50.000**
Predição	33,90	73,91	129,57	200,88
Tempo Médio Medido	32,43	72,32	157,87	201,58
Erro (%)	-4,35	-2,15	21,84	0,35
Desvio Padrão	0,93	1,08	4,36	7,66

imprecisões no modelo de predição também podem prejudicar as predições. Assim, uma alternativa para as atividades de predição de desempenho é adotar uma estimativa por intervalos (SCHOPF; BERMAN, 1997; LASTOVETSKY; TWAMLEY, 2005).

Nessa abordagem, é estimado um valor mínimo e outro máximo para o tempo de execução do programa. Assim, a predição é feita por intervalos do tipo: $[\delta - \Delta, \delta + \Delta]$. Sendo δ o tempo estimado pelo modelo de desempenho, e Δ o erro máximo previsto pelas estimativas. Essa previsão para o erro máximo é calculada através de um modelo analítico elaborado para representar a variação dos erros percentuais em cada caso de teste analisado. Com essa abordagem, a forma de estimar o tempo de execução deixa de ser feita por valores pontuais e passa a ser realizada por intervalos. Assim, para cada valor de p e n, um intervalo aceitável para o tempo de execução é determinado.

O comportamento dos erros de predição para a aplicação PGF, implementada com a estratégia VRP estática, foi analisado e modelado a fim de adaptar as estratégias do PEMPIs-Het para essa outra abordagem. Dessa forma, o erro de predição (Δ) foi modelado pela seguinte expressão:

$$\Delta(n) = 1,514 \times 10^{-9} n^2 + 4,5 \times 10^{-6} n - 4,486 \times 10^{-2} \tag{5.13}$$

Essa equação representa o erro máximo para as estimativas. Um polinômio de segundo grau foi escolhido como modelo teórico porque o comportamento dos erros está associado ao comportamento da aplicação. Como o tempo de execução previsto pelo modelo analítico (δ) é uma estimativa para a distribuição ideal das cargas computacionais, o valor calculado pelo modelo pode ser definido como o valor mínimo. Desse modo, o tempo de execução pode ser estimado da seguinte forma: $\left[\delta^{PGF}(p,n),\ \delta^{PGF}(p,n) + \Delta(n)\right]$. O resultado das predições pode ser visto na Tabela 5.12.

O gráfico da Figura 5.6 apresenta o tempo de execução estimado pelo modelo de predição e os valores medidos para os processos que demoraram mais e menos tempo, utilizando a versão estática do VRP. Como é possível observar, até 30.000 partículas o plano de distribuição se mostrou adequado, garantindo uma pequena diferença entre o tempo dos processos analisados. Além desse equilíbrio, o tempo estimado pelo modelo de predição esteve ajustado. Para as outras configurações, é possível dizer que os valores calculados para cada f_i não foram adequados, prejudicando o balanceamento.

Tabela 5.12: Predições do tempo (segundos) de execução por intervalos.

	Número de Partículas			
	20.000	**40.000**	**50.000**	
Tempo mínimo medido	31,44	153,85	196,03	**Estratégia Estática**
Tempo máximo medido	34,21	174,75	232,14	
$\delta(p,n)$ **(mín. estimado)**	33,90	129,57	200,88	
$\delta + \Delta$ **(máx. estimado)**	34,21	174,75	232,14	
Δ (erro)	0,31	45,18	31,26	
$g_{t,b}$	4,2			
$g_{t,i}$	8,1			
f	0,123	0,123	0,123	
Tempo total	41,27	191,89	251,65	
Tempo mínimo medido	32,53	127,46	198,06	**Estratégia Dinâmica**
Tempo máximo medido	33,17	129,95	201,33	
$\delta(p,n)$ **(mín. estimado)**	32,54	127,87	198,25	
$\delta + \Delta$ **(máx. estimado)**	33,17	129,54	201,14	
Δ (erro)	0,63	2,08	3,08	
$g_{t,b}$	4,25	4,24	4,24	
$g_{t,i}$	7,55	7,95	7,95	
f	0,119	0,122	0,122	
Tempo total	40,15	143,35	219,52	

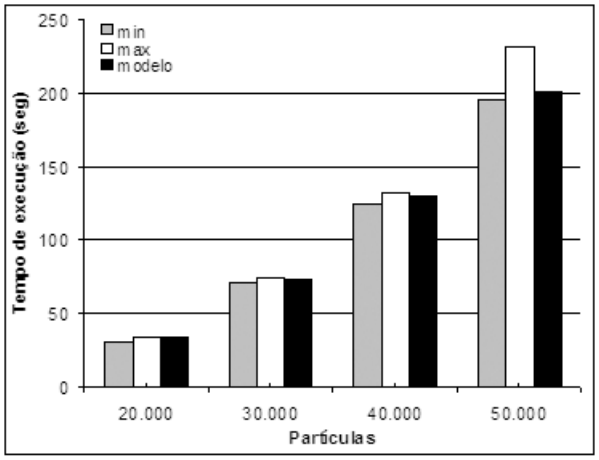

Figura 5.6: Distribuição estática.

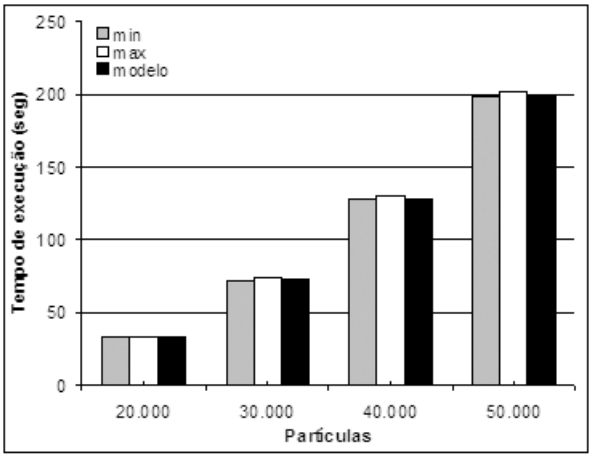

Figura 5.7: Distribuição dinâmica.

Os resultados das predições para a distribuição dinâmica são apresentados no gráfico da Figura 5.7. Como podemos observar, a distância entre os tempos mínimos e máximos diminuiu. Além disso, o tempo total de execução foi reduzido em até 14%, no caso de 50.000 partículas.

Já o gráfico da Figura 5.8 apresenta o balanceamento alcançado na distribuição das cargas computacionais. As barras mostram a distância dos tempos de execução de cada um dos processos em relação ao tempo estimado pelo modelo (DFPT[2]). A maior diferença

[2]DFPT (**D**istance **F**rom **P**redicted **T**ime) representa a distância, em porcentagem, dos tempos medidos em relação ao tempo estimado pelo modelo de desempenho.

percentual entre o tempo estimado pelo modelo e o tempo medido nos testes experimentais não chegou a 2%.

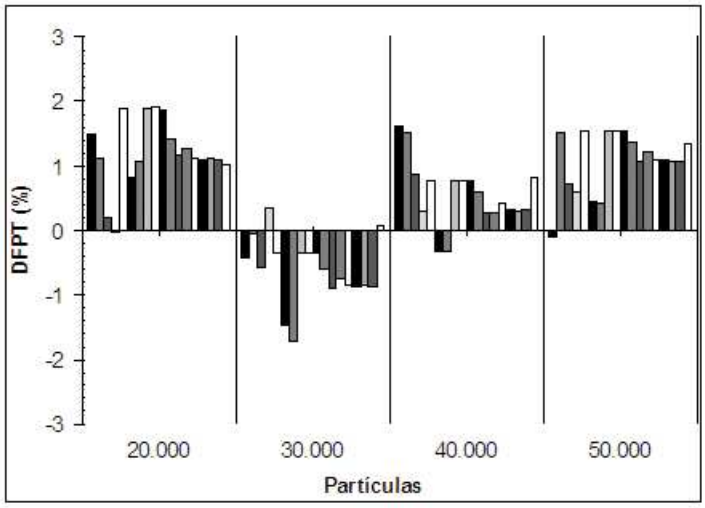

Figura 5.8: Distância entre os tempos medidos e o tempo estimado pelo modelo.

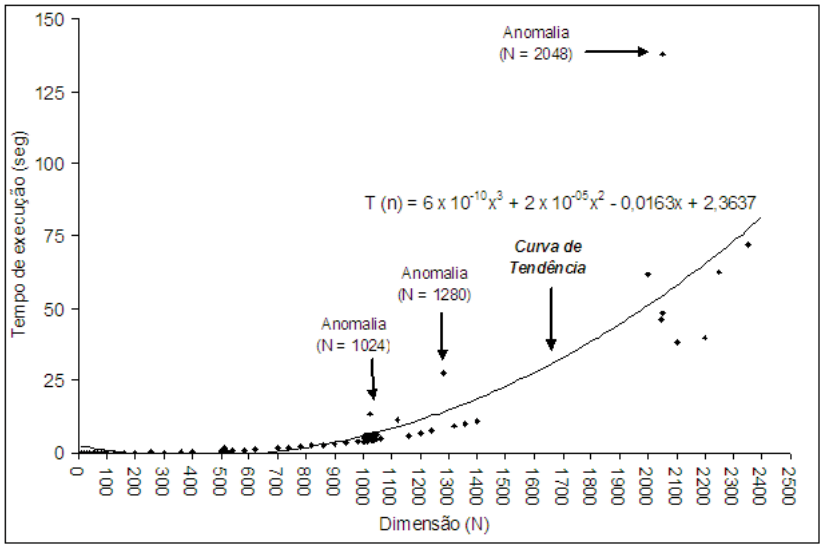

Figura 5.9: Análise detalhada do programa Mat-SS.

5.4 Dificuldades na Modelagem

O comportamento dos programas, muitas vezes, pode dificultar a construção dos modelos e comprometer a precisão dos mesmos. O gráfico da Figura 5.9 foi elaborado para ilustrar essa situação. Os resultados apresentados são do programa de multiplicação de matrizes. É possível observar que o programa modifica seu comportamento em algumas dimensões, em particular naquelas que são potência de 2, como 128, 512, 1.024 e 2.048. Uma análise detalhada foi realizada para investigar a razão do comportamento observado.

Neste estudo, descobriu-se que a causa do comportamento observado é o padrão de acesso à memória utilizado pelo algoritmo da multiplicação de matrizes.

O algoritmo tradicional utiliza uma estrutura de três loops aninhados para calcular o produto: $C = A \times B$, onde A, B e C são matrizes quadradas de ordem N. O trecho principal do algoritmo é apresentado a seguir:

```
for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        for (k=0; k<N; k++)
            C[i][j]=C[i][j] + A[i][k]*B[k][j];
```

Analisando a estrutura do algoritmo de multiplicação de matrizes, observa-se que é possível permutar os *loops* i, j e k gerando seis combinações possíveis: ijk, ikj, jik, jki, kij e kji. O programa testado utiliza a combinação apresentada no trecho de código acima, ijk. Nessa combinação, as matrizes A e C são percorridas por linha, e a matriz B, por coluna. Como a linguagem C armazena as matrizes por linha, percebe-se que o acesso à matriz B não está adequado. Testes com a versão ikj do programa de multiplicação de matrizes não apresentaram as anomalias da versão ijk.

Outro fato a ser observado diz respeito ao desempenho das máquinas de mesma configuração. O tempo de execução dos processos que executam um programa pode ser ligeiramente diferente mesmo em máquinas idênticas. As causas dessas diferenças podem, por exemplo, estar relacionadas a eventuais problemas com o *hardware* e às diferenças na carga computacional das máquinas. Para visualizar essa situação, medimos o tempo gasto em cada iteração do programa de multiplicação de matrizes implementado com a estratégia SS.

O gráfico da Figura 5.10 apresenta a variação dos tempos medidos ao dividir a execução do programa de multiplicação de matrizes entre 8 máquinas idênticas. A dimensão das matrizes utilizadas nos testes experimentais foi 1.300. É possível verificar que, além de existir uma diferença significativa entre o desempenho de cada máquina, ocorre uma variação também no tempo de execução de um mesmo processo. Pelo gráfico é possível visualizar 8 padrões diferentes em relação ao tempo de execução. Cada um desses conjuntos de dados faz referência a um processo distinto. Portanto, elaborar modelos com alta precisão é uma tarefa extremamente difícil.

5.5 Balanceamento de Carga

Nessa seção, apresentamos um tipo de análise que pode ser realizada com o auxílio da metodologia PEMPIs-Het para avaliações sobre resultados de balanceamento de carga. A abordagem adotada pela metodologia consiste em avaliar o tempo de execução de cada um dos processos que participam da execução do programa (LAINE; MIDORIKAWA, 2008a).

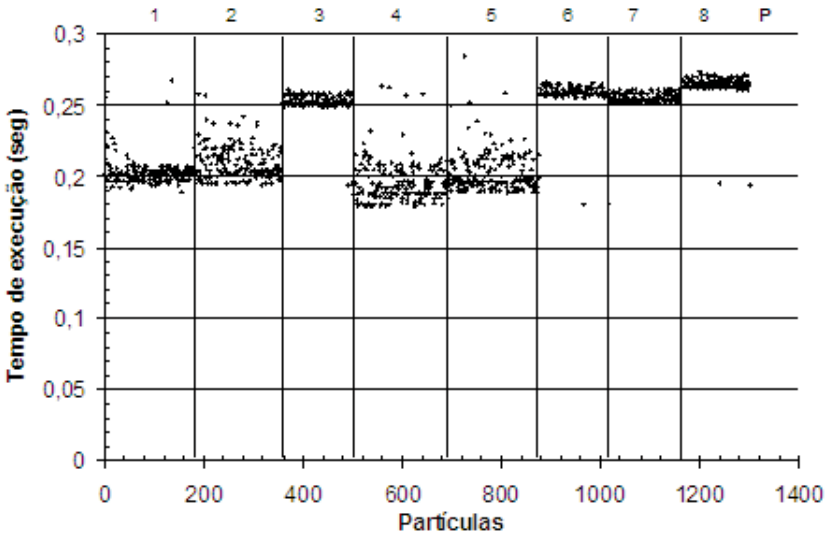

Figura 5.10: Análise por iteração do programa Mat-SS.

O intuito das estratégias, como já comentado no Capítulo 4, é aproximar o tempo de execução das tarefas para que se estabeleça um sincronismo de execução. As estratégias devem buscar essa sincronização independente da capacidade da máquina onde o processo esteja sendo executado e da quantidade de trabalho processado. Para verificar a precisão da distribuição das cargas computacionais, formulamos uma métrica denominada DFAT, detalhada na próxima seção.

Para o desenvolvimentos das avaliações, utilizamos os programas de multiplicação de matrizes e de simulação de partículas em um campo gravitacional. Embora a estratégia de balanceamento analisada tenha sido a VRP, qualquer outra abordagem poderia ter sido utilizada para demonstrar a aplicabilidade da metodologia.

5.5.1 DFAT – Uma Métrica para Avaliação da Precisão das Estratégias de Balanceamento

Para comparar os resultados preditos através dos modelos de predição e os valores reais medidos com a execução das aplicações, uma métrica denominada *DFAT* foi definida. Essa métrica representa a distância percentual do tempo de execução individual de cada processo em relação ao tempo médio de todas as tarefas que colaboram na solução do problema. A métrica $DFAT$ (*Distance From Average Time*) é definida através da seguinte equação:

$$DFAT_p = \frac{x_p - \overline{x}}{\overline{x}} \times 100\% \tag{5.14}$$

sendo p a identificação do processo, x_p o tempo de execução do mesmo e $\overline{x}$ o tempo médio de execução das tarefas.

5.5.2 Multiplicação de Matrizes

Durante os testes experimentais realizados com o programa de multiplicação de matrizes (MM), utilizamos 18 processos escravos para realizar os cálculos envolvidos na computação. O ambiente computacional contou com 4 máquinas Intel, 7 Bios e 7 Taurus. Cada processo foi executado em uma máquina distinta, e o ambiente esteve dedicado durante o processamento do programa. Duas abordagens da estratégia VRP foram avaliadas. No gráfico da Figura 5.11, apresentamos os resultados para a versão estática de balanceamento. Já no gráfico da Figura 5.12, mostramos o balanceamento gerado com a abordagem dinâmica da estratégia VRP.

Os gráficos apresentam a variação do tempo de execução de cada um dos processos em relação ao tempo médio. Essa variação (DFAT) permite observar se a distribuição das cargas computacionais foi adequada ou não. O ajuste no balanceamento é sensível ao mudar da abordagem estática para a dinâmica. Na versão estática alguns processos apresentaram um DFAT em torno de 25%. Já na versão dinâmica, o maior DFAT observado é de 5%. Além disso, os valores do DFAT para a versão dinâmica variam entre -5% a +5%, enquanto na versão estática essa variação está entre -13% a +25%. Isso mostra um equilíbrio maior entre os tempos de execução dos processos na distribuição dinâmica. Todos os tempos medidos e os respectivos DFAT estão nas Tabelas 5.13 e 5.14. Essas tabelas também apresentam o desvio padrão calculado para os valores da amostragem utilizada.

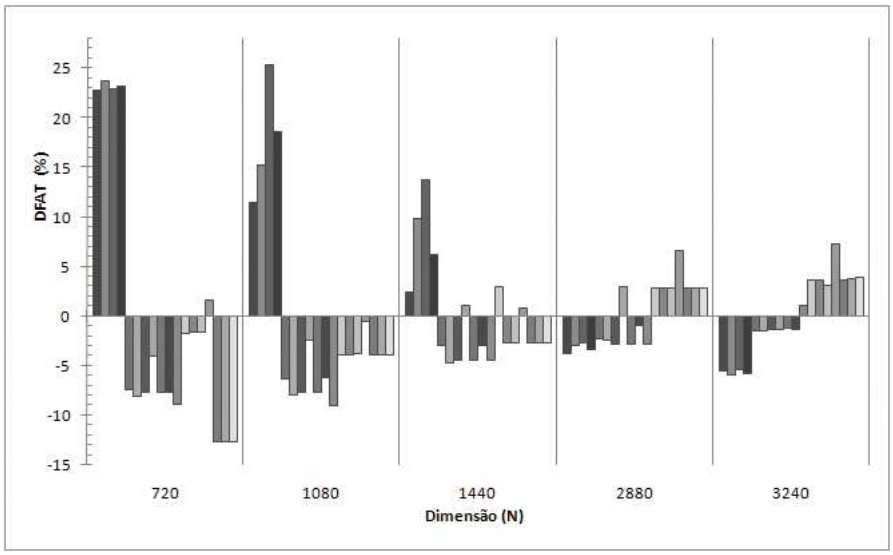

Figura 5.11: Resultados do balanceamento estático - MM.

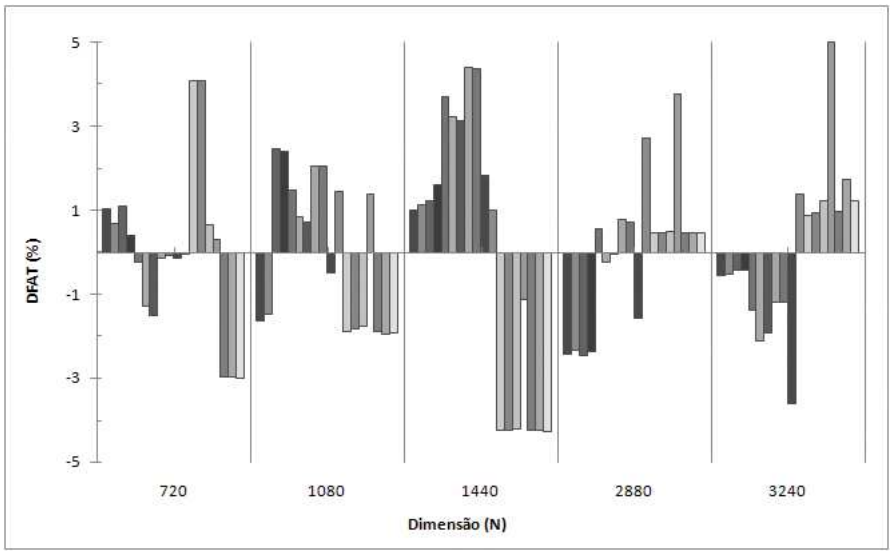

Figura 5.12: Resultados do balanceamento dinâmico - MM.

Tabela 5.13: MM: tempos medidos e estimados (segundos) com o VRP estático.

	720		**1.080**		**1.440**		**2.880**		**3.240**	
Escravo	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**
1	0,71	22,86	2,15	11,49	4,52	2,48	33,06	-3,70	47,04	-5,44
2	0,72	23,70	2,22	15,22	4,84	9,91	33,32	-2,96	46,79	-5,95
3	0,71	22,90	2,41	25,41	5,01	13,72	33,43	-2,62	47,09	-5,34
4	0,71	23,16	2,28	18,57	4,68	6,20	33,19	-3,34	46,87	-5,78
5	0,54	-7,43	1,81	-6,25	4,28	-2,98	33,54	-2,31	49,04	-1,42
6	0,53	-8,02	1,77	-7,97	4,20	-4,69	33,51	-2,39	49,00	-1,51
7	0,53	-7,71	1,78	-7,62	4,21	-4,37	33,36	-2,82	49,08	-1,34
8	0,56	-4,04	1,88	-2,37	4,45	1,07	35,35	2,97	49,12	-1,27
9	0,53	-7,70	1,78	-7,63	4,21	-4,38	33,37	-2,79	49,14	-1,23
10	0,53	-7,67	1,81	-6,21	4,28	-2,95	34,01	-0,95	49,12	-1,26
11	0,53	-8,85	1,75	-9,02	4,21	-4,38	33,38	-2,77	50,31	1,12
12	0,57	-1,67	1,85	-3,82	4,54	3,01	35,31	2,85	51,57	3,66
13	0,57	-1,57	1,85	-3,91	4,29	-2,70	35,30	2,83	51,56	3,64
14	0,57	-1,59	1,85	-3,80	4,29	-2,71	35,30	2,83	51,34	3,20
15	0,59	1,59	1,91	-0,56	4,44	0,87	36,62	6,65	53,41	7,36
16	0,51	-12,64	1,85	-3,84	4,29	-2,69	35,31	2,84	51,60	3,72
17	0,51	-12,62	1,85	-3,85	4,29	-2,69	35,31	2,86	51,66	3,85
18	0,51	-12,69	1,85	-3,85	4,29	-2,71	35,31	2,84	51,74	4,00
Desvio Padrão	**0,077**	—	**0,197**	—	**0,233**	—	**1,109**	—	**2,000**	
Tempo Médio	**0,58**	—	**1,93**	—	**4,41**	—	**34,33**	—	**49,75**	
Tempo Total	**1,05**		**3,07**		**6,39**		**38,66**		**54,21**	

Tabela 5.14: MM: tempos medidos e estimados (segundos) com o VRP dinâmico.

	720		**1.080**		**1.440**		**2.880**		**3.240**	
Escravo	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**
1	0,61	1,04	2,00	-1,64	4,53	1,02	33,42	-2,42	48,22	-0,56
2	0,61	0,70	2,00	-1,47	4,54	1,15	33,44	-2,35	48,24	-0,54
3	0,61	1,10	2,08	2,47	4,54	1,21	33,40	-2,46	48,30	-0,41
4	0,61	0,39	2,08	2,39	4,56	1,60	33,44	-2,36	48,28	-0,44
5	0,61	-0,23	2,06	1,48	4,65	3,72	34,43	0,54	47,82	-1,39
6	0,60	-1,30	2,05	0,84	4,63	3,22	34,17	-0,22	47,48	-2,11
7	0,60	-1,52	2,04	0,73	4,63	3,14	34,24	0,00	47,56	-1,93
8	0,61	-0,13	2,07	2,05	4,68	4,41	34,51	0,78	47,92	-1,18
9	0,61	-0,09	2,07	2,06	4,68	4,37	34,50	0,73	47,91	-1,20
10	0,61	-0,13	2,02	-0,49	4,57	1,83	33,70	-1,58	46,75	-3,61
11	0,61	-0,02	2,06	1,45	4,53	1,00	35,18	2,73	49,16	1,37
12	0,63	4,08	1,99	-1,90	4,29	-4,26	34,41	0,48	48,92	0,87
13	0,63	4,09	1,99	-1,82	4,29	-4,26	34,40	0,46	48,96	0,95
14	0,61	0,66	1,99	-1,76	4,30	-4,23	34,41	0,49	49,09	1,23
15	0,61	0,32	2,06	1,38	4,43	-1,13	35,54	3,78	50,93	5,01
16	0,59	-2,99	1,99	-1,90	4,29	-4,26	34,41	0,47	48,97	0,98
17	0,59	-2,97	1,99	-1,95	4,29	-4,26	34,40	0,46	49,33	1,72
18	0,59	-2,99	1,99	-1,92	4,29	-4,27	34,40	0,46	49,10	1,24
Desvio Padrão	**0,012**	—	**0,036**	—	**0,151**	—	**0,587**	—	**0,929**	
Tempo Médio	**0,61**	—	**2,03**	—	**4,48**	—	**34,25**	—	**48,50**	
Tempo Total	**0,96**		**2,82**		**5,94**		**39,08**		**55,38**	

5.5.3 Partículas sob Forças Gravitacionais - PGF

A análise realizada com o programa de multiplicação de matrizes foi repetida para o programa que simula a iteração de partículas em um campo gravitacional (PGF). O programa foi implementado para utilizar as abordagens estática e dinâmica da estratégia VRP. A quantidade de processos utilizado na execução desse programa é o mesmo dos testes realizados com a aplicação de multiplicação de matrizes. No entanto, o ambiente computacional possui 4 máquinas Intel, 6 Bios e 8 Taurus.

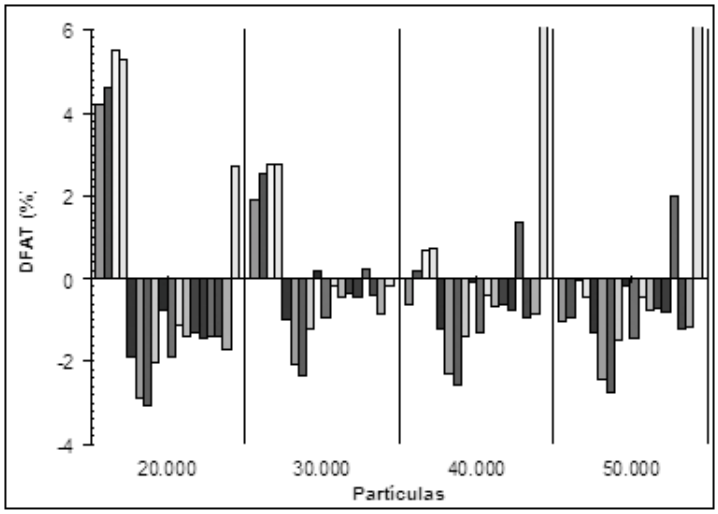

Figura 5.13: Resultados do balanceamento estático - PGF.

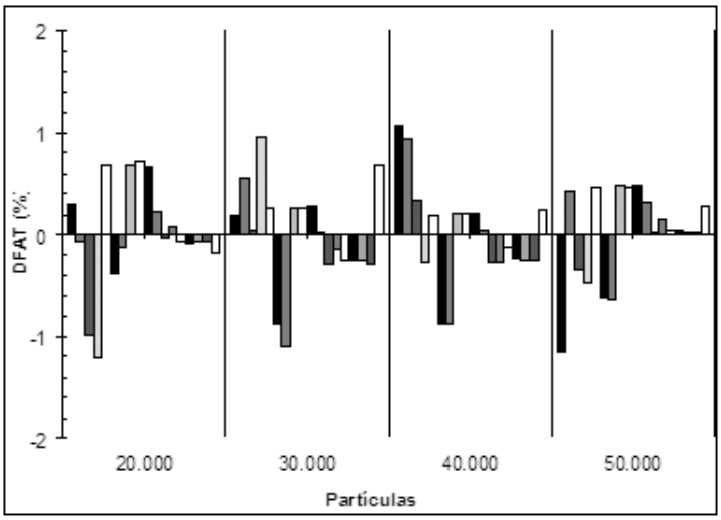

Figura 5.14: Resultados do balanceamento dinâmico - PGF.

Assim como no programa de multiplicação de matrizes, os melhores resultados foram gerados pela estratégia dinâmica de balanceamento, conforme ilustra o gráfico da Figura 5.14. Esse balanceamento gerou uma distribuição mais homogênea das cargas computacionais, reduzindo a diferença máxima de $5,5\%$ na versão estática (gráfico da Figura

5.13) para aproximadamente 1% na versão dinâmica. Da mesma forma que no programa anterior, o equilíbrio gerado com a abordagem dinâmica representa um ajuste de 80% em relação à versão estática.

Além disso, o tempo total de execução da aplicação também diminuiu com o uso da estratégia dinâmica. Para todos os valores de n, houve uma redução no tempo total medido. A maior redução ocorreu para $n = 50.000$ e representou 15%. O maior desvio padrão encontrado ocorreu para $n = 50.000$. Na versão estática, esse valor chegou a 7,66, enquanto na estratégia dinâmica ficou em 0,9. Todos os resultados podem ser visualizados nas Tabelas 5.15 e 5.16.

Tabela 5.15: PGF: tempos medidos e estimados (segundos) com o VRP estático.

	20.000		**30.000**		**40.000**		**50.000**	
Escravo	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**
1	33,794	4,217	73,714	1,923	156,884	-0,627	199,489	-1,038
2	33,914	4,586	74,148	2,522	158,181	0,194	199,665	-0,951
3	34,208	5,493	74,328	2,771	158,961	0,688	201,516	-0,032
4	34,147	5,306	74,335	2,781	159,030	0,732	200,699	-0,438
5	31,818	-1,878	71,612	-0,984	155,971	-1,206	198,957	-1,302
6	31,490	-2,888	70,842	-2,049	154,245	-2,299	196,655	-2,444
7	31,437	-3,054	70,622	-2,353	153,852	-2,548	196,032	-2,753
8	31,768	-2,031	71,448	-1,211	155,716	-1,367	198,606	-1,476
9	32,189	-0,732	72,462	0,192	157,750	-0,079	201,266	-0,157
10	31,817	-1,881	71,633	-0,955	155,852	-1,281	198,694	-1,433
11	32,061	-1,130	72,210	-0,156	157,284	-0,374	200,637	-0,469
12	31,970	-1,409	72,004	-0,441	156,806	-0,677	200,028	-0,771
13	31,997	-1,327	72,075	-0,343	156,885	-0,627	200,164	-0,704
14	31,963	-1,431	72,000	-0,448	156,706	-0,740	199,942	-0,813
15	31,971	-1,407	72,478	0,214	160,021	1,360	205,610	1,998
16	31,969	-1,411	72,008	-0,436	156,369	-0,954	199,131	-1,216
17	31,865	-1,733	71,695	-0,868	156,486	-0,880	199,247	-1,159
18	33,305	2,709	72,208	-0,159	174,747	10,687	232,138	15,158
Desvio Padrão	**0,930**	—	**1,080**	—	**4,363**	—	**7,665**	
Tempo Médio	**32,427**	—	**72,323**	—	**157,875**	—	**201,582**	
Tempo Total	**41,272**		**89,823**		**191,897**		**251,651**	

Tabela 5.16: PGF: tempos medidos e estimados (segundos) com o VRP dinâmico.

	20.000		**30.000**		**40.000**		**50.000**	
Escravo	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**	**Tempo**	**DFAT**
1	33,029	0,295	72,633	0,182	129,948	1,061	198,058	-1,149
2	32,910	-0,068	72,902	0,554	129,794	0,941	201,223	0,431
3	32,608	-0,982	72,525	0,034	129,000	0,324	199,689	-0,335
4	32,532	-1,213	73,186	0,945	128,234	-0,273	199,405	-0,476
5	33,156	0,682	72,693	0,266	128,843	0,201	201,300	0,469
6	32,804	-0,390	71,871	-0,869	127,462	-0,873	199,133	-0,612
7	32,888	-0,133	71,700	-1,104	127,462	-0,873	199,086	-0,636
8	33,158	0,686	72,691	0,263	128,845	0,203	201,326	0,482
9	33,166	0,712	72,691	0,263	128,847	0,204	201,299	0,469
10	33,151	0,666	72,694	0,268	128,849	0,206	201,315	0,477
11	33,003	0,215	72,510	0,013	128,630	0,035	200,984	0,311
12	32,923	-0,028	72,290	-0,290	128,232	-0,274	200,390	0,015
13	32,957	0,075	72,395	-0,146	128,232	-0,274	200,660	0,150
14	32,906	-0,077	72,318	-0,252	128,426	-0,123	200,440	0,040
15	32,901	-0,094	72,309	-0,264	128,283	-0,234	200,428	0,034
16	32,906	-0,078	72,311	-0,262	128,248	-0,261	200,412	0,026
17	32,905	-0,083	72,295	-0,283	128,275	-0,241	200,412	0,026
18	32,872	-0,183	72,994	0,681	128,909	0,253	200,914	0,277
Desvio Padrão	**0,168**	—	**0,356**	—	**0,628**	—	**0,911**	
Tempo Médio	**32,932**	—	**72,500**	—	**128,584**	—	**200,360**	
Tempo Total	**40,157**		**83,679**		**143,353**		**219,527**	

5.6 Analisando, Modelando e Comparando Diferentes Modelos de Soluções Paralelas

Além de permitir avaliações quantitativas sobre o desempenho de programas paralelos, a metodologia PEMPIs-Het também admite que os modelos analíticos sejam utilizados para comparar diferentes abordagens ou modelos de soluções distribuídas. Nesta comparação, o objetivo é projetar qual modelo de organização do programa é o mais eficiente para ser utilizado no ambiente. Na Figura 5.15 apresentamos o mecanismo utilizado para comparação dos modelos de soluções. Inicialmente, o problema é implementado e várias soluções são criadas para a aplicação. O modelo analítico de desempenho é gerado para cada solução, com as técnicas da metodlogia PEMPIs-Het. Análises sobre o desempenho das soluções são realizadas e os resultados são comparados. Assim, uma previsão do melhor modelo de solução é gerada. Nesta seção, demonstramos como os modelos analíticos de desempenho podem ser aplicados para este objetivo.

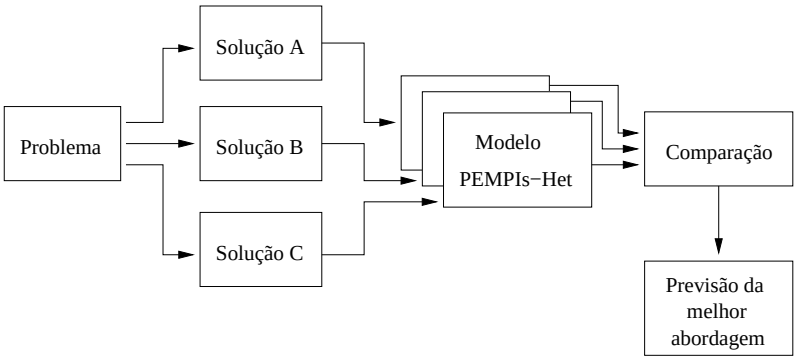

Figura 5.15: Mecanismo utilizado para comparar modelos de soluções.

5.6.1 Análise das Estruturas

Para a realização das análises propostas utilizamos quatro diferentes versões para o tradicional programa de multiplicação de matrizes, denominadas Mat-SS, Mat-VRP-CSS, Mat-VRP-CSS-Adapt e Mat-VRP. A estrutura organizacional de cada solução foi apresentada e discutida na Seção 4.10. A influência da estrutura da solução no desempenho da aplicação pode ser facilmente observada com os dados apresentados nos gráficos das Figuras 5.16 e 5.17.

Os resultados apresentados no gráfico da Figura 5.16 foram obtidos sem carga computacional extra no sistema. As versões da aplicação tiveram exclusividade no uso do ambiente (*cluster* dedicado). Já o gráfico da Figura 5.17 apresenta o desempenho de cada uma das versões no mesmo ambiente mas com carga artificial extra (*cluster* compartilhado). Essa carga foi criada para simular um ambiente de execução compartilhado, onde outras aplicações podem disputar os recursos computacionais com o programa de multi-

plicação de matrizes. Em cada máquina selecionada para executar o programa gerador da carga o consumo de CPU e memória se manteve constante durante todo o tempo. Em todos os testes experimentais utilizamos 20 unidades de processamento, sendo 4 das máquinas Intel, 6 das Taurus e 10 das máquinas Bios. A carga artificial foi executada em 1 máquina Intel e 3 máquinas Bio. Com exceção da versão Mat-VRP, todas as demais apresentaram tempo menor no *cluster* dedicado, como devia ocorrer.

Em ambos os testes a versão adaptativa Mat-VRP-CSS-Adapt foi a que apresentou os melhores índices de desempenho, levemente superior à versão Mat-VRP-CSS. Isso pode ser explicado pois embora a versão adaptativa promova o ajuste das cargas computacionais, o que melhora o desempenho do sistema, o tempo gasto na estratégia pode comprometer o desempenho da solução quando o sistema apresenta pouca variação de carga, como avaliado nos testes experimentais. Nesse caso, a otimização é pequena e de pouca influência no desempenho final da aplicação.

As versões Mat-SS e Mat-VRP tiveram um desempenho consideravelmente inferior às duas primeiras e, portanto, podem ser descartadas como opções para a organização da solução do programa. Uma explicação para os resultados ruins da versão *self-scheduling* é que ela tende a perder desempenho quando a carga do sistema se mantém constante e o tamanho do problema aumenta significativamente. Isto porque a quantidade de mensagens necessárias para a distribuição das cargas individuais aumenta na mesma proporção e, portanto, o tempo de transmissão dos pacotes na rede de interconexão pode comprometer o desempenho global da aplicação. Já a versão Mat-VRP por promover uma distribuição estática das tarefas é extremamente dependente da precisão dos modelos de desempenho utilizados para elaborar o VRP. Como a relação de desempenho entre as máquinas se altera ao variar o tamanho do problema e os valores do VRP não são modificados e tendência é que a estratégia não acompanhe essas alterações.

No cenário de *cluster* dedicado, a diferença entre o desempenho da melhor (Mat-VRP-CSS-Adapt) e da pior estratégia (Mat-VRP), com matrizes de 3000×3000 elementos, chegou próximo a 500%. O tempo de execução da versão adaptativa foi de $38,43$ segundos e da Mat-VRP foi de $227,95$ segundos. Já no cenário com a carga artificial, a versão adaptativa teve um desempenho quase 4 vezes melhor. Os tempos médios medidos para as versões foram de $199,46$ e $51,99$ segundos, respectivamente.

5.6.2 Modelagem

A partir dos resultados experimentais podemos utilizar a metodologia PEMPIs-Het para modelar o comportamento das diferentes versões avaliadas e estimar o desempenho da aplicação em função da dimensão da matriz, por exemplo. Essa predição pode ajudar a definir, extrapolando os valores associados a dimensão das matrizes, qual o melhor modelo organizacional para a solução do problema, levando em consideração as características do ambiente de execução. Conforme descrito em (LAINE; MIDORIKAWA, 2007b), os mode-

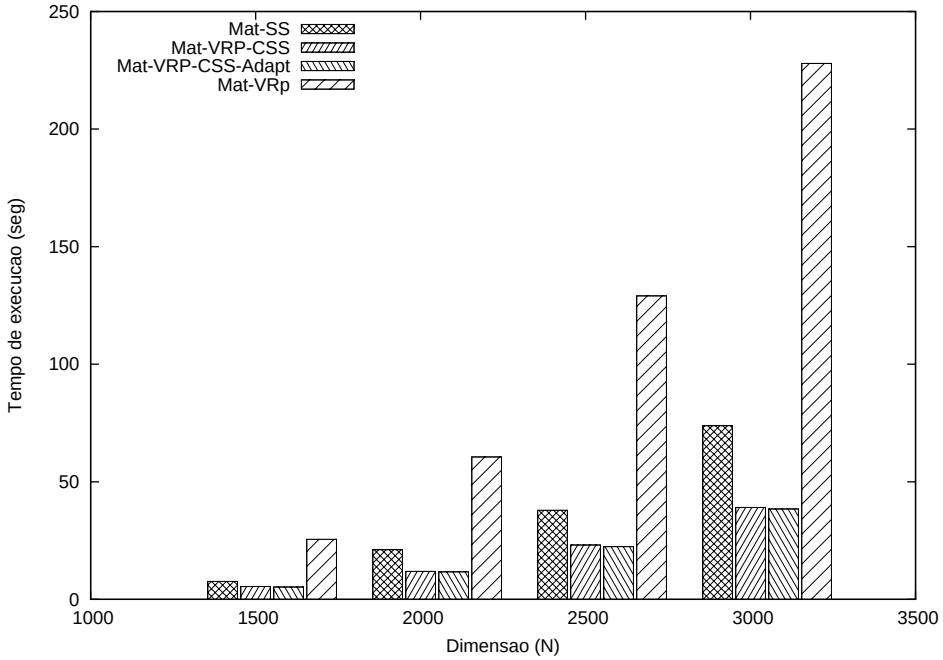

Figura 5.16: Desempenho medido no *cluster* dedicado.

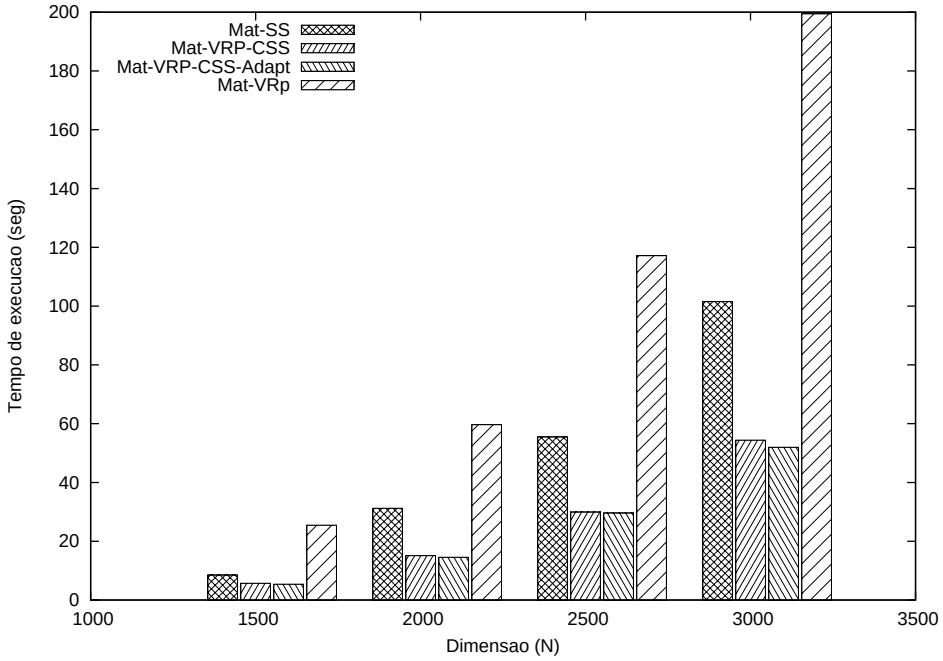

Figura 5.17: Desempenho medido no *cluster* compartilhado.

los analíticos de desempenho são gerados a partir de informações sobre a complexidade computacional do programa. Uma análise sobre o código fonte permite obter o modelo teórico da aplicação e, conseqüentemente, a estrutura geral dos modelos analíticos. No caso da multiplicação de matrizes, todas as versões fizeram uso do modelo tradicional de computação, cuja complexidade algorítmica é $O(n^3)$. Portanto, o modelo teórico da aplicação, escrito em função da dimensão das matrizes (n) e da quantidade de processos

(p), é dado por:

$$\delta^{mat}(n,p) = \frac{C3}{p} \times n^3 + \frac{C2}{p} \times n^2 + \frac{C1}{p} \times n + \frac{C0}{p} \tag{5.15}$$

Neste artigo, em particular, reduzimos a modelagem para direcionar a análise do tempo de execução em função da dimensão das matrizes apenas. Logo, o desempenho da aplicação pode ser representado através do seguinte modelo:

$$\delta^{mat}(n) = C3 \times n^3 + C2 \times n^2 + C1 \times n + C0 \tag{5.16}$$

A partir do modelo teórico podemos gerar os modelos individuais da aplicação para cada versão modelada, utilizando técnicas de ajuste de curvas e os dados obtidos nos testes experimentais (LAINE; MIDORIKAWA, 2008b). Assim, é possível obter os valores para cada uma das constantes (C_i) do modelo teórico e realizar predições de desempenho em função do tamanho das matrizes. Após aplicar as técnicas contempladas na metodologia PEMPIs-Het obtivemos as constantes apresentadas nas Tabelas 5.17 e 5.18. Portanto, a instanciação do modelo de desempenho de cada uma das versões ocorre com a substituição desses valores no modelo teórico apresentado anteriormente.

Tabela 5.17: Modelagem para o *cluster* dedicado.

	C0	**C1**	**C2**	**C3**
SS	-172,18999	0,2530733	-0,0001206	$2{,}119 \times 10^{-8}$
VRP-CSS	12,119999	-0,0166867	0,0000078	$2{,}67 \times 10^{-10}$
VRP-CSS-Ad.	0,47	0,0001133	-0,0000001	$1{,}427 \times 10^{-9}$
VRP	152,93999	-0,2136767	0,0000921	$-4{,}173 \times 10^{-9}$

Tabela 5.18: Modelagem para o *cluster* compartilhado.

	C0	**C1**	**C2**	**C3**
SS	-254,62999	0,3539733	-0,0001597	$2{,}711 \times 10^{-8}$
VRP-CSS	-32,829998	0,0479167	-0,0000234	$5{,}693 \times 10^{-9}$
VRP-CSS-Ad.	-1,29	-0,00024	0,0000002	$1{,}92 \times 10^{-9}$
VRP	48,529998	-0,07239	0,0000351	$1{,}92 \times 10^{-9}$

5.6.3 Comparando Modelos de Estruturas

A etapa seguinte consiste em estimar o desempenho de cada versão, extrapolando valores para a dimensão das matrizes. Essa atividade permite prever, de acordo com a precisão dos modelos, o comportamento das versões modeladas no ambiente utilizado para os testes experimentais. Os resultados das projeções são apresentados nos gráficos das Figuras 5.18 e 5.19.

Pelas projeções é possível notar que em ambos os cenários algumas estratégias apresentam bons resultados para menores valores de n (dimensão) mas pioram a medida que

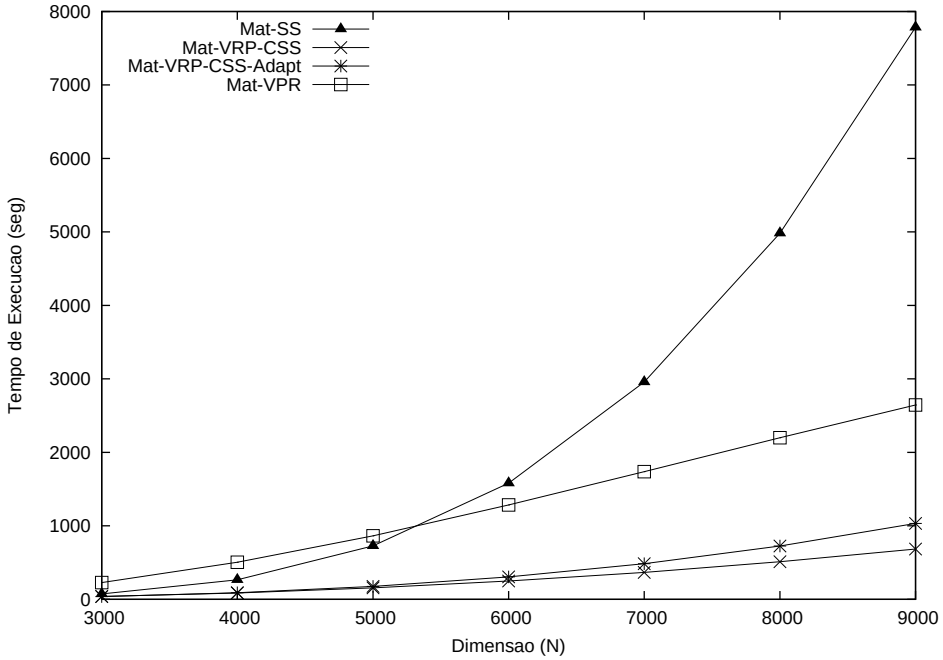

Figura 5.18: Predições no *cluster* dedicado.

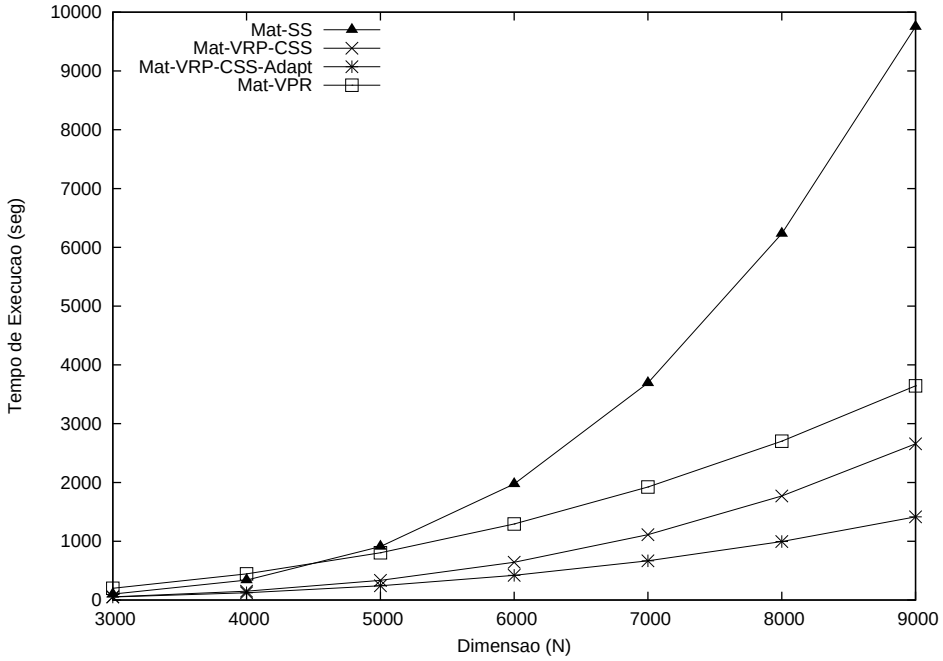

Figura 5.19: Predições no *cluster* compartilhado.

os valores aumentam. Esse é o caso, por exemplo, da versão Mat-SS. Como se esperava, pelos resultados dos testes experimentais, esse modelo não é interessante para a aplicação no ambiente avaliado, com as características que foram destacadas. No cenário de *cluster* dedicado as versões Mat-VRP-CSS-Adapt e Mat-VRP-CSS apresentam uma projeção muito semelhante para o tempo de execução. Pelos valores obtidos nos testes experimentais (Figura 5.16), esperávamos que a versão adaptativa tivesse melhores resultados. No

entanto, as projeções mostraram que a partir de n = 4000 a estratégia Mat-VRP-CSS apresenta melhores resultados (86, 60 seg. contra 90, 63 seg. da abordagem adaptativa), invertendo a tendência esperada. Isto pode ser explicado pois o método de ajuste de curvas empregado captura o grau de crescimento das curvas do tempo de execução e define o modelo baseado no ajuste dos maiores tempos. Uma possível causa do maior crescimento do tempo de execução medido, em relação à versão Mat-VRP-CSS, pode ser atribuída ao tempo gasto com a maior quantidade de cálculos para a definição da capacidade de processamento efetiva (Cpe) e do trabalho adequado, a medida que os valores de n aumentam. Isto porque a quantidade de fases necessária para a distribuição dos dados também aumenta em função do tamanho de n, e o cálculo dos valores para Cpe é realizado ao final de cada uma das fases.

Para mostrar como a estratégia adaptativa ajusta a divisão do trabalho, apresentamos, na Figura 5.20, a Cpe das máquinas que compõem o nosso ambiente de teste ao executar tarefas de mesmo tamanho. Os coeficientes indicam que as máquinas Intel possuem quase três vezes mais capacidade de processamento que as máquinas Bio, para o programa executado. Já as máquinas Bio conseguem executar quatro vezes mais trabalho que as máquinas Taurus. Esses valores foram obtidos com as máquinas dedicadas aos testes. Baseado nesses dados, a estratégia determina a quantidade de trabalho ideal a cada processo para que eles tenham o mesmo tempo de execução. Essa sincronização pode melhorar o desempenho da aplicação ao eliminar eventuais tempos gastos em esperas de execução. Na Figura 5.21, mostramos a convergência dos tempos de processamento de dois processos sendo executados em nós computacionais diferentes do *cluster*, após o ajuste realizado com base nos cálculos da Cpe. O gráfico ilustra o momento em que a estratégia de balanceamento de carga adaptativa modifica a distribuição e ajusta os tempos de execução.

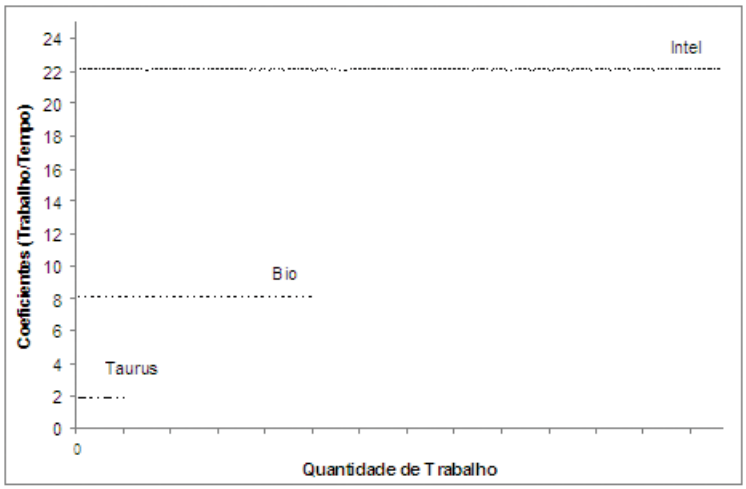

Figura 5.20: Capacidade de processamento efetiva das máquinas.

No cenário apresentado na Figura 5.19 a versão adaptativa promete melhores resulta-

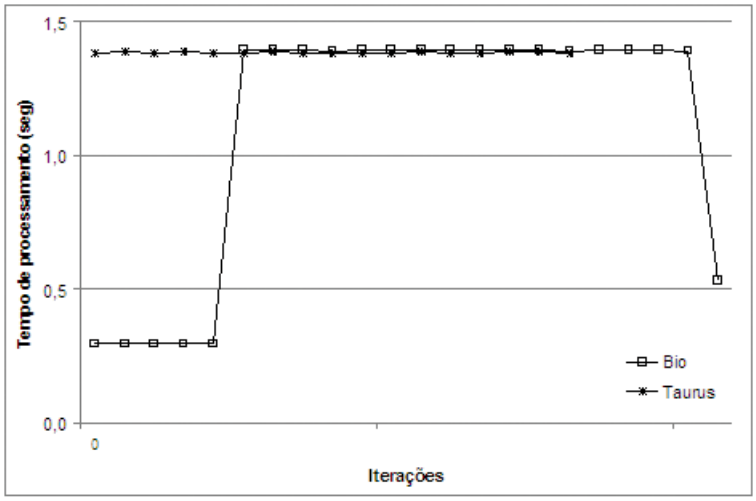

Figura 5.21: Ajuste dos tempos de execução com o cálculo da Cpe.

dos para todos os valores de n, de acordo com o modelo de desempenho obtido. Portanto, essas estimativas permitem concluir que: i) em ambos os cenários as versões Mat-SS e Mat-VRP não são boas opções para organizar a solução da aplicação no ambiente considerado; ii) no cenário com uma carga computacional extra, concorrendo pelos recursos computacionais (mais próximo de uma situação real em ambientes distribuídos), a versão adaptativa consegue melhorar o desempenho da aplicação ajustando a divisão e distribuição das tarefas.

Dessa forma, as estratégias descritas na metodologia PEMPIs-Het, aplicadas da forma como demonstramos nesse artigo, podem ser utilizadas para comparar diferentes estruturas de programas paralelos e ajudar a decidir qual modelo é o mais eficiente para a aplicação em um ambiente computacional com características conhecidas.

5.7 Considerações Finais

Este capítulo apresentou os resultados obtidos em atividades de modelagem e predição de desempenho em ambientes homogêneos e heterogêneos. Algumas abordagens para a predição de desempenho foram avaliadas. Inicialmente, o tempo de execução foi estimado por valores pontuais. A seguir, mostrou-se uma segunda opção para as estimativas, utilizando intervalos de predição.

A aplicabilidade das estratégias definida na metodologia PEMPIs-Het foi confirmada pelos bons resultados alcançados. Além disso, avaliou-se também o uso dos modelos de desempenhos em atividades de balanceamento de carga. Demonstramos também a capacidade da metodologia PEMPIs-Het em avaliar diferentes estruturas de programas paralelos e apontar qual estrutura e/ou, conseqüentemente, solução é capaz de oferecer o melhor desempenho computacional para um problema. Para as avaliações e predições de desempenho utilizamos quatro diferentes versões do tradicional programa de multiplicação

de matrizes. Cada uma das versões implementa um modelo diferente de balanceamento de carga computacional. O reflexo dessa organização é visualizado no desempenho da aplicação, conforme ilustram os gráficos das Figuras 5.16 e 5.17.

Em geral, as projeções apresentadas nas Figuras 5.18 e 5.19 confirmam a tendência do bom ou mau desempenho das estratégias avaliadas. No entanto, algumas estimativas indicam alterações no comportamento relatado pelos testes experimentais. Pelas projeções, a versão Mat-SS passa a ser a pior estratégia a medida que a dimensão das matrizes aumenta. Nos testes realizados a estratégia com o pior desempenho foi a Mat-VRP. Esse comportamento também pode ser observado entre as versões Mat-VRP-CSS e Mat-VRP-CSS-Adapt, conforme ilustra o gráfico da Figura 5.18. Pelos testes experimentais a estratégia Mat-VRP-CSS-Adapt seria a mais eficiente. Contudo, as projeções mostram que a partir de $n = 4000$ a versão Mat-VRP-CSS passa a apresentar melhor desempenho, invertendo a tendência esperada.

Portanto, essa análise permite comparar diferentes estruturas de soluções e prever, para um determinado ambiente computacional, qual estratégia é capaz de otimizar o desempenho da aplicação ao variar especificações do problema ou do sistema. Embora, nesse artigo, as análises ficaram em função do tamanho do problema, é possível modificar o modelo para considerar também a quantidade de processos. Logo, as análises de desempenho poderiam ser realizadas em função desses dois parâmetros. Uma descrição de como utilizar a metodologia para elaborar modelos em função de n (tamanho do problema) e p (número de processos) pode ser obtida em (MIDORIKAWA; OLIVEIRA; LAINE, 2005).

Ao estimar o comportamento de alternativas de soluções é possível melhorar o desenvolvimento de programas paralelos distribuídos e ajudar em decisões de implementação para sistemas complexos, na qual o desempenho computacional é um fator importante a ser considerado.

Capítulo 6

Conclusões

Neste capítulo, as principais conclusões sobre o trabalho desenvolvido são apresentadas e alguns resultados são discutidos. O capítulo também destaca as contribuições geradas pela tese e os possíveis trabalhos futuros, tanto na área de modelagem e predição de desempenho quanto em balanceamento de carga em sistemas distribuídos.

6.1 Discussão Final

O objetivo principal desta tese foi propor uma metodologia para auxiliar o desenvolvimento e a avaliação de aplicações paralelas em sistemas distribuídos. Para isso, algumas técnicas foram criadas para facilitar a modelagem e predição de desempenho das aplicações paralelas. Além disso, elaboramos estratégias de balanceamento de carga para melhorar o desempenho dos programas em ambientes distribuídos compartilhados ou dedicados.

Ao caracterizar o comportamento dos programas, é possível avaliar, através dos modelos analíticos, qual modelo de solução é mais eficiente ou adequado ao respeitar as características das aplicações e do ambiente computacional disponível. Nesse sentido, a metodologia oferece condições para analisar e comparar aspectos relacionados ao desempenho de diferentes soluções distribuídas.

Inicialmente, uma metodologia para modelagem e avaliação de desempenho de programas paralelos, em sistemas distribuídos homogêneos, foi especificada e proposta. Essa metodologia, chamada PEMPIs, não previa atividades relacionadas ao balanceamento de carga nem permitia modelagem de aplicações em ambientes heterogêneos. Os bons resultados obtidos com as técnicas propostas pelo PEMPIs e a necessidade de metodologias semelhantes para sistemas heterogêneos motivaram a especificação da metodologia PEMPIs-Het.

O processo definido no PEMPIs-Het contempla 4 módulos principais: AME, PWD, MWD e PEM. Esses módulos definem estratégias que permitem modelar as aplicações, estimar o desempenho dos programas, gerar índices que caracterizam o desempenho das aplicações em cada uma das máquinas do sistema, elaborar um plano inicial para distribuir

as cargas computacionais, monitorar a execução dos processos e acompanhar o estado das máquinas. Todos os módulos foram implementados e várias análises foram realizadas através de testes experimentais com aplicações MPI.

O processo de modelagem e predição de desempenho foi avaliado tanto em sistemas distribuídos homogêneos quanto em heterogêneos. Os resultados gerados foram consistentes e mostraram a eficiência das técnicas de modelagem. A precisão dos modelos analíticos foi demonstrada em cada um dos estudos de caso realizado. As estimativas para o tempo de execução das aplicações apresentaram pequenos erros em relação ao tempo real medido nos testes experimentais. Um dos problemas apresentado pelas técnicas de modelagem diz respeito ao método de ajuste de curvas utilizado. As análises dos resultados mostraram que o mecanismo de ajuste do método dos mínimos quadrados pode não ser o mais interessante quando um único modelo de desempenho é elaborado para representar o comportamento de toda a aplicação. Quando o intervalo da análise é grande, o modelo se torna mais preciso para a extremidade dos valores mais altos da análise. Por outro lado, o processo de modelagem e as técnicas utilizadas são simples e fáceis de serem aplicados. Algumas funções relacionadas à modelagem e análise dos dados foram implementadas com o auxílio do pacote Scilab, automatizando algumas etapas do processo.

Outra dificuldade encontrada na modelagem das aplicações diz respeito ao comportamento dos programas. Algumas aplicações podem modificar seu comportamento ao longo da execução e dificultar a elaboração de modelos precisos para a avaliação de desempenho. Uma análise detalhada de uma das versões do programa de multiplicação de matrizes mostrou que o padrão de acesso à memória pode modificar o comportamento da aplicação em função do tamanho da matriz e comprometer a qualidade dos modelos. Alterações no ambiente, como contenções na rede e na carga computacional das máquinas, também podem prejudicar as estimativas, quando se utiliza um ambiente real de execução para análises de desempenho. Uma alternativa, muitas vezes utilizada para contornar o problema, é fazer uso de simuladores. Em ambientes simulados é possível ter um controle completo sobre parâmetros intrinsicamente dinâmicos, como os citados.

Além das avaliações sobre a modelagem e predição de desempenho dos programas paralelos, vários testes foram realizados para verifiar a eficiência das estratégias criadas para balanceamento de carga. Todas as estratégias elaboradas fazem uso de índices de desempenho gerados através dos próprios modelos das aplicações. Inicialmente, o comportamento do programa é caracterizado em cada tipo de máquina do ambiente. Com os modelos individuais é possível extrair relações de desempenho entre as máquinas. Essas relações são utilizadas para planejar a distribuição da carga computacional. O princípio das estratégias é dividir o problema em pequenas partes (*chunks*) e atribuir uma quantidade de trabalho adequada à capacidade de processamento de cada um dos nós computacionais.

A metodologia também foi utilizada para comparar diferentes estruturas de soluções para o programa de multiplicação de matrizes. As estimativas geradas através das

predições podem ajudar a escolher o modelo de organização mais eficiente em relação ao desempenho.

Assim, a especificação de um processo para a modelagem e predição de desempenho de programas MPI e a formalização de técnicas para balanceamento de carga, utilizando os modelos de predição, foram as duas principais linhas de desenvolvimento deste trabalho.

6.2 Contribuições Apresentadas

As principais contribuições geradas com o desenvolvimento desta tese são:

1. As metodologias PEMPIs e PEMPIs-Het: poderão auxiliar programadores na implementação de aplicações paralelas eficientes em sistemas computacionais distribuídos. As metodologias podem estimar o desempenho das aplicações em ambientes homogêneos e heterogêneos, permitindo avaliações de escalabilidade ou análises de desempenho para diferentes soluções. Além disso, é possível planejar a distribuição das tarefas de modo a explorar mais eficientemente a capacidade computacional de cada uma das máquinas.

2. DP*Graph^{++}: simbologia gráfica que permite modelar as aplicações e detectar estruturas que comprometem o desempenho da solução. Uma alternativa de uso deste modelo seria associar estimativas de tempo para cada estrutura do modelo e, com isso, realizar uma previsão por ordem de grandeza para as aplicações. Estas informações ajudariam a decidir qual é o algoritmo mais eficiente para resolver um mesmo problema.

3. Estratégias para Balanceamento de Carga: estratégias baseadas em índices de desempenho computacional (VRP e VRP-SS) foram especificadas e implementadas para promover a divisão e distribuição de cargas computacionais no ambiente. O intuito das estratégias é associar a cada máquina do sistema uma carga computacional que seja adequada à capacidade de processamento do nó.

4. *Middleware* para Distribuição de Cargas: algumas bibliotecas foram implementadas para auxiliar as estratégias elaboradas para o balanceamento de carga do sistema. Essas bibliotecas ainda podem ser melhoradas para facilitar a chamada das funções relacionadas à divisão e distribuição das cargas no sistema. Com isso, os programas podem ser modificados com maior facilidade para utilizar as estratégias planejadas.

5. Estratégias para Avaliação de Modelos de Soluções Paralelas e Distribuídas: a metodologia PEMPIs-Het pode ser utilizada para analisar o comportamento de diferentes modelos de soluções e estimar a tendência de desempenho de cada abordagem. Essas estimativas são utilizadas para definir qual o modelo mais indicado para a aplicação que será implementada.

6. Monitor de Desempenho: foi implementado um monitor que analisa e coleta informações distribuídas sobre a situação de cada uma das máquinas do sistema. O monitor permite avaliar a carga computacional de cada um dos nós. Se a máquina é multiprocessada ou multinúcleo, informações sobre cada unidade de processamento são analisadas separadamente. Além disso, informações sobre o uso de memória principal e virtual também são coletadas. No final, a situação de cada nó é mapeada.

7. Ferramentas Scilab: um conjunto de rotinas foi implementado no Scilab para automatizar atividades relacionadas à seleção dos dados coletados nos testes experimentais, à análise estatística dos dados e à geração automática de tabelas latex para apresentação dos dados.

8. Revisão bibliográfica: uma revisão atualizada dos trabalhos relacionados ao tema de modelagem e avaliação de desempenho foi realizada.

6.3 Trabalhos Publicados

Ao longo do desenvolvimento deste trabalho conseguimos as seguintes publicações:

1. MIDORIKAWA, E. T.; OLIVEIRA, H. M.; LAINE, J. M. PEMPIs: A New Methodology for Modeling and Prediction of MPI Programs Performance. In: *16th Symposium on Computer Architecture and High Performance Computing* (SBAC-PAD 2004). Foz do Iguaçu, Brazil: IEEE Computer Society, 2004. p. 246-253. ISBN 0-7695-2240-8.

2. MIDORIKAWA, E. T.; OLIVEIRA, H.; LAINE, J. M. PEMPIs: A New Methodology for Modeling and Prediction of MPI Programs Performance. *International Journal of Parallel Programming*, Springer, v. 33, n. 5, p. 499-527, October 2005.

3. LAINE, J. M.; MIDORIKAWA, E. T. Analisando a Predição de Desempenho com os Modelos Analíticos Gerados pela Metodologia PEMPIs-Het. *WSCAD'07 - VIII Workshop em Sistemas Computacionais de Alto Desempenho*, Gramado, Brasil, 2007.

4. LAINE, J. M.; MIDORIKAWA, E. T. Using Analytical Models to Load Balancing in a Heterogeneous Network of Computers. In: *Parallel Computing Technologies - PaCT 2007*. Springer, 2007. (Lecture Notes in Computer Science, v. 4671), p. 559-568.

5. LAINE, J. M.; MIDORIKAWA, E. T. Uso de Modelos Analíticos na Modelagem de Aplicações Paralelas Distribuídas. WPerformance 2008 - *Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, Belém - PA, Brasil, 2008.

6. LAINE, J. M.; MIDORIKAWA, E. T. Efficient Strategies for Workload Distribution in Heterogeneous Computing Systems. In: *Proceedings of 2008 IEEE 11th International Conference on Computational Science and Engineering*. São Paulo, Brazil, 2008.

7. LAINE, J. M.; MIDORIKAWA, E. T. Avaliando e Comparando Diferentes Estruturas de Programas Paralelos Através de Modelos Analíticos de Desempenho. *WSCAD'08 - IX Workshop em Sistemas Computacionais de Alto Desempenho*, Campo Grande - MS, Brasil, 2008.

6.4 Trabalhos Futuros

Para dar continuidade a este trabalho e aperfeiçoar os processos definido na metodologia PEMPIs-Het, os seguintes trabalhos futuros foram pensados:

1. Construir uma ferramenta integrada para modelagem e análise de desempenho: as funções elaboradas com o auxílio do Scilab podem ser integradas em uma ferramenta para ajudar no processo de modelagem das aplicações. Essa ferramenta pode automatizar a instrumentação do códido da aplicação com os monitores de tempo e gerar o modelo gráfico DP*Graph^{++} da aplicação. A partir dos resultados experimentais, funções de análises estatísticas podem ser aplicadas na seleção dos dados coletados e no desenvolvimento dos modelos de predição. Por fim, estimativas podem ser realizadas pela própria ferramenta, gerando gráficos para a apresentação dos resultados. A idéia é integrar essas funções em uma ferramenta que poderia ser acessível via *web*, por exemplo.

2. Elaborar bibliotecas completas com as funções de balanceamento: as funções de divisão e distribuição das cargas computacionais, implementadas pelas estratégias elaboradas, podem ser disponibilizadas em bibliotecas. Assim, as aplicações poderiam adicionar as bibliotecas e invocar as funções para promover o balanceamento.

3. Avaliar as estratégias de modelagem e balanceamento de carga em um ambiente de *Grid*: as estratégias definidas na metodologia PEMPIs-Het podem ser avaliadas em um ambiente extremamente dinâmico e heterogêneo, como é o caso dos *grids* computacionais. A aplicabilidade das estratégias de modelagem e balanceamento de carga pode ser analisada neste outro ambiente. Para isso, os testes realizados e apresentados nesta tese podem ser repetidos se as aplicações forem ajustadas para utilizar o MPICH-G2, que é uma implementação MPI disponível para *grid*. Essa distribuição utiliza alguns serviços do Globus Toolkit (SOTOMAYOR; CHILDERS, 2005) que permite agrupar máquinas de arquiteturas diferentes e executar aplicações MPI utilizando trocas de mensagens.

4. Estender os símbolos do DP*Graph^{++} para permitir uma análise qualitativa: a simbologia do DP*Graph^{++} pode ser estendida para proporcionar outras análises sobre as aplicações. Avaliações qualitativas poderiam ser realizadas sobre o modelo de organização utilizado em diferentes soluções para um mesmo problema. Para isso, o DP*Graph^{++} deve ser capaz de associar a cada estrutura uma estimativa de sua complexidade computacional. Desse modo, além de ajudar a compreender o código da aplicação, o modelo permite estimar qual solução é mais eficiente já na modelagem do programa.

5. Avaliar outros métodos de ajuste de curvas: o método dos mínimos quadrados apresenta alguns problemas em relação à precisão das estimativas ao gerar um único modelo para todo o intervalo de dados. Os problemas foram relatados e discutidos durante a apresentação dos resultados. Por isso, avaliar a possibilidade de utilizar outros métodos para a elaboração dos modelos analíticos é algo a ser pensado. Os programas modelados neste trabalho podem ser caracterizados através de outros métodos de ajuste de curvas e os resultados podem ser comparados.

6. Integrar as estratégias de predição e distribuição em escalonadores de processos para ambientes amplamente distribuídos, como os *Grids* Computacionais: as estimativas geradas com os modelos de desempenho podem ser utilizadas em tomadas de decisões dos escalonadores de processos. O planejamento do escalonamento pode levar em consideração as estimativas sobre a liberação de recursos ou finalização de tarefas que, *a priori*, podem ser obtidas através dos modelos analíticos da cada aplicação. Através desses modelos é possível estimar quanto tempo de processamento a aplicação ainda vai gastar e quais os nós computacionais serão liberados com o fim da execução. Essas informações podem melhorar o planejamento do escalonador.

7. Avaliar outros modelos de programação paralela: as estratégias da metodologia PEMPIs-Het poderiam ser utilizadas para avaliar programas paralelos implementados com o *OpenMP* ou *threads*, por exemplo.

8. Avaliar a flexibilidade da metodologia PEMPIs-Het aplicando as estratégias de modelagem e predição de desempenho em programas PVM. A princípio, entendemos que as técnicas especificadas na metodologia podem ser facilmente aplicadas na análise de programas PVM. Isso porque tanto os programas MPI quanto os programas PVM fazem uso de primitivas de passagem de mensagem para comunicação distribuída. Nossa sugestão é que se refaça a análise de alguma aplicação MPI avaliada neste trabalho usando as primitivas de comunicação da biblioteca PVM.

Referências Bibliográficas

ADVE, S. V.; GHARACHORLOO, K. Shared Memory Consistency Models: A Tutorial. *IEEE Computer*, v. 29, n. 12, p. 66–76, 1996.

AGARWAL, A. Performance Tradeoffs in Multithreaded Processors. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Press, Piscataway, NJ, USA, v. 3, n. 5, p. 525–539, 1992. ISSN 1045-9219.

BADIA, R. M.; RODRÍGUEZ, G.; LABARTA, J. Deriving Analytical Models from a Limited Number of Runs. In: *Parallel Computing Conference (PARCO)*. [S.l.: s.n.], 2003. p. 769–776.

BADIA, R. M. et al. Performance Monitoring of Grid Superscalar with OCM-G/G-PM: Tuning and Improvements. In: *CoreGRID Workshop on Grid Programming Model Grid and P2P Systems Architecture, Grid Systems, Tools and Environments*. [S.l.: s.n.], 2007.

BAL, H. E.; HAINES, M. Approaches for Integrating Task and Data Parallelism. *IEEE Concurrency*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 6, n. 3, p. 74–84, 1998. ISSN 1092-3063.

BANICESCU, I. et al. Overhead Analysis of a Dynamic Load Balancing Library for Cluster Computing. In: *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 1*. Washington, DC, USA: IEEE Computer Society, 2005.

BANINO, C. et al. Scheduling Strategies for Master-Slave Tasking on Heterogeneous Processor Platforms. *IEEE Trans. Parallel Distributed Systems*, v. 15, n. 4, p. 319–330, 2004.

BARNEY, B. M. Introduction to Parallel Computing. 2007. Disponível em: <http://www.llnl.gov/computing/tutorials/parallel_comp>.

BEAUMONT, O. et al. Matrix-Matrix Multiplication on Heterogeneous Platforms. In: *ICPP '00: Proceedings of the Proceedings of the 2000 International Conference on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2000. p. 289. ISBN 0-7695-0768-9.

BEAUMONT, O.; LEGRAND, A.; ROBERT, Y. The Master-Slave Paradigm with Heterogeneous Processors. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, USA, v. 14, n. 9, p. 897–908, 2003. ISSN 1045-9219.

BEGUELIN, A. et al. *A User's Guide to PVM Parallel Virtual Machine*. Knoxville, TN, USA, 1991.

BEN-ARI, M. *Principles of Concurrent and Distributed Programming*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1990. ISBN 0-13-711821-X.

BRANCO, K. R. L. J. C. *Índices de Carga e Desempenho em Ambientes Paralelos/Distribuídos - Modelagem e Métricas*. Tese (Doutorado) — Universidade de São Paulo, São Carlos, SP, 2004.

BREWER, E. A. et al. *PROTEUS: A High-Performance Parallel-Architecture Simulator*. Cambridge, MA, USA, 1991.

BUCHHOLZ, P. Hierarchical Structuring of Superposed GSPNs. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 25, n. 2, p. 166–181, 1999. ISSN 0098-5589.

BURNS, G.; DAOUD, R.; VAIGL, J. LAM: An Open Cluster Environment for MPI. In: *Proceedings of Supercomputing Symposium*. [S.l.: s.n.], 1994. p. 379–386.

BUYYA, R. *High Performance Cluster Computing: Architectures and Systems*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0130137847.

BUYYA, R. *High Performance Cluster Computing: Programming and Applications*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0130137855.

CASAVANT, T. L.; KUHL, J. G. A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 2, p. 141–154, 1988. ISSN 0098-5589.

CHAN, E. et al. SuperMatrix: a multithreaded runtime scheduling system for algorithms-by-blocks. In: *PPoPP '08: Proceedings of the 13th ACM SIGPLAN Symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM, 2008. p. 123–132. ISBN 978-1-59593-795-7.

CHANDRA, R. et al. *Parallel Programming in OpenMP*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2001. ISBN 1-55860-671-8.

CHING, L. K.; GAUDIOT, J.-L.; SATO, L. M. Performance Prediction Methodology for Parallel Programs with MPI in NOW Environments. In: *IWDC '02: Proceedings of the*

4th International Workshop on Distributed Computing, Mobile and Wireless Computing. London, UK: Springer-Verlag, 2002. p. 268–279. ISBN 3-540-00355-X.

CIORBA, F. M. et al. Dynamic Multi Phase Scheduling for Heterogeneous Clusters. In: *20th International Parallel and Distributed Processing Symposium (IPDPS)*. Rhodes Island, Greece: IEEE Computer Society, 2006. p. 10pp. ISBN 1-4244-0054-6.

CIRNE, W. et al. Running Bag-of-Tasks Applications on Computational Grids: The MyGrid Approach. *ICCP'2003 - International Conference on Parallel Processing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 407, 2003. ISSN 0190-3918.

CLARK, D. L. et al. *Scheduling of parallel jobs on dynamic, heterogeneous networks*. [S.l.], 1995.

COOPER, R. B. Queueing Theory. In: *ACM 81: Proceedings of the ACM '81 conference*. New York, NY, USA: ACM Press, 1981. p. 119–122. ISBN 0-89791-049-4.

CRAVEIRO, G. D. S.; SATO, L. M. CPAR-Cluster: A Runtime System for Heterogeneous Clusters With Mono and Multiprocessor Nodes. In: *CCGRID '04: Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2004. p. 387–394. ISBN 0-7803-8430-X.

CULLER, D. et al. LogP: Towards a Realistic Model of Parallel Computation. In: *PPOPP '93: Proceedings of the fourth ACM SIGPLAN symposium on Principles and practice of parallel programming*. New York, NY, USA: ACM Press, 1993. p. 1–12. ISBN 0-89791-589-5.

CULLER, D.; SINGH, J. P.; GUPTA, A. *Parallel Computer Architecture: A Hardware/Software Approach (The Morgan Kaufmann Series in Computer Architecture and Design)*. [S.l.]: Morgan Kaufmann, 1998. Hardcover. ISBN 1558603433.

DAVE, N. et al. Hardware Acceleration of Matrix Multiplication on a Xilinx FPGA. In: *MEMOCODE '07: Proceedings of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign*. Washington, DC, USA: IEEE Computer Society, 2007. p. 97–100. ISBN 1-4244-1050-9.

DHAKAL, S. et al. Dynamic Load Balancing in Distributed Systems in the Presence of Delays: A Regeneration-Theory Approach. *IEEE Transactions on Parallel and Distributed Systems*, IEEE Computer Society, Los Alamitos, CA, USA, v. 18, n. 4, p. 485–497, 2007. ISSN 1045-9219.

DIAS, J. E. A. O. *Performance Prediction and Tuning in a Multi-Cluster Environment*. Tese (Doutorado), Barcelona, Spain, 2006.

DONGARRA, J.; LASTOVETSKY, A. An Overview of Heterogeneous High Performance and Grid Computing. *Engineering the Grid: Status and Perspective*, American Scientific Publishers, February 2006.

DU, C.; SUN, X.-H.; WU, M. Dynamic Scheduling with Process Migration. In: *CCGRID '07: Proceedings of the Seventh IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2007. p. 92–99. ISBN 0-7695-2833-3.

DUNCAN, R. A Survey of Parallel Computer Architectures. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 23, n. 2, p. 5–16, 1990. ISSN 0018-9162.

FAHRINGER, T. *Automatic Performance Prediction for Parallel Programs on Massively Parallel Computers*. Tese (Doutorado) — University of Vienna, 1994.

FERRARI, D. *Measurement and Tuning of Computer Systems*. Englewood Cliffs, NJ: Prentice Hall, 1983.

FERSCHA, A. A Petri Net Approach for Performance Oriented Parallel Program Design. *Journal of Parallel and Distributed Computing.*, v. 15, n. 3, p. 188–206, 1992.

FLYNN, M. J. Some Computer Organizations and their Effectiveness. *IEEE Transactions on Computers*, C-21, n. 9, p. 948–960, 1972.

FLYNN, M. J.; RUDD, K. W. Parallel Architectures. *ACM Comput. Surv.*, ACM Press, New York, NY, USA, v. 28, n. 1, p. 67–70, 1996. ISSN 0360-0300.

FOSTER, I.; KESSELMAN, C. *The Grid 2: Blueprint for a New Computing Infrastructure*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003. ISBN 1558609334.

FOSTER, I. et al. *The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration*. [S.l.], June 2002.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, v. 15, n. 3, 2001.

FRANK, M. P. Approaching the Physical Limits of Computing. In: *ISMVL '05: Proceedings of the 35th International Symposium on Multiple-Valued Logic (ISMVL'05)*. Washington, DC, USA: IEEE Computer Society, 2005. p. 168–185. ISBN 0-7695-2336-6.

FRANKLIN, M.; GOVINDAN, V. The N-Body Problem: Distributed System Load Balancing and Performance Evaluation. In: *Proceedings of the 6th International Conference on Parallel and Distributed Computing Systems*. [S.l.]: PDCS, 1993.

FUJIMOTO, R. M. Parallel and Distributed Discrete Event Simulation: Algorithms and Applications. In: *WSC '93: Proceedings of the 25th conference on Winter simulation*. New York, NY, USA: ACM, 1993. p. 106–114. ISBN 0-7803-1381-X.

GEIST, A. et al. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Networked Parallel Computing*. Cambridge, MA, USA: MIT Press, 1994. ISBN 0-262-57108-0.

GEMUND, A. J. C. van. Performance Prediction of Parallel Processing Systems: the PAMELA Methodology. In: *ICS '93: Proceedings of the 7th international conference on Supercomputing*. New York, NY, USA: ACM Press, 1993. p. 318–327. ISBN 0-89791-600-X.

GEMUND, A. J. C. van. Compiling Performance Models from Parallel Programs. In: *ICS '94: Proceedings of the 8th international conference on Supercomputing*. New York, NY, USA: ACM Press, 1994. p. 303–312. ISBN 0-89791-665-4.

GEMUND, A. J. C. van. Symbolic Performance Modeling of Parallel Systems. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 14, n. 2, p. 154–165, 2003. ISSN 1045-9219.

GETOV, V.; GRAY, P.; SUNDERAM, V. MPI and Java-MPI: contrasts and comparisons of low-level communication performance. In: *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*. New York, NY, USA: ACM Press, 1999. p. 21. ISBN 1-58113-091-0.

GRAHAM, R. L. et al. Open MPI: A High-Performance, Heterogeneous MPI. In: *Proceedings of Fifth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*. Barcelona, Spain: [s.n.], 2006.

GRAHAM, R. L.; WOODALL, T. S.; SQUYRES, J. M. Open MPI: A Flexible High Performance MPI. In: *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*. Poznan, Poland: [s.n.], 2005.

GROPP, W. et al. *MPI — The Complete Reference: Volume 2, the MPI-2 Extensions*. [S.l.]: MIT Press, 1998.

GROVE, D. A. *A Performance Modeling System for Message-Passing Parallel Programs*. Tese (Doutorado) — University of Adelaide, Department of Computer Science, Adelaide, 2003.

GROVE, D. A.; CODDINGTON, P. D. Communication Benchmarking and Performance Modelling of MPI Programs on Cluster Computers. *J. Supercomput.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 34, n. 2, p. 201–217, 2005. ISSN 0920-8542.

GROVE, D. A.; CODDINGTON, P. D. Modeling Message-Passing Programs With a Performance Evaluating Virtual Parallel Machine. *Performance Evaluation*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, v. 60, n. 1-4, p. 165–187, 2005. ISSN 0166-5316.

GUBITOSO, M. D. *Modelos de Desempenho para Sistemas de Memória Compartilhada Virtual*. Tese (Doutorado) — Universidade de São Paulo, São Paulo, SP, 1996.

HARRIS, T. J. A Survey of PRAM Simulation Techniques. *ACM Comput. Surv.*, ACM Press, New York, NY, USA, v. 26, n. 2, p. 187–206, 1994. ISSN 0360-0300.

HSU, J. M.; BANERJEE, P. Performance Measurement and Trace Driven Simulation of Parallel CAD and Numeric Applications on a Hypercube Multicomputer. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 3, n. 4, p. 451–464, 1992. ISSN 1045-9219.

HUSS-LEDERMAN, S. et al. Implementation of Strassen's algorithm for matrix multiplication. In: *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*. Washington, DC, USA: IEEE Computer Society, 1996. p. 32. ISBN 0-89791-854-1.

IBRAHIM, M.; XINDA, L. Performance of Dynamic Load Balancing Algorithm on Cluster of Workstations and PCs. In: *ICA3PP '02: Proceedings of the Fifth International Conference on Algorithms and Architectures for Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 2002. p. 44–47. ISBN 0-7695-1512-6.

JAIN, R. *The Art of Computer Systems Performance Analysis*. [S.l.]: Wiley & Sons, 1991.

JUHÁSZ, S. Modelling Asynchronous Message Passing in Small Cluster Environments. *Int. J. Comput. Appl.*, ACTA Press, Anaheim, CA, USA, v. 28, n. 1, p. 43–49, 2006. ISSN 1206-212X.

KANT, K. *Structured Computer Organization*. [S.l.]: McGraw-Hill, Inc, 1992.

KARONIS, N. T.; TOONEN, B.; FOSTER, I. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 63, n. 5, p. 551–563, 2003. ISSN 0743-7315.

KLEIJNEN, J. P. C. Monte Carlo Simulation and its Statistical Design and Analysis. *SIGSIM Simul. Dig.*, ACM Press, New York, NY, USA, v. 6, n. 1, p. 23–29, 1974. ISSN 0163-6103.

KNOPP, J.; REICH, M. A Workpool Model for Parallel Computing. In: *Proceedings of the First International Workshop on High Level Programming Models and Supportive Environments (HIPS)*. [S.l.: s.n.], 1996.

KUMARAN, S.; QUINN, M. J. Divide-and-Conquer Programming on MIMD Computers. In: *IPPS '95: Proceedings of the 9th International Symposium on Parallel Processing*. Washington, DC, USA: IEEE Computer Society, 1995. p. 734–741. ISBN 0-8186-7074-6.

LAINE, J. M. *Desenvolvimento de Modelos de Predição para Programas Paralelos MPI*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2003.

LAINE, J. M.; MIDORIKAWA, E. T. Analisando a Predição de Desempenho com os Modelos Analíticos Gerados pela Metodologia PEMPIs-Het. *WSCAD'07 - VIII Workshop em Sistemas Computacionais de Alto Desempenho*, Gramado, Brasil, 2007.

LAINE, J. M.; MIDORIKAWA, E. T. Using Analytical Models to Load Balancing in a Heterogeneous Network of Computers. In: *Parallel Computing Technologies (PaCT 2007)*. [S.l.]: Springer, 2007. (Lecture Notes in Computer Science, v. 4671), p. 559–568.

LAINE, J. M.; MIDORIKAWA, E. T. Efficient Strategies for Workload Distribution in Heterogeneous Computing Systems. In: *Proceedings of 2008 IEEE 11th International Conference on Computational Science and Engineering*. São Paulo, Brazil: [s.n.], 2008. p. 194–199.

LAINE, J. M.; MIDORIKAWA, E. T. Uso de Modelos Analíticos na Modelagem de Aplicações Paralelas Distribuídas. *WPerformance 2008 - Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, Belém - PA, Brasil, 2008.

LAINE, J. M.; OLIVEIRA, H. M.; MIDORIKAWA, E. T. Modeling Repetition Structures in MPI Programs Using a New Graphical Model. In: *International Conference on Parallel and Distributed Processing Techniques and Applications - PDPTA'03*. Las Vegas, Nevada: [s.n.], 2003. p. 1696–1701.

LASTOVETSKY, A.; MKWAWA, I.-H.; O'FLYNN, M. An Accurate Communication Model of a Heterogeneous Cluster Based on a Switch-Enabled Ethernet Network. In: *ICPADS '06: Proceedings of the 12th International Conference on Parallel and Distributed Systems*. Washington, DC, USA: IEEE Computer Society, 2006. p. 15–20. ISBN 0-7695-2612-8.

LASTOVETSKY, A.; TWAMLEY, J. Towards a Realistic Performance Model for Networks of Heterogeneous Computers. In: *Proceedings of IFIP TC5 Workshop, World Computer Congress*. [S.l.]: Springer, 2005. p. 39–58. ISBN 0-387-24048-9.

LAZOWSKA, E. D. et al. *Quantitative System Performance: Computer System Analysis Using Queueing Network Models*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1984. ISBN 0-13-746975-6.

LEVESON, N. G.; STOLZY, J. L. Safety Analysis Using Petri Nets. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 13, n. 3, p. 386–397, 1987. ISSN 0098-5589.

LI, H. et al. Locality and Loop Scheduling on NUMA Multiprocessors. In: *Proceedings of the 1993 International Conference on Parallel Processing*. Boca Raton, FL: CRC Press, 1993. II - Software, p. II–140–II–147.

LI, K. C. *Análise e Predição de Desempenho de Programas Paralelos em Redes de Estações de Trabalho*. Tese (Doutorado) — Universidade de São Paulo, São Paulo, SP, 2001.

LIU, X.; GOLDSMITH, A. Load Balancing and Switch Scheduling. *ICC 2005. IEEE International Conference on Communications*, v. 2, p. 1010–1014, 2005.

LO, V. M.; MACHE, J.; WINDISCH, K. J. A Comparative Study of Real Workload Traces and Synthetic Workload Models for Parallel Job Scheduling. In: *IPPS/SPDP '98: Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing*. London, UK: Springer-Verlag, 1998. p. 25–46. ISBN 3-540-64825-9.

LUO, Q.; DRAKE, J. B. A Scalable Parallel Strassen's Matrix Multiplication Algorithm for Distributed-Memory Computers. In: *SAC '95: Proceedings of the 1995 ACM symposium on Applied computing*. New York, NY, USA: ACM, 1995. p. 221–226. ISBN 0-89791-658-1.

MAHESHWARI, P. A Dynamic Load Balancing Algorithm for a Heterogeneous Computing Environment. In: *HICSS '96: Proceedings of the 29th Hawaii International Conference on System Sciences (HICSS'96) Volume 1: Software Technology and Architecture*. Washington, DC, USA: IEEE Computer Society, 1996. p. 338. ISBN 0-8186-7324-9.

MEIRA, J. W.; LEBLANC, T. J.; POULOS, A. Waiting Time Analysis and Performance Visualization in Carnival. In: *SPDT '96: Proceedings of the SIGMETRICS symposium on Parallel and distributed tools*. New York, NY, USA: ACM, 1996. p. 1–10. ISBN 0-89791-846-0.

MENASCÉ, D. A.; BARROSO, L. A. A Methodology for Performance Evaluation of Parallel Applications on Multiprocessors. *J. Parallel Distrib. Comput.*, Academic Press, Inc., Orlando, FL, USA, v. 14, n. 1, p. 1–14, 1992. ISSN 0743-7315.

MIDORIKAWA, E. T.; OLIVEIRA, H.; LAINE, J. M. PEMPIs: A New Methodology for Modeling and Prediction of MPI Programs Performance. *International Journal of Parallel Programming - IJPP*, Springer, v. 33, n. 5, p. 499–527, October 2005.

MIDORIKAWA, E. T.; OLIVEIRA, H. M.; LAINE, J. M. PEMPIs: A New Methodology for Modeling and Prediction of MPI Programs Performance. In: *16th Symposium on Computer Architecture and High Performance Computing (SBAC-PAD 2004)*. Foz do Iguaçu, Brazil: IEEE Computer Society, 2004. p. 246–253. ISBN 0-7695-2240-8.

MPI-FORUM. *MPI: A Message-Passing Interface Standard*. Knoxville, TN, USA, 1994.

MUELLER, F. A Library Implementation of POSIX Threads Under Unix. In: *Proceedings of theWinter 1993 USENIX Technical Conference and Exhibition*. San Diego, CA, USA: [s.n.], 1993. p. 29–41.

MURATA, T.; SHATZ, S. M.; SHENKER, B. Detection of Ada Static Deadlocks Using Petri Net Invariants. *IEEE Trans. Softw. Eng.*, IEEE Press, Piscataway, NJ, USA, v. 15, n. 3, p. 314–326, 1989. ISSN 0098-5589.

NÉMETH, Z.; SUNDERAM, V. A Formal Framework for Defining Grid Systems. In: *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'2002)*. Berlin: IEEE Computer Society Press, 2002. p. 202–211.

NÉMETH, Z.; SUNDERAM, V. Characterizing Grids: Attributes, Definitions, and Formalisms. *Journal of Grid Computing*, v. 1, n. 1, p. 9–23, 2003.

NUTT, G. J. Tutorial: Computer System Monitors. *SIGMETRICS Perform. Eval. Rev.*, ACM Press, New York, NY, USA, v. 5, n. 1, p. 41–51, 1976. ISSN 0163-5999.

OLIVEIRA, H. M.; LAINE, J. M.; MIDORIKAWA, E. T. Performance Analysis and Prediction of some MPI Communication Primitives. In: *International Conference on Parallel and Distributed Processing Techniques and Applications - PDPTA'02*. Las Vegas, Nevada: [s.n.], 2002.

OLIVEIRA, H. M. de. *Modelagem e Predição de Desempenho de Primitivas de Comunicação MPI*. Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo, 2003.

OLIVEIRA, J. A. de. *Um Estudo Comparativo de Cargas de Trabalho e Políticas de Escalonamento para Aplicações Paralelas em Clusters e Grids Computacionais*. Tese (Doutorado) — Universidade de São Paulo, São Carlos, SP, 2006.

ORLANDI, R. C. G. S. *Ferramenta para Análise de Desempenho de Sistemas Computacionais Distribuídos*. Dissertação (Mestrado) — ICMC-USP, São Carlos, SP, 1995.

PENMATSA, S.; CHRONOPOULOS, A. T. Cooperative Load Balancing for a Network of Heterogeneous Computers. In: *20th IEEE International Parallel and Distributed Processing Symposium (IPDPS), 15th Heterogeneous Computing Workshop*. Rhodes Island, Greece: IEEE Computer Society, 2006. p. 10pp.

PETERSON, J. L. Petri Nets. *ACM Comput. Surv.*, ACM Press, New York, NY, USA, v. 9, n. 3, p. 223–252, 1977. ISSN 0360-0300.

PETERSON, T. G. A Comparison of Software and Hardware Monitors. *SIGMETRICS Perform. Eval. Rev.*, ACM Press, New York, NY, USA, v. 3, n. 2, p. 2–5, 1974. ISSN 0163-5999.

PFNEISZL, H.; KOTSIS, G. Synthetic Workload Generation for Parallel Processing Systems. In: *Proceedings of the Third International ACPC Conference with Special Emphasis on Parallel Databases and Parallel I/O*. London, UK: Springer-Verlag, 1996. p. 136–147. ISBN 3-540-61695-0.

POLYCHRONOPOULOS, C. D.; KUCK, D. J. Guided Self-Scheduling: A Practical Scheduling Scheme for Parallel Supercomputers. *IEEE Transactions on Computers*, C-36, n. 12, p. 1425–1439, December 1987.

QIN, X.; XIE, T. An Availability-Aware Task Scheduling Strategy for Heterogeneous Systems. *IEEE Transactions on Computers*, IEEE Computer Society, Los Alamitos, CA, USA, v. 57, n. 2, p. 188–199, 2008. ISSN 0018-9340.

QUINN, M. J. *Parallel Programming in C with MPI and OpenMP*. [S.l.]: McGraw-Hill Education Group, 2003. ISBN 0071232656.

SÁNCHEZ, D.; MACÍAS, E. M.; SUÁREZ, Á. An Application Level Load Balancing Mechanism for Heterogeneous Clusters Programming. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'02)*. [S.l.: s.n.], 2002. p. 872–878.

SCHOPF, J. M. *Performance Prediction and Scheduling for Parallel Applications on Multi-User Clusters*. Tese (Doutorado), La Jolla, CA, USA, 1998.

SCHOPF, J. M.; BERMAN, F. *Performance Prediction Using Intervals*. [S.l.], May 1997.

SHIH, W.-C.; YANG, C.-T.; TSENG, S.-S. A Performance-Based Approach to Dynamic Workload Distribution for Master-Slave Applications on Grid Environments. In: *Grid and Pervasive Computing (GPC)*. [S.l.: s.n.], 2006. p. 73–82.

SHIH, W.-C.; YANG, C.-T.; TSENG, S.-S. A Performance-Based Parallel Loop Scheduling on Grid Environments. *J. Supercomput.*, Kluwer Academic Publishers, Hingham, MA, USA, v. 41, n. 3, p. 247–267, 2007. ISSN 0920-8542.

SILVA, L. M.; BUYYA, R. *Parallel Programming Models and Paradigms*. New Jersey, USA: Prentice Hall, PTR, 1999. In: BUYYA, R. High Performance Cluster Computing: Programming and Applications.

SNIR, M.; OTTO, S. *MPI — The Complete Reference: The MPI Core*. Cambridge, MA, USA: MIT Press, 1998. ISBN 0262692155.

SOTOMAYOR, B.; CHILDERS, L. *Globus® Toolkit 4: Programming Java Services (The Elsevier Series in Grid Computing)*. [S.l.]: Morgan Kaufmann, 2005. Paperback. ISBN 0123694043.

SQUYRES, J. M.; LUMSDAINE, A. A Component Architecture for LAM/MPI. In: *Proceedings, 10th European PVM/MPI Users' Group Meeting*. Venice, Italy: Springer-Verlag, 2003. (Lecture Notes in Computer Science, 2840), p. 379–387.

STALLINGS, W. *Computer Organization and Architecture: Designing for Performance*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999. ISBN 0130812943.

STALLINGS, W. *Arquitetura e Organização de Computadores: Projeto para o Desempenho*. São Paulo, SP, Brasil: Prentice Hall, 2002.

TAM, M.-C.; SMITH, J. M.; FARBER, D. J. A taxonomy-based comparison of several distributed shared memory systems. *SIGOPS Oper. Syst. Rev.*, ACM Press, New York, NY, USA, v. 24, n. 3, p. 40–67, 1990. ISSN 0163-5980.

TANENBAUM, A. S.; GOODMAN, J. R. *Structured Computer Organization*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998. ISBN 0130959901.

TRAN, V. D.; HLUCHY, L.; NGUYEN, G. T. Parallel Programming with Data Driven Model. *8th Euromicro Workshop on Parallel and Distributed Processing (PDP)*, IEEE Computer Society, Los Alamitos, CA, USA, v. 00, p. 205, 2000. ISSN 1066-6192.

TZEN, T. T.; NI, L. M. Trapezoidal Self-Scheduling: A Practical Scheduling Scheme for Parallel Compilers. *IEEE Transactions on Parallel and Distributed Systems*, v. 4, n. 1, p. 87–98, January 1993.

VALIANT, L. G. A Bridging Model for Parallel Computation. *Commun. ACM*, ACM Press, New York, NY, USA, v. 33, n. 8, p. 103–111, 1990. ISSN 0001-0782.

WABNIG, H.; HARING, G. Performance Prediction of Parallel Systems With Scalable Specifications: Methodology and Case Study. *SIGMETRICS Perform. Eval. Rev.*, ACM Press, New York, NY, USA, v. 22, n. 2-4, p. 46–62, 1995. ISSN 0163-5999.

WILKINSON, B.; ALLEN, M. *Parallel Programming: Techniques and Applications Using Networked Workstations and Prallel Computers*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1999. ISBN 0-13-671710-1.

WILLIAM, G.; LUSK, E. *User's Guide for MPICH, a Portable Implementation of MPI.* [S.l.], 1996.

WILLIAM, G.; LUSK, E. *Portable MPI Model Implementation*. 2007. Disponível em: <citeseer.ist.psu.edu/article/gropp96users.html>.

XU, M.-J.; CHUNG, M.-M. J. Predicting the Performance of Synchronous Discrete Event Simulation. *IEEE Trans. Parallel Distrib. Syst.*, IEEE Press, Piscataway, NJ, USA, v. 15, n. 12, p. 1130–1137, 2004. ISSN 1045-9219.

YANG, C.-T.; CHANG, S.-C. A Parallel Loop Self-Scheduling on Extremely Heterogeneous PC Clusters. In: *International Conference on Computational Science*. [S.l.: s.n.], 2003. p. 1079–1088.

YANG, C.-T.; CHENG, K.-W.; SHIH, W.-C. On Development of an Efficient Parallel Loop Self-Scheduling for Grid Computing Environments. *Parallel Comput.*, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, v. 33, n. 7-8, p. 467–487, 2007. ISSN 0167-8191.

YANG, C.-T.; SHIH, W.-C.; TSENG, S.-S. A Dynamic Partitioning Self-scheduling Scheme for Parallel Loops on Heterogeneous Clusters. In: *International Conference on Computational Science (1)*. [S.l.: s.n.], 2006. p. 810–813.

YANG, W. et al. Implementation and Evaluation of Matrix-matrix Multiplication on FT64 Stream Processor. In: *ICCIT '07: Proceedings of the 2007 International Conference on Convergence Information Technology (ICCIT 2007)*. Washington, DC, USA: IEEE Computer Society, 2007. p. 1445–1451. ISBN 0-7695-3038-9.