Uso de Modelos Analíticos na Modelagem de Aplicações Paralelas Distribuídas

Jean M. Laine, Edson T. Midorikawa

[1]Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, trav. 3, 158 – São Paulo – SP – 05508-900, Brazil

{jean.laine, edson.midorikawa}@poli.usp.br

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. The PEMPIs-Het methodology allows performance modeling, evaluation and prediction of parallel programs in homogeneous and heterogeneous environments. In this paper some applications are modeled and the accuracy of predictions is verified by experimental tests. An evaluation about different models of distributed solutions is executed and a performance prediction is generated to each approach.*

Resumo. *Uma metodologia para desenvolvimento de programas paralelos deve especificar mecanismos capazes de caracterizar o comportamento das aplicações e permitir estudos sobre o desempenho de diferentes modelos de soluções. A metodologia PEMPIs-Het permite a modelagem, avaliação e predição de desempenho de programas paralelos em ambientes homogêneos e heterogêneos. Neste artigo, algumas aplicações são modeladas e a precisão das estimativas geradas para o tempo de execução das aplicações é verificada através de testes experimentais. Uma avaliação sobre diferentes modelos de soluções distribuídas é realizada e uma estimativa de desempenho é gerada para cada abordagem.*

1. Introdução

Nos últimos anos, com o aumento da capacidade de processamento dos computadores pessoais e o desenvolvimento das redes de alta velocidade, sistemas computacionais como os *clusters* e os *grids* [Németh and Sunderam 2003, Nemeth and Sunderam 2002, Foster and Kesselman 2003, Foster et al. 2002, Foster et al. 2001] e, mais recentemente, os *clouds*, se tornaram atraentes para a execução de aplicações de larga escala e/ou de alto desempenho. Neste contexto, tais ambientes se constituíram uma alternativa atraente aos tradicionais computadores paralelos centralizados [Clark 1995]. Uma comparação entre os ambientes tradicionais de computação distribuída e os *grids* computacionais, em particular, pode ser encontrada em [Németh and Sunderam 2002].

Conseqüentemente, surgiram algumas abordagens para o desenvolvimento de aplicações paralelas distribuídas, algumas bibliotecas para comunicação remota foram criadas, como o PVM [Geist et al. 1994] e o MPI [Snir and Otto 1998], e novos modelos de programação foram definidos para atender as necessidades e particularidades da organização de cada uma dessas plataformas. Hoje em dia, podemos citar os modelos

mestre-escravo, *bag of tasks*, *peer-to-peer* e cliente-servidor [Buyya 1999] como sendo os mais utilizados para organização do processamento paralelo e distribuído.

Se a programação paralela em sistemas multiprocessados ou multi-núcleos, baseada em variáveis de memória compartilhada, já é uma tarefa complicada, ainda mais difícil é o desenvolvimento de sistemas paralelos em arquiteturas distribuídas heterogêneas, como discutido em [Badia et al. 2007]. Isso porque é necessário considerar diversos aspectos que não estão presentes na implementação das aplicações seqüenciais. Alguns desses aspectos dizem respeito a comunicação entre os processos remotos, as sincronizações estabelecidas entre as tarefas e as 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.

Normalmente, as aplicações paralelas fazem uso do modelo de programação mestre-escravo. Nesta organização, o processo mestre é o responsável por organizar a divisão e a distribuição do trabalho entre os escravos. Estes processos executam a tarefa recebida do mestre e devolvem os resultados parciais para que a solução final possa ser encontrada. Durante a distribuição do trabalho aos escravos é possível fazer uso de alguma estratégia de balanceamento de carga para maximizar a utilização dos recursos computacionais e melhorar o desempenho final da aplicação.

Neste contexto, a existência de metodologias de análise, avaliação e predição de desempenho pode ser fundamental para o desenvolvimento de aplicações paralelas otimizadas. Uma metodologia capaz de analisar, avaliar e estimar o desempenho de diferentes modelos de soluções pode auxiliar os programadores a encontrarem soluções mais eficientes para um mesmo problema. Os modelos de predição podem estimar o comportamento das aplicações em diversas situações projetadas, bem como avaliar mecanismos de distribuição de carga e escalabilidade do sistema. Além disso, a utilização de métodos e ferramentas de modelagem, especificadas por uma metodologia, agregam valores ao sistema implementado.

Várias metodologias foram propostas para auxiliar atividades relacionadas a estudos de desempenho de aplicações paralelas e distribuídas [de Oliveira Dias Júnior 2006, Grove 2003, Culler et al. 1993, Schopf 1998]. No entanto, a maioria desses trabalhos propõe estratégias para analisar somente trechos da aplicação paralela, como as estruturas de repetição ou as primitivas de comunicações, por exemplo. Embora este tipo de análise seja capaz de ajudar a descobrir eventuais gargalos para o desempenho da solução ela não ajuda os programadores a decidirem qual paradigma ou modelo de programação é mais adequado para a solução de um problema.

Neste intuito, definimos uma metodologia chamada PEMPIs-Het (*Performance Estimation of MPI Programs in Heterogeneous Systems*) [Laine and Midorikawa 2007a, Laine and Midorikawa 2007b]. Esta metodologia especifica um conjunto de técnicas de modelagem, avaliação e predição de desempenho que permite avaliar diferentes modelos de soluções e definir a estratégia mais otimizada para a solução. Em trabalhos publicados anteriormente focamos na apresentação e discussão da metodologia PEMPIs-Het e nas técnicas que a metodologia utiliza para a modelagem das aplicações paralelas e distribuídas. Neste artigo nosso objetivo é demonstrar a capacidade da metodologia PEMPIs-

Het em modelar e avaliar o desempenho de diferentes modelos de soluções distribuídas e prever, ao longo do tempo, o comportamento das soluções analisadas.

Este trabalho está organizado da seguinte forma: a seção 2 apresenta alguns trabalhos relacionados. A seção 3 descreve algumas formas de estruturar uma solução distribuída. Na seção 4 descrevemos a metodologia PEMPIs-Het. Na seção 5 apresentamos a aplicação das estratégias do PEMPIs-Het na modelagem e predição de desempenho de aplicações paralelas distribuídas. As conclusões do trabalho são apresentadas na seção 7.

2. Trabalhos Relacionados

Uma metodologia para análise e predição de desempenho deve ser capaz de identificar os prováveis fatores que podem influenciar o desempenho das aplicações e, além disso, mostrar como estes fatores interagem. Neste intuito, alguns autores têm utilizado a modelagem analítica para alcançar este objetivo [Tam and Wang 1999, Badia et al. 2003]. Com este enfoque, elaboramos modelos matemáticos capazes de representar o comportamento das aplicações em sistemas distribuídos heterogêneos. A idéia destes modelos é estimar o tempo de execução das aplicações não só em função do tamanho do problema mas da quantidade de processos também.

Na literatura existem vários trabalhos sobre modelagem e predição de desempenho de aplicações paralelas. No entanto, grande parte das publicações acabam restringindo as análises para alguns elementos do programa, como as estruturas de repetição ou as primitivas de comunicação.

Em [Lastovetsky et al. 2006], o autor apresenta uma modelagem das primitivas de comunicação, levando em consideração o impacto da heterogeneidade dos processadores no desempenho da transmissão dos dados. Modelos analíticos, em função do tamanho da mensagem, são gerados para caracterizar o comportamento das operações ponto-a-ponto e coletivas. Em [Lastovetsky and Twamley 2005] os autores caracterizam o comportamento da aplicação em uma rede não dedicada de computadores heterogêneos. Para isso, curvas de desempenho baseadas em intervalos são utilizadas. A estratégia determina que o desempenho da aplicação pode ser estimado por um limite superior e outro inferior (intervalo de predição).

Em [Yang et al. 2007] e [Shih et al. 2007] os autores apresentam uma modelagem para laços de repetição de programas paralelos e também implementam mecanismos de balanceamento de carga do tipo *self-scheduling* para estas estruturas em ambientes de *grid*.

Em [Grove 2003, Grove and Coddington 2005a] os autores propõem um sistema para modelar o desempenho de programas paralelos que utilizam o modelo de programação baseado em passagem de mensagem. 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 and Coddington 2005b]. A estrutura do programa é dividida em segmentos de código seqüenciais e de comunicação, sendo cada trecho modelado separadamente.

Em [Laine et al. 2003] realizamos a modelagem de estruturas de repetição, simples e aninhadas, de programas paralelos MPI e prevemos o tempo de execução dos

laços de repetição em função do número de iteração. Em [Oliveira et al. 2002] caracterizamos e modelamos as primitivas de comunicação, ponto-a-ponto e coletivas, da implementação LAM-MPI e verificamos a precisão dos modelos através de testes experimentais. As primeiras análises sobre modelagem de aplicações MPI foram apresentadas em [Oliveira et al. 2003]. Entretanto, as estratégias para a modelagem de aplicações completas foi formalizada com a especificação da metodologia PEMPIs [Midorikawa et al. 2004, Midorikawa et al. 2005]. No entanto, esta metodologia restringia as análises para ambientes distribuídos homogêneos. Para permitir a modelagem e a predição de desempenho em sistemas heterogêneos a metodologia foi estendida e o PEMPIs-Het (*Performance Estimation of MPI Programs in Heterogeneous Systems*) [Laine and Midorikawa 2007a] foi especificado.

3. Estruturas de Programas Paralelos

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.

Com este propósito, preparamos três diferentes versões para o programa de multiplicação de matrizes: Self-Scheduling, VRP dinâmico e VRP-SS. Todas as versões utilizam o algoritmo tradicional de multiplicação de matrizes (O(n^3)). No entanto, cada solução implementa uma estratégia diferente para balanceamento de carga. As soluções geradas com a organização promovida pelas estratégias *Self-Scheduling (SS)*, *VRP* dinâmico e *VRP-SS* são completamente diferentes. As modificações promovidas no modelo da arquitetura da solução estão diretamente relacionadas ao desempenho final da aplicação, conforme mostra os resultados apresentados no gráfico da Figura 2.

3.1. Self-Scheduling (SS)

Essa estratégia utiliza o modelo de comunicação mestre-escravo para organizar a solução distribuída. Inicialmente, o trabalho é dividido em pequenas tarefas que serão enviadas aos processos escravos para que sejam executadas. Estas tarefas representam cargas unitárias de processamento. À medida que os processos terminam o processamento e ficam ociosos, eles enviam requisições ao mestre solicitando mais uma fatia de trabalho. Novamente, o mestre atende a solicitação e envia mais uma pequena tarefa ao processo requisitante. Esta dinâmica continua até que todo o trabalho tenha sido processado. 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 and Kuck 1987, Tzen and Ni 1993, Shih et al. 2006, Yang et al. 2006, Yang and Chang 2003].

3.2. Vetor de Desempenho Relativo (VRP)

Uma das estratégias elaboradas para promover o balanceamento de carga do sistema faz uso dos modelos analíticos individuais da aplicação ($\delta(n,p)$). Estes modelos são elaborados para estimar o desempenho da solução, em função do tamanho do problema (n) e da quantidade de processos (p), em cada uma das máquinas do sistema. As relações entre cada um desses modelos permite definir a capacidade de processamento relativa das

máquinas e, conseqüentemente, a proporção de trabalho ideal que cada nó computacional deve receber em relação ao trabalho total. Para determinar a capacidade de processamento da máquina x (φ_x) em relação a y, considerando valores específicos para n e p, a seguinte relação deve ser calculada:

$$\varphi_x = \frac{\delta_x(n,p)}{\delta_y(n,p)} \tag{1}$$

A partir dessas relações geramos um vetor de capacidades relativas, denominado VRP. Este vetor caracteriza a proporção de trabalho ideal de cada uma das máquinas, ajustada a capacidade estimada pelos modelos de desempenho. Cada máquina do ambiente é numerada e tem sua capacidade representada em posições distintas do VRP, conforme ilustrado a seguir:

$$VRP = [\varphi_1, \varphi_2, \varphi_3, ..., \varphi_{m-1}, \varphi_m] \tag{2}$$

onde:

φ_m: representa o valor de desempenho relativo da m-ésima máquina no sistema;

φ_1: geralmente representa o desempenho da máquina mais lenta do sistema. Se esta ordem for estabelecida, a máquina de menor capacidade computacional tem seu parâmetro representado como sendo 1 e a capacidade das demais são calculadas em função desta.

A distribuição do trabalho é feita utilizando o conceito de carga unitária. A carga unitária representa a menor unidade de trabalho computável para a aplicação que está sendo modelada e testada. Na metodologia PEMPIs-Het, essa carga unitária é determinada da seguinte forma:

$$ul = \frac{\tau}{\sum_{i=1}^{m} VRP[i]} \tag{3}$$

onde ul representa a carga unitária e τ a quantidade de trabalho total a ser computado pelo programa.

A partir dessas informações é possível calcular a quantidade de trabalho (Δ_i) que deve ser enviado para a i-ésima máquina listada no VRP, segundo a seguinte expressão:

$$\Delta_i = \lceil ul \times VRP[i] \rceil \tag{4}$$

Portanto, quanto maior a precisão dos modelos analíticos, mais ajustado estará os valores calculados para o VRP. Na versão estática de balanceamento, os valores determinados para o VRP permanecem os mesmos em todas as configurações de testes, independente dos valores usados para N. Na abordagem dinâmica, os valores do VRP estão em função do tamanho do problema (N). Esta estratégia obtém melhores resultados pois está sintonizada às modificações de comportamento que as aplicações podem apresentar ao variar o tamanho do N. A versão estática pode oferecer bons resultados em ambientes homogêneos mas não consegue repetir os resultados da estratégia dinâmica em sistemas heterogêneos.

3.3. VRP Self-Scheduling (VRP-SS)

A estratégia *self-scheduling* e as variações GSS, TSS, entre outras, não conseguem alcançar bons resultados no balanceamento de carga em ambientes distribuídos extremamente heterogêneos [Yang et al. 2007]. Isso porque o tempo gasto com o grande número de divisões e distribuições de cargas unitárias entre as máquinas pode comprometer o desempenho da estratégia.

Por outro lado, a estratégia VRP consegue reduzir o número de mensagens trocadas pela rede e ajustar a divisão do trabalho à capacidade individual de cada uma das máquinas. Logo, o tempo total de execução das aplicações pode ser reduzido. Entretanto, a estratégia VRP é menos flexível às possíveis mudanças apresentadas pelo ambiente, quando comparada ao modelo *self-scheduling* puro. Isso porque se uma máquina ou mais máquinas tiver sua carga de trabalho alterada, ao longo do tempo, os índices do VRP não permanecerão ajustados adequadamente. Como na versão estática do VRP a distribuição do trabalho é feita toda no início do processamento, esta variação pode comprometer o desempenho final da solução.

Outra situação problemática para a estratégia VRP pode ser observada na Figura 1. Este gráfico representa a variação do tempo de execução em função do número de iterações para distribuição do trabalho na estratégia *self-scheduling*. Durante o processamento foi utilizado máquinas e situações idênticas de processamento e mesmo assim nota-se uma variação significativa entre o tempo de execução apresentado pelas máquinas para processar uma carga idêntica de trabalho. Isto mostra que mesmo em máquinas idênticas a capacidade de processamento é algo particular e relacionado ao momento do nó. Portanto, o ajuste dos índices do VRP é extremamente importante para o êxito da estratégia de balanceamento.

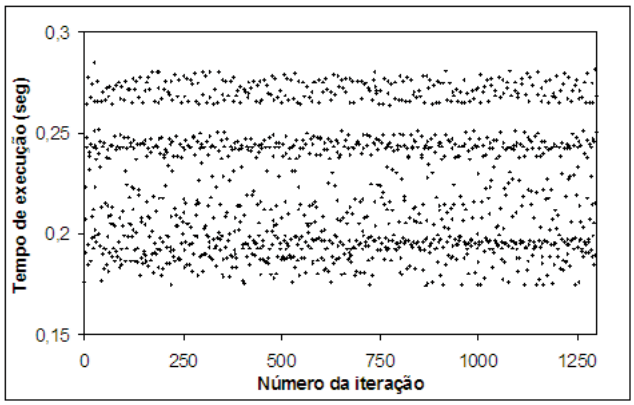

Figura 1. Análise por iteração do programa de multiplicação de matrizes.

Para minimizar estes problemas, uma versão híbrida, mesclando princípios das estratégias VRP e SS, foi especificada e implementada. Na estratégia, denominada VRP-SS, a distribuição das cargas é realizada por fases, como na estratégia *self-scheduling*, mas a quantidade de trabalho enviada a cada nó é calculada de acordo com os valores especificados no VRP. Assim, a quantidade de mensagens transmitidas para a distribuição de todo o trabalho é menor, se comparada ao modelo SS, e ajustada a capacidade individual de cada máquina. Além disso, a estratégia consegue reagir às eventuais sobrecargas individuais das máquinas atribuindo mais trabalho aos nós que apresentam uma melhor

condição de processamento. Isto é possível pois o mecanismo *self-scheduling* permite um maior dinamismo nas atribuições de trabalho. A estrutura principal do algoritmo que implementa a estratégia VRP-SS é apresentada a seguir:

```
Ler os índices do VRP
Para cada processo escravo faça {
  Divide o trabalho proporcionalmente a capacidade da máquina
  Envia a carga de trabalho calculada
}
Enquanto não receber todo o trabalho faça {
  Recebe o resultado parcial do processo escravo
  Identifica o processo que enviou o resultado
  Se existir trabalho a ser calculado então {
    Lê o índice da posição do processo no VRP
    Determina a quantidade de trabalho adequada a capacidade da máquina
    Envia a carga de trabalho calculada
  }
}
```

3.4. Comparando as Estruturas SS, VRP-SS e VRP

O desempenho de cada uma das versões comprovam as conseqüências do modelo adotado na organização da solução. Os testes experimentais com o programa de multiplicação de matrizes foram feitos em um ambiente heterogêneo formado por 4 máquinas intel, 4 máquinas bio e 6 máquinas taurus. Em cada uma das máquinas foi associado um único processo para colaborar na execução do programa. Portanto, a quantidade de processos foi fixado em 14 (p) e os valores avaliados para N foram: 500, 1000, 1500, 2000 e 2500. Uma análise sobre o desempenho de cada abordagem pode ser realizada através do gráfico da Figura 2, que compara o tempo de execução dos programas. A versão VRP-SS apresentou melhor desempenho que as demais em quase todas as configurações de teste. Somente para $N = 500$ e 1000 o desempenho da solução não foi o melhor. Isto pode ser explicado pois o modelo SS, em geral, é mais eficiente no processamento de pequenos trabalhos, onde a quantidade de mensagens trocadas entre os processos é menor. Para grandes valores de N, como $N = 2500$, o tempo de execução das versões VRP e SS supera em 60% o da versão VRP-SS, o que ilustra a eficiência da estratégia na organização da solução.

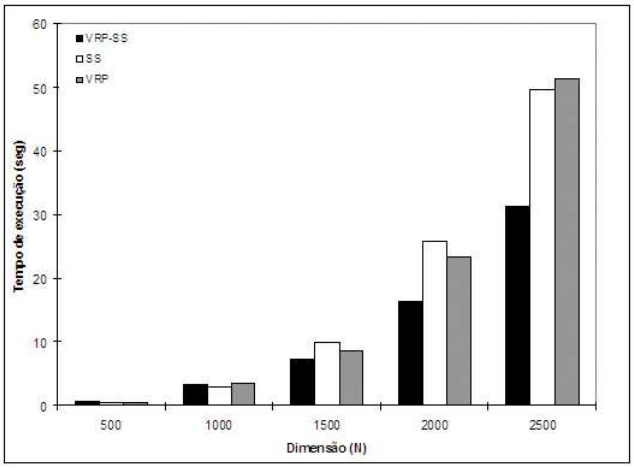

Figura 2. Resultados gerados pelas estratégias VRP, SS e VRP-SS.

4. A Metodologia PEMPIs-Het

A metodologia PEMPIs-Het [Laine and Midorikawa 2007a] define um processo estruturado de fácil aplicação capaz de auxiliar o desenvolvimento de aplicações de alto desempenho. Este processo compreende atividades de modelagem e avaliação de desempenho de programas paralelos distribuídos em ambientes homogêneos e heterogêneos. Além disso, a metodologia especifica um conjunto de técnicas para balanceamento de carga utilizando índices de desempenhos gerados através dos modelos analíticos. A organização da metodologia PEMPIs-Het é ilustrada na Figura 3. Conforme apresentado neste diagrama, os seguintes módulos foram definidos e implementados:

- AME - ***A**pplication **M**odeling **E**nvironment*: é o subprocesso da metodologia PEMPIs-Het que especifica um conjunto de atividades relacionadas à modelagem das aplicações MPI;
- PWD - ***P**erformance Estimation and **W**orkload **D**istribution*: é o subprocesso responsável por especificar e realizar atividades associadas a predição de desempenho da aplicação e determinar os pré-requisitos necessários para as atividades de balanceamento de carga do sistema;
- MWD - ***M**iddleware for **W**orkload **D**istribution*: este subprocesso gera um plano de execução inicial capaz de otimizar o desempenho das aplicações, aproveitando ao máximo a capacidade de processamento que os recursos do sistema podem oferecer;
- PEM - ***PE**rformance **M**onitor*: é uma ferramenta implementada para acompanhar, em tempo real, o desempenho das aplicações em execução no sistema e a carga de trabalho das máquinas. As informações de desempenho, coletadas pelo monitor, retornam ao MWD, que avalia se o planejamento inicial está sendo alcançado com sucesso.

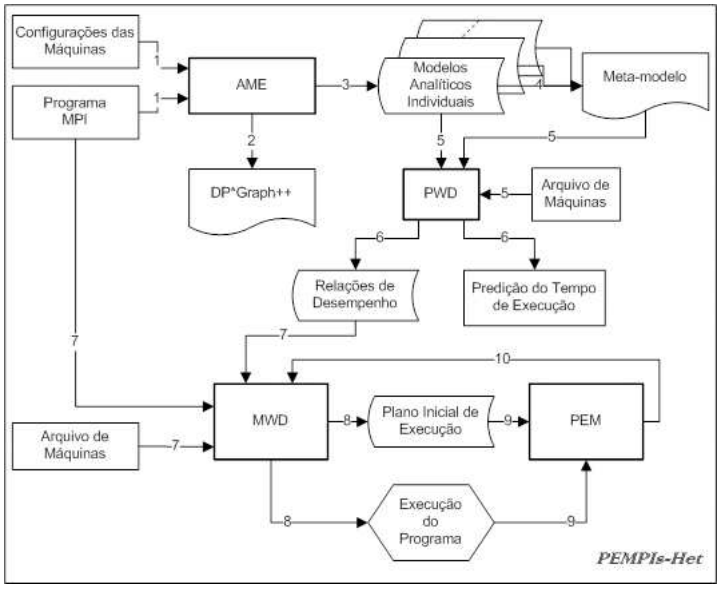

Figura 3. Metodologia PEMPIs-Het.

4.1. 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. Estas 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^{++} [Midorikawa et al. 2005]. A simbologia deste modelo gráfico auxilia a modelagem da aplicação e facilita o entendimento estrutural do código do programa;
4. com a ajuda do modelo gráfico DP*Graph^{++} é realizado um estudo sobre a complexidade algorítmica da aplicação e o modelo teórico de desempenho é elaborado;
5. as configurações do ambiente de teste são definidas. Neste momento um arquivo, chamado *hostfile*, é criado com o nome das máquinas que serão utilizadas pelo LAM [Squyres and Lumsdaine 2003, Burns et al. 1994] na execução do programa MPI. As configurações dos testes são planejadas, especificando a quantidade de processos p e o tamanho do problema n, para que as análises de desempenho sejam realizadas. Normalmente, um *script* é criado para controlar a execução dos casos de testes;
6. os testes experimentais planejados são realizados. Cada configuração prevista é executada um certo número de vezes e os tempos medidos são armazenados em arquivos individuais. A quantidade de vezes que o programa é executado não é um valor fixo, mas o suficiente para gerar uma massa de dados representativa e confiável;
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 destes 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. Estes 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 possível criar um meta-modelo para a aplicação. Este meta-modelo generaliza o comportamento da aplicação no ambiente modelado mas não especifica a quantidade de máquinas utilizada para cada tipo de nó computacional modelado;

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 meta-modelo permite realizar estudos sobre a escalabilidade da aplicação e/ou do sistema;
11. relações de desempenho são obtidas através dos modelos analíticos individuais. Instanciando 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. Estes índices são utilizados no planejamento das atividades de distribuição de carga do sistema;
12. um plano inicial de execução é elaborado para a aplicação. Este 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;
13. o desempenho dos processos distribuídos são monitorados em tempo real, através das funções implementadas no PEM. O objetivo é avaliar se a distribuição de carga planejada está adequada a 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.

5. Modelagem de Aplicações

Algumas aplicações foram utilizadas para avaliar a aplicabilidade e precisão das estratégias definidas na metodologia PEMPIs-Het na modelagem e predição de desempenho de aplicações completas.

Os testes experimentais foram realizados em 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 e o sistema operacional instalado em todas as máquinas é o Fedora Core 6.

As próximas seções descrevem cada uma das aplicações e apresentam os modelos analíticos de desempenho e as estimativas geradas para o tempo de execução dos programas paralelos avaliados.

5.1. Programa Heat

O programa Heat simula a propagação de calor em uma superfície metálica. Na simulação, o valor da temperatura de cada um dos pontos da superfície é mapeado através de uma matriz. 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 (método de Jacobi). A temperatura de cada um dos pontos $t_{i,j}$ 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 utiliza o modelo mestre-escravo e os cálculos da simulação são divididos entre todos os nós de processamento. Cada processo escravo recebe um subconjunto dos dados, representado 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 mapaemento da temperatura. Cada tarefa paralela é mapeada em um único processador.

O código da aplicação foi mapeado com os símbolos do modelo gráfico DP*Graph^{++} e está represenado na Figura 4. Na representação é possível identificar tanto os trechos de computação (simbolizados através dos retângulos na cor cinza) e as estruturas de repetição (indicadas pelas linhas pontilhadas) como as comunicações estabelecidas entre os processos (triângulos). O processamento da simulação é feito quase que exclusivamente nos escravos e está mapeado nos trechos de VIII a XIII.

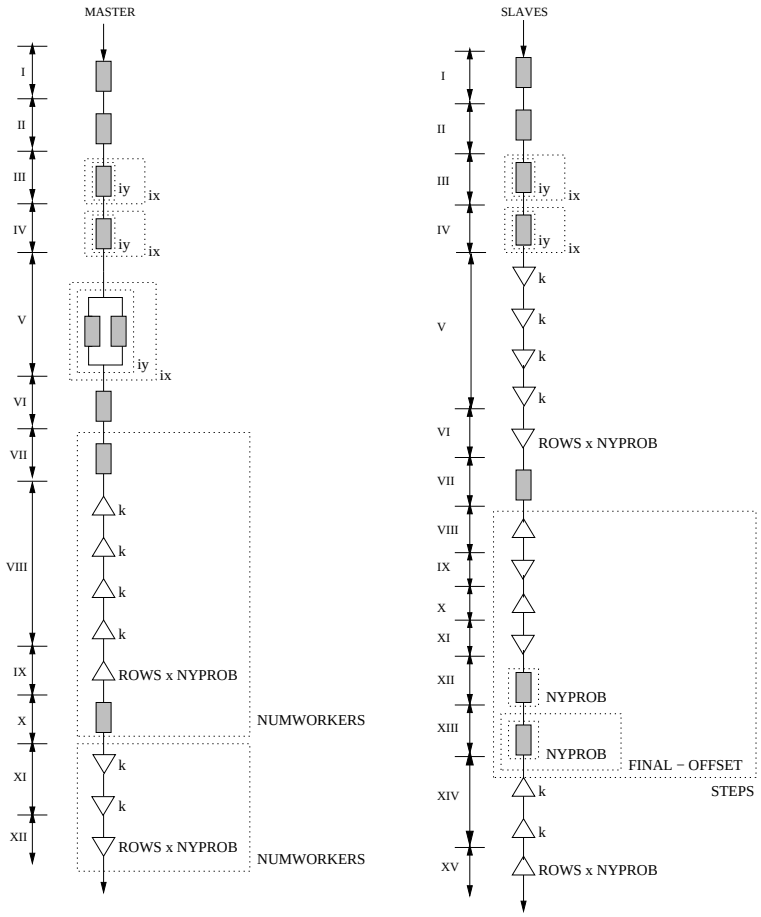

Figura 4. Modelo gráfico DP*Graph^{++} do programa Heat.

Na modelagem analítica do programa Heat equações parciais foram elaboradas para representar o comportamento de cada trecho representado no modelo gráfico DP*Graph^{++}. Estes modelos permitem estimar, individualmente, o tempo gasto em cada fase do processamento.

Para o desenvolvimento dos modelos de predição foram realizados testes experimentais com 4, 5 e 6 processos no *cluster* bio. Os valores utilizados para a matriz de mapeamento da superfície metálica foram 500, 750, 1000, 1250, 1500, 1750 e 2000. Cada configuração utilizada no teste experimental foi executada 50 vezes no ambiente e uma média dos tempos de execução selecionados foi calculada.

Um estudo sobre a complexidade das principais primitivas do LAM-MPI foi realizado em [Oliveira et al. 2002]. Neste estudo, modelos teóricos de algumas rotinas MPI foram especificados. A rotina MPI_Init, trecho I do modelo gráfico da Figura 4, pode ser representada por uma função linear em p (número de processos utilizados na execução do programa). Assim, na análise do programa Heat, o comportamento da operação MPI_Init foi modelado pela seguinte função:

$$T_I(p) = -1,9232 \times 10^{-3} + 4,3877 \times 10^{-3} \times p \quad (5)$$

As variáveis p, N e m, que aparecem nas equações, representam o número de processos, o tamanho do problema e o tamanho da mensagem transmitida (em bytes), respectivamente. Como o processamento final da aplicação é feito pelo mestre, a modelagem é realizada sobre os trechos identificados para este processo.

Um polinômio de segundo grau pode ser utilizado para representar os trechos de II a VI. Isto porque esta parte do código está associada ao processamento de dois laços aninhados [Laine et al. 2003]. Portanto, o modelo de predição elaborado para representar estas estruturas é dado por:

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

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, de acordo com [Oliveira et al. 2002], por um modelo linear em m. Assim, o tempo de execução deste trecho pode ser representado por:

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

O último trecho do processo mestre é composto somente por operações do tipo MPI_Recv. Esta primitiva, assim como o MPI_Send, pode ser representada por uma função linear em m. Assim, o tempo de execução deste trecho é estimado pelo seguinte modelo:

$$T_{XI-XII}(m) = (p-1) \times (-4,487 \times 10^{-4} + 4,899 \times 10^{-7} \times m) \quad (8)$$

Embora os modelos parciais para o processo mestre já tenham sido elaborados, é necessário incluir no modelo final de predição o tempo gasto no processamento da simulação. Este processamento é representado pelos trechos de VIII a XIII do processo escravo. Para modelar este trecho do programa foram elaborados três modelos. O primeiro representa o tempo gasto em comunicações do tipo *send* e *receive*. O segundo e o terceiro determinam o tempo gasto na execução de instruções dentro de estruturas de repetição simples (sl) e aninhada (nl). Assim, os modelos para cada um dos trechos são:

$$T_{comm}(m) = 3,5 \times 10^{-4} + 1,3 \times 10^{-6} \times m \quad (9)$$

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

$$T_{nl}(N) = -3,6341 \times 10^{-3} - 8,1 \times 10^{-6} \times N + 1,66 \times 10^{-7} \times N^2 \quad (11)$$

Finalmente, o modelo analítico final da aplicação Heat é determinado pela composição de todos os modelos parciais apresentados. Com este modelo para o tempo total de execução é possível estimar o desempenho da aplicação em diferentes configurações de p e N.

$$T_{total}(N,p,m) = T_I(p) + T_{II-VI}(N) + T_{VII-X}(m) + T_{XI-XII}(m) + T_{comm}(m) + T_{sl}(N) + T_{nl}(N) \quad (12)$$

Tabela 1. Comparação entre os tempos medidos e os estimados (segundos).

Dimensão da Placa	6 processos			8 processos		
	Estimado	Medido	Erro (%)	Estimado	Medido	Erro (%)
2000	14,938	15,338	-2,61	14,93	14,745	1,27
2500	23,527	24,653	-4,57	23,52	23,914	-1,64
3000	34,090	39,021	-12,64	34,08	38,682	-11,89
3500	46,627	53,137	-12,25	46,62	52,547	-11,28
4000	61,139	68,420	-10,64	61,13	68,080	-10,20

Dimensão da Placa	10 processos			12 processos		
	Estimado	Medido	Erro (%)	Estimado	Medido	Erro (%)
2000	14,926	14,786	0,95	14,92	14,866	0,36
2500	23,515	23,790	-1,16	23,51	23,953	-1,85
3000	34,078	38,415	-11,29	34,07	39,505	-13,75
3500	46,616	51,727	-9,88	46,61	53,014	-12,08
4000	61,128	66,836	-8,54	61,12	67,253	-9,12

A Tabela 1 apresenta os valores estimados, medidos e o erro percentual da predição. 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 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.

5.2. 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 and Govindan 2003]. 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 uma das partículas.

A paralelização da aplicação consiste em distribuir entre os nós de processamento um conjunto de partículas para que sejam calculadas as interações gravitacionais. O processo mestre mantém uma estrutura de dados que armazena o estado atual de cada uma das partículas. Esta 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 estas informações ele retransmite a estrutura novamente a todas as tarefas. A memória utilizada para armazenar as estruturas de dados são alocadas dinamicamente, o que otimiza o uso da memória.

Na modelagem das aplicações em ambientes heterogêneos o programa deve ser caracterizado em cada tipo de máquina do sistema. Os modelos analíticos individuais são combinados para estimar o desempenho do programa no ambiente. Portanto, os aspectos modelados não estão associados somente às características da aplicação (*software*). O modelo também contempla, implicitamente, a influência de elementos relacionados à heterogeneidade do *hardware* no desempenho do programa. Como descrito na apresentação da metodologia, antes de gerar o modelo de predição final é preciso definir o modelo teórico da aplicação. Para o programa analisado, em particular, o seguinte modelo é válido:

$$\delta^{PGF}(n,p) = \frac{C3}{p} \times n^2 + \frac{C2}{p} \times n + \frac{C1}{p} \tag{13}$$

Tabela 2. Coeficientes em função de n (intel).

C1	-1,125736	5,790233	8,578502
C2	0,000073	-0,000157	-0,000247
C3$\times 10^{-7}$	3,206325	2,163252	1,640802

Tabela 3. Coeficientes em função de p (intel).

C1	18,569761- 39,199647$\times$(1/p)
C2	0,0005777+0,0012943$\times$(1/p)
C3	0,0756705$\times 10^{-7}$ + 6,26157$\times 10^{-7}$(1/p)

Durante os testes experimentais foram utilizados 2, 3 e 4 processos sobre as máquinas do tipo intel. O processo de modelagem apresentado para o programa Heat foi repetido para gerar o modelo final de desempenho do programa PGF. Inicialmente, a modelagem foi feita em função de n e depois em função de p. Analisando o desenvolvimento do modelo para as máquinas *intel*, as Tabelas 2 e 3 apresentam os coeficientes encontrados para elaborar o modelo de predição através da aplicação dos métodos de modelagem analítica descritos na metodologia PEMPIs-Het. Após a substituição dos coeficientes apresentados na Tabela 3 no modelo teórico da aplicação (Equação 13), o tempo de execução dos processos atribuídos às máquinas intel pode ser estimado através da seguinte equação:

$$\delta^{PGF}(p,n) = (7,57 \times 10^{-9} + \frac{6,26 \times 10^{-7}}{p})n^2 + (-5,78 \times 10^{-4} + \frac{1.3 \times 10^{-3}}{p})n + (-1,99 + \frac{8,74}{p}) \tag{14}$$

6. Avaliando Estruturas de Soluções Diferentes

A partir das predições geradas para as aplicações, decidimos modelar diferentes estruturas de soluções para um mesmo problema e comparar o desempenho de cada uma através dos modelos analíticos gerados com a metodologia PEMPIs-Het. O intuito desta análise é verificar se os modelos gerados com a metodologia PEMPIs-Het podem ser utilizados para determinar qual modelo de solução é mais adequado na implementação de uma aplicação específica. Para isso, implementamos três versões diferentes do programa de multiplicação de matrizes - Self-Scheduling, VRP dinâmico e VRP-SS - e modelamos cada uma das versões para realizar as análises de desempenho.

6.1. Modelagem

As versões implementadas foram avaliadas experimentalmente em um ambiente heterogêneo formado por 4 máquinas intel, 4 máquinas bio e 6 máquinas taurus. Os valores usados para a dimensão das matrizes foram: 500, 1000, 1500, 2000 e 2500. A partir dos resultados experimentais, aplicamos as estratégias de modelagem definidas na metodologia PEMPIs-Het sobre os resultados de cada versão do programa e os seguintes modelos de desempenho foram elaborados:

VRP-SS:

$$\delta^{MM}_{VRP-SS}(n) = -2,14494 + 7,445 \times 10^{-3} \times n - 5,2 \times 10^{-6} \times n^2 + 3,008 \times 10^{-9} \times n^3 \quad (15)$$

SS:

$$\delta^{MM}_{SS}(n) = 2,42258 - 6,5303 \times 10^{-3} \times n + 4,5 \times 10^{-6} \times n^2 + 2,29 \times 10^{-9} \times n^3 \quad (16)$$

VRP:

$$\delta^{MM}_{VRP}(n) = -4,48182 + 1,64541 \times 10^{-2} \times n - 1,61 \times 10^{-5} \times n^2 + 7,38 \times 10^{-9} \times n^3 \quad (17)$$

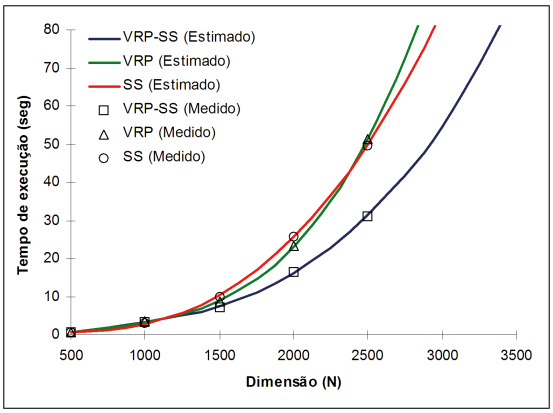

Figura 5. Precisão dos modelos das estratégias VRP, SS e VRP-SS.

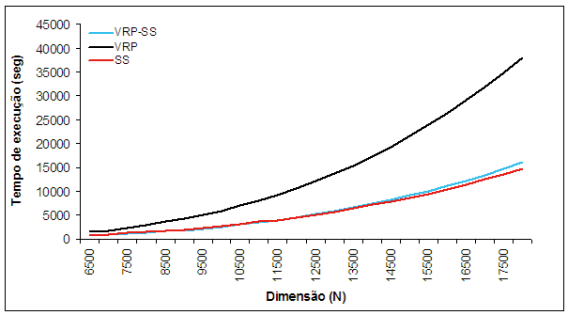

Figura 6. Projeção das predições de cada estrutura de solução.

O gráfico da Figura 5 apresenta os valores pontuais medidos durante os testes experimentais e as curvas geradas com os modelos de predição destacados nesta seção. É possível observar a precisão da modelagem e o comportamento de cada uma das curvas de predição. Até $N = 1115$ a solução VRP-SS não é a que apresenta os melhores índices de desempenho. No entanto, a partir deste valor existe uma inversão na posição das curvas de predição e a solução VRP-SS passa a ser melhor.

Até $N = 2400$ a versão SS gera resultados piores que a versão VRP, mas a partir deste ponto o comportamento das soluções é invertido e o desempeho da versão SS passa a ser melhor, conforme mostra as curvas exibidas no gráfico da Figura 5. Portanto, estas e outras análises podem ser realizadas em relação a projeção do desempenho de cada uma das versões. Uma continuação destas análises é realizada na seção 6.3 para valores extrapolados de N.

6.2. Precisão dos Modelos

Para verificar a precisão dos modelos gerados podemos analisar os resultados apresentados nos gráficos da Figura 7. Neste gráfico apresentamos os erros percentuais das estimativas de desempenho geradas para cada uma das versões do programa. Como é possível observar, os maiores erros percentuais estão localizados nos menores valores de N. Isso porque o ajuste realizado pelo método dos mínimos quadrados prioriza os valores mais significativos da análise. Com os resultados é possível observar que o modelo de desempenho da versão VRP-SS está melhor ajustado que os demais. Pelos resultados, 86% das predições apresentaram erros menores que 10%, o que é muito satisfatório para uma modelagem analítica.

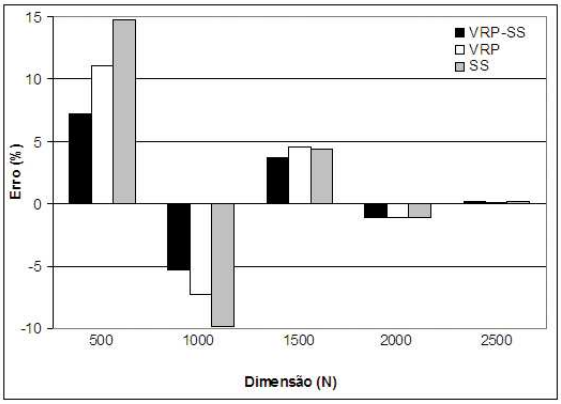

Figura 7. Erros obtidos nas predições de desempenho.

6.3. Projeção de Desempenho

Pelas curvas de projeção, apresentadas na Figura 6, é possível verificar que o desempenho das soluções VRP-SS e SS é semelhante ao longo de toda análise. Até $N = 11.500$, o desempenho do programa estruturado com a abordagem VRP-SS é ligeiramente melhor que o SS. Entretanto, a partir deste valor a curva de projeção da solução SS cruza a da versão VRP-SS e começa a apresentar melhores resultados de desempenho. Portanto, a estratégia VRP-SS inicia com um desempenho melhor mas tende a ser superada pela versão SS, segundo as previsões geradas pelos modelos. Mesmo assim, o desempenho

das duas versões segue próximo e, portanto, é um indicativo de que qualquer uma destas estratégias poderia ser utilizada na organização da solução para o problema. É possível concluir também que a única que deve ser desconsiderada como uma opção viável é a versão VRP, pois tem um desempenho sempre pior que as demais. Assim, podemos verificar que os modelos gerados com as estratégias definidas na metodologia PEMPIs-Het também podem ser utilizados para analisar estruturas de soluções diferentes de um mesmo problema.

7. Conclusões

Neste artigo apresentamos o uso de modelos analíticos, elaborados com as técnicas propostas pela metodologia PEMPIs-Het, na modelagem e predição de desempenho de aplicações paralelas completas. Além de estimar o desempenho das aplicações demonstramos que é possível gerar modelos parciais para estruturas específicas das aplicações, como trechos de processamento ou comunicações. A precisão das estimativas geradas pelos modelos foi verificada através dos testes experimentais. Para o programa Heat, o maior erro percentual das predições do tempo de execução foi de $13,75\%$.

Os modelos analíticos também foram utilizados para comparar o desempenho de diferentes versões de um mesmo programa. O intuito desta modelagem é estimar qual arquitetura de solução distribuída é mais eficiente para a aplicação modelada. Dessa forma, é possível realizar uma predição sobre o comportamento do programa e estimar, através dos modelos analíticos, qual a estratégia capaz de oferecer melhor desempenho no ambiente de execução. A precisão desta análise está diretamente relacionada a acurácia dos modelos gerados. O maior problema desta atividade é que o comportamento da aplicação tende a ser alterado ao modificar os valores de N e p, por questões relacionadas ao *hardware* das máquinas, como quantidade de memória principal e tamanho do cache, principalmente. Como os modelos analíticos são elaborados com base em resultados de execuções para intervalos restritos de N e p, a precisão das estimativas pode ser comprometida.

Neste intuito, fizemos uma previsão para o desempenho de três diferentes modelos de soluções para o programa de multiplicação de matrizes: SS, VRP-SS e VRP. Através das estimativas verificamos que o desempenho das versões SS e VRP-SS tende a ser muito próximo ao variar o tamanho de N. Embora até $N = 11.500$ a solução VRP-SS seja um pouco melhor, a partir deste valor a solução SS passa a oferecer melhor desempenho. Assim, demonstramos que é possível aplicar os modelos de predição para analisar estruturas de soluções diferentes de um mesmo problema.

Referências

Badia, R. M., Rodríguez, G., and Labarta, J. (2003). Deriving analytical models from a limited number of runs. In *PARCO*, pages 769–776.

Badia, R. M., Sirvent, R., Bubak, M., Funika, W., and Machner, P. (2007). 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*, number TR-0080.

Burns, G., Daoud, R., and Vaigl, J. (1994). LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386.

Buyya, R. (1999). *High Performance Cluster Computing: Programming and Applications*. Prentice Hall PTR, Upper Saddle River, NJ, USA.

Cirne, W., Paranhos, D., Costa, L., Santos-Neto, E., Brasileiro, F., Sauve, J., Silva, F. A. B., Barros, C. O., and Silveira, C. (2003). Running bag-of-tasks applications on computational grids: The mygrid approach. *icpp*, 00:407.

Clark, D. (1995). Scheduling of parallel jobs on dynamic heterogenous networks.

Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R., and von Eicken, T. (1993). 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*, pages 1–12, New York, NY, USA. ACM Press.

de Oliveira Dias Júnior, E. A. (2006). *Performance prediction and tuning in a multicluster environment*. PhD thesis, Barcelona, Spain.

Foster, I. and Kesselman, C. (2003). *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Foster, I., Kesselman, C., Nick, J. M., and Tuecke, S. (2002). The physiology of the grid: An open grid services architecture for distributed systems integration. Technical Report OGSI WG, Global Grid Forum.

Foster, I., Kesselman, C., and Tuecke, S. (2001). The anatomy of the grid: Enabling scalable virtual organizations. *International Journal of High Performance Computing Applications*, 15(3).

Franklin, M. and Govindan, V. (2003). The n-body problem: Distributed system load balancing and performance evaluation. Technical Report 93-16, Department of Computer Science and Engineering, Washington University, St. Louis.

Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., and Sunderam, V. (1994). *PVM: Parallel virtual machine: a users' guide and tutorial for networked parallel computing*. MIT Press, Cambridge, MA, USA.

Grove, D. A. (2003). *A Performance Modeling System for Message-Passing Parallel Programs*. PhD thesis, University of Adelaide, Department of Computer Science, Adelaide.

Grove, D. A. and Coddington, P. D. (2005a). Communication benchmarking and performance modelling of mpi programs on cluster computers. *J. Supercomput.*, 34(2):201–217.

Grove, D. A. and Coddington, P. D. (2005b). Modeling message-passing programs with a performance evaluating virtual parallel machine. *Perform. Eval.*, 60(1-4):165–187.

Laine, J. M. and Midorikawa, E. T. (2007a). Analisando a predição de desempenho com os modelos analíticos gerados pela metodologia pempis-het. *WSCAD'07 - VIII Workshop em Sistemas Computacionais de Alto Desempenho*.

Laine, J. M. and Midorikawa, E. T. (2007b). Using analytical models to load balancing in a heterogeneous network of computers. In Malyshkin, V. E., editor, *PaCT*, volume 4671 of *Lecture Notes in Computer Science*, pages 559–568. Springer.

Laine, J. M., Oliveira, H. M., and Midorikawa, E. T. (2003). Modeling repetition structures in mpi programs using a new graphical model. In *International Conference on Parallel and Distributed Processing Techniques and Applications - PDPTA'03*, pages 1696–1701, Las Vegas, Nevada.

Lastovetsky, A., Mkwawa, I.-H., and O'Flynn, M. (2006). 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*, pages 15–20, Washington, DC, USA. IEEE Computer Society.

Lastovetsky, A. and Twamley, J. (2005). Towards a realistic performance model for networks of heterogeneous computers. pages 39–58. Springer.

Midorikawa, E. T., Oliveira, H., and Laine, J. M. (2005). Pempis: A new methodology for modeling and prediction of mpi programs performance. *International Journal of Parallel Programming*, 33(5):499–527.

Midorikawa, E. T., Oliveira, H. M., and Laine, J. M. (2004). 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)*, pages 246–253, Foz do Iguaçu, Brazil. IEEE Computer Society.

Németh, Z. and Sunderam, V. (2002). A comparison of conventional distributed computing environments and computational grids. In *Proceedings of the ICCS2002*, volume II, pages 729–738, Amsterdam. Springer.

Nemeth, Z. and Sunderam, V. (2002). A formal framework for defining grid systems. In *2nd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGRID'2002)*, pages 202–211, Berlin. IEEE Computer Society Press.

Németh, Z. and Sunderam, V. (2003). Characterizing grids: Attributes, definitions, and formalisms. *Journal of Grid Computing*, 1(1):9–23.

Oliveira, H. M., Laine, J. M., and Midorikawa, E. T. (2002). 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.

Oliveira, H. M., Laine, J. M., and Midorikawa, E. T. (2003). Algumas contribuições para modelagem de programas paralelos mpi. In *WPERFORMANCE03 - II Workshop em Desempenho de Sistemas Computacionais e de Computação*, Campinas, São Paulo, Brazil.

Polychronopoulos, C. D. and Kuck, D. J. (1987). Guided self-scheduling: a practical scheduling scheme for parallel supercomputers. *IEEE Transactions on Computers*, C-36(12):1425–1439.

Schopf, J. M. (1998). *Performance prediction and scheduling for parallel applications on multi-user clusters*. PhD thesis, La Jolla, CA, USA.

Shih, W.-C., Yang, C.-T., and Tseng, S.-S. (2006). A performance-based approach to dynamic workload distribution for master-slave applications on grid environments. In *GPC*, pages 73–82.

Shih, W.-C., Yang, C.-T., and Tseng, S.-S. (2007). A performance-based parallel loop scheduling on grid environments. *J. Supercomput.*, 41(3):247–267.

Snir, M. and Otto, S. (1998). *MPI — The Complete Reference: The MPI Core*. MIT Press, Cambridge, MA, USA.

Squyres, J. M. and Lumsdaine, A. (2003). A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy. Springer-Verlag.

Tam, A. T. C. and Wang, C.-L. (1999). Realistic communication model for parallel computing on cluster. In *IWCC '99: Proceedings of the 1st IEEE Computer Society International Workshop on Cluster Computing*, page 92, Washington, DC, USA. IEEE Computer Society.

Tzen, T. T. and Ni, L. M. (1993). Trapezoidal self-scheduling: A practical scheduling scheme for parallel compilers. *IEEE Transactions on Parallel and Distributed Systems*, 4(1):87–98.

Yang, C.-T. and Chang, S.-C. (2003). A parallel loop self-scheduling on extremely heterogeneous pc clusters. In *International Conference on Computational Science*, pages 1079–1088.

Yang, C.-T., Cheng, K.-W., and Shih, W.-C. (2007). On development of an efficient parallel loop self-scheduling for grid computing environments. *Parallel Comput.*, 33(7-8):467–487.

Yang, C.-T., Shih, W.-C., and Tseng, S.-S. (2006). A dynamic partitioning self-scheduling scheme for parallel loops on heterogeneous clusters. In *International Conference on Computational Science (1)*, pages 810–813.