AVALIAÇÃO DE ESTRATÉGIAS DE BALANCEAMENTO DE CARGA DO TIPO MESTRE-ESCRAVO PARA APLICAÇÕES SPMD EM *CLUSTERS* E *GRIDS* COMPUTACIONAIS

André Lucio de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Cláudio Luis de Amorim, Ph.D.

Prof. Ricardo Cordeiro de Farias, Ph.D.

Prof. Simone de Lima Martins, D.Sc.

Prof. Alexandre Plastino de Carvalho, D.Sc.

RIO DE JANEIRO, RJ - BRASIL

SETEMBRO DE 2008

OLIVEIRA, ANDRÉ

Avaliação de Estratégias de Balanceamento de Carga do Tipo Mestre-Escravo para Aplicações SPMD em *Clusters* e *Grids* Computacionais [Rio de Janeiro] 2008

XV, 121p. 29,7 cm (COPPE/UFRJ, M.Sc., Engenharia de Sistemas e Computação, 2008)

Dissertação - Universidade Federal do Rio de Janeiro, COPPE

1. Balanceamento de Carga
2. Aplicações SPMD
3. *Clusters* e *Grids* Computacionais

I. COPPE/UFRJ II. Título (série)

Dedicatória

Dedico este trabalho ao meu querido pai
que faleceu em outubro do último ano,
deixando-nos uma imensa saudade.
Você estará sempre presente nas minhas
lembranças e no meu coração.

Agradecimentos

Eis que chegou o momento de expressar sinceros agradecimentos a todos que diretamente ou indiretamente contribuíram para o desenvolvimento deste trabalho.

Em primeiro lugar, não poderia deixar de agradecer a Deus pela vida e por ter me dado uma família tão maravilhosa. Obrigado aos meus amados pais Elias e Elena pelo imenso amor com que sempre cuidaram de mim e pelo apoio que sempre me deram nas escolhas da minha vida. Obrigado minha querida irmã Fernanda e meus queridos sobrinhos Gabriel e Raphael por entenderem a importância de todas as vezes que não pude dar atenção à vocês. Peço desculpas pelas vezes que o tio André não pôde brincar e fazer molequices com vocês. Te amo família.

Obrigado ao samba e ao carnaval por serem peças fundamentais no equilíbrio da minha vida. Durante este mestrado foram muitas noitadas de samba, ensaios e desfiles que contribuíram para aliviar minha cabeça nos momentos mais difíceis.

Obrigado aos meus amados amigos, presentes sempre na minha vida e no meu coração. Sambistas ou não, ruins da cabeça ou do pé, não importa, amo muito vocês: Fernanda Kelly, Dna. Marlene, Ney, Luciano Tinoco, Nair, Cleber, Rodrigo, Gilberto, Fernanda "Carraro", Leandro, Diego Araújo, Big, Dedeco, Vini, Dna. Ivanisia, Elaine, Dida, Vlad, Marcela, Cleide, Igor, Ivan, Ney. Para não haver brigas listei vocês na ordem em que os conheci.

Agradeço de coração também aos novos amigos que fiz na COPPE durante este mestrado, são eles: Cristiane, André Nathan, Luciana, Gustavo, Vivian, Ivomar, Bernardo, Patricia e João Vitor. E também aos meus amigos do trabalho que sempre me apoiaram bastante: Alon, Raphael, Zé Luiz, Mônica, Fafá, Flávio e Solange. Um agradecimento especial ao meu chefe Luiz Cláudio pela liberação parcial que me concedeu nos momentos em que mais precisei.

Obrigado ao meu orientador Claudio Amorim por ter aceitado me orientar quando a professora Inês foi para Portugal. Muito obrigado a querida professora Inês Dutra pelo apoio que sempre me deu, mesmo após o seu desligamento da COPPE Sistemas. Um obrigado bastante especial aos queridos professores da UFF Alexandre Plastino e Simone Martins. Vocês foram fundamentais para o desenvolvimento deste trabalho. Agradeço pela força que sempre me deram e por sempre acreditarem no meu potencial.

Um agradecimento a dois amigos que contribuíram para o desenvolvimento deste trabalho. Obrigado ao Jacques, doutorando da UFF na área de processamento paralelo, que sempre me socorreu quando eu mais tive problemas neste mundo paralelo. Sua atuação no laboratório da UFF foi fundamental para a realização dos experimentos apresentados neste trabalho. Obrigado ao meu amigo Gilberto Medeiros, mestrando da PUC Rio na aréa de computação gráfica, por ter me fornecido a versão seqüencial da aplicação *Ray Tracing*, utilizada neste trabalho. Gostaria de agradecer também ao Graziano, da universidade de San Diego, pelo auxílio na utilização da ferramenta NWS.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

AVALIAÇÃO DE ESTRATÉGIAS DE BALANCEAMENTO DE CARGA DO TIPO MESTRE-ESCRAVO PARA APLICAÇÕES SPMD EM *CLUSTERS* E *GRIDS* COMPUTACIONAIS

André Lucio de Oliveira

Setembro/2008

Orientador: Claudio Luis de Amorim

Programa: Engenharia de Sistemas e Computação

O desempenho de aplicações paralelas do tipo SPMD (*Single Program Multiple Data*) é fortemente afetado pelos fatores dinâmicos de desequilíbrio de carga. O uso de um algoritmo de balanceamento de carga adequado é essencial para superar os efeitos destes fatores. O principal objetivo deste trabalho é avaliar o desempenho de algumas estratégias de balanceamento de carga baseadas principalmente na abordagem mestre-escravo. Foram implementados algoritmos estáticos e dinâmicos baseados ou adaptados a partir de estratégias de escalonamento bastante utilizadas na literatura. Os dois algoritmos estáticos implementados utilizam diferentes métricas para distribuir tarefas para o processadores. O primeiro deles simplesmente distribui as tarefas de forma igualitária entre os processadores. O segundo utiliza a ferramenta *Network Weather Service* (NWS) para recuperar informações mais precisas sobre as características de uma dada máquina e sobre a rede, efetuando assim uma distribuição mais eficiente. As estratégias dinâmicas utilizam o modelo mestre-escravo e foram divididas em três grupos principais: (1) nas estratégias deste grupo o mestre distribui aos seus escravos blocos de tarefas de tamanho fixo, (2) neste grupo, foram implementadas seis estratégias diferentes, onde foi utilizada uma abordagem de redução de bloco durante a execução da aplicação, (3) a estratégia mestre escravo hieráquica, que utiliza uma topologia de processadores. O desempenho dos algoritmos foi analisado a partir da execução de duas aplicações reais com diferentes características e em diferentes ambientes de execução. Os resultados obtidos mostram a importância de se escolher a estratégia adequada para uma determinada condição.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

EVALUATING MASTER-SLAVE LOAD BALANCING STRATEGIES FOR SPMD APPLICATION IN COMPUTATIONAL CLUSTERS AND GRIDS.

André Lucio de Oliveira

September/2008

Advisor: Claudio Luis de Amorim

Department: Systems Engineering and Computer Science

The performance of SMPD (Single Program Multiple Data) applications is strongly affected by load imbalance. The use of a suitable load balancing algorithm is essential to overcome the effects of the imbalance factors. The main objective of this work is to evaluate the performance of some load balancing strategies based on static and master-slave models. Static and dynamic algorithms were implemented and adapted on well-known scheduling strategies used in the literature. The two static algorithms implemented use two different metrics to distribute work. The first one simply distributes the tasks equally between machines. The second one uses the Network Weather Service (NWS) to take more precise information about the characteristics of a given machine and about the network. The dynamic strategies use the master-slave model and are divided in three main groups: (1) the algorithm distributes work by using a fixed work block size, (2) in this group, six different algorithms were implemented reducing the work block size during the execution in order to have better performance by reducing the grain size of the work along the execution time, (3) the algorithm distributes work in an hierarchical fashion. The impact of the algorithms was analyzed by running two real applications with different parallel patterns on two different distributed environments. The results show the importance of using proper load balancing strategies for specific environment conditions.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Aplicações paralelas podem ser desenvolvidas seguindo-se dois modelos de programação: paralelismo de função, também conhecido como paralelismo de controle, ou paralelismo de dados. No primeiro, diferentes operações são executadas sobre conjuntos de dados distintos, enquanto que no segundo uma única função atua sobre os dados. O paralelismo de dados apresenta maior facilidade de implementação, pois um único programa é executado em diversos processadores. Neste modelo, uma tarefa consiste na execução do código comum sobre uma fração do conjunto de dados.

Quando uma aplicação utiliza o paralelismo de dados em arquiteturas paralelas sem sincronismo no nível de instrução, é caracterizada como sendo do tipo SPMD (*Single Program, Multiple Data*). Neste tipo de aplicação, o mesmo programa é executado de forma assíncrona nos diferentes processadores, manipulando conjuntos distintos de dados [55]. A facilidade na implementação deste modelo, determinada principalmente pela necessidade de controle sobre um único código executável, e a existência de um grande número de aplicações do tipo SPMD [13, 16, 38] faz com que esse tipo de abordagem seja amplamente utilizada.

O desempenho de uma aplicação paralela do tipo SPMD pode ser prejudicado por diversos fatores que causam o desequilíbrio de carga, tais como: a heterogeneidade da arquitetura paralela adotada, o desconhecimento da quantidade de processamento envolvida em cada tarefa, a criação dinâmica de tarefas, além da heterogeneidade e da variação da carga externa à aplicação nos diversos processadores em um ambiente de execução não dedicado. A adequada distribuição de tarefas entre os processadores é fun-

damental para minimizar o tempo de execução de uma aplicação SPMD e, conseqüentemente, maximizar a utilização dos recursos computacionais disponíveis. A identificação desta distribuição ideal de tarefas caracteriza o problema de balanceamento de carga e a utilização de uma estratégia de balanceamento de carga adequada é fundamental para a redução dos efeitos dos diversos fatores de desequilíbrio [25, 27, 37, 52, 62].

Nas últimas décadas, foram propostas e implementadas muitas estratégias para resolver o problema da manutenção do equilíbrio de carga de uma aplicação SPMD. As estratégias de balanceamento de carga podem ser divididas basicamente em dois grandes grupos: estáticas e dinâmicas. Uma estratégia é dita estática quando utiliza uma política de distribuição de tarefas que distribui as tarefas no início da execução da aplicação, sem levar em consideração possíveis desequilíbrios de carga que possam ocorrer durante a mesma. Já as estratégias dinâmicas visam garantir o equilíbrio de carga durante toda a execução. Para isto, utilizam políticas de distribuição mais elaboradas, associadas, ou não, à políticas de transferência de tarefas.

Além da grande variedade de estratégias de balanceamento de carga que podem ser implementadas utilizando-se as abordagens estática e dinâmica, os fatores externos de desequilíbrio, já mencionados, também representam uma grande dificuldade no momento de se definir qual é a estratégia mais adequada para uma determinada aplicação SPMD. Sendo assim, as características das tarefas de uma aplicação somadas as condições do ambiente de execução têm grande influência no desempenho de uma estratégia. Por este motivo, é de grande importância a realização de um estudo que avalie o desempenho dos algoritmos de balanceamento de carga sob estas diversas circunstâncias.

O paradigma mestre-escravo tem sido amplamente utilizado para resolver problemas de balanceamento de carga. A utilização deste tipo de abordagem para implementar algoritmos de balanceamento de carga tem sido motivada pelos bons resultados apresentados e pela sua facilidade de implementação [9, 26, 29, 31, 48, 49]. Em estratégias deste tipo, o processador mestre, que contém o conjunto inicial de tarefas, distribui blocos de tarefas para os seus escravos e toda vez que um escravo termina de executar o bloco recebido solicita um novo bloco ao mestre. Desta forma, o equilíbrio de carga tende a ser garantido pela envio de tarefas de forma proporcional à capacidade de cada processador escravo, através de uma distribuição sob demanda.

Estratégias do tipo mestre-escravo podem apresentar algumas desvantagens. A centralização do controle em um único ponto pode sobrecarregar o mestre e dificultar a implementação de uma política de tolerância a falhas. Além disso, a definição do tamanho do bloco de tarefas a ser enviado para um escravo também pode afetar o desempenho do algoritmo de balanceamento de carga e, conseqüentemente, da aplicação.

O objetivo principal deste trabalho é avaliar o desempenho das estratégias de balanceamento de carga. Um total de dez estratégias diferentes foram implementadas, sendo oito delas dinâmicas do tipo mestre-escravo e as outras duas do tipo estático. A análise de desempenho destas estratégias levou em consideração as características de cada uma das aplicações SPMD desenvolvidas e dos ambientes onde as mesmas foram executadas.

Para avaliar o desempenho das estratégias de balanceamento de carga em diferentes cenários, foram utilizadas três aplicações SPMD distintas: uma aplicação física que calcula a dispersão térmica em meios porosos, uma aplicação da área de computação gráfica para renderização de imagens com a técnica de *Ray Tracing* e uma aplicação sintética utilizada para simular aplicações com diferentes características. Os experimentos realizados com estas aplicações tiveram como propósito a análise do desempenho das estratégias quando se varia o tempo de computação e o tamanho de cada tarefa e o número total de tarefas. Além destes fatores, a heterogeneidade de uma aplicação, determinada pelo tempo de execução das suas tarefas, também foi levada em consideração.

Os experimentos foram realizados em diversos ambientes computacionais. No que diz respeito à arquitetura paralela adotada, estes experimentos foram conduzidos em um *cluster* e, posteriormente, em um *grid* computacional constituído por dois *clusters* geograficamente distantes. No *cluster*, onde os processadores eram todos homogêneos, enfatizou-se a análise do desempenho das estratégias de balanceamento de carga quando os processadores possuíam ou não cargas externas à aplicação. Aspectos como o grau de concorrência por CPU e variação da carga externa no decorrer da execução de uma aplicação também foram considerados. A simulação de um ambiente dinâmico de concorrência foi possível através de uma ferramenta geradora de carga externa, denominada *Playload* [22]. Nos experimentos realizados no *grid*, onde existe heterogeneidade apenas entre processadores de *clusters* distintos, optou-se por variar a capacidade dos

canais de comunicação entre os *clusters*. Para isto, foi utilizada a ferramenta TC (*Traffic Control*) que permite a configuração destes canais.

Outra questão considerada na avaliação do desempenho das estratégias de balanceamento de carga neste trabalho diz respeito ao tamanho dos blocos em uma estratégia do tipo mestre-escravo. A necessidade deste tipo de avaliação se deve ao fato de que o desempenho de uma aplicação pode ser diretamente afetado pelos desequilíbrios de carga gerados a partir da distribuição de blocos de tamanhos indevidos. Blocos de tamanho pequeno podem necessitar de um número grande de envios, aumentando assim o tempo de comunicação. Em contrapartida, blocos de tamanho grande podem gerar desequilíbrios de carga no final da execução da aplicação quando os últimos blocos forem enviados para alguns escravos, enquanto os demais podem ficar ociosos.

Para tentar resolver este problema, foram propostas na literatura algumas variações da estratégia mestre-escravo que adotam uma abordagem de redução de bloco [33, 34, 51]. Nestas estratégias, os blocos iniciais possuem um tamanho grande que vão se reduzindo gradativamente durante a execução, até que um bloco contenha apenas uma tarefa. Esta medida visa garantir possíveis desequilíbrios e, além disso, tirar do usuário a responsabilidade por definir o tamanho do bloco.

As estratégias *Guided Self Scheduling*(GSS) [51] e *Factoring* [33] são estratégias do tipo mestre-escravo com redução de bloco, propostas para ambientes de memória compartilhada. A estratégia *Weighted Factoring* [34] consiste em uma variação da estratégia *Factoring*, proposta para ambientes distribuídos, a qual associa um peso a cada escravo como forma de definir a capacidade de processamento de cada um deles. A redução de bloco também é implementada nesta estratégia e o tamanho dos blocos são definidos de acordo com o peso associado a cada escravo.

Neste trabalho, todas as estratégias foram implementadas e avaliadas com o auxílio do SAMBA-Grid [47], uma ferramenta que tem por finalidade facilitar a implementação e a execução de aplicações SPMD paralelas. Das dez estratégias avaliadas três já estavam disponíveis na biblioteca de balanceamento de carga desta ferramenta: a estratégia estática simples, que possui uma política de distribuição de tarefas bem simples, dividindo igualmente o conjunto inicial de tarefas entre todos os processadores; a estratégia mestre-escravo com bloco fixo, onde o mestre sempre envia blocos de mesmo tamanho para os seus escravos; e a estratégia mestre-escravo hierárquico, que propõe

uma topologia hieráquica de processadores como forma de diminuir o tempo de comunicação. As demais estratégias avaliadas consistem em variações destas. Uma variação da estratégia estática implementada foi a denominada estática com NWS que, em vez de distribuir a mesma quantidade de tarefas para todos os processadores, procura distribuí-las de acordo com a capacidade de processamento de cada um deles. Para obter a capacidade de processamento de cada processador, foi utilizada a ferramenta *Network Weather Service* (NWS) proposto por Wolski [57], a qual permite a monitoração de alguns tipos de recursos computacionais, fornecendo previsões do comportamento destes no decorrer do tempo. As demais estratégias do tipo mestre-escravo implementadas utilizam a abordagem com redução de bloco. Além das estratégias GSS e *Factoring*, também foram implementadas a *Weighted Factoring* e mais três variações desta, onde foram utilizadas novas técnicas para definição do peso de um escravo. Algumas destas técnicas utilizam as medições fornecidas pelo NWS como parâmetro para auxiliar na definição do peso de cada escravo.

Esta dissertação está organizada da seguinte forma. No Capítulo 2, são apresentadas as ferramentas de apoio à implementação, execução e avaliação das estratégias de balanceamento de carga: o *framework* SAMBA-Grid [4, 5, 47], que facilitou a implementação e avaliação de todas as estratégias testadas, o *Network Weather Service* (NWS) [57], que possibilita a monitoração de recursos computacionais, utilizado em algumas estratégias para se tomar decisões para a distribuição de tarefas, o *Playload* [22], uma ferramenta geradora de carga externa que possibilita a avaliação das estratégias em um ambiente dinâmico de concorrência, e o TC (*Traffic Control*) [2], que possibilitou a simulação de uma rede com canais de comunicação mais lentos. No Capítulo 3, são apresentadas com detalhes todas as estratégias de balanceamento de carga implementadas e avaliadas neste trabalho. No Capítulo 4, são apresentadas as aplicações SPMD utilizadas neste trabalho, caracterizando-as no contexto SPMD e explicando o padrão de execução paralela de cada uma delas. No Capítulo 5, são apresentados os resultados dos experimentos e a análise do desempenho das estratégias para as aplicações exploradas. Finalmente, no Capítulo 6, são apresentadas as conclusões e perspectivas de trabalhos futuros.

No Apêndice A, podem ser encontradas as tabelas com os tempos computacionais e alguns dados estatísticos sobre as execuções das aplicações.

Capítulo 2

Ferramentas Utilizadas

Neste capítulo, serão apresentadas as ferramentas utilizadas neste trabalho. Na Seção 2.1, apresenta-se o SAMBA-Grid, um *framework* que deu suporte ao desenvolvimento dos algoritmos de balanceamento de carga e facilitou a execução de aplicações paralelas do tipo SPMD. Na Seção 2.2, será apresentado o *Network Weather Service (NWS)*, uma ferramenta para sistemas distribuídos que permite a monitoração de recursos computacionais e que auxiliou na elaboração de novas estratégias de balanceamento de carga. Na Seção 2.3, será apresentado o *Playload*, um gerador de carga externa que possibilitou maior diversificação dos experimentos computacionais. Por fim, na Seção 2.4, há uma breve explicação sobre a ferramenta *Traffic Control* (TC), do Linux, utilizada para controlar o tráfego em redes.

2.1 *Framework* SAMBA-Grid

Uma aplicação SPMD é caracterizada por uma única função que opera sobre conjuntos distintos de dados. A execução das operações definidas nesta função, sobre cada conjunto de dados, constitui uma tarefa desta aplicação. Sendo assim, uma aplicação SPMD pode ser caracterizada por três componentes: (a) um código único que é replicado para a execução das tarefas; (b) uma estratégia de balanceamento de carga e (c) uma estrutura central (que inicializa e termina a aplicação, gerencia tarefas e controla a execução das outras partes). A primeira é específica a cada aplicação, mas as outras duas partes geralmente não são. Desta forma, aplicações do tipo SPMD podem ser modeladas por um *framework*.

Frameworks capturam decisões que sejam comuns a todas as aplicações de um dado domínio [28]. Eles promovem reusabilidade em alto grau, pois não só as partes do código são reutilizadas, mas também a arquitetura da aplicação.

Neste trabalho, a avaliação do desempenho das estratégias de balanceamento de carga foi facilitada pelo uso de um *framework* denominado SAMBA-Grid [4, 5], que consiste em uma das adaptações da ferramenta SAMBA, proposta em [47]. A versão SAMBA-Grid permite o desenvolvimento e a execução de aplicações do tipo SPMD também em *grids* computacionais, uma vez que a versão inicial permitia a execução apenas em *clusters*.

É importante ressaltar que, em um *framework*, algumas partes são deixadas propositalmente incompletas. Estas partes, chamadas *hot spots*, representam a diferença entre as aplicações distintas de um mesmo domínio. O desenvolvedor precisa completar estes *hot spots* para obter um programa completo. Este processo é conhecido como instanciação do *framework*. Assim, o usuário típico do *framework* é um desenvolvedor de aplicações.

O *framework* SAMBA-Grid modela a classe de aplicações paralelas SPMD. Seus principais *hot spots* são responsáveis pela geração de tarefas, execução de uma tarefa e tratamento de resultados. O SAMBA-Grid permite ao desenvolvedor de aplicações paralelas utilizar diferentes algoritmos de balanceamento de carga, de uma maneira *plug-and-play*. Conseqüentemente, possibilita a geração de versões distintas da aplicação para diferentes estratégias de balanceamento de carga sem custo de reprogramação.

Este *framework* é ilustrado na Figura 2.1 através da especificação das suas classes e seus relacionamentos, usando uma notação gráfica baseada em UML [14]. Os métodos que definem os *hot spots* estão sublinhados. A seguir serão discutidas as principais classes no *framework*.

A *classe Mediador* ativa e controla a execução da aplicação SPMD. Seu comportamento é definido basicamente por dois métodos: um que inicialmente gera as tarefas e outro que trata os resultados da execução. O objeto mediador possui dois outros objetos: um repositório de tarefas e outro responsável pelo balanceamento de carga.

A *classe Repositório* é responsável pelo gerenciamento das tarefas a serem executadas. O objeto repositório possui um conjunto de zero ou mais tarefas. Seu comportamento é definido por dois métodos principais, um para inserção e outro para

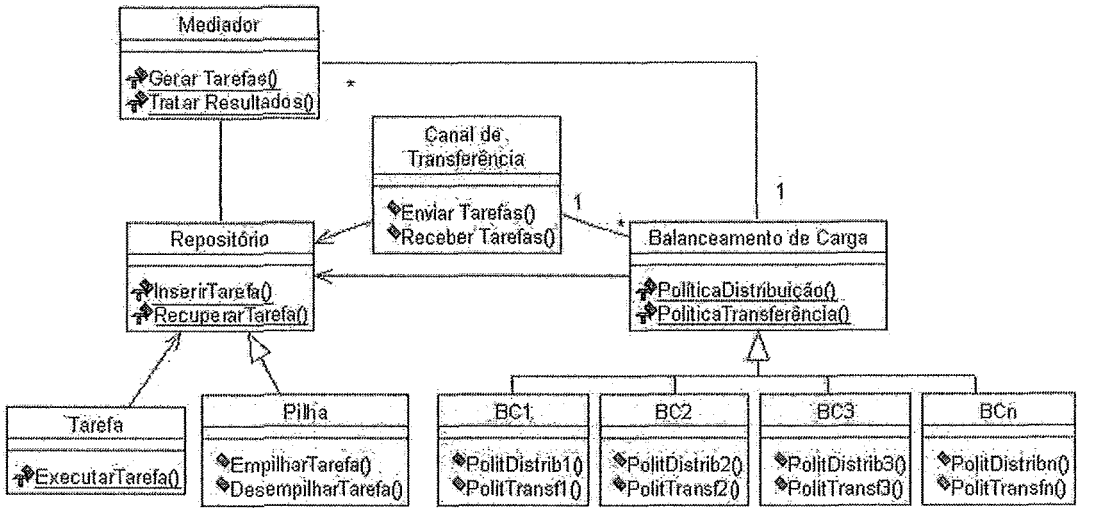

Figura 2.1: Estrutura do *framework* SAMBA-Grid.

recuperação de tarefas. A organização existente no repositório (uma pilha) é implementada por uma subclasse da classe repositório. Outra estrutura pode ser utilizada implementando-se uma nova subclasse.

A *classe Tarefa* representa uma tarefa da aplicação. Seu principal método é o responsável pela execução das tarefas - o código único SPMD. Objetos desta classe precisam acessar o repositório, pois uma tarefa em execução pode gerar novas tarefas a serem inseridas neste.

A *classe Balanceamento de Carga* é responsável pelo balanceamento de carga entre os processadores. Seus dois principais métodos implementam políticas de distribuição e transferência de tarefas. Objetos desta classe possuem um canal de transferência que é usado para enviar e receber tarefas. Objetos de balanceamento de carga precisam acessar o repositório para recuperar tarefas que serão executadas. Cada subclasse desta classe implementa uma estratégia diferente de balanceamento de carga. Quando o *framework* é instanciado, o usuário pode escolher uma destas estratégias ou implementar uma nova.

A política de distribuição é responsável pela forma como as tarefas serão distribuídas entre os processadores. Já a política de transferência é utilizada por algumas estratégias com o intuito de realizar a transferência de tarefas de um processador para outro, quando um processador acaba suas tarefas e outros processadores ainda têm tarefas em seus repositórios. A composição destas duas políticas determina as ações em uma

estratégia de balanceamento de carga.

A *classe Canal de Transferência* fornece dois métodos que serão usados para enviar e receber tarefas. Objetos desta classe precisam também acessar o repositório para recuperar tarefas que serão enviadas e inserir tarefas que serão recebidas.

Os métodos ExecutarTarefa() (o núcleo de uma aplicação), GerarTarefas() e TratarResultados() são específicos de cada aplicação. Estes *hot spots* precisam ser implementados quando uma nova aplicação SPMD é criada.

Para estabelecer a comunicação entre os processos que executam uma aplicação paralela SPMD, o SAMBA-Grid utiliza a biblioteca MPI (*Message Passing Interface*) [30, 43], que possibilita a troca de mensagens entre os processadores. O padrão MPI define uma biblioteca de funções que possibilitam trocas de dados entre processadores. Essas funções incluem comunicação ponto-a-ponto, na qual uma operação de envio é usada para iniciar uma transferência de dados entre dois processos e uma operação de recebimento correspondente é usada para extrair os dados da estrutura de dados no espaço de memória da aplicação. O principal objetivo do MPI é otimizar a comunicação e aumentar o desempenho de aplicações paralelas ou distribuídas em máquinas paralelas, como supercomputadores e *cluster* de processadores.

Para possibilitar a execução de aplicações SPMD em um *grid* computacional, a ferramenta SAMBA-Grid utiliza a infra-estrutura do *middleware* Globus Toolkit [44]. O Globus é um conjunto integrado de ferramentas de software que visam facilitar a criação de aplicações que possam explorar as capacidades avançadas de um *grid.* Para isso, dispõe de serviços tais como: submissão e controle de processos, gerenciamento e descoberta de recursos, protocolos de comunicação, serviços de escalonamento, protocolos de autenticação, movimentação de dados e facilidades de acesso a dados remotos.

2.2 *Network Weather Service*

O NWS é um sistema distribuído proposto em [57] que realiza o monitoramento periódico dos principais recursos de um sistema computacional. Para isso, são registradas medições e geradas previsões da situação destes recursos ao longo do tempo. Estas medições e previsões têm sido freqüentemente utilizadas como parâmetros para tomada de decisões em algoritmos dinâmicos de escalonamento. Em [7, 10, 20, 31, 35, 39, 53, 60],

podem ser encontrados exemplos do uso do NWS com esse objetivo.

Entre os objetivos básicos do NWS estão: monitorar alguns tipos de recursos computacionais, fornecer uma previsão da disponibilidade de um dado recurso monitorado, utilizar minimamente os recursos durante o processo de monitoração (com baixo grau de intrusão) e estar sempre disponível quando uma medição for solicitada.

O NWS informa dados estatísticos de acordo com as variações que ocorrem em cada relatório de sensoriamento, o que permite a requisição de uma previsão estatística da possível tendência de uma determinada máquina, para uma característica específica em um dado intervalo de tempo.

2.2.1 Arquitetura do NWS

A arquitetura do NWS é composta de três processos que atuam de forma distribuída e interagem entre si. São os seguintes:

- **Processo servidor de nome**: Implementa um sistema de diretórios que é responsável pelo registro de todos os outros componentes da arquitetura, além de realizar a ligação destes com informações de contato de baixo nível (como o endereço IP dos componentes e identificação da porta de comunicação);

- **Processo de estado persistente ou memória**: Responsável pelo armazenamento das medições feitas pelos sensores;

- **Processo sensor** - Responsável pela coleta de informações sobre a disponibilidade dos recursos sendo monitorados pelo NWS.

Na Figura 2.2, são apresentados três exemplos de configuração dos três processos NWS. Na configuração 2.2(a), observa-se que existe uma máquina exclusiva para executar o processo servidor de nome (SN) e uma para o processo de memória (M). Os processos sensores (S) monitoram os recursos nas quatro máquinas restantes. As setas pontilhadas indicam registros de processos no servidor de nome ou a qual processo de memória um determinado sensor deverá se reportar quando desejar armazenar suas medições. As setas com dupla orientação (não pontilhadas) foram usadas apenas para representar a comunicação entre os sensores quando estiverem sendo feitas monitorações de rede. Em 2.2(b) nota-se uma pequena diferença, pois existem dois processos

de memória e um deles está hospedado em uma máquina que executa um sensor. Repare que um sensor armazena suas medições por meio de um processo de memória e os demais através de outro. Uma configuração mais centralizada pode ser observada em 2.2(c) onde, em uma única máquina, estão residentes os três tipos de processo NWS e nas demais apenas os sensores. Neste caso, uma única máquina fica responsável pelo registro de processos e pela coleta das medições.

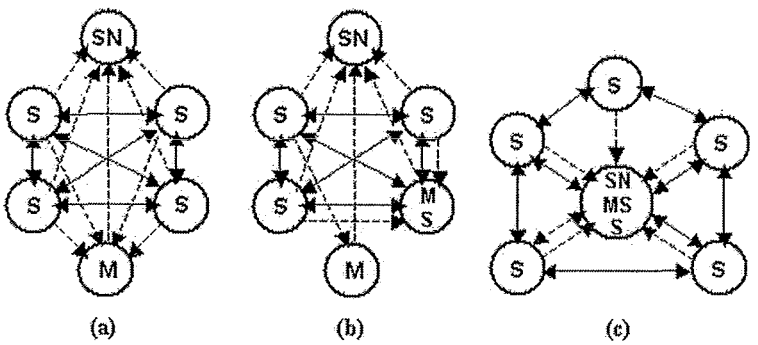

Figura 2.2: Exemplos de configurações dos processos NWS.

Cada uma destas configurações tem vantagens e desvantagens, ficando a cargo do administrador do sistema avaliar os benefícios que cada uma delas pode trazer. Não é objetivo deste trabalho fazer este tipo de análise e em todos os experimentos foi utilizada uma configuração centralizada (Figura 2.2(c)), devido à maior simplicidade apresentada por esta.

2.2.2 Registro no Servidor de Nome

Como foi visto anteriormente, o servidor de nome é responsável pelo registro de cada um dos processos que constituem a arquitetura, além de realizar a ligação destes com informações de baixo nível, como endereços IP e a porta onde esse processo pode ser encontrado. Por isso, o endereço do servidor de nomes é o único que precisa ser conhecido por todos os processos da arquitetura, e todos os outros podem ser obtidos a partir dele.

Os registros do NWS seguem o modelo do LDAP (*Lightweight Directory Access Protocol*) [32]. Neste modelo, os registros são feitos na forma de objetos que são constituídos de uma série de atributos, cada um com um nome e um valor. Cada objeto

possui um atributo *name* que o identifica de forma única. Além disso, cada objeto possui o atributo *objectclass*, que especifica o tipo do objeto. Os demais atributos são especificados de acordo com o tipo de objeto em questão.

O *objectclass nwsHost*, por exemplo, identifica um objeto que tem por objetivo registrar no servidor de nome as informações de um processo NWS (servidor de nome, memórias e sensores). Os objetos de registros de processos NWS possuem também os seguintes atributos: *hostType*, *ipAddress*, *owner*, *port*, *started*, *version*, *systemType*, que representam o tipo de *host* registrado, o endereço IP do *host*, o usuário que iniciou o processo, a porta onde ele receberá conexões, a data em que foi iniciado, a versão do NWS e o tipo de sistema operacional que estão sendo usados. Na Tabela 2.1, encontra-se um exemplo de registro para cada tipo de processo NWS: um servidor de nome, uma memória e um sensor [40].

name :n08.par.inf.puc-rio.br:8090	name :n01.par.inf.puc-rio.br:8050	name :n02.par.inf.puc-rio.br:8060
objectclass:nwsHost	objectclass:nwsHost	objectclass:nwsHost
hostType :**nameserver**	hostType :**memory**	hostType :**sensor**
ipAddress :139.82.36.10	ipAddress :139.82.36.10	ipAddress :139.82.36.32
owner :andre	owner :andre	owner :andre
port :8090	port :8050	port :8060
started :Mar19,2008-10:52:16	started :Mar19,2008-10:52:19	started :Mar19,2008-10:52:31
version :2.12.2	version :2.12.2	version :2.12.2
systemType :Linux	systemType :Linux	systemType :Linux
timestamp :1205937796	timestamp :1205937439	timestamp :1205936551
expiration :1205939596	expiration :1205939239	expiration :1205938351

Tabela 2.1: Exemplos de objetos para registro de processos do NWS.

2.2.3 Armazenamento das Medições

O processo de estado persistente (ou simplesmente memória) é o responsável pelo armazenamento das medições feitas pelos sensores instalados nos recursos monitorados. Para isso, ele oferece um serviço de armazenamento e envio de *strings* de texto que contêm as medições e um marcador de tempo, que indica o momento em que essas medições foram realizadas.

Quando o usuário deseja recuperar as medições de um recurso monitorado, solicita ao servidor de nome e este identifica qual processo de memória armazena os dados coletados pelo sensor que realiza esta monitoração. A configuração mostrada na Figura 2.2(c) apresenta uma vantagem na recuperação de medidas, pois é centralizada e os processos servidor de nome e de memória são únicos e estão hospedados na mesma máquina.

Como as medições do NWS são usadas durante um intervalo limitado de tempo, e depois perdem sua validade, a memória armazena uma quantidade limitada de medições e as descarta assim que um determinado número de registros de medições é alcançado. A eliminação é feita usando-se um sistema de fila circular e o número de medições que deve ser armazenada, o qual é determinado no momento de criação do servidor de memória.

2.2.4 Sensores e Recursos Monitorados

Os sensores NWS registram objetos com *objectclass* do tipo *nwsSkill.* Um objeto deste tipo é registrado no servidor de nome com o objetivo de identificar que tipo de recurso um sensor está monitorando. Sendo assim, um único sensor pode ter vários registros de recursos no servidor de nome. Além disso, cada objeto *nwsSkill* possui um conjunto de atributos que indica a maneira como a monitoração do recurso será configurada [40]. Na Tabela 2.2, ilustra-se um exemplo de um objeto *nwsSkill* que identifica a monitoração de CPU no processador n02.par.inf.puc-rio.br.

name :n02.par.inf.puc-rio.br:8060.cpuMonitor
objectclass:nwsSkill
host :n02.par.inf.puc-rio.br:8060
option :nice
skillName :cpuMonitor
nice :0-to-10-int
timestamp :1205937451
expiration :1205939251

Tabela 2.2: Exemplo de um objeto *nwsSkill* para identificar monitoração de CPU.

Na Tabela 2.3, estão listados todos os *nwsSkills* que são atualmente suportados pelo NWS [59]. Neste trabalho, foram utilizados apenas os *skills cpuMonitor* e *tcpMessageMonitor.*

O *skill cpuMonitor* monitora a fração de CPU disponível para um novo processo através da medição *availableCpu* e a fração de CPU disponível para um processo que já esteja em execução através da *currentCpu* [40]. Este *skill* possui o atributo *nice* que determina o nível de prioridade de um processo durante a monitoração da disponibilidade de CPU. Este atributo pode ser configurado com um valor inteiro de 0 a 10, de forma que 0 indica que se deseja monitorar a disponibilidade de CPU para um processo com um nível alto de prioridade. Assim, quanto maior o valor do atributo *nice* menor

NwsSkill	Objetivo
cpuMonitor	Monitorar a fração de CPU disponível.
availabilityMonitor	Monitorar o tempo desde o último *boot* de uma máquina.
diskMonitor	Monitorar o espaço disponível em disco rígido.
startMonitor	Registrar o número de segundos desde que último sensor foi posto em execução.
fileSystemMonitor	Monitorar o desempenho de um sistema de arquivos especificado.
memoryMonitor	Monitorar o espaço disponível em memória principal.
memorySpeedMonitor	Monitorar velocidade de acesso a memória (acesso randômico ou seqüencial).
tcpConnectMonitor	Monitorar o tempo necessário para se estabelecer uma conexão TCP com um *host* especificado.
tcpMessageMonitor	Monitorar a largura de banda e a latência entre pares de *hosts*.

Tabela 2.3: nwsSkills disponíveis no NWS.

a prioridade de um processo.

Para realizar as medições do tempo disponível de CPU, o sensor do NWS combina informações obtidas através dos comandos *vmstat* e *uptime* do Unix com medições ativas que são realizadas periodicamente. A disponibilidade de CPU é definida como a fração de tempo de CPU que um processo pode obter.

Tipicamente, o comando *uptime* reporta a carga do sistema como uma média do número de processos na fila de prontos nos últimos 1, 5 e 15 minutos. Com isso, o sensor utiliza o valor determinado pelo *uptime* para calcular o tempo de disponibilidade da CPU para um processo que seja executado no momento que a medição do *uptime* foi realizada. Da saída gerada pelo comando *vmstat*, o sensor considera os valores das medições do tempo ocioso (*idle time*), tempo alocado para processos do usuário (*user time*) e do tempo alocado para processos do sistema (*system time*). Através da combinação desses valores, ele estima a fração de tempo que a CPU estará disponível. Como esses utilitários geram seus relatórios baseados em variáveis internas do próprio Unix, eles não consomem muitos recursos, sendo uma forma relativamente não intrusiva de medição.

Os comandos *uptime* e *vmstat* desconsideram algumas informações que podem ser

importantes para determinar a disponibilidade de CPU, como por exemplo as prioridades dos processos. Por este motivo, para obter medições mais precisas, o sensor de CPU incorpora medições ativas em seus cálculos. Ele executa, periodicamente, um processo *cpu-bound* que calcula a disponibilidade de CPU através da relação entre o tempo total de sua execução e o tempo em que ele ocupou a CPU. O sensor então compara esses dados com os valores obtidos com o *uptime* e o *vmstat* e utiliza a medição que estiver gerando o resultado mais preciso. Embora seja conhecido que processos *cpu-bound* são muito intrusivos, os autores desta ferramenta afirmam não ser significativo o *overhead* ocasionado por estes processos nas monitorações do NWS.

Além disso, o sensor utiliza heurística para ajustar a freqüência com que as medições ativas são realizadas, visando sempre diminuir a interferência do sensor no sistema. Enquanto as diferenças entre as medições passivas se mantêm relativamente estáveis, as ativas vão sendo realizadas em espaços de tempo maiores. Em sistemas estáveis, as medições ativas chegam a ser executadas apenas uma vez a cada hora, da mesma forma que, sempre que há grande diferença entre estas, o sistema passa a realizar medidas ativas com mais freqüência [57].

O *skill tcpMessageMonitor* monitora a largura de banda (medição *bandwidthTcp*) e latência (medição *latencyTcp*) entre cada par de máquinas do sistema. O tamanho da mensagem, do *buffer* do *socket* de envio e recebimento, assim como o tamanho dos *buffers* internos usados em cada chamada de sistema *send()* e *recv()*, podem ser parametrizados para cada sensor através dos atributos *buffer* e *size*.

Os sensores de rede do NWS se baseiam exclusivamente em medidas ativas. Cada medição se baseia em uma operação de rede cronometrada, como o tempo para envio de mensagens de tamanho fixo ou o tempo necessário para estabelecer e desfazer uma conexão TCP. Em intervalos regulares, cada sensor de rede se conecta com sensores localizados em outras estações e realiza uma ou mais séries de diferentes testes para coletar suas medições. Para conseguir medições de desempenho ponto-a-ponto de qualquer tipo, N sensores necessitam de $N^2 - N$ testes. Para evitar que isso interfira no desempenho geral da rede, os sensores são organizados de forma hierárquica, fazendo com que os sensores tenham medições ponto-a-ponto de subgrupos representativos da população de sensores. E, através dos valores obtidos para esses subgrupos, é possível determinar o desempenho de um dado par de sensores [57].

É importante ressaltar que as medições de rede fornecidas pelo NWS podem ser insuficientes para aplicações com padrões de comunicação onde múltiplos nós competem pelo mesmo *link*. Por exemplo, se dois pares de nós se comunicam, o NWS irá monitorar o desempenho de cada par separadamente. Se os dois canais de comunicação têm algum *link* em comum, estas previsões serão muito otimistas, já que a largura de banda deverá ser compartilhada. A versão atual do NWS não resolve este problema, porque carece de uma informação de topologia de rede que possibilite o conhecimento de quais *links* são compartilhados. Em [6], foi implementada uma ferramenta denominada TOPOMON que utiliza a infra-estrutura do NWS somado a uma topologia de rede que prevê este compartilhamento de *links*. Os autores afirmam que esta ferramenta fornece uma medição mais precisa da disponibilidade dos canais de comunicação. No entanto, esta ferramenta não está disponível para utilização pela comunidade acadêmica.

2.2.5 Controle dos Sensores

Os *skills* operam de acordo com os objetos registrados com *objectclass* do tipo *nwsControl*, os quais definem a freqüência e a duração dos experimentos. Diferentes *controls* suportam diferentes *skills*, e cada *control* tem um conjunto de opções de configuração que pode ser usado para guiar sua operação [40]. Os controles presentes atualmente no NWS são:

- **Clique**: Este controle coordena medições entre um conjunto de sensores do NWS, chamados de membros do clique. Esses membros se revezam fazendo medições, e só pode haver um membro realizando medições de cada vez. Isso é útil para evitar concorrência quando são realizadas medições das condições da rede entre máquinas. Esse controle pode ser usado tanto com a *skill tcpConnectMonitor* quanto com a *tcpMessageMonitor*. Ele possui o atributo *member:2-to-100-sensor*, que indica que devem ser especificados de 2 a 100 sensores do NWS como membros participantes do clique e o atributo *period:0-to-1-int*, que indica o número de segundos entre tentativas de conexões entre dois pares de nós.

- **Periodic**: Este controle pode ser utilizado com qualquer uma das *skills* quando se deseja coletar medições em tempos fixos. Possui o atributo opcional *period:0-to-1-int* que permite especificar a freqüência com que as medições serão realizadas.

Na Tabela 2.4, pode ser observado um objeto que descreve um *nwsControl* do tipo clique, indicando que o processador n00.par.inf.puc-rio.br se comunica apenas com os processadores de sua clique para realizar monitorações de rede (*skills tcpConnectMonitor* e *tcpMessageMonitor*).

name :n00.par.inf.puc-rio.br:8060.clique
objectclass:**nwsControl**
controlName:clique
host :n00.par.inf.puc-rio.br:8060
option :member,period
skillName :tcpConnectMonitor,tcpMessageMonitor
member :2-to-100-sensor
period :0-to-1-int
timestamp :1205935651
expiration :1205937451

Tabela 2.4: Exemplo de um objeto *nwsControl* para controle de uma clique.

Um objeto denominado *nwsActivity* também deve ser registrado no servidor de nome. Este objeto associa uma *skill* a um determinado controle (*nwsControl*), com o objetivo de configurar a monitoração do recurso. Na Tabela 2.5, pode ser verificado um exemplo de um objeto *nwsActivity*. Observa-se que a atividade de monitoração de CPU (*skill cpuMonitor*) no processador n00.par.inf.puc-rio.br é controlada pelo *nwsControl* do tipo *periodic*, determinando que as medições relacionadas à CPU serão coletadas a cada cinco segundos (*period*). O atributo *resource* identifica os tipos de medições coletadas por esta atividade.

name :CPU-n00.par.inf.puc-rio.br
objectclass:**nwsActivity**
controlName:periodic
host :n00.par.inf.puc-rio.br:8060
option :period
resource :availableCpu,currentCpu
skillName :cpuMonitor
period :5
timestamp :1205934754
expiration :1205936554

Tabela 2.5: Exemplo de um objeto *nwsActivity*

O registro de uma clique também fica armazenado na forma de objeto do tipo *nwsActivity*. Este objeto especifica, além dos membros da clique, as medições coletadas (largura de banda e latência), o tamanho dos pacotes de envio e o período de execução dos experimentos. Na Tabela 2.6, pode ser observado um exemplo deste objeto.

objectclass:nwsActivity
controlName:**clique**
host :n00.par.inf.puc-rio.br:8060
option :size,period,member
resource :bandwidthTcp,latencyTcp
skillName :tcpMessageMonitor
size :32
period :1
member : n00.par.inf.puc-rio.br:8060, n01.par.inf.puc-rio.br:8060, n02.par.inf.puc-rio.br:8060
cycleTime :0
timestamp :1205949541
expiration :1205951341

Tabela 2.6: Exemplo de um objeto *nwsActivity* do tipo clique.

2.2.6 Recuperação de Medições e Previsões

Todas as medidas coletadas e calculadas por um sensor são armazenadas em arquivos. Estes arquivos ficam hospedados no processador que contém o processo de memória responsável por guardar os dados deste sensor. Além disso, os arquivos são identificados por objetos do tipo *nwsSeries* que descrevem o conteúdo de cada um deles, especificando o tipo de medida nele contido e a quais processadores esta medida está relacionada. Na Tabela 2.7, é mostrado um exemplo de um objeto *nwsSeries* que identifica um arquivo de séries que armazena a disponibilidade de CPU para um novo processo (*availableCpu*) a ser executado no processador n02.par.inf.puc-rio.br.

name :n02.par.inf.puc-rio.br:8060.availableCpu.0
objectclass:**nwsSeries**
host :n02.par.inf.puc-rio.br:8060
label :CPU-Fraction
resource :availableCpu
activity :CPU-n02.par.inf.puc-rio.br
option :nice
nice :0
memory :n01.par.inf.puc-rio.br:8050
timestamp :1205934756
expiration :1205943756

Tabela 2.7: Exemplo de um objeto *nwsSeries*.

O NWS disponibiliza, além das medições coletadas, uma consulta de previsões (*forecast*) que indicam o comportamento futuro de um recurso monitorado. A metodologia de previsão utilizada pelo NWS assume que o desempenho de cada recurso pode ser medido quantitativamente. Definir uma previsão de forma qualitativa, indicando que um recurso vai estar mais ou menos sobrecarregado no futuro, pode não ser suficiente para o desenvolvimento de um escalonamento efetivo. Por este motivo, as previsões do

NWS são calculadas com base em registros históricos que descrevem o comportamento do recurso durante um período de tempo passado, fornecendo também a margem de erro associada a cada previsão. Em [58, 59], são apresentados estudos sobre o método utilizado para calcular as previsões do NWS e são feitas análises para mostrar que estes cálculos não são intrusivos.

Além da consulta de registros e medições/previsões via linha de comandos pelo Shell Linux, o NWS fornece uma API (nws-api.h), implementada na linguagem C, onde o usuário dispõe de métodos que capturam as medições e previsões coletadas. Assim, uma aplicação que incluir esta API poderá facilmente acessar os dados registrados pelo NWS. Neste trabalho, esta API foi utilizada com o objetivo de obter as medições que auxiliem nas decisões de balanceamento de carga.

2.3 *Playload*

Para se fazer uma análise consistente do desempenho de um algoritmo de balanceamento de carga, se faz necessária a realização de experimentos em diversos ambientes de execução. Um ambiente de execução está relacionado aos aspectos físicos do sistema computacional na qual uma aplicação será executada, tais como, velocidade de CPU, memória, latência de rede, etc.

A velocidade de CPU pode ser considerada um dos fatores mais importantes dentro de um sistema computacional. Sendo assim, o compartilhamento deste recurso por um grupos de processos pode afetar significativamente o desempenho de cada um destes processos. O grau de concorrência por este recurso pode sofrer grandes variações, uma vez que novos processos podem ser iniciados, enquanto que outros podem terminar de executar suas operações.

Uma avaliação mais ampla sobre uma estratégia de balanceamento de carga deve ser feita por meio da realização de experimentos em ambientes dinâmicos de concorrência. Para reproduzir este tipo de ambiente foi utilizada a ferramenta *Playload* [22], que implementa a técnica denominada *load trace playback* em sistemas Unix. O *Playload* tem por objetivo reproduzir, em um processador, a variação de carga de trabalho coletada a partir de um outro processador. Para isto, monitora a média de carga de um processador e registra estes valores em um arquivo de *traces*, o qual reflete o

histórico da variação da carga neste processador durante um determinado período de tempo. Assim, este arquivo servirá como entrada para o *Playload* quando o usuário desejar reproduzir esta mesma variação de carga em um outro processador [41].

O *Playload* foi implementado na linguagem C e utiliza grupos de processos que executam *loops* para simular o histórico da variação de carga, definido em um arquivo de *traces*. Além disso, esta ferramenta pode ser configurada para operar no modo *time-based*, onde uma determinada configuração de carga persiste apenas por um período de tempo pré-definido no arquivo de *traces*, ou no modo *work-based* que considera a existência de cargas externas aos processos gerados pelo *Playload* e persiste com uma determinada configuração de carga enquanto o trabalho definido por esta não for finalizado.

Em [42], são disponibilizados diversos arquivos de *traces* coletados de algumas máquinas do *cluster* de um laboratório da universidade de Chicago, nos Estados Unidos. Cada trace tem uma característica particular e é classificado segundo dois parâmetros. O primeiro se refere ao consumo médio de CPU e o segundo à variação deste consumo no decorrer do tempo. A Tabela 2.8, define as combinações possíveis destes dois parâmetros, o que caracteriza cada tipo de *trace*. Cada *trace* é representado por duas letras, a primeira indicando o *load average* da CPU (L - *low* ou H - *high*) e a segunda, a variação da carga (L - *low* ou H - *high*). Um trace é considerado com *load average* baixo (L) quando tem valores compreendidos entre 0,046 e 0,25, e alto (H) quando seus valores estão entre 0,96 e 2,28. Já a variação de carga em um *trace* é dita baixa (L) quando está entre 0,05 e 0,16, e alta (H), entre 0,28 e 0,57.

Tipo do Trace	**Significado**
LL	Baixo consumo de CPU e baixa variação de carga.
LH	Baixo consumo de CPU e alta variação de carga.
HL	Alto consumo de CPU e baixa variação de carga.
HH	Alto consumo de CPU e alta variação de carga.

Tabela 2.8: Tipos de *traces*.

A ferramenta *Playload* também foi utilizada por [61] com o objetivo de avaliar o desempenho dos algoritmos de escalonamento em um ambiente de execução não exclusivo. Os autores selecionaram alguns tipos de traces, disponibilizados em [42], para reproduzir um ambiente dinâmico com a execução de cargas externas à sua aplicação.

2.4 *Traffic Control* (TC)

As versões mais recentes do núcleo do sistema Linux oferecem um grande conjunto de funções de controle de tráfego de rede, conhecidas como TC (*Traffic Control*) [2], capazes de configurar o compartilhamento das filas de envio de pacotes das interfaces de rede. O controle é composto pelos seguintes componentes conceituais: disciplinas de escalonamento, classes e filtros de classificação e policiamento. Cada interface de rede tem uma fila associada, que por sua vez tem associada uma classe e sua política de escalonamento. O escalonamento pode definir novas classes, cada uma com uma política distinta de escalonamento, em uma estrutura de escalonamento hierárquico, onde as classes folhas na hierarquia são responsáveis pelo escalonamento de pacotes a elas pertencentes. Filtros de classificação são utilizados para distinção de um pacote e sua distribuição nas diversas classes, as quais podem ter associadas ações de policiamento e moldagem de tráfego.

O TC permite decidir a forma com que os pacotes devem ser enfileirados e quando eles devem ser descartados, por exemplo, em uma situação em que o tráfego excede um certo limite. É possível também definir a ordem de envio desses pacotes, aplicando-se prioridades aos fluxos e, por último, retardar o envio de alguns pacotes para limitar a taxa de dados do tráfego de saída. Executadas todas essas operações, cada pacote pode ser entregue ao *driver* da interface para a transmissão pela rede. Com este conjunto de ferramentas, é possível configurar vários tipos de políticas para o escalonamento dos pacotes, distribuídas em uma estrutura hierárquica.

Neste trabalho, serão utilizadas apenas as funções do TC responsáveis por embutir retardos nos envios de pacotes. Nos experimentos executados no *grid* computacional formado por dois clusters geograficamente distantes e com um canal de comunicação inter-*cluster* bastante rápido, o TC foi utilizado para emular canais de comunicação com latências de rede mais elevadas. O objetivo dessas intervenções na rede foi possibilitar uma análise do comportamento das estratégias de balanceamento de carga em ambientes com canais de comunicação mais lentos.

Capítulo 3

Estratégias de Balanceamento de Carga

O objetivo de um algoritmo de balanceamento de carga, dependendo do contexto no qual está inserido, é equilibrar a carga dos nós em um sistema distribuído ou controlar a carga de uma aplicação nos diversos processadores de um ambiente paralelo. Neste ultimo contexto, os algoritmos diferenciam-se pelo modelo de programação paralela adotado na aplicação: paralelismo de controle ou de dados. Existem, portanto, algoritmos projetados para balancear a carga de aplicações definidas com paralelismo de controle e outros projetados para controlar paralelismo de dados. Neste trabalho, todas as estratégias de balanceamento avaliadas visam a manutenção do equilíbrio de carga para aplicações com paralelismo de dados do tipo SPMD (Single Program, Multiple Data).

Neste capítulo serão apresentadas as estratégias de balanceamento de carga avaliadas neste trabalho. Conforme já foi mencionado, o desenvolvimento dos algoritmos de balanceamento de carga e a realização dos experimentos tiveram suporte da ferramenta SAMBA-Grid. Das estratégias avaliadas, três delas já estavam disponíveis na biblioteca de balanceamento de carga desta ferramenta (Estático Simples, Mestre-Escravo com Bloco Fixo e Mestre-Escravo Hierárquico), outras três foram implementadas de acordo com propostas apresentadas na literatura (GSS, *Factoring* e *Weighted Factoring* Simples) e as últimas quatro foram propostas e implementadas neste trabalho (Estático com NWS, *Weighted Factoring* com NWS Estático, *Weighted Factoring* com NWS

dinâmico e *Weighted Factoring with Dynamic Feedback*). Este capítulo está organizado da seguinte forma: na Seção 3.1, serão apresentadas as estratégias estáticas e na Seção 3.2 as estratégias dinâmicas.

3.1 Estratégias Estáticas

3.1.1 Estático Simples

A estratégia de balanceamento de carga estática simples é considerada a mais trivial, pois é caracterizada apenas por uma política de distribuição que divide igualmente o conjunto inicial de tarefas entre todos os processadores. Esta divisão é feita no início da execução e não há uma política de transferência que permita a troca de tarefas entre os processadores para reduzir os efeitos dinâmicos de desequilíbrio de carga.

A vantagem desta política é a baixa comunicação entre os processadores, uma vez que, em um sistema formado por N nós executando uma aplicação que não gere novas tarefas, serão necessários apenas N envios de tarefas. A desvantagem é facilmente observada quando executamos esta estratégia em um ambiente não exclusivo. A presença de cargas externas à aplicação ocasiona desbalanceamentos deixando alguns processadores ociosos enquanto outros estão sobrecarregados.

A implementação deste algoritmo utilizou como parâmetro de ativação apenas o *host* da máquina que contém o conjunto inicial de tarefas.

3.1.2 Estático com NWS

Visando diminuir o impacto da presença de cargas externas sobre um algoritmo de balanceamento de carga estático simples, foi implementado, neste trabalho, uma versão deste algoritmo, denominado Estático com NWS, que utiliza o *Network Weather Service* (NWS), apresentado no Capítulo 2.

A única diferença entre os dois algoritmos estáticos está na forma da divisão do conjunto inicial de tarefas entre os processadores. O estático com NWS divide o conjunto inicial de tarefas de acordo com um peso associado a cada processador. Este peso é definido pela velocidade de processamento de uma máquina e pela sua disponibilidade no momento da distribuição.

A velocidade de processamento é medida em MHz e obtida a partir do arquivo de sistema */proc/cpuinfo* do Linux e a disponibilidade de CPU através da medição *availableCpu* do NWS, que fornece a fração disponível de CPU para um novo processo a ser executado. A composição destes dois parâmetros define o peso de cada processador e, conseqüentemente, a divisão do conjunto inicial de tarefas. A fórmula $L_p = W_p \times N$ calcula o tamanho do lote L_p de tarefas destinado a um processador p com peso W_p associado. A partir do conjunto inicial de tarefas N, cada processador p receberá um lote L_p de tarefas de acordo com a sua capacidade de processamento. O cálculo do peso W_p de um processador p está definido na Fórmula 3.1, onde V_p é a velocidade de processamento e F_p é a fração CPU disponível (obtida através do NWS). Vale lembrar que a fração F_p é uma previsão calculada estatísticamente pelo NWS e por isso possui margem de erro.

$$W_p = \frac{V_p \times F_p}{\sum_{k=1}^{k=P} V_k \times F_k} \tag{3.1}$$

A principal vantagem desta estratégia é permitir uma melhor distribuição das tarefas quando existir algum processador carregado com carga externa. Da mesma forma, o tempo de comunicação entre os processadores permanece o mesmo do algoritmo estático simples.

Embora seja necessário acessar o arquivo de sistema para obter a velocidade de processamento e buscar nos registros NWS a fração de CPU disponível, estes procedimentos não geram *overhead* considerável no tempo total de execução de uma aplicação, uma vez que os cálculos dos lotes L_p só são feitos uma vez, no início da execução.

Esta estratégia de balanceamento pode não ser eficiente quando executada em ambientes onde as cargas externas têm alta variação durante a execução da aplicação. Quando isso ocorre, a previsão inicial da fração de CPU disponível não refletirá a capacidade real dos processadores, ocasionando desbalanceamentos.

A implementação deste algoritmo utilizou como parâmetro de ativação apenas o *host* da máquina que contém o conjunto inicial de tarefas. Além disso, os sensores NWS devem estar monitorando o recurso CPU em todas as máquinas que executarão tarefas.

3.2 Estratégias Dinâmicas

Neste trabalho, todas as estratégias de balanceamento de carga dinâmicas implementadas são do tipo mestre-escravo. Na literatura podem ser encontrados diversos trabalhos que utilizaram esta abordagem de forma bastante eficiente no escalonamento de tarefas [9, 26, 29, 31, 48, 49]. Em uma estratégia do tipo mestre-escravo, o processador mestre, que possui o conjunto inicial de tarefas, distribui para os seus processadores escravos blocos de tarefas para que estes os executem. Toda vez em que um escravo termina de executar um bloco recebido, solicita um novo ao mestre. Este processo se repete até que não existam mais tarefas pendentes de execução no repositório de tarefas do mestre.

Conforme mencionado no Capítulo 2, na ferramenta SAMBA-Grid, uma estratégia de balanceamento de carga é composta por uma política de distribuição e uma política de transferência de tarefas. A maioria das implementações de estratégias do tipo mestre-escravo são implementadas apenas por uma política de distribuição, uma vez que o balanceamento de carga tende a ser garantido pela abordagem de distribuição sob demanda, a qual visa garantir que escravos não fiquem ociosos enquanto outros estão trabalhando.

De acordo com o tamanho do bloco enviado por um mestre aos seus escravos pode-se ter muitas variações de uma estratégia do tipo mestre-escravo. Observando essas variações foi implementada neste trabalho uma política de distribuição mais flexível, onde um parâmetro de entrada define a forma de distribuição dos blocos de tarefas durante a execução de uma aplicação SPMD. Para isto, a política de distribuição foi implementada por apenas um código fonte não havendo a necessidade de gerar um código executável (código de aplicação + código da política de distribuição + código gerenciador de repositório) para cada uma das variações de mestre-escravo. Vale ressaltar que esta unificação não gerou *overhead* significante sobre nenhuma das variações de mestre-escravo contempladas nesta política de distribuição. Nas Subseções 3.2.1 e 3.2.2, serão apresentadas todas estas variações de mestre-escravo.

3.2.1 Mestre-Escravo com Bloco Fixo (MEBF)

Implementada em [9, 29, 31, 47], a estratégia Mestre-Escravo com Bloco Fixo (MEBF) é a forma mais simples de uma estratégia do tipo mestre-escravo. Neste algoritmo, os blocos de tafefas que o mestre envia para um escravo é sempre o mesmo durante a execução da aplicação, salvo no final da execução quando o mestre envia as últimas tarefas disponíveis.

Vale lembrar, que no MEBF o processador que hospeda o processo mestre também pode hospedar um processo escravo com o objetivo de evitar a perda de recursos computacionais.

O principal problema desta estratégia é a dificuldade na definição do um tamanho ideal do bloco de tarefas destinado a cada escravo. Quanto menor for o tamanho do bloco mais envios de tarefas serão necessários, o que pode implicar em um aumento do tempo de comunicação entre o mestre e seus escravos. No entanto, quanto maior for o tamanho do bloco maior a probabilidade de ocorrerem desequilíbrios de carga no final da execução. Este desbalanceamento pode acontecer após o envio do ultimo bloco de tarefas, pois o escravo que recebeu este bloco poderá ficar sobrecarregado enquanto os demais estarão ociosos.

3.2.2 Mestre-Escravo com Redução de Bloco (MERB)

Conforme mecionado, a definição do tamanho do bloco não é uma tarefa simples. A partir deste pressuposto foram implementadas as estratégias que foram denominadas neste trabalho como sendo do tipo Mestre-Escravo com Redução de Bloco (MERB). A idéia principal destas estratégias é distribuir blocos de tamanho maior no início da execução e, à medida que o mestre recebe novas solicitações de tarefas, o tamanho dos blocos vai reduzindo gradativamente até o final da execução da aplicação. O objetivo principal é diminuir o *overhead* de comunicação, sem gerar desbalanceamentos e tirar do usuário a responsabilidadede por definir o tamanho do bloco. Em [33, 34, 51] podem ser encontradas implementações e análises de estratégias de balanceamento de carga deste tipo. Nas Subseções 3.2.2.1 a 3.2.2.6, serão apresentadas as estratégias do tipo mestre-escravo com esta característica.

3.2.2.1 MERB1 - *Guided Self-Scheduling*

Proposta em [51], o *Guided Self-Scheduling* (GSS) consiste em uma estratégia do tipo mestre-escravo com redução de bloco, originalmente proposto para sistemas de memória compartilhada, na qual o tamanho do i-ésimo bloco B_i de tarefas destinado a um escravo é definido pela Fórmula 3.2, onde i indica o número da solicitação, P o número de processadores escravos e N o conjunto inicial de tarefas. Observe que, a cada novo envio, um bloco contém $1/P$ das tarefas remanescentes no mestre.

$$B_i = \left\lceil \left(1 - \frac{1}{P}\right)^i \times \frac{N}{P} \right\rceil \tag{3.2}$$

A Tabela 3.1 mostra o tamanho dos 14 primeiros blocos destinados aos escravos a cada solicitação de tarefas ao mestre. Foram considerados neste exemplo 100 tarefas ($N = 100$) e 4 processadores escravos ($P = 4$).

bloco	0	1	2	3	4	5	6	7	8	9	10	11	12	13
B_p	25	19	14	11	8	6	5	3	3	2	1	1	1	1

Tabela 3.1: Tamanhos dos blocos de tarefas na estratégia GSS.

3.2.2.2 MERB2 - *Factoring*

Apresentada em [33], também inicialmente para sistemas de memória compartilhada, a estratégia mestre-escravo *Factoring* distribui as tarefas a partir de lotes formados por P blocos de mesmo tamanho, onde P é o número de processadores escravos. O tamanho de um bloco B_p dentro do i-ésimo lote é definido pela Fórmula 3.3, onde i indica o número de identificação do lote vigente, p o processador escravo que irá receber o bloco e N o conjunto inicial de tarefas. Assim, a cada P requisições por tarefas o identificador i do lote será incrementado e os blocos deste lote reduzirão de tamanho em relação aos blocos do lote anterior. Todos os blocos de um mesmo lote terão sempre o mesmo tamanho. Na prática, um novo lote deve conter a metade das tarefas remanescentes no mestre.

$$B_p = \left\lceil \left(\frac{1}{2}\right)^{i+1} \times \frac{N}{P} \right\rceil \tag{3.3}$$

A Tabela 3.2 mostra o tamanho dos blocos (B_p) destinados aos escravos (p) a cada solicitação de tarefas recebida pelo mestre, a partir do lote vigente i. Foram considerados neste exemplo 100 tarefas ($N = 100$) e 4 processadores escravos ($P = 4$).

i	0				1				2				3				4			
bloco	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
B_p	13	13	13	13	6	6	6	6	3	3	3	3	2	2	2	2	1	1	1	1

Tabela 3.2: Tamanhos dos blocos de tarefas na estratégia *Factoring*.

3.2.2.3 MERB3 - *Weighted Factoring* Simples

Proposta em [34] a estratégia *Weighted Factoring Simples* é uma variação da estratégia *Factoring* para um sistema de multicomputadores. Nesta estratégia, o tamanho de um bloco dentro de um lote é determinado na proporção de um peso W_p associado a cada escravo p e não como um valor constante, conforme a estratégia *Factoring*. O tamanho do bloco é determinado pela Fórmula 3.4.

$$B_p = \left\lceil \left(\frac{1}{2}\right)^{i+1} \times N \times \frac{W_p}{\sum_{k=1}^{k=P} W_k} \right\rceil \tag{3.4}$$

Na Tabela 3.3, é possível verificar o tamanho dos blocos distribuidos para 4 processadores escravos ($P = 4$) durante a execução de uma aplicação com 100 tarefas ($N = 100$). Cada escravo p possui o seguinte valor de *clock*: escravo1 = 400 Mhz, escravo2 = 800 Mhz, escravo3 = 1200 Mhz e escravo4 = 1600 Mhz.

i	0				1				2				3				4			
bloco	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
p	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
B_p	5	10	15	20	3	5	8	10	2	3	4	5	1	2	2	3	1	1	-	-

Tabela 3.3: Tamanhos dos blocos de tarefas na estratégia *Weighted Factoring Simples*.

Assim como a estratégia *Factoring*, na fórmula que calcula o tamanho B_p de um bloco, o índice i determina o lote vigente de tarefas. Nota-se que para $i = 0$ tem-se um lote de tamanho $N/2$, para $i = 1$, um lote de tamanho $N/4$ e assim por diante. Os autores deste algoritmo incrementam o valor de i a cada P requisições de tarefas ao mestre. No entanto, esta medida pode não ser eficiente quando um mestre têm muitos

escravos (valor alto de P). Neste caso, o mestre ficará muito tempo distribuindo tarefas dentro de um mesmo lote, o que pode gerar desbalancemantos de carga, pois o mestre distribuirá blocos de tamanho grande durante um longo período de tempo. Este problema poderá ser agravado em um ambiente não dedicado ou heterogêneo, uma vez que escravos menos capacitados poderiam estar recebendo blocos maiores do que seriam capazes de executar. Depois da realização de alguns experimentos foi confirmada esta deficiência e a implementação desta estratégia seguiu uma outra abordagem, conforme será apresentado.

O Algoritmo 1, apresentado de forma simplificada, implementa esta estratégia. Este algoritmo será executado todas as vezes que um escravo solicitar tarefas ao mestre. O peso (W_p) refere-se à velocidade de processamento de um escravo p obtida a partir do arquivo de sistema */proc/cpuinfo* que fornece, entre outras informações, o *clock* da CPU em MHz; P é o número de escravos; N é total de tarefas do conjunto inicial de tarefas e $PercTar_p$ é o percentual de tarefas destinado a um escravo p, o qual é calculado apenas no inicio da execução, a partir do peso W_p. No inicio da execução $IDlote = 1$ e $tamanhoDoLote = 0$.

```
1 início
2 |  se tamanhoDoLote <= 0 então
3 |  |  tamanhoDoLote ⟵ ⌈(1/2)^IDlote × N⌉;
4 |  |  tamanhoInicialDoLote ⟵ tamanhoDoLote;
5 |  |  IDlote ⟵ IDlote + 1;
6 |  bloco ⟵ tamanhoInicialDoLote × PercTar_p;
7 |  tamanhoDoLote ⟵ tamanhoDoLote − bloco;
8 |  retorna bloco;
9 fim
```

Algoritmo 1: Algoritmo *Weighted Factoring* Simples

Observa-se que, neste algoritmo, a vigência de um lote não está mais associada ao número de requisições feitas ao mestre. O tamanho de um lote é calculado através da fórmula descrita na terceira linha do algoritmo e todas as vezes que um escravo solicita tarefas ao mestre é debitado do lote vigente a quantidade de tarefas enviadas (sétima linha do algoritmo). Quando o lote vigente estiver vazio (verificação feita na segunda linha do algoritmo) será recalculado o tamanho do próximo lote que, conforme observado, reduz exponencialmente.

Esta estratégia é interessante pois o peso W_p possibilita uma melhor definição dos blocos, considerando a velocidade de processamento de cada processador. No entanto, este peso pode não refletir a capacidade real de processamento dos processadores quando estes não estiverem dedicados a execução da aplicação SPMD executada. Ainda que um processador tenha um valor alto de *clock*, a presença de carga externa diminuiria a capacidade de processamento e isto não seria refletido na definição do peso, podendo gerar desbalanceamentos.

3.2.2.4 MERB4 - *Weighted Factoring* com NWS Estático

A estratégia *Weighted Factoring* com NWS Estático, proposta e implementada neste tabalho, é uma versão do MERB3 que utiliza os valores de disponibilidade de CPU, fornecidos pelo NWS, como parâmetros para definir o peso W_p de cada processador escravo. Este peso é calculado pela fórmula $W_p = clockCpu_p \times availableCpu_p$, onde $clockCpu_p$ é o valor do *clock* do escravo p obtido a partir do arquivo de sistema do Unix /proc/cpuinfo e $availableCpu_p$ é a medição NWS que fornece a fração de *clocks* disponíveis para um novo processo a ser executado no escravo p.

A única diferença deste algoritmo de balanceamento de carga para o MERB3 é forma de definição do peso de cada processador. Conforme já foi mecionado, apenas o valor do *clock* pode não ser suficiente para determinar a capacidade de processamento de um processador quando executamos uma aplicação em um ambiente não exclusivo. Nesta estratégia o desbalanceamento pode ser minimizado com a utilização do parâmetro *availableCpu*, pois, desta forma, cada processador tende a receber o bloco de tarefas que é capaz de executar.

Embora a utilização da medição NWS *availableCpu* minimize o desbalanceamento de carga, o fato do cálculo do peso W_p ser feito apenas no início da execução da aplicação pode não ser o suficiente para retratar a situação de um processador no decorrer do tempo. Freqüentemente, em um ambiente não exclusivo, processos são iniciados e finalizados. Desta forma, a medição *availableCpu* obtida no início da execução da aplicação logo estará defasada. Por este motivo, esta estratégia tende a ser eficiente apenas em ambientes onde não ocorra muita variação das cargas externas, ou seja, na quantidade de processos concorrentes.

3.2.2.5 MERB5 - *Weighted Factoring* com NWS Dinâmico

A estratégia *Weighted Factoring* com NWS Dinâmico é uma versão do MERB4 que atualiza os pesos dos processadores escravos durante a execução da aplicação. O objetivo desta atualização é possibilitar que os pesos possam ser ajustados durante a execução, fazendo com que a entrada e a saída de processos concorrentes à aplicação sejam percebidas e consideradas na definição do novo peso do escravo.

O cálculo do peso W_p é o mesmo da estratégia MERB4 e é feito no início da execução da aplicação e todas as vezes que um novo lote vigente é recalculado. Porém, apenas o cálculo de peso feito no início da execução utiliza a medição NWS *availableCpu*. As atualizações de peso utilizam a medição *currentCpu*, que fornece a disponibilidade de CPU para um processo que já esteja em execução. O número de recálculos de peso são efetuados de acordo com um parâmetro de entrada R, o qual determina que um recálculo só será efetuado quando as tarefas disponíveis no mestre forem superiores a $R \times P$. A utilização deste parâmetro visa permitir a diminuição do *overhead* provocado por recálculos excessivos de pesos.

A principal desvantagem desta estratégia pode ser verificada quando executamos um número muito grande de tarefas e em muitos processadores. O *overhead* de recuperação das medições NWS poderá ser significativo, pois serão feitas muitas atualizações de peso durante a execução da aplicação, em especial no fim da execução quando os tamanhos dos lotes são menores e esses se esvaziam rapidamente.

3.2.2.6 MERB6 - *Weighted Factoring with Dynamic Feedback*

A estratégia *Weighted Factoring with Dynamic Feedback* também foi proposta e implementada neste trabalho e segue uma abordagem diferente para definir a capacidade de processamento de um escravo. Diferente das abordagens anteriores que determinam o peso W_p de um processador com base no *clock* da CPU, associada ou não à fração de CPU disponível (fornecida pelo NWS), neste algoritmo o valor do W_p é determinado pelo *feedback* de execução do escravo p. Este *feedback* consiste no tempo médio de computação de uma tarefa (TMCT) retornado por um escravo durante a execução da aplicação.

Para se tomar decisões de escalonamento nesta estratégia deve-se obter o TMCT

de cada processador. Como geralmente no início da execução de uma aplicação não se conhece o TMCT, o algoritmo que implementa esta estratégia adota inicialmente uma política de distribuição independente da capacidade de processamento dos escravos. Todas as vezes que um escravo solicita novas tarefas ao mestre, envia junto o TMCT referente ao bloco executado imediatamente antes da solicitação. O mestre, então, registra este valor e verifica se todos os demais escravos também já deram seu *feedback*. Em caso afirmativo, o algoritmo adotará, daí em diante, uma abordagem de redução de blocagem de forma semelhante a usada pela estratégia MERB5, onde o peso da cada escravo será o TMCT (em ms) mais atual. Em caso negativo, seguirá a distribuição sem considerar a capacidade de processamento dos escravos.

Conforme citado, a distribuição inicial não leva em consideração a capacidade de processamento de um escravo por não conhecer ainda o seu TMCT. Optou-se por fazer uma distribuição inicial bem simples, na qual o tamanho do lote é definido através da fórmula $tamanhoDoLote = IDLote \times P$, onde $tamanhoDoLote$ é tamanho do lote vigente, $IDLote$ identifica o lote vigente (inicialmente igual a 1) e P é o número de processadores escravos. Se um lote ficar vazio e nem todos os escravos tiverem dado pelo menos um *feedback* (envio do TMCT) ao mestre, o identificador do lote será incrementado em uma unidade e o próximo lote vigente será recalculado. Assim, enquanto os lotes tiverem sendo calculados nesta distribuição, seguirão a seguinte seqüência de tamanho: P, $2P$, $3P$, etc. Já o tamanho do bloco destinado a um escravo a cada solicitação será $tamanhoInicialDoLote/P$, onde $tamanhoInicialDoLote$ é o tamanho inicial do lote, isto é, o valor no momento em que este foi calculado.

De acordo com o que foi exposto, conclui-se que o TMCT será utilizado como peso W_p assim que todos os escravos o tiverem retornado ao mestre pelo menos uma vez. No entanto, de maneira diferente da estratégia MERB5, que considera como peso a quantidade de *clocks* de CPU disponíveis e envia mais tarefas para o escravo com maior peso, nesta estratégia será enviado mais tarefas para o processador com menor peso, isto é, menor TMCT. Assim, o valor do percentual de tarefas destinadas a um escravo deverá ser calculado a partir do Sistema de Equações 3.5 que estabelece uma relação entre o valores de $TMCT_i$ e p_i, grandezas inversamente proporcionais. O valor $TMCT_i$ é o TMCT de um escravo i e p_i é o percentual de tarefas que devem ser destinadas a este.

$$\begin{cases} \sum_{n=1}^{n=P} p_n = 1 & (1) \\ TMCT_1 p_1 = TMCT_2 p_2 & (2) \\ TMCT_1 p_1 = TMCT_3 p_3 & (3) \\ \ldots & \\ TMCT_1 p_1 = TMCT_n p_n & (n) \end{cases} \tag{3.5}$$

Isolando-se os valores de p_i, para $i \neq 1$, obtém-se.

$$p_2 = \frac{TMCT_1}{TMCT_2} \times p_1, \quad p_3 = \frac{TMCT_1}{TMCT_3} \times p_1, \quad \ldots, \quad p_n = \frac{TMCT_1}{TMCT_n} \times p_1 \tag{3.6}$$

Substituindo-se p_1, p_2, p_3, ..., p_n na Equação 3.5 (1), obtém-se:

$$p_1 + \frac{TMCT_1}{TMCT_2} \times p_1 + \frac{TMCT_1}{TMCT_3} \times p_1 + \ldots + \frac{TMCT_1}{TMCT_n} \times p_1 = \sum_{n=1}^{n=P} \frac{TMCT_1}{TMCT_n} \times p_1 \tag{3.7}$$

Logo:

$$p_1 \times \left(1 + \frac{TMCT_1}{TMCT_2} + \frac{TMCT_1}{TMCT_3} + \ldots + \frac{TMCT_1}{TMCT_n}\right) = 1 \tag{3.8}$$

E conseqüentemente:

$$p_1 = \frac{1}{\left(1 + \frac{TMCT_1}{TMCT_2} + + \frac{TMCT_1}{TMCT_3} + \ldots + \frac{TMCT_1}{TMCT_n}\right)} \tag{3.9}$$

Da mesma forma que na estratégia MERB5, todas as vezes que um lote de tarefas terminar, deverá ser realculado o percentual de tarefas destinado a cada processador escravo. Este cálculo será feito através das Fómulas 3.9 e 3.6, e também será utilizado o parâmetro R para limitar o número de recálculos. Com o valor de p_1 obtém-se facilmente os demais percentuais (p_2, p_3, ... p_n) por substituição nas Equações 3.6. O Algoritmo 2, exibido de forma simplificada, implementa esta estratégia.

```
1  início
2      Registrar feedback(TMTC) do escravo que está solicitando mais tarefas;
3      se Todos os escravos já deram feedback então
4          feedback ⟵ true;
5          IDlote ⟵ 1;
6          N ⟵ Total de tarefas não executadas;
7      se not feedback então
8          se tamanhoDoLote <= 0 então
9              tamanhoDoLote ⟵ IDlote × P;
10             tamanhoInicialDoLote ⟵ tamanhoDoLote;
11             IDlote ⟵ IDlote + 1;
12         bloco ⟵ tamanhoInicialDoLote/P;
13     senão
14         se tamanhoDoLote <= 0 então
15             CalcularPercTar;
16             tamanhoDoLote ⟵ ⌈(1/2)^IDlote × N⌉;
17             tamanhoInicialDoLote ⟵tamanhoDoLote;
18             IDlote ⟵ IDlote + 1;
19         bloco ⟵ tamanhoInicialDoLote × PercTar_i;
20     tamanhoDoLote ⟵ tamanhoDoLote − bloco;
21     retorna bloco;
22 fim
```

Algoritmo 2: Algoritmo *Weighted Factoring with Dynamic Feedback*

Este algoritmo será executado pelo mestre todas as vezes que um escravo lhe solicitar novas tarefas. Na segunda linha do algoritmo, o mestre registra o TMCT do escravo que solicitou mais tarefas. Na terceira linha, é verificado se todos os demais escravos já retornaram o seu TMCT ao menos uma vez. Em caso afirmativo, a variável lógica *feedback* receberá o valor *true* (linha 4), a identificação do lote (*IDlote* na linha 5) será iniciada com o valor 1 e o conjunto inicial de tarefas N receberá a quantidade de tarefas ainda não executadas. Na linha sete é verificado o valor da variável lógica *feedback* com o objetivo de escolher a forma de distribuição de tarefas. Enquanto esta variável contiver o valor *false* o tamanho do bloco a ser enviado para o escravo será determinado de acordo com a fórmula da linha 12. Na linha oito é verificado se o lote vigente de tarefas está vazio. Se estiver, o novo lote de tarefas será calculado de acordo com a fórmula descrita na linha nove. Se *feedback* igual a *true* o bloco de tafefas será calculado de acordo com a fórmula da linha 18, que utiliza a abordagem *Weighted Factoring*. Da mesma forma, quando o lote vigente de tarefas estiver vazio (verificação

feita na linha 14), será recalculado na linha 15 o percentual de tarefas associado a cada escravo, com base nos TMCT atualizados (De acordo com as fórmulas 3.9 e 3.6), e o novo lote através da fórmula descrita na linha 16. Por fim, será retornado na linha 21 o tamanho do bloco destinado ao escravo. É importante lembrar que, no início da execução, $IDlote = 1$ e $tamanhoDoLote = 0$ e que P é o número total de processadores escravos.

A implementação deste algoritmo seguiu duas abordagens distintas: a primeira considerando o TMCT como sendo apenas o tempo médio de execução de uma tarefa e a segunda levando em conta, além do fator execução, a latência de rede para enviar uma tarefa do mestre para o escravo. Nesta segunda abordagem o TMCT é calculado da seguinte forma: $TMCT_p = exec_p + lat_p$, onde $exec_p$ é o tempo médio de execução de uma tarefa no escravo p e lat_p a latência para enviar a tarefa para este escravo. A latência é obtida a partir do medição *latencyTcp* do NWS.

O fato de esta estratégia considerar o tempo médio de execução de uma tarefa pode não retratar a atual a capacidade de processamento de um escravo, uma vez que este valor se refere a uma situação verificada no passado. Este problema poderá ser observado quando cargas externas forem iniciadas ou finalizadas imediatamente após o envio do TMCT ao mestre, pois o mesmo estará desatualizado. Na abordagem que utiliza a latência, apenas o $exec_p$ estará dasatualizado, pois a latência lat_p não se refere a uma medida defasada e sim a um previsão estatística da situação da rede (via *forecaster* NWS).

3.2.3 Mestre-Escravo Hierárquico (MEH)

A estratégia mestre-escravo hierárquico (MEH) proposta em [4] e discutida anteriormente em [1, 23] consiste em uma variação da estratégia mestre-escravo com bloco fixo que utiliza uma topologia hierárquica de processadores. Assim como as estratégias com redução de bloco apresentadas anteriomente, esta estratégia tem por objetivo diminuir o *overhead* de comunicação entre o mestre e os seus escravos. O MEH foi elaborado visando a sua utilização em *grids* computacionais onde a comunicação entre os processadores muitas vezes é relevante no desempenho de aplicações.

Nesta política, a raiz da hierarquia representa o mestre global, as folhas da hie-

rarquia representam os escravos e os nós internos representam submestres. O mestre global distribui as tarefas sob demanda entre os seus filhos. Cada submestre também distribui sob demanda suas tarefas aos seus filhos e, sempre que estas são concluídas, este solicita mais tarefas ao seu pai. Cada escravo, após executar as tarefas recebidas, solicita mais tarefas ao seu pai. Conclui-se, então, que o mestre global e os submestres podem ter como filhos submestres e escravos, o que é determinado pelo número de níveis de uma hierárquia. Na Figura 3.1, apresenta-se uma hieraquia com 3 níveis.

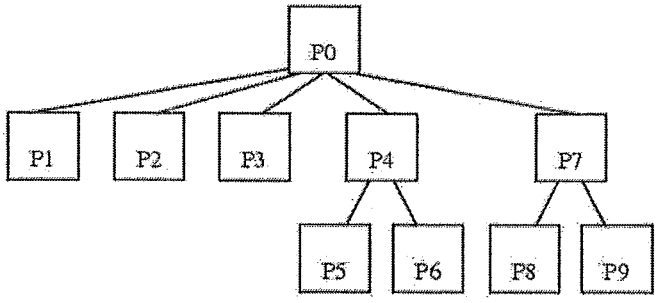

Figura 3.1: Exemplo de uma topologia para MEH em três níveis.

Em *grids* computacionais, escravos podem se tornar ociosos com freqüência em função do mestre estar fisicamente distante e conectado por meio de reduzida capacidade de transmissão ou sujeito a congestionamentos. Utilizando-se a estratégia MEH, o mestre global, submestres e escravos podem ser alocados de tal forma que se utilize menos os canais de comunicação que apresentam pior desempenho. Considere três *clusters* A, B e C, formados respectivamente pelos processadores (P0,P1,P2,P3), (P4,P5,P6) e (P7,P8,P9). Tem-se ainda que as tarefas encontram-se inicialmente localizadas no *cluster* A e que os demais canais de comunicação entre os *clusters* apresentam alta taxa de latência e baixa taxa de transmissão comparados com os canais internos de cada *cluster*. A Figura 3.1 representa uma possível solução hieráquica para reduzir a utilização dos canais entre os *clusters*, tendo em vista que P4 e P7 farão solicitações em bloco a P0, distribuindo as tarefas recebidas entre os processadores P5, P6, P8 e P9, evitando a comunicação destes com o *cluster* A.

Na implementação desta estratégia a hierarquia de processadores é definida em um arquivo de topologia, fornecido como parâmetro de entrada. A quantidade de níveis da hieraquia e de escravos por submestres pode ser alterada em função das características da aplicação e da topologia do *grid*. Vale lembrar, que os processadores que hospedam

o mestre global ou os submestres também podem hospedar processos escravos com o objetivo de evitar a perda de recursos computacionais.

Esta estratégia já se encontrava disponível na biblioteca de estratégias de balanceamento de carga do *framework* SAMBA-Grid. Neste trabalho, ela foi incluída no código fonte da política de distibuição que agrega todas as estratégias do tipo mestre-escravo. Para executar esta estratégia basta fornecer um parâmetro que a identifica e um arquivo de topologia.

Capítulo 4

Aplicações SPMD

Segundo [47], as aplicações SPMD podem ser divididas em três classes segundo a forma com que as tarefas que as constituem se comunicam durante sua execução. Essas classes são: aplicações sem dependência, aplicações com comunicação síncrona e as aplicações com comunicação assíncrona.

Em aplicações SPMD sem dependência, os processadores não se comunicam no decorrer da execução das tarefas que constituem a aplicação, pois a execução de uma tarefa não depende dos dados resultantes da execução das demais. Esse tipo de aplicação permite que as tarefas possam ser executadas em qualquer processador e em qualquer ordem. Além da facilidade de programação, vários problemas podem ser resolvidos utilizando-se essa abordagem. As características dessa classe facilitam o trabalho do algoritmo de balanceamento, já que a transferência de carga pode ser realizada sem maiores problemas devido à grande independência existente entre as tarefas da aplicação.

A classe de aplicações SPMD com comunicação síncrona é constituída por aplicações cujas tarefas se comunicam em pontos específicos e bem definidos durante sua execução, chamados pontos de sincronismo. A necessidade constante de trocas de informações, como por exemplo, para atualizar dados comuns, é uma característica marcante dessa classe. A execução das tarefas entre dois pontos de sincronismo corresponde, tipicamente, à execução de uma iteração da aplicação. Por esse motivo, aplicações deste tipo também são conhecidas como aplicações paralelas iterativas. Dessa forma, uma iteração é dependente dos dados da iteração imediatamente anterior a ela e o tempo total

de execução da aplicação SPMD será definido pela soma dos tempos de cada iteração adicionado dos tempos de comunicação. Um fator que caracteriza o desbalanceamento nessa classe é a diferença entre os tempos de execução das iterações realizadas em cada processador, pois quanto maior essa diferença, maior o desbalanceamento de carga no sistema. O balanceamento de carga nessa classe é realizado normalmente nos pontos de sincronismo, aproveitando o momento utilizado para a sincronização entre as tarefas.

A última classe de aplicações SPMD é a com comunicação assíncrona. A comunicação entre as tarefas que constituem as aplicações dessa categoria se caracteriza pela falta de uniformidade, sendo muito dependente do tipo da aplicação. Dessa forma, o padrão de comunicação, muitas vezes irregular e imprevisível, pode dificultar a implementação deste tipo de aplicação. Essa disparidade no padrão de comunicação muitas vezes inviabiliza a criação de um algoritmo de balanceamento de carga genérico, sendo necessário definir algoritmos próprios para as características individuais de cada aplicação.

Neste trabalho, será avaliado o desempenho das estratégias de balanceamento de carga utilizando duas aplicações SPMD do tipo sem dependência. A primeira, denominada aplicação dos *termions* e apresentada na Seção 4.1, trata de um problema da área de física, e a segunda, apresentada na Seção 4.2, é da área de computação gráfica e implementa uma técnica de renderização de imagens chamada *Ray Tracing* distribuído.

4.1 Aplicação dos *Termions*

A aplicação dos *termions* tem como objetivo o cálculo da dispersão térmica em meios porosos. O estudo deste problema tem demonstrado ser de grande interesse em várias áreas como medicina, controle ambiental, recuperação de petróleo, entre outras. Detalhes sobre o tema podem ser encontrados em [54].

O meio poroso é caracterizado pela união de várias células fundamentais, conforme observado na Figura 4.1, chamado meio periódico. Cada célula fundamental é composta de elementos sólidos (partes escuras) e por fluidos (partes brancas).

A dispersão térmica é calculada pelo movimento aleatório de partículas hipotéticas, chamadas *termions*, que carregam uma quantidade fixa de energia. A natureza do movimento das moléculas é determinada pela agitação térmica que depende da temperatura

e das propriedades térmicas do meio.

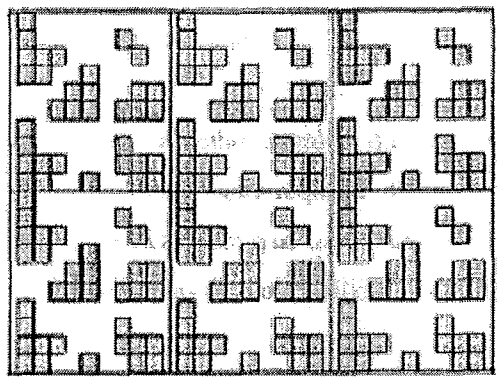

Figura 4.1: Exemplo de um meio periódico.

Cada *térmion* é representado pela sua posição (x,y) no espaço. Esta posição determina em que tipo de meio a partícula está. A direção e o sentido do movimento das partículas são determinados aleatoriamente e a velocidade de locomoção delas depende do meio em que estão, ou seja, se as partículas estiverem em um meio fluido elas terão velocidade maior do que se estivessem em um meio sólido.

Inicialmente, um grande número de *termions* é liberado e a posição de cada um é atualizada a cada iteração de acordo com a direção e o comprimento do passo. A direção da partícula é escolhida aleatoriamente, para cima, para baixo, para esquerda ou para a direita. O comprimento de cada passo depende das propriedades térmicas do meio em que a partícula está, da velocidade do fluxo do fluido, caso a partícula esteja na parte fluida, e das dimensões do meio periódico. Sólido e fluido tem propriedades térmicas diferentes.

Como ilustrado na Figura 4.2, quando um *térmion* alcança uma fronteira (região entre as partes fluidas e sólidas), verifica-se a probabilidade de esta partícula atravessar de um meio para outro. Esta probabilidade de transição (P) depende das propriedade térmicas do meio (sólido/fluido). Se um número aleatório (r) escolhido é menor que esta probabilidade, a partícula irá atravessar a fronteira, caso contrário ocorrerá um choque elástico. Como conseqüência destes cálculos, um maior esforço computacional é exigido para cruzar fronteiras do que quando uma partícula permanece no mesmo meio.

Cada *térmion* irá atravessar um caminho aleatório durante um determinado tempo.

A avaliação do caminho total de cada *término* pode requerer diferentes cargas computacionais. Isto se deve ao número de vezes que uma partícula tenta cruzar fronteiras e também das diferentes propriedades térmicas de partes sólidas distintas.

Depois de um número de passos, que depende das propriedades térmicas do meio, a partícula alcança sua posição final. A partir da posição final de cada *término* obtém-se a dispersão térmica.

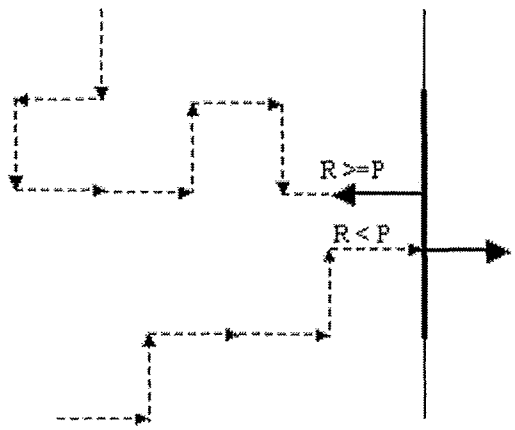

Figura 4.2: Transição entre meios distintos.

Na execução da aplicação, cada tarefa é constituída pela computação do caminho percorrido por um *termion*, ou seja, uma tarefa é a realização de todo o processo de cálculo da trajetória, da sua posição inicial à sua posição final. Observa-se que a aplicação dos *termions* possui a característia de ser SPMD e do tipo sem dependência, pois a computação de cada *término* é independente dos outros. Esta aplicação, no entanto, possui um alto custo computacional e, por este motivo, foi paralelizada utilizando a ferramenta SAMBA-Grid.

Para que se possa obter um comportamento mais confiável da dispersão térmica no meio, é necessário utilizar, na execução da aplicação, um número suficiente de *termions* e fazê-los caminhar por um determinado tempo. Além disso, cada tarefa pode demandar quantidades de processamento diferentes se as propriedades térmicas das diferentes partes sólidas forem distintas.

4.2 Aplicação *Ray Tracing*

Os estudos na área de computação gráfica têm demonstrado bastante interesse no que diz respeito ao foto-realismo, ou seja, geração de imagens com alto grau de realismo. A técnica denominada *Ray Tracing*, inicialmente apresentada em [56], oferece uma ferramenta simples e poderosa para esse estudo e tornou-se rapidamente bastante popular devido à qualidade dos seus resultados e à simplicidade de seus métodos. O *Ray Tracing* consiste em um poderoso método de renderização de imagens e se baseia no traçado de raios, simulando o processo de propagação da luz no sentido inverso a como ela ocorre.

Com a técnica *Ray Tracing*, os objetos são construídos e suas cores calculadas, através das interseções destes objetos com os raios de luz simulados, que partem do ponto de vista do usuário e passam por cada *pixel* da tela. Com isso, devem-se considerar as fontes luminosas presentes na cena, os materiais que compõem as superfícies dos objetos (com todas as suas propriedades), as distâncias dos objetos à tela de projeção e a posição da tela em relação ao ponto de vista do observador [3]. Todos estes procedimentos de cálculos foram obtidos a partir de observações da natureza e, obviamente, simplificados a ponto de se tornarem factíveis com os equipamentos atuais. Apesar das simplificações, o método apresenta um resultado bastante satisfatório e, muitas vezes, surpreendente.

A Figura 4.3 ilustra o modelo descrito anteriormente. De uma forma bem resumida, cada *pixel* fornece a direção de um raio, que parte do ponto de vista do observador e se dirige para a cena composta de objetos modelados pelo usuário. Verificam-se quais objetos fazem interseção com este raio e obtêm-se as distâncias destes pontos de interseção até o ponto de vista do observador. A menor distância indica o objeto mais próximo e, conseqüentemente, o que será visível naquele *pixel*. Os demais pontos de interseção ficam ocultos pelo primeiro.

Tendo-se o ponto de interseção e as informações sobre o objeto (forma, material da superfície, etc.), calcula-se a sua intensidade luminosa. Esta intensidade é obtida a partir da influência das fontes luminosas da cena neste ponto. O cálculo da influência é feito através do vetor normal à face do objeto e dos raios que são traçados até as fontes de luz, utilizando-se as características do material (transparência, reflexividade,

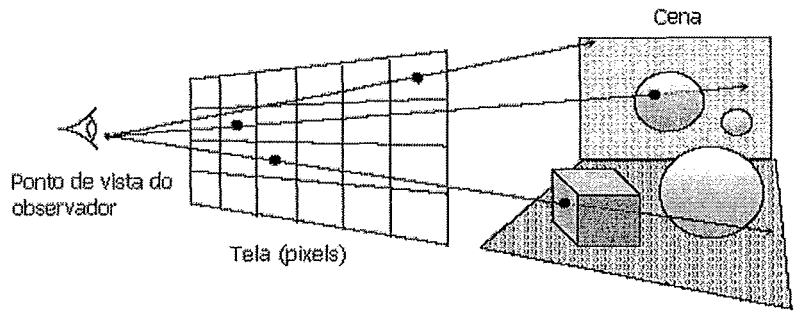

Figura 4.3: Idéia da técnica de *Ray Tracing*.

contribuição para a iluminação difusa, etc.) [3].

O Algoritmo 3 implementa a técnica *ray tracing* tradicional. Este algoritmo, além de permitir uma melhor visualização da imagem, pode resolver de forma efetiva e relativamente simples vários fenômenos luminosos como sombras, reflexão e refração.

```
1 início
2   para cada pixel p da tela faça
3     Traçar um raio que une o observador ao pixel p;
4     para cada objeto da cena faça
5       se o raio intercepta o objeto e o ponto de interseção encontra-se
        mais próximo do observador do que o ponto de interseção até
        agora encontrado então
6         Registrar o ponto de interseção e o objeto interceptado;
7     Atribuir ao pixel a cor do objeto interceptado no ponto de interseção
      registrado;
8 fim
```

Algoritmo 3: Algoritmo *Ray Tracing* Tradicional

Uma vez determinada a interseção de um raio com o objeto mais próximo, têm então lugar os cálculos de iluminação no ponto de interseção para atribuir ao *pixel* da imagem a cor desse ponto. Estes cálculos têm por objetivo determinar a intensidade luminosa proveniente do ponto de interseção que é propagada no sentido do centro de projeção, isto é, no sentido contrário ao raio. Esta luz é o resultado das contribuições de várias origens que atingem o ponto de interseção. Uma das componentes é devida à reflexão da luz proveniente das fontes de luz existentes na cena e que iluminam diretamente o objeto no ponto de interseção. Neste caso, é necessário determinar se cada uma das fontes de luz da cena ilumina realmente o ponto de interseção ou se existe um outro objeto que se interpõe entre os dois e faz com que o ponto se encontre na sombra

em relação a essa fonte. O ponto de interseção pode também ser iluminado pela luz refletida por outros objetos. Finalmente, se o objeto interceptado é transparente ou translúcido, existe uma outra componente para a sua iluminação devido à transmissão (com refração na superfície do objeto) da luz através do objeto. Assim, e em resumo, existem três componentes que contribuem para a iluminação:

- Luz proveniente de outros objetos e que foi refletida na direção do objeto cuja iluminação se pretende determinar.
- Luz refratada transmitida através de um objeto transparente e que ilumina o ponto.
- Luz proveniente diretamente das fontes de luz e que é refletida pelo objeto.

A determinação de cada uma destas componentes requer cálculos próprios, sendo necessário inspecionar as origens prováveis de cada uma delas. Para isto empregam-se os chamados raios secundários, traçados em sentido inverso ao da propagação da luz correspondente a cada uma das componentes. Uma vez que cada um destes raios secundários tem objetivos diferentes e o seu cálculo é também diferente, convém então separá-los em três tipos: raios refletidos, raios refratados e raios de iluminação direta ou de sombra (*shadow feelers*).

Percebe-se que os raios lançados no método de *Ray Tracing* seguem o caminho inverso ao da luz. Os raios são lançados na direção inversa, pois de uma fonte luminosa partem bilhões de fótons que são os formadores dos raios luminosos. Porém apenas uma pequeníssima parte deles chega aos olhos de um observador. Simular tal mecanismo seria impraticável para o computador, fazendo que uma simples cena levasse anos para ser renderizada. Lançando raios a partir do observador, garantimos que estaremos calculando apenas os raios que efetivamente serão captados pelo observador.

O algoritmo do *Ray Tracing* gasta a maior parte do seu tempo (entre 75% e 95%) determinando as interseções com os objetos [3]. Por este motivo, a eficiência da rotina de interseção entre raios e objetos afeta significativamente a eficiência do algoritmo, a qual se reflete no tempo de geração da cena. Por outro lado, este é um algoritmo que pode ser facilmente paralelizado, pois os cálculos são feitos de forma independente

para cada um dos pontos da imagem. De acordo com [45], esta paralelização pode ser feita de diversas formas, entre as quais pode-se destacar:

- **Paralelismo de componentes:** os cálculos de um simples raio podem ser paralelizados. Por exemplo, cálculos de reflexão, refração e interseção, envolvem o cálculo de componentes (x,y,z) de vetores ou pontos. Estes três componentes podem ser calculadas em paralelo alcançando aceleração de um fator de três.

- **Paralelismo da imagem:** as interseções entre raios e objetos podem ser calculadas em processadores separados, pois são independentes. Para tomar partido do paralelismo é necessário que todos os processadores tenham acesso a toda a base de dados dos objetos que compõem a cena, pois cada raio pode atingir qualquer uma das regiões da cena.

- **Paralelismo de objetos:** os objetos podem ser distribuídos espacialmente por múltiplos processadores. Cada processador fica responsável por todos os raios que passam por sua região. O processador calcula a interseção entre o raio e o objeto (se o raio atinge um objeto) e envia o raio para o próximo processador, se necessário.

Embora atualmente existam diversos recursos de hardware que efetuam os cálculos do *Ray Tracing* com extrema rapidez, o alto custo inviabiliza a sua utilização em larga escala. Em razão das vantagens oferecidas pelos computadores paralelos comerciais, baixo custo e ambientes maduros de programação, a pesquisa em *Ray Tracing* paralelo encaminhou-se para o desenvolvimento de algoritmos eficientes que funcionassem nestas máquinas.

É importante ressaltar que o algoritmo tradicional de *Ray Tracing* é baseado numa técnica de amostragem pontual onde um único raio é lançado por cada *pixel* da imagem. A técnica de amostragem pontual é, porém, inadequada para o tratamento de outros fenômenos que enriquecem ainda mais o realismo das imagens geradas, como profundidade de campo e, principalmente, na solução do efeito provocado pela discretização da imagem no espaço da tela, conhecido como *aliasing* [3].

O tamanho de um *pixel* define o limite superior para as freqüências que podem ser exibidas. Freqüências mais altas causam *aliasings*, como o aparecimento de linhas

serrilhadas na imagem, ao invés de linhas contínuas [3]. Uma solução simples para reduzir este problema é conhecida como "super-amostragem", que consiste em subdividir o *pixel* em vários *subpixels*, fazendo com que se tenha mais informações para se determinar a intensidade do *pixel* em questão. Quanto mais sub-regiões são consideradas, melhor o resultado, mas também mais cálculos serão realizados. Esta solução não elimina o problema do *aliasing*, mas pode levá-lo a dimensões insignificantes.

Baseando-se no princípio de que o *aliasing* é uma conseqüência da regularidade da grade de amostragem, em [19], foi proposto um aperfeiçoamento no algoritmo *Ray Tracing* tradicional, denominado *Ray Tracing* distribuído, que utiliza a técnica de amostragem estocástica onde os raios são distribuídos de forma aleatória. Nesta técnica, é adicionado um deslocamento aleatório (conhecido como *jitter*) às localizações de uma distribuição de amostragem e, para cada raio obtido, aplica-se o processo definido no algoritmo de *Ray Tracing* tradicional. Na Figura 4.4 é exibida uma "super-amostragem" para um *pixel* que foi subdivido em 16 partes, onde são mostrados dois exemplos de distribuição de raios. Na amostragem regular os raios incidem sempre no centro de cada subpixel. Já na amostragem estocática a incidência é feita de forma aleatória.

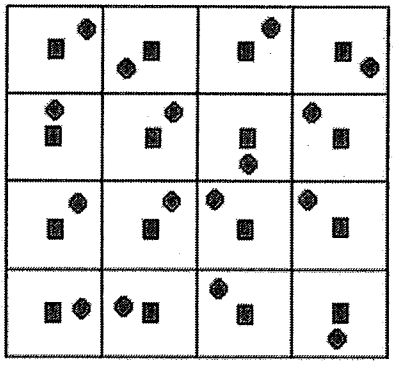

Figura 4.4: Amostragem regular (quadrados) e amostragem estocástica (círculos).

O algoritmo *Ray Tracing* distribuído permite calcular cores médias mais corretas, pois os raios encontram-se melhor distribuídos dentro da área correspondente a cada *pixel*, mas introduz ruído devido à distribuição aleatória. No entanto, como a visão humana é muito mais tolerante ao ruído aleatório do que a singularidades locais, esta técnica produz resultados de melhor qualidade visual, principalmente em zonas de sombra parcial (penumbra) onde não existem arestas vivas.

O poder da técnica de *Ray Tracing* distribuído pode ser percebido com a quantidade de efeitos que podem ser obtidos, como *anti-aliasing* e controle da profundidade de campo, descritos anteriormente, *motion-blur*, penumbra, reflexões borradas (*gloss*) e translucidez. Para se obter estes últimos, é necessário distribuir os raios do espaço da tela para a dimensão do tempo, distribuir os raios de sombra para as superfícies das fontes de luz, distribuir os raios refletidos ao redor do raio de reflexão e distribuir os raios refratados ao redor do raio de refração [3].

Conforme observado, o custo computacional será maior do que a técnica de *Ray Tracing* tradicional, uma vez que um *pixel* é subdividido em vários *subpixels* e, para cada um destes, os cálculos são feitos na forma tradicional. Conclui-se, então, que o tempo computacional terá um aumento exponencial, de acordo com a ordem da subdivisão. Uma subdivisão de *pixels* na ordem 2×2, por exemplo, produz quatro *subpixels*, enquanto que uma ordem 4×4 produz 16 *subpixels*, e assim sucessivamente. Para se conseguir imagens realistas, com a percepção dos efeitos citados e sem a presença de *aliasing*, não é necessário a utilização de ordens de divisão muito elevadas.

Uma imagem a ser renderizada pela técnica *Ray Tracing* pode ser constituída por diversos objetos que compõem a cena. Sendo assim, as tarefas desta aplicação serão heterogêneas, pois o tempo de computação de cada *pixel* dependerá da sua localização na imagem. Os *pixels* localizados em região de *background* basicamente não irão consumir muito tempo de computação, enquanto que, os pixel localizados em regiões de objetos demandarão um alto custo computacional.

Neste trabalho, foi utilizado o *Ray Tracing* do tipo distribuído, o qual foi paralelizado utilizando-se a ferramenta SAMBA-Grid, apresentada no Capítulo 2. A versão paralela foi implementada seguindo-se a abordagem de paralelismo de imagem, onde o processamento do cálculo da cor de cada *pixel* caracteriza uma tarefa desta aplicação SPMD. Devido ao fato desta aplicação ser constituída por tarefas heterogêneas, acredita-se que, a inclusão de forma aleatória de suas tarefas no repositório de tarefas do SAMBA-Grid poderia reduzir os efeitos de desbalanceamento. Porém, como o objetivo deste trabalho é avaliar o desempenho das estratégias de balanceamento de carga, optou-se por uma inclusão seqüencial, isto é, seguindo-se a disposição dos *pixels* na imagem.

Nas Figuras 4.5, 4.6, 4.7 e 4.8 são apresentados dois exemplos de imagens antes e

depois da renderização utilizando a técnica de *Ray Tracing* distribuído. Efeitos como sombras, reflexão e *anti-aliasing* são facilmente observados nestas imagens, após a renderização.

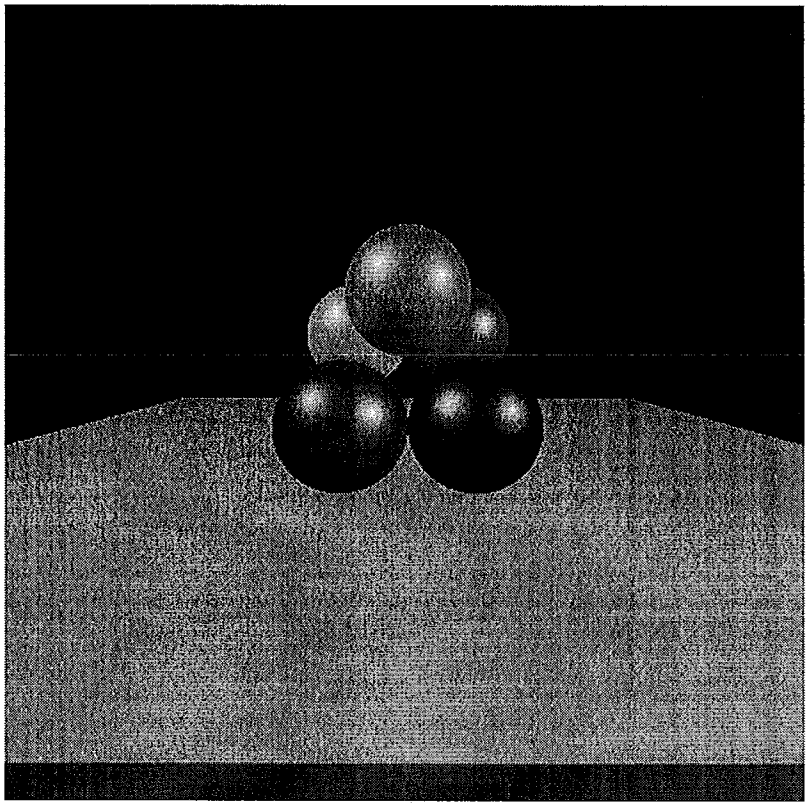

Figura 4.5: Imagem *5balls*.

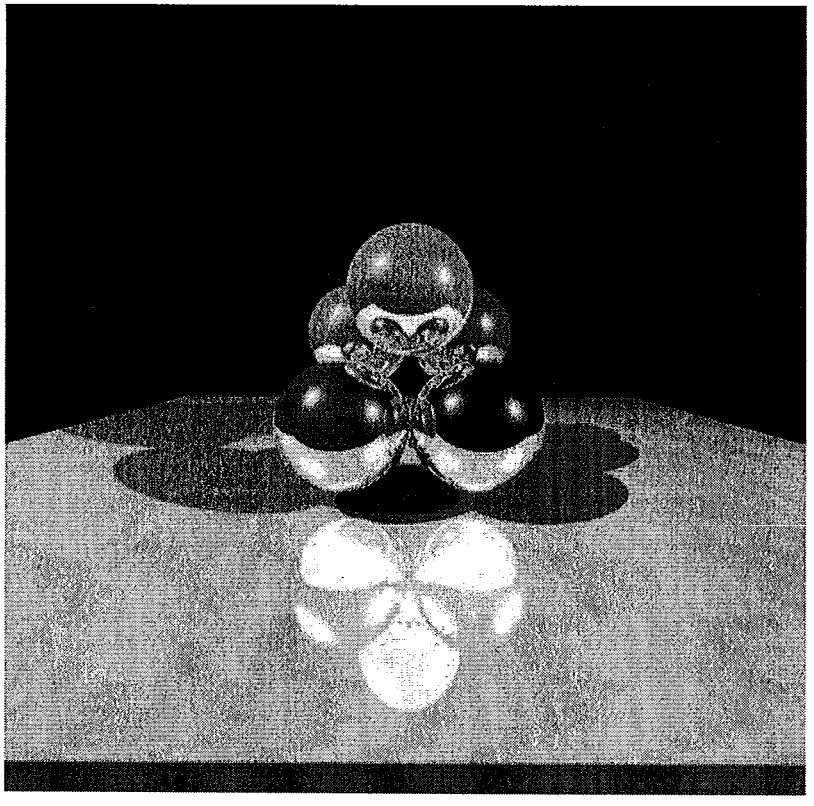

Figura 4.6: Imagem *5balls* depois da renderização com *Ray Tracing*.

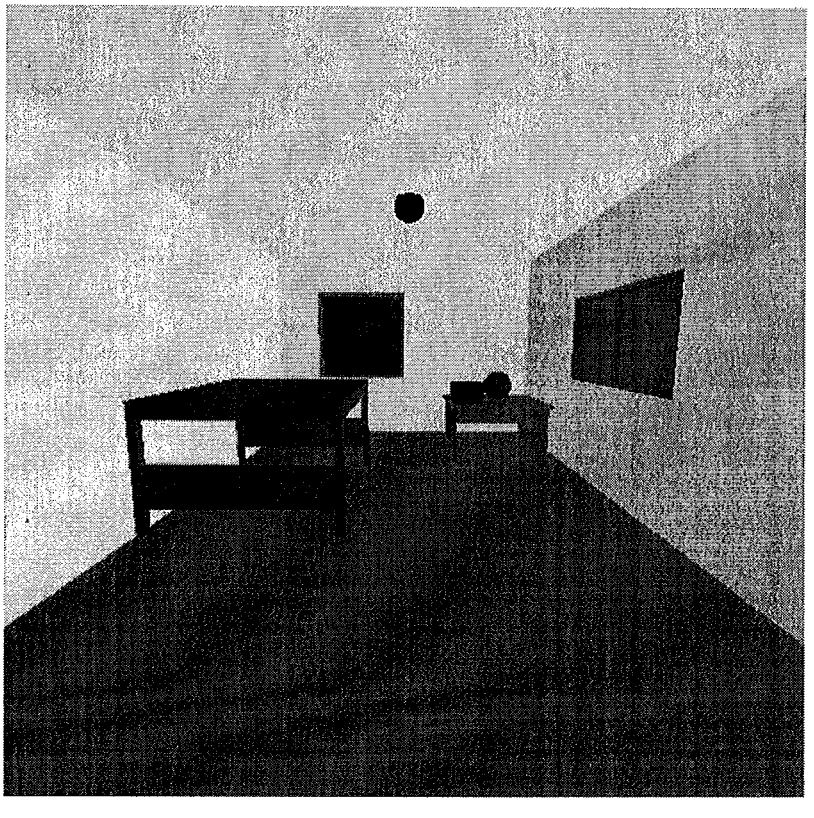

Figura 4.7: Imagem *room*.

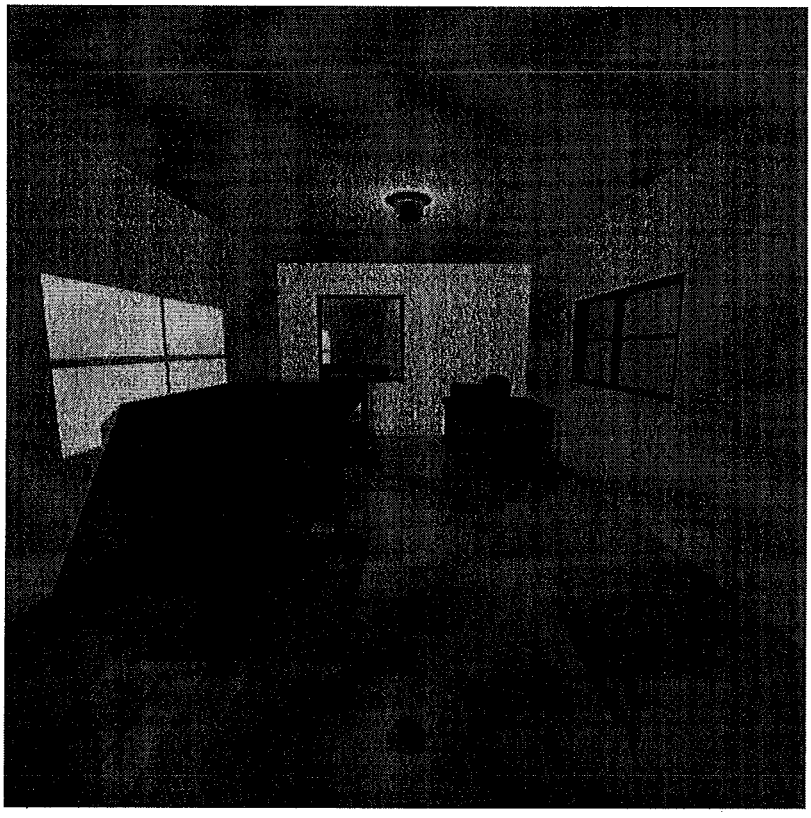

Figura 4.8: Imagem *room* depois da renderização com *Ray Tracing*.

Capítulo 5

Metodologia Experimental, Resultados e Análise

Neste capítulo, serão apresentados os resultados e as análises dos experimentos computacionais realizados com as estratégias de balanceamento de carga mostradas no Capítulo 3. Na Seção 5.1, é apresentada a fórmula que calcula o índice de desbalanceamento de carga de uma estratégia, utilizado também como parâmetro para avaliação do desempenho das estratégias implementadas. Na Seção 5.2, são mostrados os resultados e as análises de desempenho das estratégias quando executadas em um *cluster*. Na Seção 5.3, são apresentados os resultados e as análises dos experimentos executados em um *grid* computacional.

5.1 Índice de Desbalanceamento de Carga (IDC)

A avaliação do desempenho de uma estratégia de balanceamento de carga pode ser auxiliada pela utilização de um índice de desbalanceamento de carga (IDC), utilizado em [48, 50]. O IDC referente à execução de uma aplicação paralela SPMD que utiliza P processadores pode ser calculada a partir da Fórmula 5.1, onde t_i é o tempo final de execução do i-ésimo processador ($1 \leq i \leq P$). Pode-se observar que esta fórmula é dada por uma relação entre a média dos tempos de ócio dos processadores (numerador da fórmula) e o tempo de execução do último processador a terminar suas tarefas (t_f - o denominador da fórmula).

$$IDC = \frac{\frac{\sum_{i=1}^{i=P}(t_f - t_i)}{(P-1)}}{t_f} \quad (5.1)$$

O IDC será utilizado como um parâmetro auxiliar na análise do desempenho de uma estratégia de balanceamento de carga.

5.2 Experimentos Realizados no *Cluster*

Os experimentos apresentados nesta subseção foram realizados no *cluster* da PUC (Pontifícia Universidade Católica do Rio de Janeiro) com a utilização de até 16 processadores. Todos os processadores neste *cluster* são Pentium 1715 MHz, interconectados por uma rede local *Fast Ethernet* e utilizam a versão 7.0.4 do *mpilam* para executar aplicações paralelas. Foi utilizada a versão 2.12.2 da ferramenta *Network Weather Service* (NWS), que foi apresentada no Capítulo 2. Vale lembrar, que a configuração de processos NWS utilizada em todos os experimentos foi a do tipo centralizada, conforme mostrada na Figura 2.2.

As estratégias de balanceamento foram executadas e analisadas utilizando as duas aplicações apresentadas no Capítulo 4: aplicação dos *termions* e aplicação *Ray Tracing*. Para ambas as aplicações variou-se o número de processadores (em 4, 8 e 16 processadores) para efeito de análise de escalabilidade. Além disso, todos os experimentos realizados com as estratégias do tipo mestre-escravo foram executados com duas configurações de processos: na primeira, um processador fica exclusivamente dedicado a executar o processo mestre (denominada SMT - sem o mestre trabalhando) e na segunda este processador também hospeda um processo escravo (denominada CMT - com o mestre trabalhando). Estas duas opções foram testadas com o objetivo de analisar se a coexistência dos processos mestre e escravo em um único processador prejudica o desempenho de uma estratégia. Na Tabela 5.1, estão identificadas todas as estratégias avaliadas nesta subseção.

Para se avaliar o comportamento das estratégias de balanceamento de carga em diferentes ambientes de execução foram realizados os experimentos em 5 ambientes diferentes, conforme apresentado na Tabela 5.2. Na primeira coluna são mostrados os identificadores dos ambientes de execução, na segunda coluna é apresentada a descrição

ID da Estratégia	Descrição
Est	Estratégia estática simples
EstNWS	Estratégia estática com NWS
MEBF1	Estratégia mestre-escravo com bloco fixo de tamanho 1
MEBF2	Estratégia mestre-escravo com bloco fixo de tamanho 2
MEBF4	Estratégia mestre-escravo com bloco fixo de tamanho 4
MEBF8	Estratégia mestre-escravo com bloco fixo de tamanho 8
MERB1	Estratégia mestre-escravo com redução de bloco – *Guided Self Scheduling* (GSS)
MERB2	Estratégia mestre-escravo com redução de bloco – *Factoring*
MERB3	Estratégia mestre-escravo com redução de bloco – *Weighted Factoring* Simples
MERB4	Estratégia mestre-escravo com redução de bloco – *Weighted Factoring* com NWS – Estático
MERB5	Estratégia mestre-escravo com redução de bloco – *Weighted Factoring* com NWS – Dinâmico
MERB6	Estratégia mestre-escravo com redução de bloco – *Weighted Factoring* com *feedback* de execução

Tabela 5.1: Identificação das estratégias avaliadas.

de cada um deles e na terceira coluna informa-se o número de vezes que foi executado o conjunto de estratégias para cada uma das aplicações testadas neste ambiente. Este conjunto de estratégias de balanceamento de carga é formado por todas as estratégias listadas na Tabela 5.1.

A partir dos dados da Tabela 5.2 conclui-se que, para obtermos a configuração para 8 e 16 processadores nas execuções com carga externa, basta replicarmos respectivamente duas e quatro vezes as configurações de carga apresentadas.

É importante ressaltar que o ambiente de execução CC-0.1.2.3 possui a presença de carga externa, mas não apresenta nenhum tipo de variação destas cargas no decorrer da execução de uma aplicação. No ambiente CC-LL.LH.HL.HH tem-se uma configuração de carga bastante heterogênea onde são utilizados todos os tipos de *traces* do *Playload*. Já nos ambientes CC-LL.LL.HL.HL e CC-LH.LH.HH.HH, independentemente do número de processadores, metade dos processadores estão carregados com cargas com baixo *load average* e a outra metade com alto *load average*. No entanto, em CC-LL.LL.HL.HL todas as cargas possuem baixa variação enquanto que em CC-LH.LH.HH.HH todas as cargas possuem alta variação.

ID do Ambiente	Descrição	Nº de Execuções
SC	Todos os processadores estão exclusivos e sem a presença de nenhuma carga externa à aplicação.	*Termions* (3 vezes) / *Ray Tracing* (5 vezes)
CC-0.1.2.3	A cada 4 processadores tem-se: um processador com 0 carga, um com 1 carga, um com 2 cargas e um com 3.	*Termions* (3 vezes) / *Ray Tracing* (5 vezes)
CC-LL.LH.HL.HH	Utiliza o playload para gerar as cargas externas e a cada 4 processadores tem-se: um processador com cargas carregadas a partir de *traces* do tipo LL, um com LH, um com HL e um com HH.	*Termions* (20 vezes) / *Ray Tracing* (20 vezes)
CC-LL.LL.HL.HL	Utiliza o playload para gerar as cargas externas e a cada 4 processadores tem-se: dois processadores com cargas carregadas a partir de *traces* do tipo LL e os outros dois com traces do tipo HL.	*Termions* (20 vezes) / *Ray Tracing* (20 vezes)
CC-LH.LH.HH.HH	Utiliza o playload para gerar as cargas externas e a cada 4 processadores tem-se: dois processadores com cargas carregadas a partir de *traces* do tipo LH e os outros dois com traces do tipo HH.	*Termions* (20 vezes) / *Ray Tracing* (20 vezes)

Tabela 5.2: Identificação dos ambientes de execução.

Na Subseção 5.2.1, são mostrados o resultados obtidos com a aplicação dos *termions* e, na Subseção 5.2.2, os resultados da aplicação *Ray Tracing*. A avaliação do desempenho de uma estratégia de balanceamento de carga levou em consideração a média dos tempos totais da execução de cada estratégia e o seu índice de desbalanceamento de carga, calculado a partir da Fórmula 5.1. Os gráficos que exibem as médias dos tempos de cada estratégia são mostrados no fim de cada subseção. Nestes gráficos, as barras vazadas representam as execuções SMT e, as barras preenchidas, as CMT. Nas Tabelas A.2 a A.11, localizadas no Apêndice A, podem ser verificados todos os valores utilizados na confecção de cada um dos gráficos apresentados, além dos índices de desbalanceamentos de carga (IDC) e o desvio padrão da média dos tempos de execução de cada estratégia.

5.2.1 Experimentos realizados com a aplicação dos *termions*

Conforme mencionado no Capítulo 4, o esforço computacional exigido por uma tarefa dos *termions* depende das propriedades físicas do meio periódico no qual estão inseridos. Para analisar o desempenho das estratégias de balanceamento de carga em uma aplicação com tarefas homogêneas, isto é, com cada tarefa levando aproximadamente o mesmo tempo para ser executada, foi estabelecida uma configuração nas propriedades físicas do meio, de modo que as propriedades térmicas das diferentes partes sólidas fossem as mesmas.

Nos experimentos realizados com esta aplicação, além da variação do número de processadores (4, 8 e 16), variou-se também o número de tarefas em 250, 500 e 1000 *termions* com o objetivo de avaliar a escalabilidade das estratégias com relação à variação do tamanho do conjunto de dados a ser executado.

A primeira análise será uma comparação entre os resultados obtidos nas execuções das estratégias do tipo mestre-escravo quando o mestre não trabalha (SMT) e quando o mestre trabalha (CMT).

Observando os gráficos das Figuras 5.1 a 5.15, pode-se concluir que, independente do ambiente de execução e do número de tarefas (*termions*), nas execuções CMT os tempos, em geral, foram menores. Além disso, também é possível notar que, à medida que se aumenta o número de processadores, a diferença entre as execuções SMT e CMT vai diminuindo, pois a ausência de um processo escravo tende a ter menos impacto quando existem muitos processadores. Em alguns casos, nas execuções com 16 processadores, os tempos SMT e CMT foram quase os mesmos, não havendo diferença significativa quando o mestre trabalha.

Devido ao bom desempenho apresentado pelas estratégias do tipo mestre-escravo CMT, a partir de agora, todas as comparações entre as estratégias serão feitas com os tempos obtidos nesta abordagem.

O fato de o tempo de computação de cada *térmion* ser aproximadamente o mesmo, permite uma análise de escalabilidade com relação à variação do número de tarefas executadas. Comparando as execuções para um mesmo número de processadores e variando-se a quantidade de *termions* pode-se concluir que, independente do ambiente onde foi realizado o experimento, o tempo total de execução dobra de 250 para 500

termions e de 500 para 1000 *termions*.

Os experimentos realizados no ambiente SC (gráficos das Figuras 5.1, 5.2 e 5.3) apresentaram um comportamento bastante previsível. As duas estratégias estáticas obtiveram um desempenho semelhante indicando que o EstNWS distribuiu as tarefas da mesma forma que o Est, após verificar que os processadores estavam completamente disponíveis. Vale lembrar, que neste ambiente todos os processadores eram homogêneos e estavam exclusivos para estes experimentos.

As estratégias estáticas apresentaram um desempenho melhor do que as estratégias do tipo mestre-escravo SMT. Além da vantagem de terem todas as máquinas executando tarefas, estas estratégias estabelecem menos comunicações entre os processadores, uma vez que são enviados apenas $P-1$ blocos de tarefas no início da execução. No entanto, a vantagem de poder contar com mais um processador foi diminuindo à medida que se aumentou o número de processadores. Nas execuções CMT, as estratégias estáticas também obtiveram um desempenho superior ao da maioria das estratégias do tipo mestre-escravo, com ganhos entre 0,17% e 10,45%.

De maneira geral, observando os gráficos das Figuras 5.1, 5.2 e 5.3, que refletem as execuções sem carga, pode-se concluir que a maioria das estratégias apresentaram um comportamento semelhante. A estratégia MEBF8 foi a que apresentou o pior desempenho neste ambiente, pois o envio de último bloco com oito tarefas possivelmente gerou desequilíbrios de carga. Este comportamento foi evidenciado principalmente nas execuções com 250 e 500 termions, onde o este desequilíbrio pode ocasionar um maior impacto no tempo total de execução. As estratégias do tipo MERB se mostraram eficientes e com desempenho bastante semelhantes. Apenas as execuções com 4 processadores e 1000 termions das estratégias MERB1, MERB2 e MERB3, não apresentaram um bom desempenho, sendo 12,5%, 11,6% e 7,6%, respectivamente, mais lentas que as demais estratégias MERB.

No ambiente SC, as estratégias do tipo MEBF podem ser avaliadas de acordo com o impacto do tamanho do bloco no tempo total de computação. O tamanho dos blocos e, conseqüentemente, a quantidade de blocos enviados foram fatores determinantes para que a maioria das estratégias com blocos maiores do que 1 obtivessem pequenos ganhos quando comparadas com a estratégia MEBF1. Nestas estratégias, a redução do tempo de comunicação, embora não muito significativo, possibilitou, na maioria dos casos,

ganhos entre 0,61% e 8,75%.

Também é possível observar (gráficos das Figuras 5.1, 5.2 e 5.3) que utilizar blocos de tamanho grande na estratégia MEBF não é uma solução para diminuir esse pequeno *overhead* de comunicação. Analisando os tempos de execução obtidos com o MEBF8 e associando-os com os IDC calculados para esta estratégia percebe-se um desbalanceamento bastante alto, principalmente nas execuções com menos *termions* e 16 processadores. (250 *termions* – IDC entre 7,8% e 34,12%, 500 termions – IDC entre 2,59% e 18,79%, e 1000 termions – IDC entre 1,69% e 9,88%). Isto se deve ao fato de que muitas vezes o mestre envia o último bloco de tarefas para um escravo enquanto que os demais ficam ociosos até que este termine de executar as oito tarefas recebidas. Em alguns casos, também notou-se que blocos de tamanho quatro também proporcionaram pequenos desbalanceamentos.

Os experimentos realizados no ambiente CC-0.1.2.3 (gráficos das Figuras 5.4, 5.5 e 5.6) evidenciaram, principalmente, o problema de se executar a estratégia estática simples em um ambiente com a presença de carga externa. Observa-se que o desbalanceamento nesta estratégia foi bastante alto (IDC entre 24,9% e 39,2%), o que explica o motivo do mau desempenho desta estratégia neste ambiente.

Para tentar amenizar este problema, utilizou-se a estratégia estática com NWS (EstNWS) que utiliza as medições de disponibilidade de CPU, fornecidas pelo NWS, como parâmetros que auxiliam na distribuição de tarefas entre os processadores. Embora o tempo total de computação nesta estratégia tenha sido melhor do que o estático simples (para 4 processadores com 250, 500 e 1000 *termions* ganhos de: 38,9%, 52,75% e 37,6%, respectivamente – Para 8 processadores com 250, 500 e 1000 *termions* ganhos de: 95%, 94,9% e 97,1%, respectivamente – Para 16 processadores com 250, 500 e 1000 *termions* ganhos de: 96,1%, 101,25% e 98,1%, respectivamente), as execuções com quatro processadores apresentaram altos IDC (em média 25% de desbalanceamento com 4 processadores), o que mostra que esta estratégia não apresentou um bom desempenho quando executada com poucos processadores. Vale ressaltar, que o EstNWS obteve um desempenho melhor do que o da maioria das estratégias do tipo mestre-escravo SMT e um desempenho equivalente ao das melhores estratégias do tipo mestre-escravo CMT nas execuções com 8 e 16 processadores.

Nos gráficos das Figuras 5.4, 5.5 e 5.6 pode-se observar que a maioria das estratégias

do tipo mestre-escravo obtiveram desempenhos semelhantes e eficientes. Além disso, novamente, a estratégia MEBF8 não apresentou um bom desempenho, pelo mesmo motivo citado na avaliação feita no ambiente SC. Da mesma forma, em algumas execuções da MEBF4 também foi possível verificar desbalanceamentos significativos. A estratégia MERB1 também não obteve desempenho satisfatório em nenhuma das execuções. Por meio de uma observação mais detalhada dos resultados foi possível concluir que a fórmula que define o tamanho dos blocos nesta estratégia não é adequada, pois na maioria das vezes alguns processadores recebem blocos grandes demais e ficam extremamente sobrecarregados.

Neste ambiente, as estratégias MERB4, MERB5 e MERB6 tiveram um comportamento bastante semelhante, apresentando um ótimo desempenho. O melhor desempenho destas estratégias pode ser observado principalmente nas execuções com 4 e 8 processadores onde estas estratégias, quando comparadas com a demais estratégias do tipo mestre-escravo (com exceção das que não apresentaram bom desempenho: MEBF8 e MERB1), chegaram a ter ganhos entre 0,04% e 12,19%.

Os experimentos realizados no ambiente CC-LL.LH.HL.HH (gráficos das Figuras 5.7, 5.8 e 5.9) tiveram por objetivo analisar o desempenho das estratégias em um ambiente onde os processadores possuem cargas externas bastante variadas. Para isto, foram executados vários tipos de cargas externas através da ferramenta *playload* (tipos de *traces*: LL, LH, HL e HH). De maneira geral, pode-se concluir que neste ambiente a maioria das estratégias avaliadas apresentaram um desempenho bastante semelhante.

Embora o ambiente CC-LL.LH.HL.HH apresente cargas diversificadas algumas considerações podem ser feitas mediante os resultados apresentados. A primeira observação diz respeito ao desempenho da estratégia EstNWS em comparação com as estratégias de tipo mestre-escravo CMT. Ao comparar o desempenho desta estratégia nos ambientes CC-0.1.2.3 e CC-LL.LH.HL.HH, percebeu-se que no primeiro ocorreram diferenças bastante significativas entre o EstNWS e as estratégias do tipo mestre-escravo quando foram utilizados apenas 4 processadores. Para 8 e 16 processadores ocorreu exatamente o contrário, pois não havia sido observado diferenças significativas entre o EstNWS e as estratégias mestre-escravo no ambiente CC-0.1.2.3 e no ambiente CC-LL.LH.HL.HH verificou-se, na maioria dos casos, diferenças consideráveis. A maior variação das cargas externas do ambiente CC-LL.LH.HL.HH pode ter gerado um impacto nas execuções

com um número maior de processadores, pois nestes casos é menor o tempo total de execução da aplicação.

Uma segunda observação pode ser feita quanto à repetição do mau desempenho das estratégias MEBF8 e MERB1, especialmente nas execuções com oito e 16 processadores. Novamente, o alto IDC (que apresentou valores entre 5,8% e 32,67%) justificou o alto tempo de computação para estas estratégias, principalmente a MERB1.

Assim como foi verificado nos experimentos realizados nos ambientes avaliados anteriormente, a estratégia MEBF8 e MERB1 também não apresentaram um bom desempenho no ambiente CC-LL.LL.HL.HL (gráficos das Figuras 5.10, 5.11 e 5.12). Além disso, foi possível observar uma piora no desempenho da estratégia EstNWS quando comparada com a estratégia Est e com a maioria das estratégias do tipo mestre-escravo (comparação com os resultados obtidos na execução CC-0.1.2.3). No ambiente CC-LL.LL.HL.HL, metade dos processadores possui cargas leves (L - com baixo *load average*), a outra metade cargas pesadas (H - com alto *load average*) e todas essas cargas não sofrem grandes variações no decorrer da execução da aplicação. Mesmo assim, pequenas variações em processadores com cargas pesadas podem ter prejudicado a estratégia EstNWS.

A maioria das estratégias do tipo mestre-escravo apresentaram um comportamento semelhante (com exceção das estratégias MEBF8 e MERB1). Apenas nos experimentos com 1000 *termions*, principalmente na execução com 16 processadores, observou-se uma diferença significativa entre as estratégias MERB2 e MERB3 e as melhores estratégias MERB (diferença de aproximadamente 44%). Isto ocorreu porque estas duas estratégias não efetuam recálculo de peso durante a execução e, além disso, as pequenas variações das cargas geradas pelo *playload* neste ambiente podem ter tido uma maior impacto nas execuções com 16 processadores, pois o tempo total de computação é menor.

No ambiente CC-LH.LH.HH.HH (gráficos das Figuras 5.13, 5.14 e 5.15) a estratégia EstNWS apresentou o seu pior desempenho em comparação com as estratégias do tipo mestre escravo. Percebe-se que a diferença entre esta estratégia e a Est diminuiu ainda mais, o que indica que esta estratégia não conseguiu se adaptar bem a um ambiente onde as cargas apresentam uma alta variação. Este comportamento foi verificado, pois as tarefas são divididas no início da execução com base na situação do ambiente

naquele instante, não considerando possíveis modificações neste durante a execução da aplicação.

A maioria das estratégias MERB apresentaram um bom desempenho, novamente tendo a MERB1 como exceção. As estratégias MERB5 e MERB6, que utilizam a técnica de reajuste de peso durante a execução, apresentaram, na maioria dos casos, pequenos ganhos quando comparadas com as estratégias MERB3 e MERB4, que definem o peso no início da execução e não o atualizam durante a execução (com ganhos médios de 3,33%, chegando a 15,18%). Este resultado também pode ser explicado pela característica de alta variação das cargas externas neste ambiente.

Uma observação interessante pode ser feita quanto ao comportamento das estratégia MERB6 para 16 processadores com 500 e 1000 termions. Conforme visto, esta estratégia define e redefine o peso de um processador a partir do *feedback* de execução (tempo médio de execução de uma tarefa). O fato do *feedback* se tratar de uma situação verificada no passado pode ter prejudicado esta estratégia, pois a informação recebida se tornou desatualizada devido à alta variação das cargas neste ambiente. Desta forma, o peso calculado não estaria refletindo a capacidade real de processamento de um escravo, o que gerou desbalanceamentos (IDC igual a 11,94% para 500 e IDC igual a 15,09% para 1000 *termions*). Além desse fator, o *overhead* gasto no recálculo dos pesos a cada fim de lote também pode ter contribuído para uma queda no desempenho. Este fator também pode ter prejudicado a estratégia MERB5 (vide execuções com 250 e 500 *termions*), uma vez que esta estratégia redefine periodicamente os pesos com base nas medições coletadas do NWS. Estes resultados foram verificados apenas nas execuções com 16 processadores, pois um desbalanceamento tende a ter um impacto maior quando o tempo total de execução for menor. Além disso, o recálculo de pesos se torna mais custoso à medida que se aumenta a quantidade de processadores escravos.

Dentre as estratégias MEBF (no ambiente CC-LH.LH.HH.HH) pode-se destacar o bom desempenho da MEBF1 e MEBF2, as quais obtiveram desempenhos semelhantes aos das melhores estratégias MERB. As execuções da estratégia MEBF8 novamente apresentaram altos IDC (variando entre 10,4% e 26,8%), o que também ocorreu nas execuções da estratégia MEBF4 com 16 processadores, principalmente nas execuções com menos tarefas (IDC igual a 17,5% para 250 *termions*, IDC igual a 11,56% para 500 *termions* e IDC igual a 7,43% para 1000 *termions*).

Outra estratégia que apresentou um bom desempenho neste ambiente foi a MERB2 (*Factoring*), com resultados semelhantes às melhores estratégias do tipo mestre-escravo. Apenas nas execuções com 16 processadores e 1000 *termions* foi possível notar um mau desempenho desta estratégia. Nestas execuções, as estratégias MERB2 e MERB3 obtiveram desempenhos bastante semelhantes, sendo 29,6% piores do que a estratégia MERB5, que obteve o melhor desempenho.

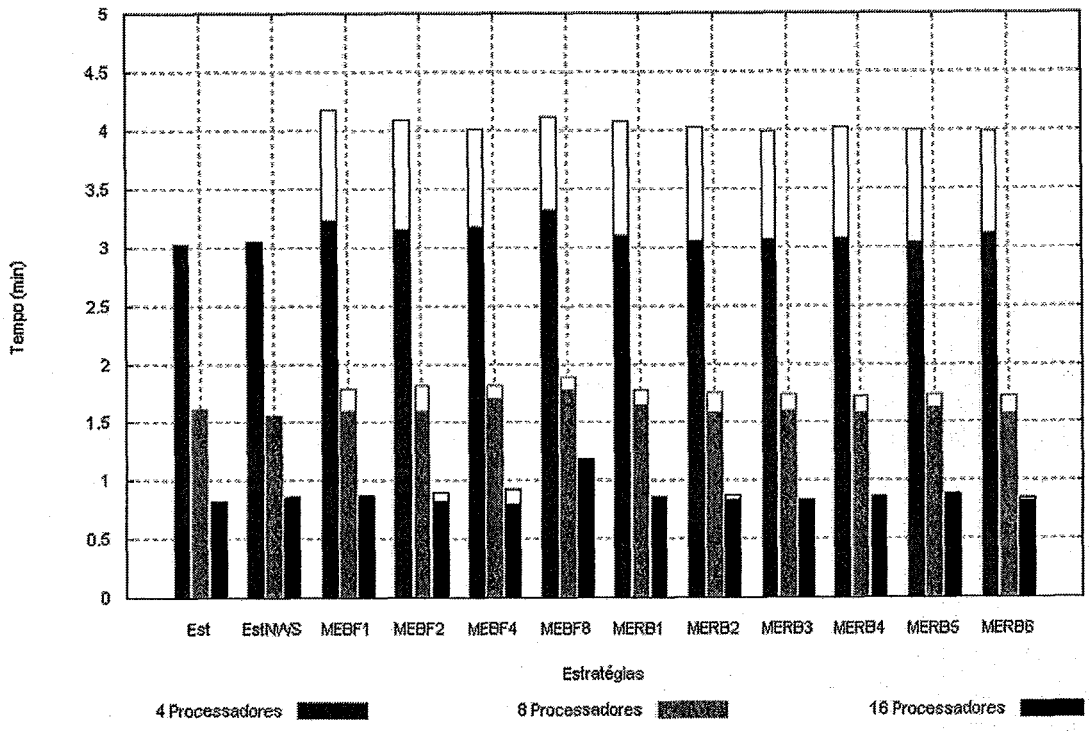

Figura 5.1: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente SC) com 250 *termions*

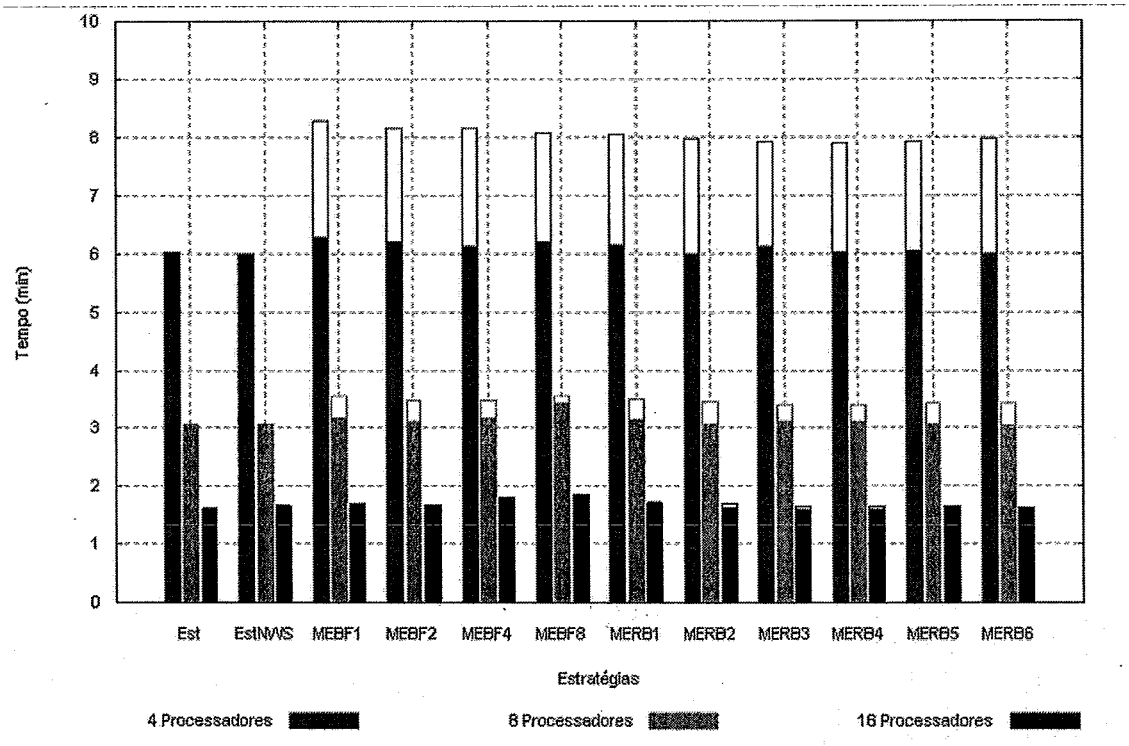

Figura 5.2: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente SC) com 500 *termions*

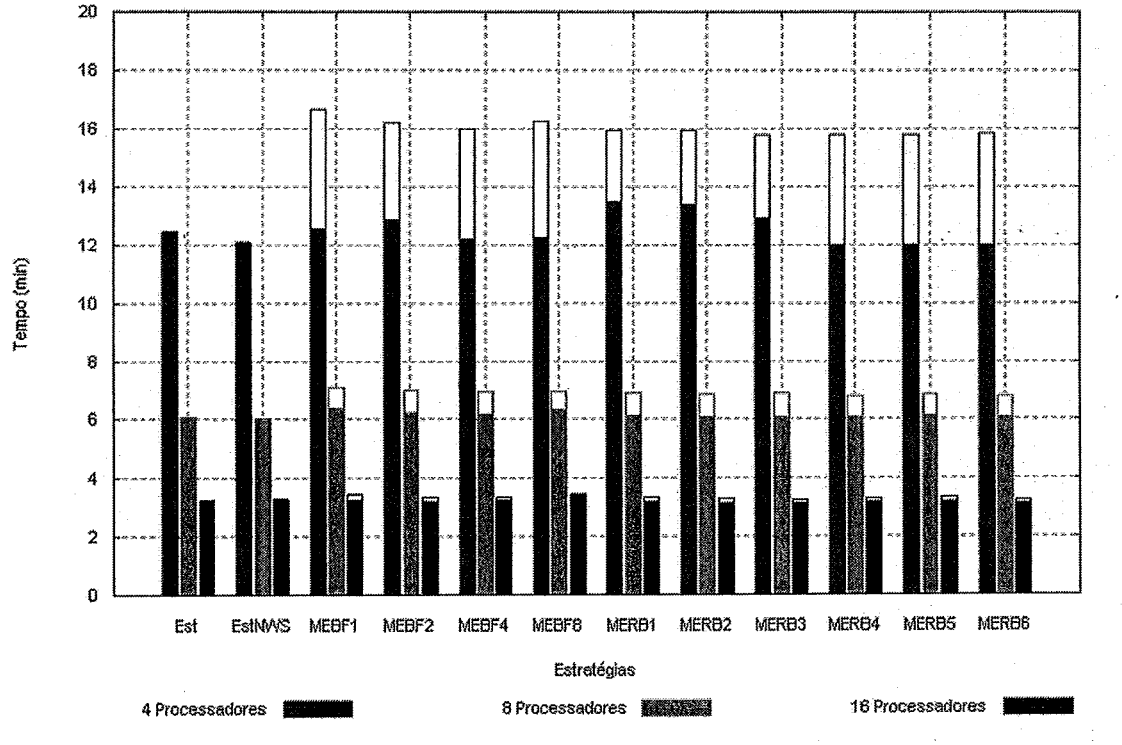

Figura 5.3: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente SC) com 1000 *termions*

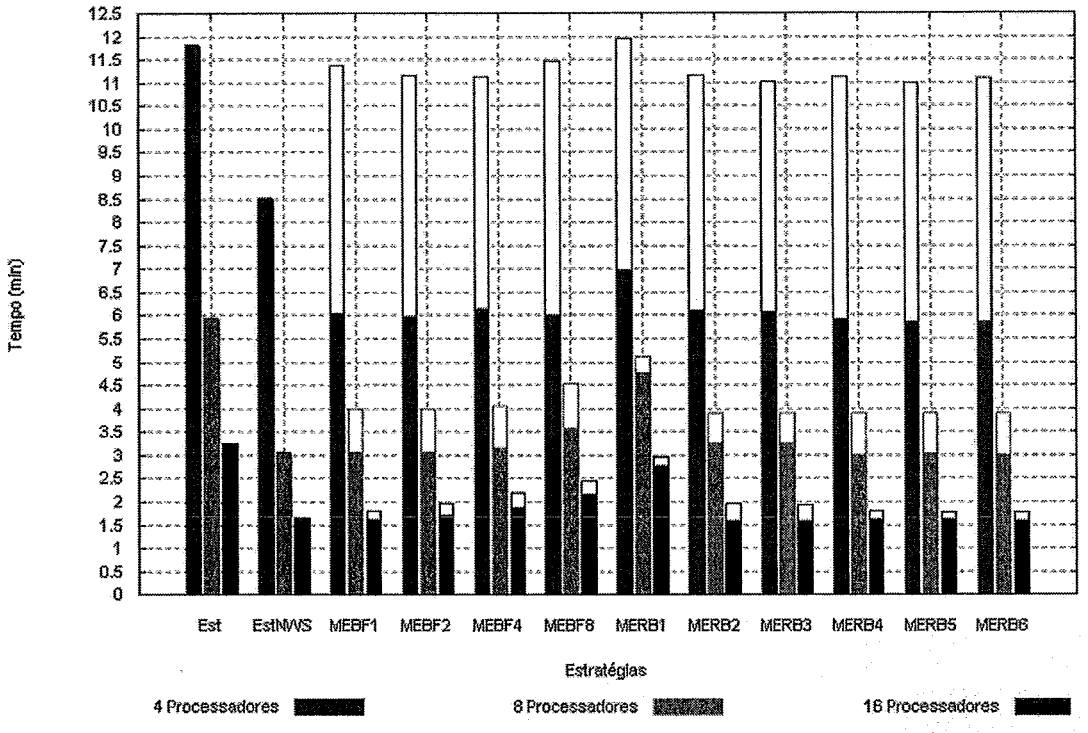

Figura 5.4: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-0.1.2.3) com 250 *termions*

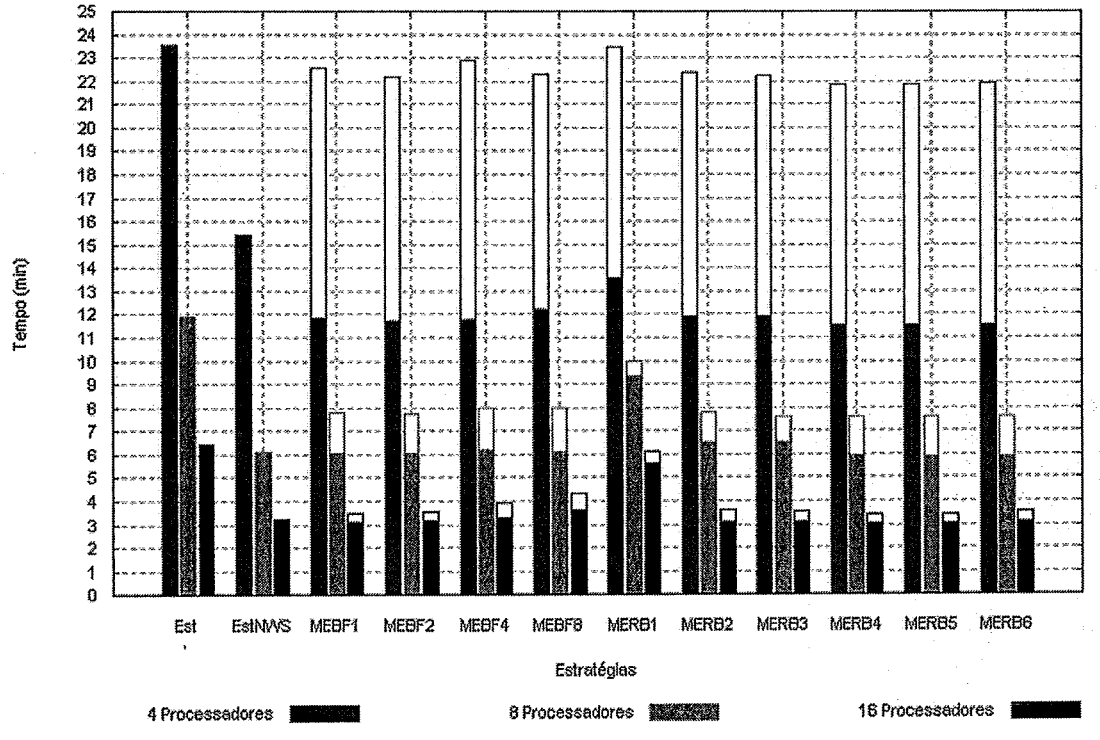

Figura 5.5: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-0.1.2.3) com 500 *termions*

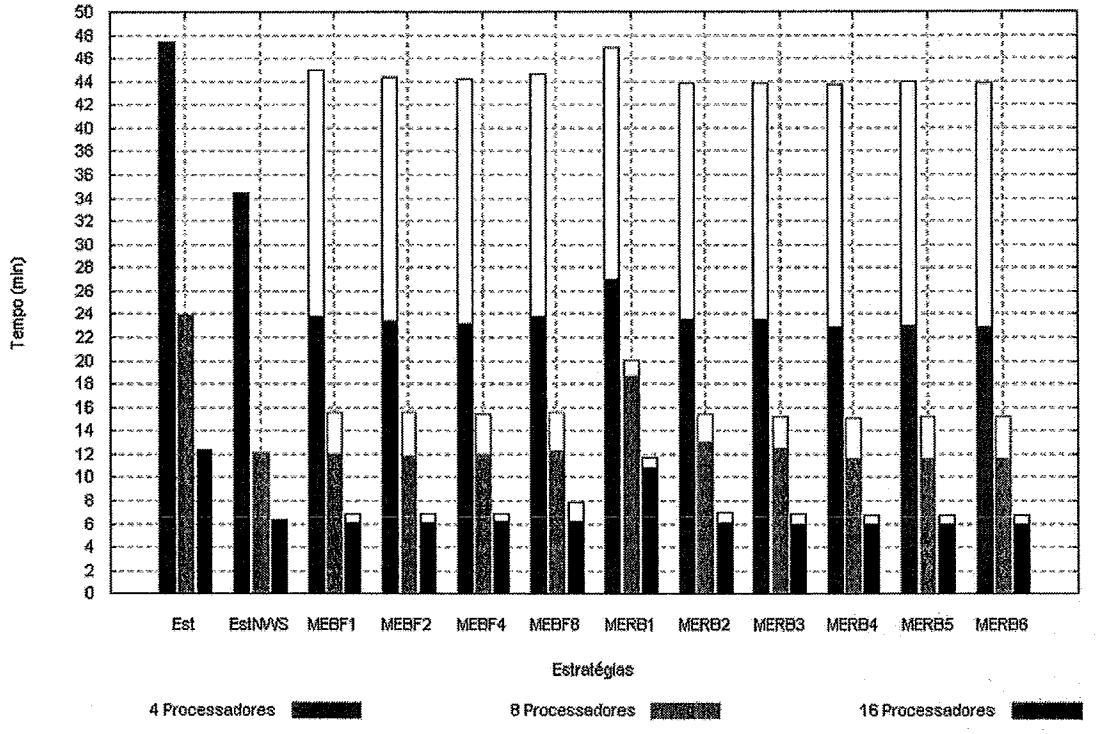

Figura 5.6: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-0.1.2.3) com 1000 *termions*

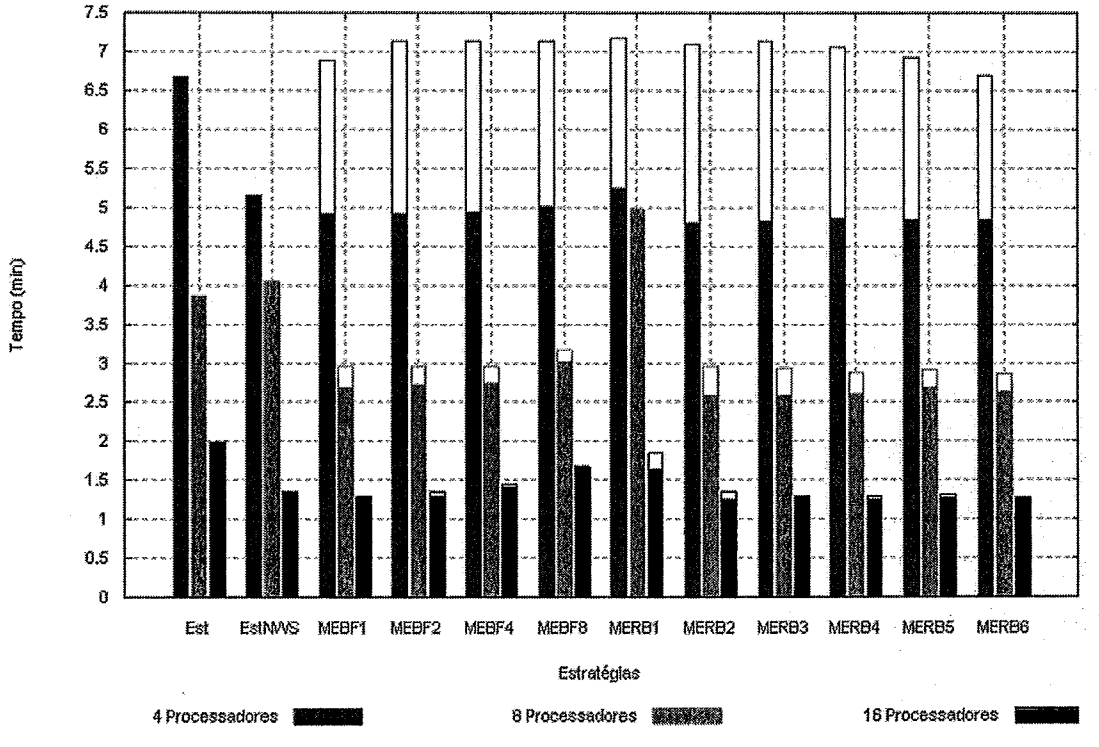

Figura 5.7: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LL.LH.HL.HH) com 250 *termions*

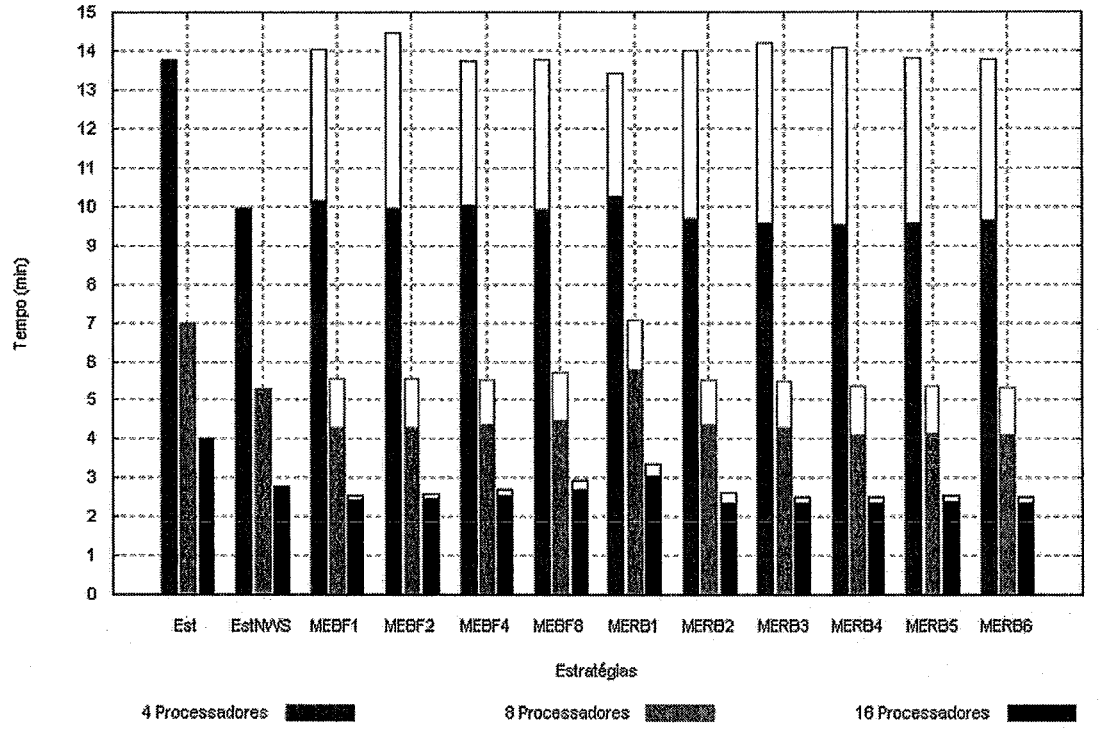

Figura 5.8: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LL.LH.HL.HH) com 500 *termions*

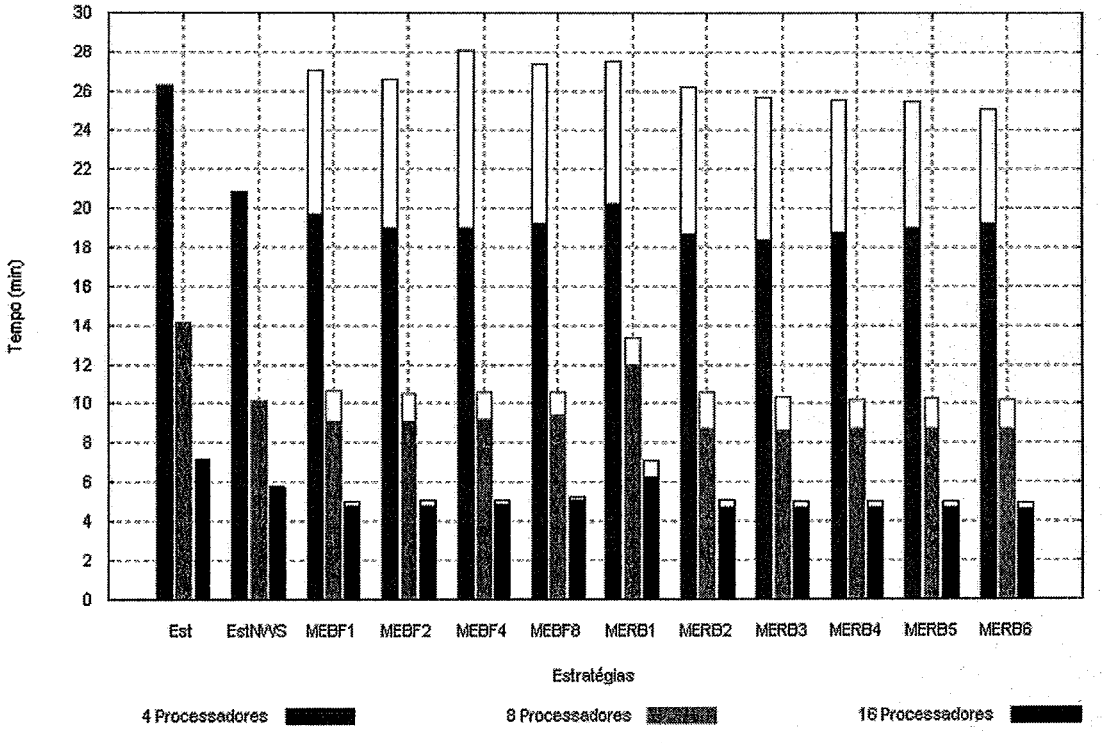

Figura 5.9: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LL.LH.HL.HH) com 1000 *termions*

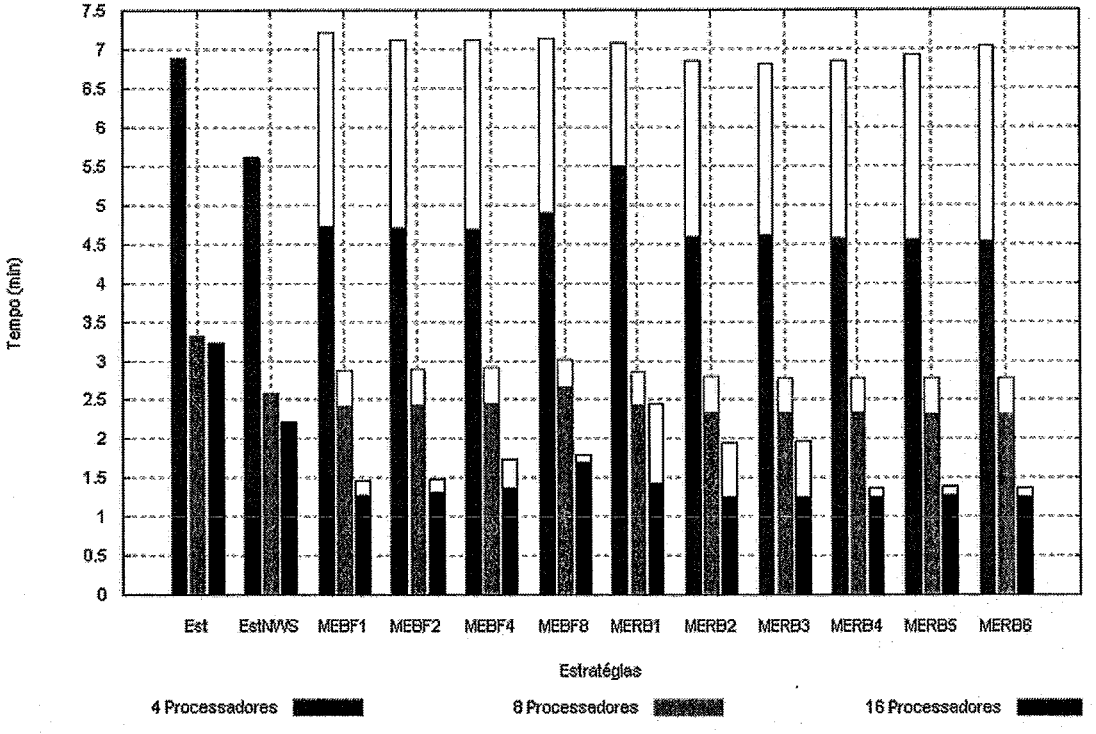

Figura 5.10: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LL.LL.HL.HL) com 250 *termions*

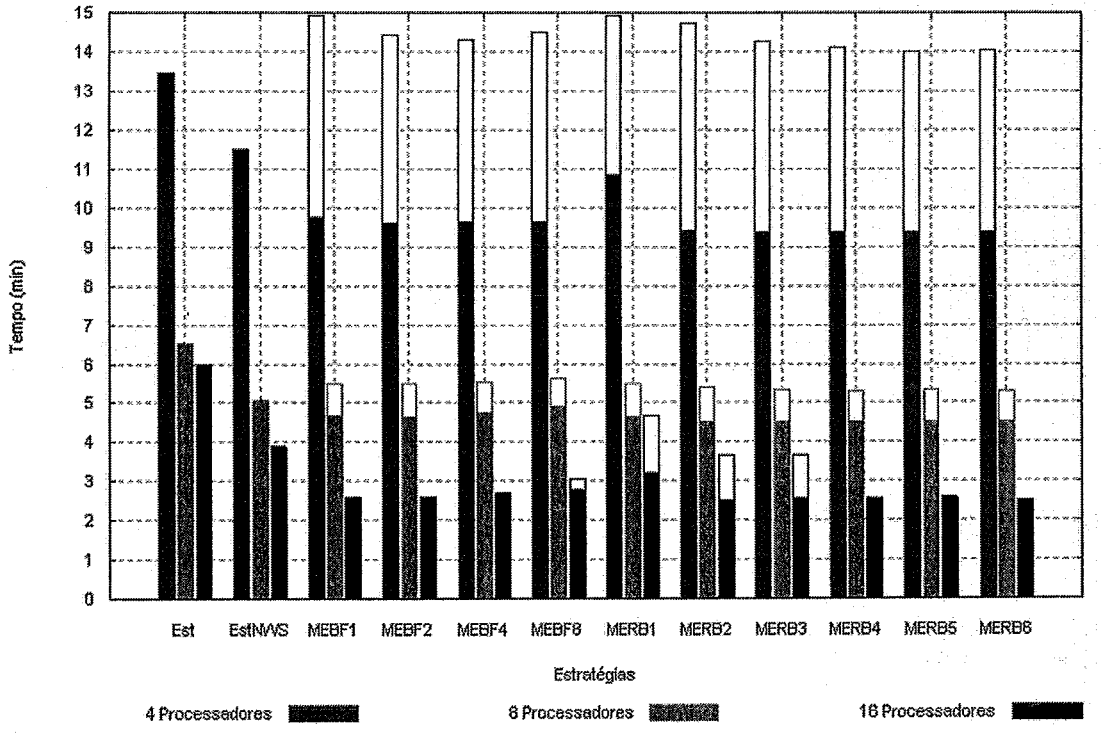

Figura 5.11: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LL.LL.HL.HL) com 500 *termions*

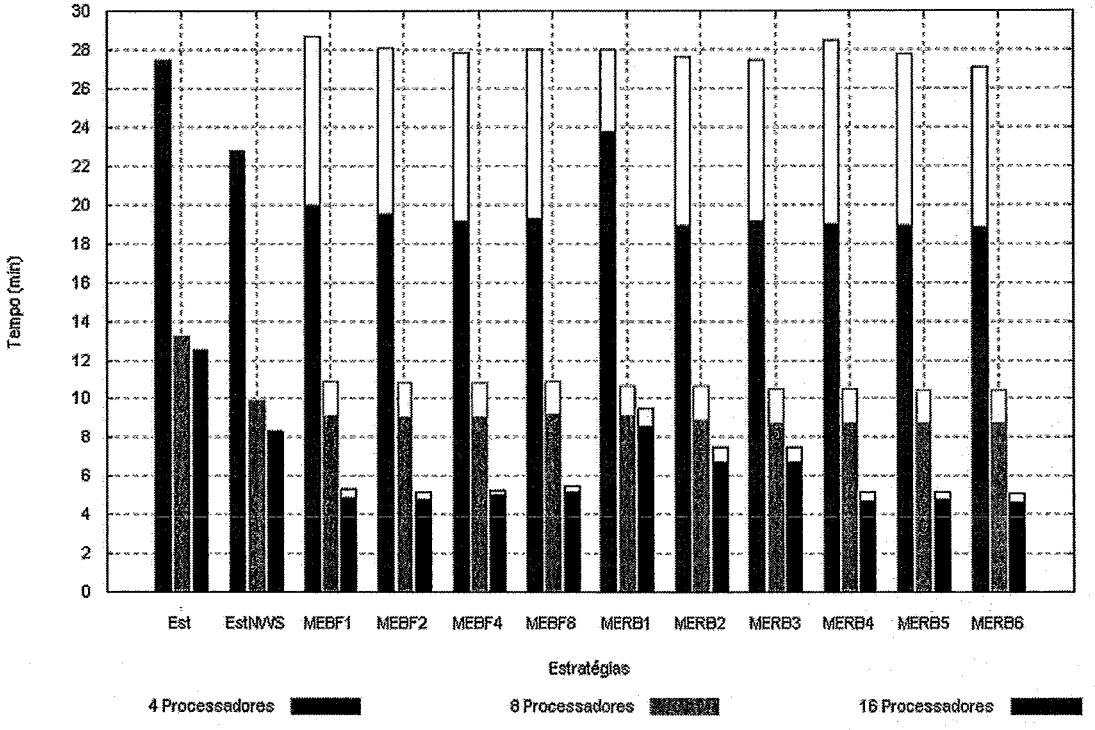

Figura 5.12: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LL.LL.HL.HL) com 1000 *termions*

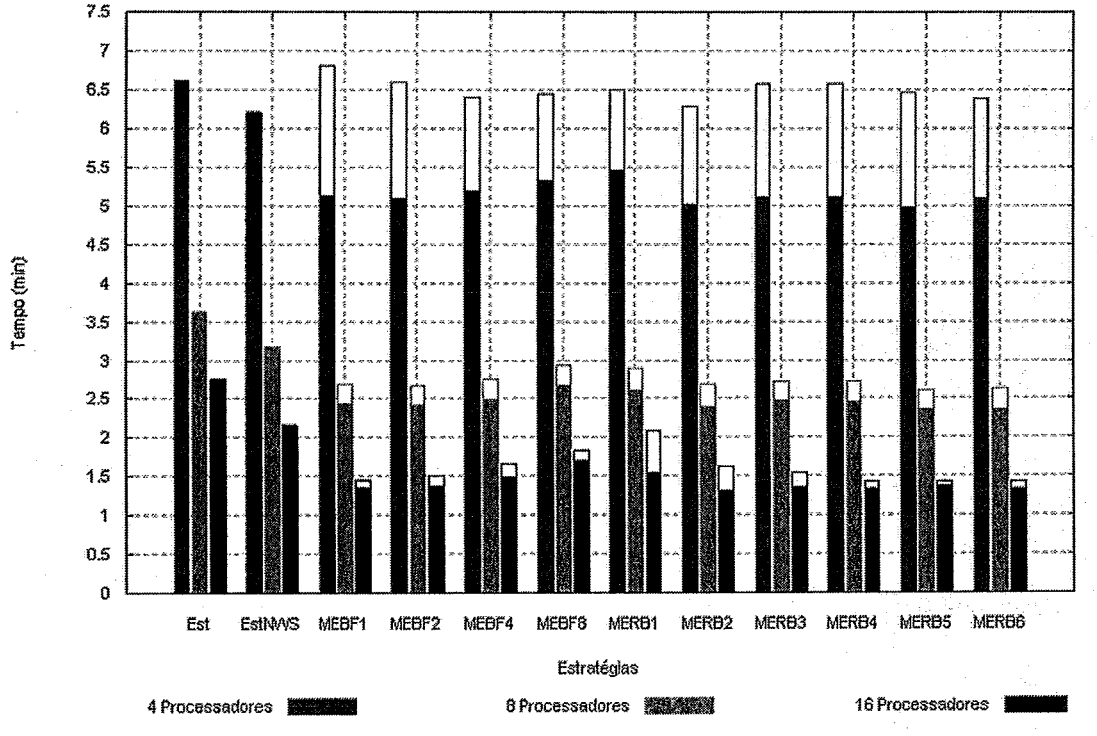

Figura 5.13: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LH.LH.HH.HH) com 250 *termions*

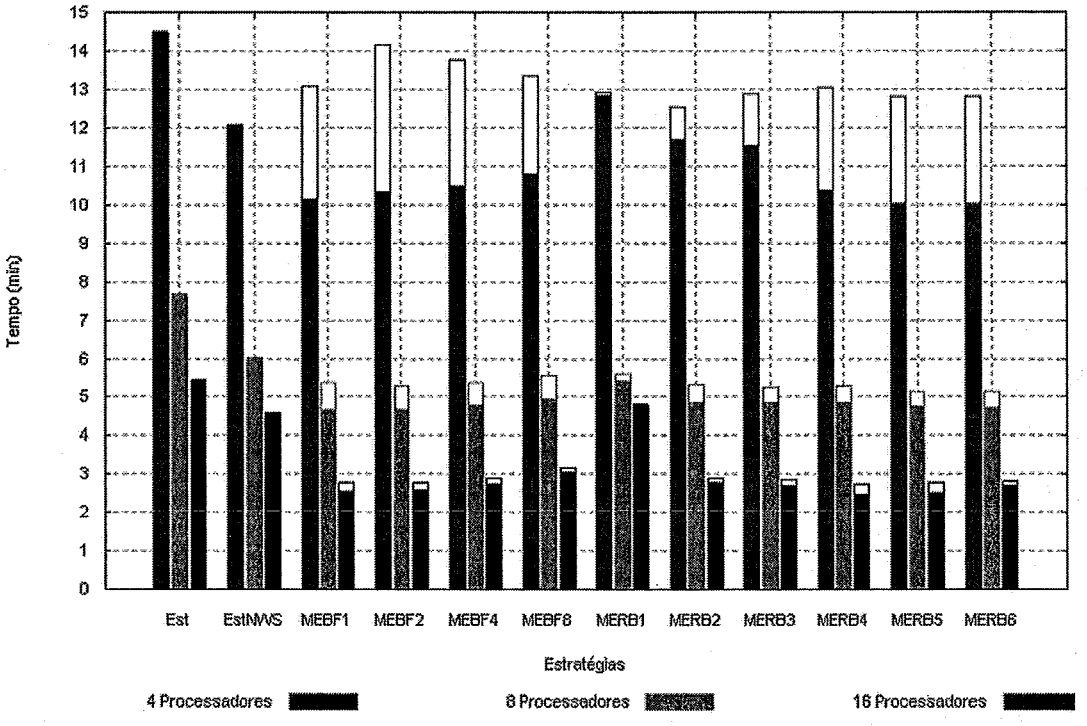

Figura 5.14: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LH.LH.HH.HH) com 500 *termions*

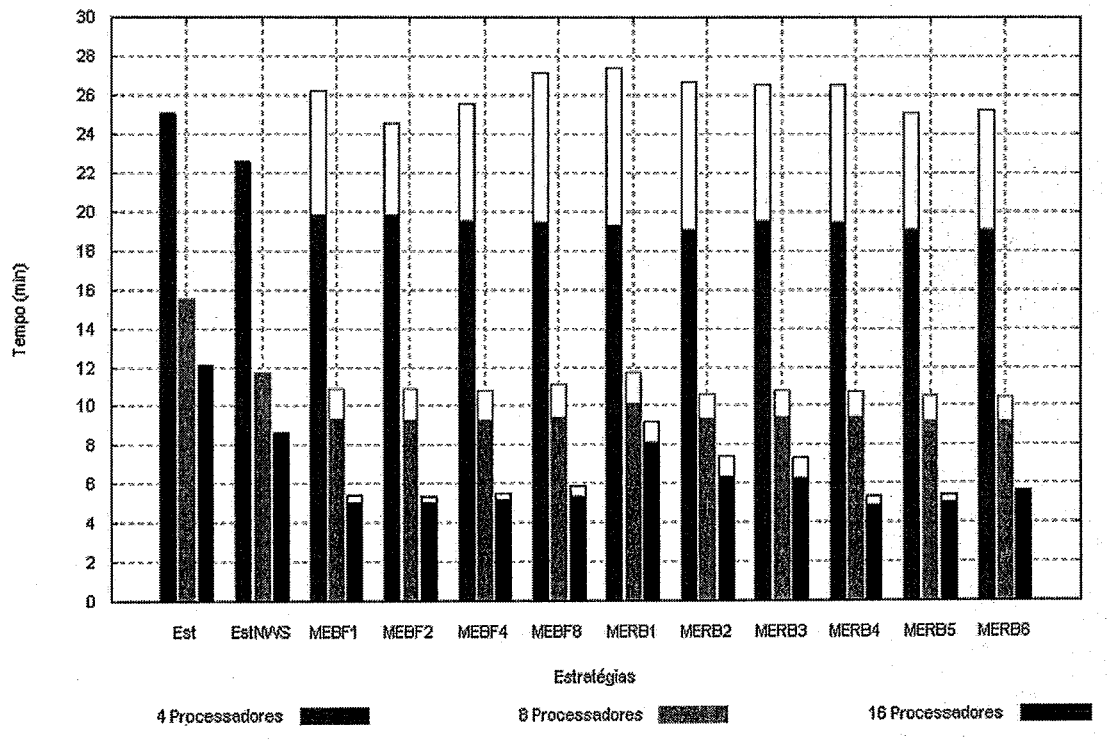

Figura 5.15: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *cluster* (ambiente CC-LH.LH.HH.HH) com 1000 *termions*

5.2.2 Experimentos realizados com a aplicação *Ray Tracing*

De acordo com o que foi mostrado no Capítulo 4, a aplicação *Ray Tracing* consiste em uma técnica que possibilita a renderização de imagens. A utilização da técnica de subdivisão de um *pixel* resulta no aumento do tempo de computação de cada *pixel* em uma imagem, pois, para cada *subpixel*, será calculada a técnica de *Ray Tracing* tradicional. Os experimentos apresentados nesta subseção adotaram uma subdivisão de *pixel* na ordem 4×4, indicando que um *pixel* foi dividido em 16 sub-regiões (*subpixels*).

Uma imagem pode ser constituída por diversos objetos que compõem a cena. Sendo assim, o desempenho de uma estratégia de balanceamento de carga vai depender das características da imagem que está sendo renderizada. A quantidade de objetos, bem como a distribuição destes na cena, pode ser um fator de desequilíbrio de carga. Como os *pixels* são inseridos no repositório de tarefas de maneira seqüencial (isto é, seguindo a seqüência da sua posição na matriz de *pixels*) e a heterogeneidade desta aplicação depende das características da imagem renderizada, a renderização de um bloco contíguo de *pixels* pode demandar um alto custo computacional quando estiverem em uma região onde se concentram os objetos da cena. Da mesma forma, podem demandar um baixo custo computacional caso estejam em uma região de *background*.

Os experimentos mostrados nesta subseção foram feitos com a utilização das duas imagens apresentadas no Capítulo 4, denominadas *5balls* e *room*, ambas de tamanho 500×500 (isto é, com 25000 *pixels*). A aplicação *Ray Tracing* é altamente paralelizável. Sua versão seqüencial executada em um Pentium 1715 MHz levou 8,01 minutos para renderizar a imagem *5balls* e 4,20 minutos para a imagem *room*. Ambas as renderizações foram feitas com a divisão de cada *pixel* na ordem 4×4.

Assim como na aplicação dos *termions*, os experimentos com esta aplicação foram realizados com 4, 8 e 16 processadores, com as mesmas estratégias (Tabela 5.1) e nos mesmos ambientes de execução (Tabela 5.2). Os gráficos das Figuras 5.16 a 5.25, exibidos no final desta subseção, mostram os resultados obtidos com esta aplicação.

Novamente, a primeira análise será feita sobre as estratégias do tipo mestre-escravo quando o mestre não trabalha (SMT) e quando o mestre trabalha (CMT). Conforme mencionado, os tempos das execuções SMT são exibidos nos gráficos através de barras vazadas e os tempos CMT com barras preenchidas. Da mesma forma como ocorreu na

aplicação dos *termions*, a configuração CMT apresentou melhor desempenho, principalmente na execução com 4 processadores. No entanto, uma pequena diferença pode ser observada, para ambas imagens utilizadas, nas execuções com 16 processadores. Percebe-se que, na maioria das vezes, o fato de o mestre executar um processo escravo não representou nenhuma vantagem, ou o ganho foi insignificante. Conclui-se que, a medida que se aumenta o número de processadores, diminui-se o impacto gerado pela ausência de um processador na abordagem SMT.

Este comportamento pode ser explicado pelo caráter heterogêneo das tarefas desta aplicação, o qual tende a causar um maior impacto nas execuções com mais processadores devido ao menor tempo total de computação. Esta afirmativa pode ser confirmada observando-se todos os gráficos apresentados no final desta subseção.

Mesmo sendo uma aplicação com tarefas heterogêneas, as execuções com 4, 8 e 16 processadores apresentaram uma boa escalabilidade, reduzindo à metade o tempo total nas execuções de 4 para 8 processadores e também de 8 para 16.

Os experimentos realizados no ambiente SC (gráficos das Figuras 5.16 e 5.17) evidenciaram que a aplicação *Ray Tracing* tem um caráter heterogêneo, pois o tempo de computação de cada *pixel* pode ser variado. Esta característica pôde ser confirmada, principalmente, na renderização da imagem *room* onde foi possível observar que as estratégias estáticas não apresentaram um bom desempenho. Com esta imagem, a maioria das estratégias do tipo mestre-escravo (com exceção apenas da MEBF1 com 16 processadores que foi pior do que as estáticas) obtiveram ganhos entre 7% e 41,8% em relação às estratégias estáticas. Esta observação demonstra que a imagem *room* é bastante heterogênea no que diz respeito ao tempo de computação de cada *pixel* da imagem. Um *pixel* que compõe um objeto e está localizado em uma área que requer cálculos de reflexão, por exemplo, pode demandar um maior custo computacional. Já na renderização da imagem *5balls*, observa-se que todas as estratégias apresentaram um desempenho bastante semelhante, o que evidencia um caráter mais homogêneo desta imagem. Embora mais heterogênea que a imagem *5balls*, a renderização da imagem *room* é 80% mais rápida. Isto ocorre porque a renderização da imagem *5balls* necessita de cálculos mais completos para a maioria dos seus *pixels*.

Todas as estratégias apresentaram um comportamento bastante semelhante na renderização da imagem *5balls* (gráfico da Figura 5.16). Mesmo com um desempenho

satisfatório, as estratégias estáticas apresentaram IDCs relativamente altos, variando entre 9,12% e 11,11%. Já as estratégias do tipo mestre-escravo obtiveram, na sua maioria, desempenhos praticamente iguais.

Na renderização da imagem *room*(gráfico da Figura 5.17), a estratégia MEBF1 não apresentou um bom desempenho, especialmente nas execuções com 16 processadores. O mau desempenho desta estratégia pode ser explicado pela maior heterogeneidade das tarefas quando a aplicação é executada para esta imagem, conforme já foi mencionado, somado ao maior *overhead* de comunicação através do envio de blocos unitários de tarefas.

As renderizações das imagens *5balls* e *room* (gráficos das Figuras 5.18 e 5.19), no ambiente CC-0.1.2.3, e utilizando a estratégia Est, apresentaram um péssimo desempenho devido à presença de carga externa. A estratégia EstNWS, que utiliza as previsões do NWS para detectar a presença de carga externa nos escravos, apresentou um bom desempenho para a imagem *5balls* e um desempenho regular para a imagem *room*. Esta estratégia obteve um melhor comportamento para a imagem *5balls* devido à característica mais homogênea desta imagem em comparação com a imagem *room*. Na imagem *5balls* nota-se que a estratégia EstNWS obteve menores diferenças de desempenho em relação às melhores estratégias do tipo mestre-escravo, principalmente nas execuções com 8 e 16 processadores (comparação com as estratégias do tipo mestre-escravo com melhor desempenho: 9,7% pior do que a MERB6 com 4 processadores; 8,95% pior que a MERB6 com 8 processadores e 9,7% pior que a MERB4 com 16 processadores). Já a imagem *room* apresentou diferenças maiores (25% pior do que a MERB2 com 4 processadores; 43,1% pior que a MERB4 com 8 processadores e 47,4% pior que a MERB3 com 16 processadores). Esta maior diferença para a imagem *room* pode ser explicada pelo maior desbalanceamento na renderização desta imagem utilizando o EstNWS (vide os IDCs nas Tabelas A.7 a A.11 do Apêndice A).

As execuções das estratégias do tipo mestre-escravo no ambiente CC-0.1.2.3 apresentaram um comportamento bastante eficiente e semelhante na renderização da imagem *5balls* (gráfico da Figura 5.18 – Uma exceção pode ser observada nas execuções da estratégia MERB1, que não obteve um bom desempenho, também ocasionado pela forma como esta estratégia determina o tamanho dos blocos) e pequenas variações para a imagem *room* (gráfico da Figura 5.19). Para a imagem *room*, as piores estratégias

foram a MEBF8 e a MERB1, pois apresentaram os piores desempenhos com 8 e 16 processadores. Isto pode ser observado a partir dos valores mais altos dos IDCs nestas estratégias (que variaram entre 7,32% e 26,31%).

Os experimentos realizados no ambiente CC-LL.LH.HL.HH com a imagem *5balls* (gráfico da Figura 5.20) apresentaram resultados semelhantes às execuções feitas no ambiente CC-0.1.2.3 (gráfico da Figura 5.18). Todas as estratégias (com exceção da MERB1) obtiveram tempos finais de execução bem próximos. Novamente a estratégia MERB1 não se mostrou eficiente, tendo inclusive um desempenho inferior ao mostrado no ambiente com cargas estáticas (CC-0.1.2.3), independente do número de processadores utilizados. Nota-se que a diversidade de tipos de cargas externa deste ambiente influenciaram o desempenho desta estratégia. Isto não foi observado na renderização da imagem *room* (gráfico da Figura 5.21), pois esta estratégia obteve um ótimo desempenho, particularmente com 8 e 16 processadores (apenas com 4 processadores esta estratégia não obteve um bom desempenho). Neste caso, este comportamento pode ser explicado por dois fatores: menor tempo gasto na computação de uma tarefa, o que tende a evitar desbalanceamentos e pela característica mais heterogênea desta imagem.

O comportamento das estratégias do tipo mestre-escravo na renderização da imagem *room* no ambiente CC-LL.LH.HL.HH (gráfico da Figura 5.21), embora sem grandes variações, destacou um pior desempenho das estratégias MEBF1 e MEBF8 nas execuções com 4, 8 e 16 processadores. No caso da estratégia MEBF8 o mau desempenho pode ser explicado novamente pelo tamanho inadequado do bloco de tarefas, o que ocasionou altos desbalanceamentos (IDCs de 4,11% para 4 processadores, 11,68% para 8 e 23,79% para 16). Já a MEBF1 também foi prejudicada pela heterogeneidade das tarefas e pela característica diversificada das cargas externas deste ambiente. O baixo tempo de renderização desta imagem pode ter contribuído para evidenciar o *overhead* de comunicação. Embora as tarefas nesta aplicação possuam um tamanho pequeno (2 bytes) e os canais de comunicação sejam bem rápidos, as 500 requisições de tarefas feitas pelos escravos ao mestre podem ter embutido um pequeno *overhead.*

A diversificação das cargas externas neste ambiente (CC-LL.LH.HL.HH) também pode ter prejudicado o desempenho da estratégia EstNWS. Percebe-se, para ambas imagens, uma diminuição na diferença dos tempos desta estratégia e da Est quando comparadas com as execuções no ambiente CC.0.1.2.3.

O desempenho de todas as estratégias no ambiente CC-LL.LL.HL.HL (gráficos das Figuras 5.22 e 5.23) foi bem semelhante ao apresentado no ambiente CC-LL.LH.HL.HH para as duas imagens renderizadas. As principais observações na renderização da imagem *5balls* no ambiente CC-LL.LL.HL.HL em relação ao CC-LL.LH.HL.HH foram: (a) a estratégia EstNWS apresentou um desempenho pior do que a Est nas execuções com 16 processadores, com uma diferença de 14,07% no tempo total, (b) embora a estratégia MERB1 tenha repetido o mau desempenho, foi melhor do que no ambiente CC-LL.LH.HL.HH. Isto possivelmente ocorreu porque as cargas no ambiente CC-LL.LL.HL.HL variam menos, (c) com 4 processadores, a estratégia MERB6 não se mostrou eficiente, o que não foi observado com 8 e 16 processadores. As principais observações na renderização da imagem *room* no ambiente CC-LL.LL.HL.HL em relação ao CC-LL.LH.HL.HH foram: (a) pequeno aumento na diferença dos tempos das estratégias Est e EstNWS, possivelmente devido a uma menor variação das cargas no ambiente CC-LL.LL.HL.HL (uma exceção pode ser verificada na execução da estratégia EstNWS com 4 processadores, onde a Est foi 2,1% melhor que a EstNWS), (b) a estratégia MERB5 não apresentou um bom desempenho na execução com 16 processadores. Este resultado pode ser explicado pelo *overhead* gasto na coleta de medições NWS a cada recálculo de peso. Percebe-se que a execução desta mesma estratégia para a imagem *5balls* foi ligeiramente menos eficiente do que as melhores estratégias. No entanto, esse *overhead* gerou um impacto maior na imagem *room*, pois o tempo total para renderizá-la é bem menor (praticamente a metade) do que para a *5balls*.

A utilização de cargas com maior variação no ambiente CC-LH.LH.HH.HH (gráficos das Figuras 5.24 e 5.25) possibilitou uma maior variação também no desempenho das estratégias avaliadas, mais notadamente na renderização da imagem *5balls*. Com esta imagem, novamente as estratégias do tipo mestre-escravo obtiveram o melhor desempenho. No entanto, quatro estratégias não apresentaram um bom desempenho: MEBF8 (apenas com 8 e 16 processadores), MERB1, MERB2 e MERB3. A estratégia MERB1 confirmou a ineficiência na sua forma de distribuição das tarefas, já a MEBF8 foi prejudicada principalmente pela maior variação das cargas, o que acentuou o desbalanceamento (IDC de 6,95% com 8 processadores e de 22,22% com 16 processadores). A estratégia MERB2, que é uma variação da MERB1, mostrou sensibilidade às grandes variações das cargas externas neste ambiente. Este mesmo fator prejudicou a estratégia

MERB3, que calcula os pesos dos processadores escravos apenas no início da execução da aplicação (o peso é o *clock* da CPU), não prevendo possíveis alterações no ambiente e, conseqüentemente, distribui as tarefas indevidamente. Nesta estratégia, foram verificadas pequenas diferenças entre o seu tempo e os das melhores estratégias MERB (14,9% pior com 4 processadores, 2,58% com 8 e 12,4% com 16). Embora a estratégia MERB4 também calcule os pesos apenas no início da execução, o uso da disponibilidade de CPU, fornecida pelo NWS, favoreceu esta estratégia.

No ambiente CC-LH.LH.HH.HH as estratégias, com ambas imagens, apresentaram um comportamento interessante, também devido à maior variação das cargas neste ambiente. Percebe-se, nas execuções com 8 e 16 processadores, uma perda de escalabilidade nas execuções das estratégias estáticas, pois, mesmo com o dobro de processadores, os tempos totais de execução nesta duas configurações foram bem mais próximos. Além disso, pode-se notar que a estratégia EstNWS obteve um desempenho pior na renderização da imagem *room*, principalmente com 4 processadores, onde obteve o pior desempenho dentre todas as estratégias. Outra observação pode ser feita quanto ao desempenho das estratégias MERB que obtiveram um excelente desempenho para a imagem *room* quando comparadas com as MEBF.

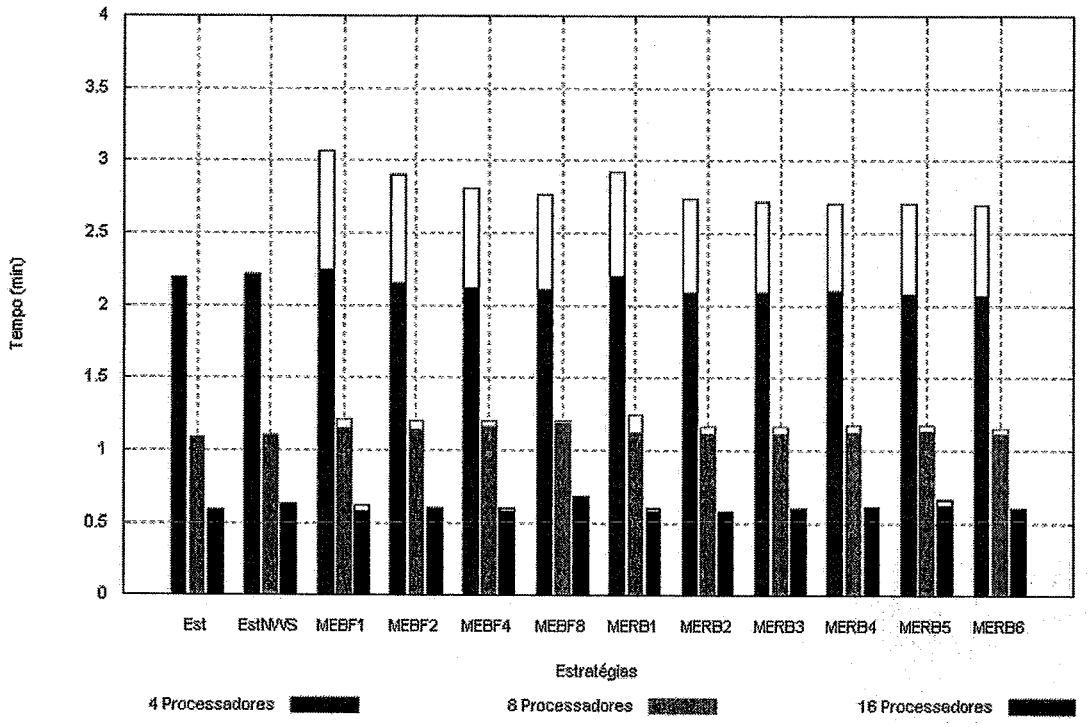

Figura 5.16: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente SC) com a imagem *5balls*

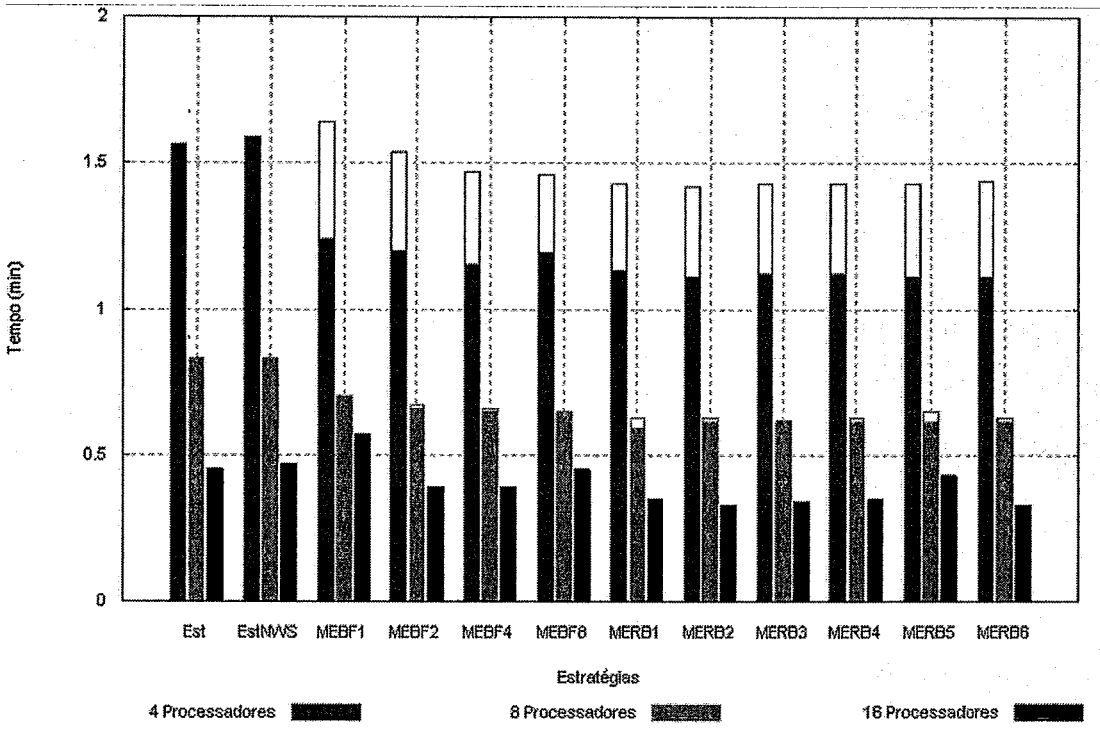

Figura 5.17: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente SC) com a imagem *room*

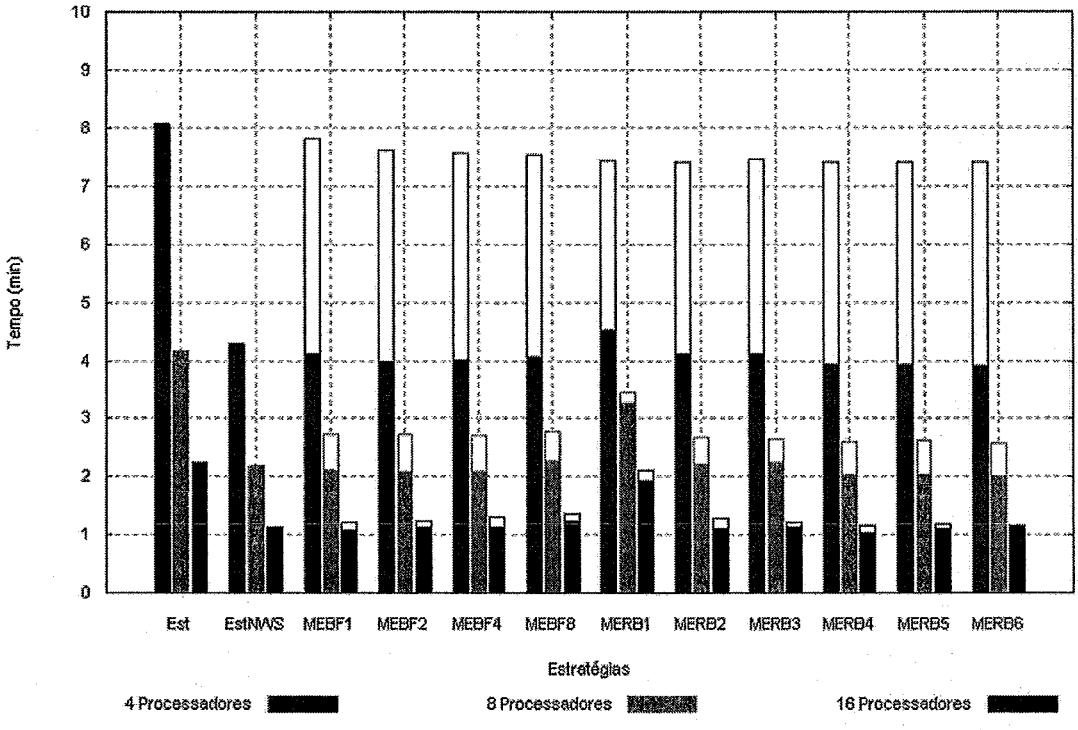

Figura 5.18: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-0.1.2.3) com a imagem *5balls*

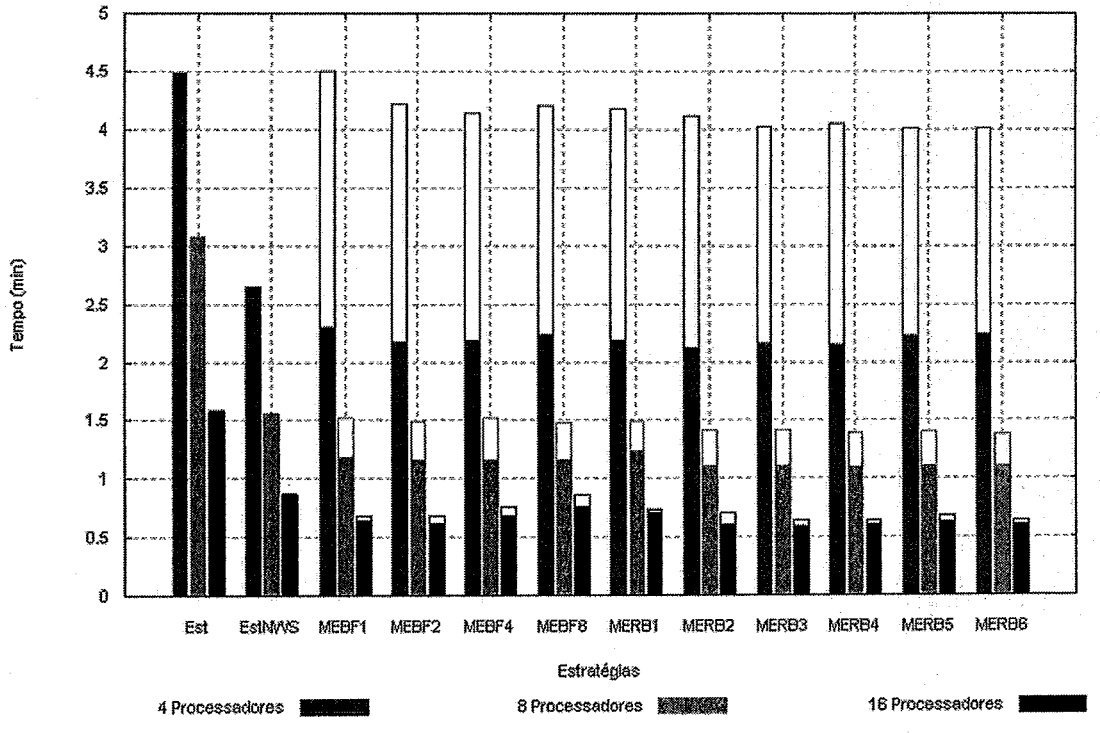

Figura 5.19: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-0.1.2.3) com a imagem *room*

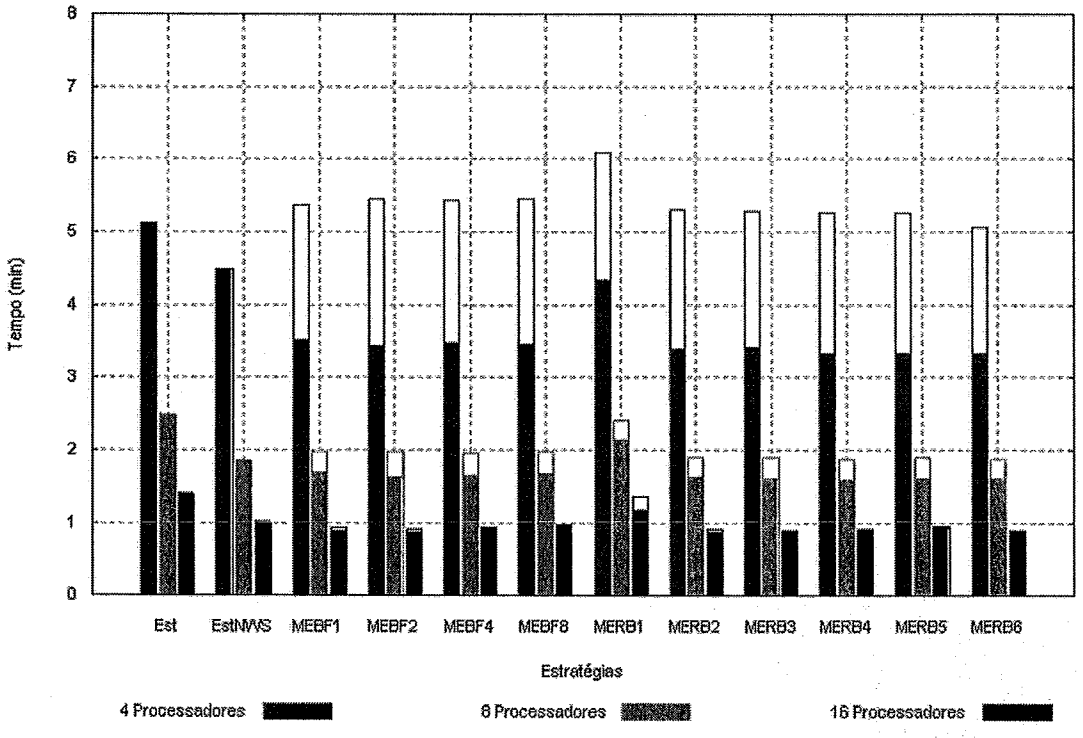

Figura 5.20: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-LL.LH.HL.HH) com a imagem *5balls*

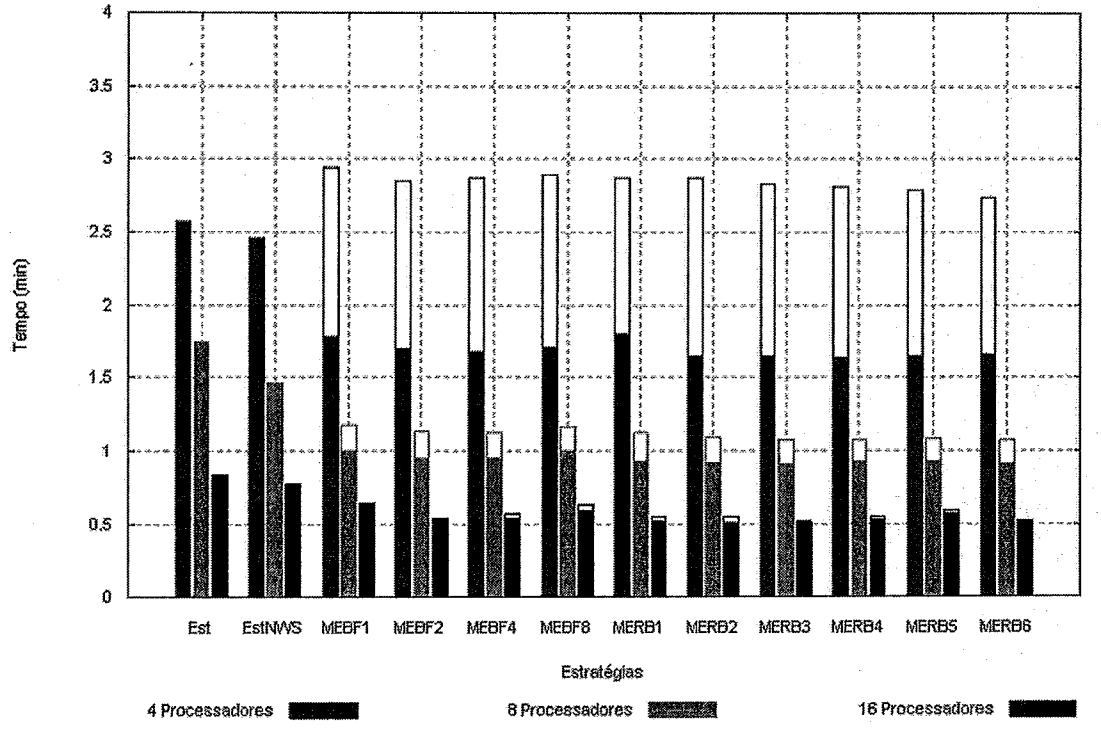

Figura 5.21: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-LL.LH.HL.HH) com a imagem *room*

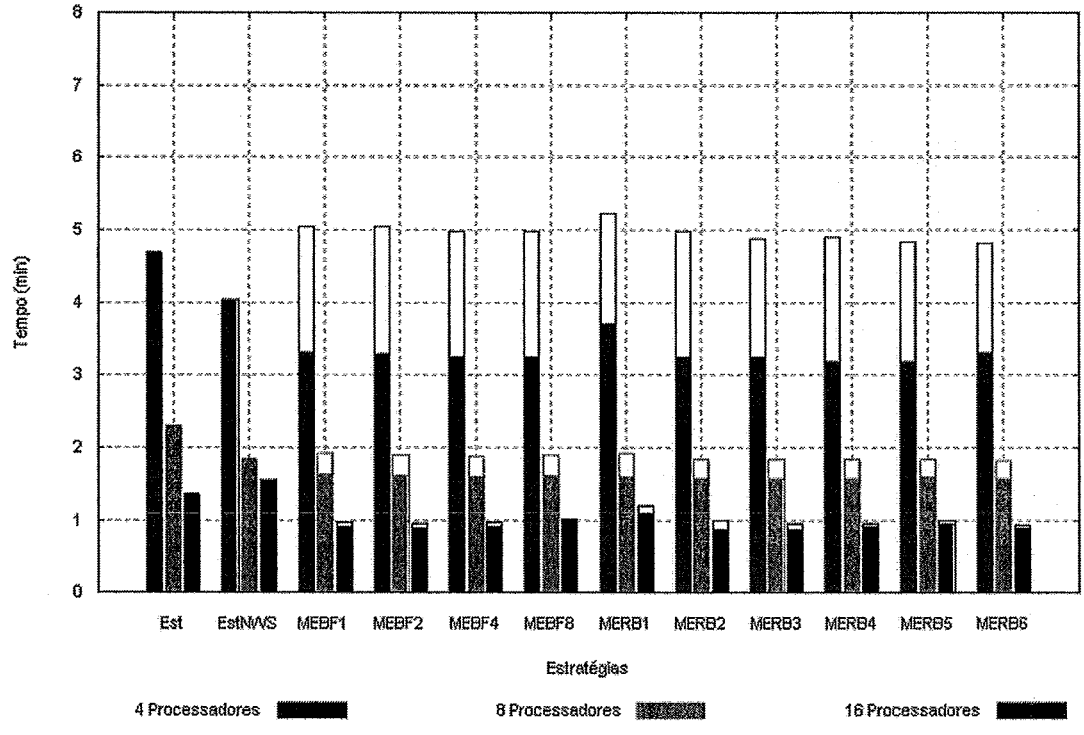

Figura 5.22: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-LL.LL.HL.HL) com a imagem *5balls*

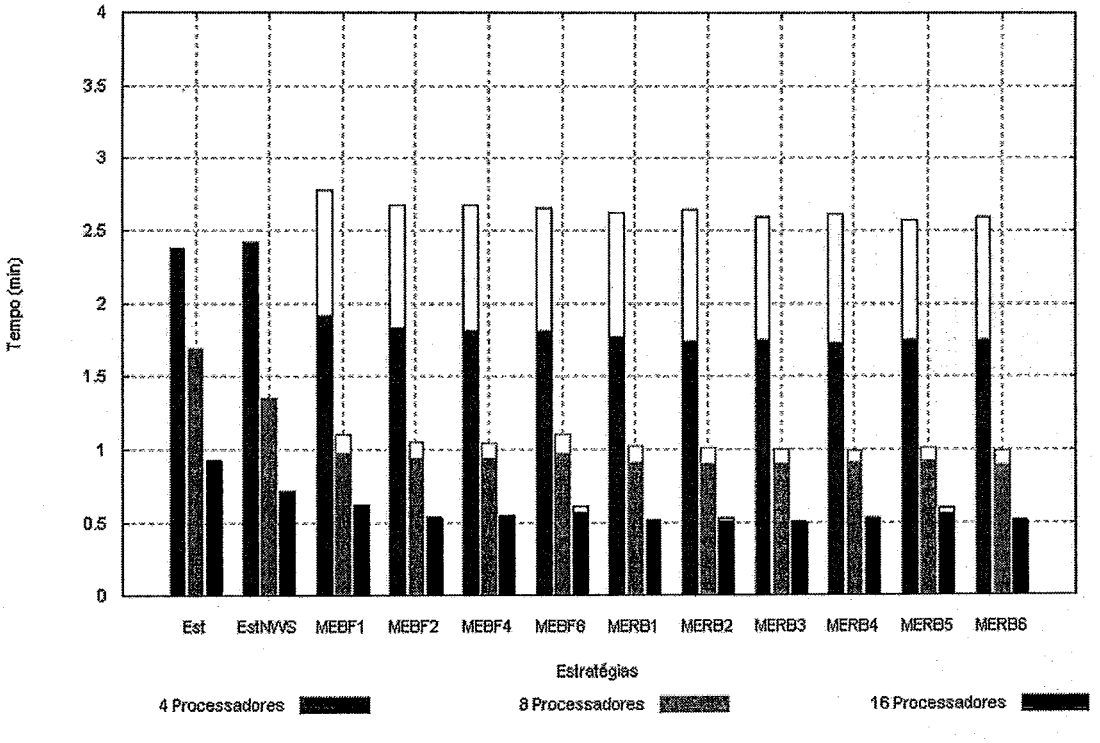

Figura 5.23: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-LL.LL.HL.HL) com a imagem *room*

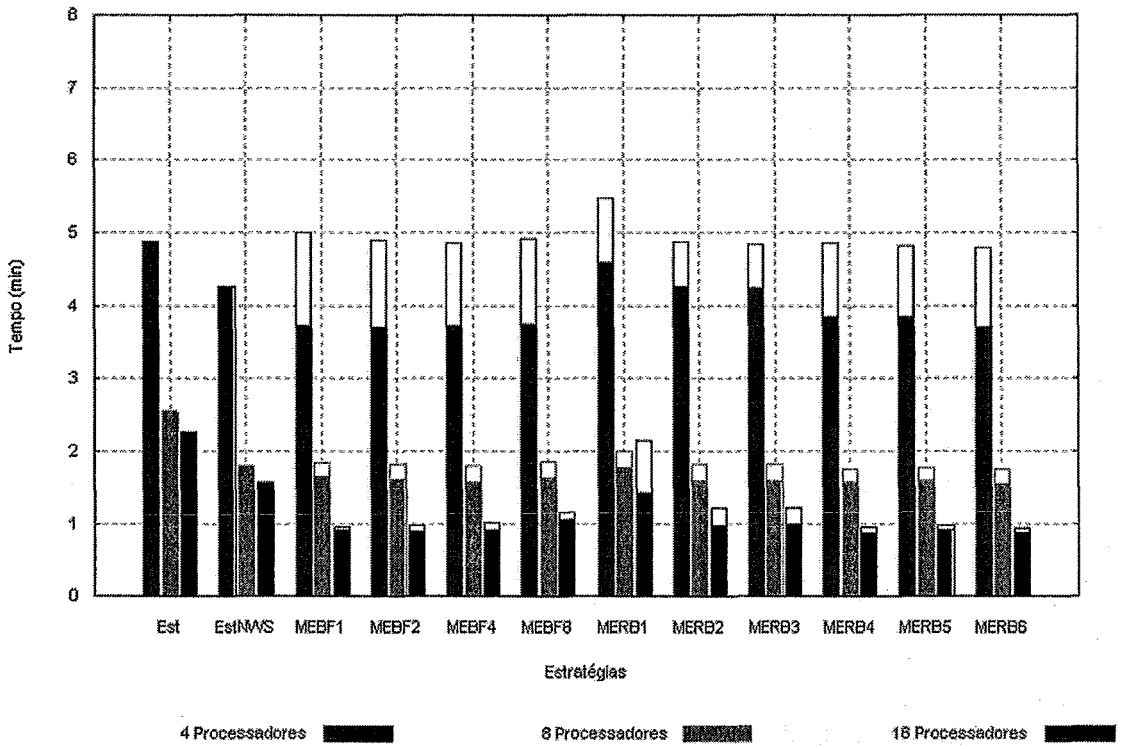

Figura 5.24: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-LH.LH.HH.HH) com a imagem *5balls*

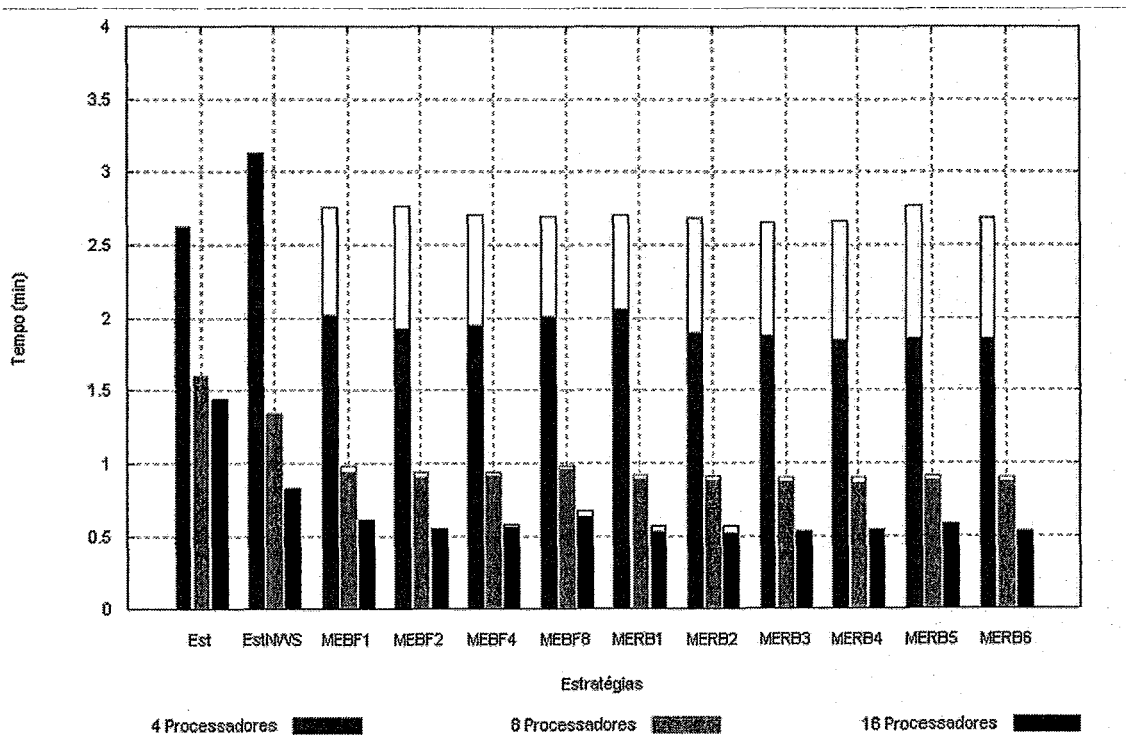

Figura 5.25: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* executada no *cluster* (ambiente CC-LH.LH.HH.HH) com a imagem *room*

5.3 Experimentos Realizados no *Grid*

Grids computacionais são sistemas que acoplam recursos heterogêneos distribuídos, oferecendo acesso consistente aos recursos, independentemente de sua localização geográfica. A tecnologia de computação em *grid* visa agregar recursos computacionais variados e dispersos em um único "supercomputador virtual", acelerando assim a execução de aplicações paralelas. *Grids* se tornaram possíveis nos últimos anos, devido à grande melhoria de desempenho e redução de custo, tanto de redes de computadores quanto de microprocessadores [23].

Vários tipos de aplicações são indicadas para serem executadas em um *grid*, como por exemplo: aplicações que demandam um alto custo computacional, aplicações colaborativas, onde o *grid* viabiliza a comunicação entre pessoas, e aplicações intensivas de dados, nas quais o foco é o acoplamento de recursos de dados distribuídos visando o acesso a grandes bases de dados [8, 15].

Além dos fatores de desequilíbrio de carga observados em um *cluster*, tais como, heterogeneidade dos processadores e a presença de carga externa, o fator comunicação pode ser relevante devido à maior heterogeneidade dos canais de comunicação. As constantes trocas de mensagens através de canais de comunicação mais lentos podem prejudicar o desempenho de uma aplicação devido aos atrasos nos envios e recepções de dados.

Nesta seção, serão apresentados os resultados e as análises sobre os experimentos realizados em um *grid* computacional. As estratégias implementadas foram executadas em um *grid* formado por dois *clusters* de duas universidades do Estado do Rio de Janeiro, assim identificados: *cluster*PUC (*cluster* localizado na Pontifícia Universidade Católica do Rio de Janeiro) e *cluster*UFF (*cluster* localizado na Universidade Federal Fluminense). O *grid* foi formado por 30 máquinas, sendo 15 no *cluster*PUC com processadores Pentim II (398 MHz) e 15 no *cluster*UFF com processadores Pentium IV (2594 MHz). As máquinas dentro de cada *cluster* são conectadas por uma rede *Fast Ethernet* e os dois *clusters* são interconectados via internet. Para possibilitar a comunicação entre os processadores dos dois *clusters* foi utilizado o *middleware* Globus Toolkit [44] (versão 2.4 nas máquinas da UFF e versão 2.2 nas máquinas da PUC) e a biblioteca de troca de mensagens mpilam [30, 43] (versão 7.0.6 em ambos os *clusters*).

Além disso, foi utilizada a mesma versão do NWS dos experimentos realizados no *cluster* (versão 2.12.2).

Além das estratégias avaliadas no *cluster* foram executadas duas outras no *grid*. A primeira trata-se de uma versão do *Weighted Factoring with Dynamic Feedback* (MERB6) que também utiliza a latência de rede (monitorada pelo NWS) como parâmetro auxiliar na definição do peso de um processador e a segunda é uma estratégia mestre-escravo hierárquica, ambas foram apresentadas no Capítulo 3. A primeira será identificada por MERB6b e a segunda, por MEH. Estas duas estratégias foram implementadas com o objetivo de proporcionar um balanceamento de carga eficiente em *grids* computacionais e por este motivo fizeram parte desses experimentos. Vale lembrar que a estratégia MERB6, que não utiliza a latência de rede e que foi avaliada nos experimentos realizados no *cluster*, será identificada nesta seção como MERB6a.

A avaliação das estratégias implementadas no *grid* utilizou, além das duas aplicações testadas no *cluster* (*termions* e *Ray Tracing*), uma aplicação sintética onde podem ser configurados os seguintes parâmetros: número total de tarefas, tempo de computação de cada tarefa (em segundos) e o tamanho da tarefa (ou seja, quantidade de bytes associados a uma tarefa). A aplicação sintética foi utilizada com o objetivo de permitir uma análise do desempenho das estratégias de balanceamento de carga quando executadas em aplicações com um maior número de tarefas e com tarefas de tamanho maior, já que na aplicação dos *termions* cada tarefa tem um tamanho de 20 bytes e na aplicação *Ray Tracing* apenas 4 bytes. Cada tarefa nesta aplicação consiste na execução de um *loop*, onde em cada iteração deste é executada uma operação aritmética bem simples (esta operação é feita apenas com número inteiros).

Diferentemente dos experimentos realizados no *cluster*, onde variou-se o número de tarefas na aplicação dos *termions* (250, 500 e 1000 *termions*) e onde foram renderizadas duas imagens distintas com o *Ray Tracing* (*5balls* e *room*), no *grid* a primeira aplicação citada foi executada com 2000 *termions* e a segunda apenas para a imagem *5balls* (novamente com tamanho 500 $\times$ 500 *pixels*). O aumento do número dos *termions* se deve ao fato de que o tempo total de computação seria muito baixo com poucas tarefas e 30 processadores (número total de processadores do *grid*), o que poderia prejudicar a análise. Na aplicação *Ray Tracing* a imagem *5balls* foi escolhida por necessitar de um tempo maior para ser renderizada. Além disso, como não foi possível aumentar o

número de *pixels* da imagem, aumentou-se a ordem de cada *pixel* de 4×4 para 8×8, aumentando assim o tempo de computação de cada um deles.

A aplicação sintética foi sempre executada com 20000 tarefas de tamanho 1kb e o tempo de computação de cada tarefa foi configurado para durar aproximadamente 2s no processador mais lento do *grid*, ou seja, nos processadores do *cluster*PUC, que são Pentium II (398MHz).

Conforme foi explicado no Capítulo 4, a estratégia MEH necessita de uma topologia de processadores como parâmetro de entrada. Em uma topologia, deve existir um único mestre global (que contém o conjunto inicial de tarefas e as distribui para escravos ou submestres), submestres (que distribuem tarefas para seus escravos ou para outros submestres e se reportam ao mestre global ou a outros submestres para solicitar tarefas) e escravos (que executam tarefas e se reportam ao mestre global ou a um submestre para solicitar tarefas para execução).

Os experimentos realizados no *grid* utilizaram uma topologia de apenas três níveis, onde o mestre global está localizado no *cluster* que contém o conjunto inicial de tarefas e todos o demais processadores deste *cluster* são escravos que se reportam diretamente ao mestre global para solicitar tarefas. No outro *cluster* existe um submestre que se reporta ao mestre global para solicitar um subconjunto de tarefas e as distribui para os seus escravos, localizados no mesmo *cluster*. O tamanho do bloco de tarefas que o submestre solicita para o mestre global ficou assim definido: 45 tarefas na aplicação dos *termions* (ou 3 tarefas para cada escravo), 30 tarefas no *Ray Tracing* (ou 2 tarefas para cada escravo) e 105 tarefas na aplicação sintética (ou 7 tarefas para cada escravo).

Testes prévios, realizados com as estratégias MERB4 e MERB5 no *grid* formado pelos 30 processadores (15 em cada *cluster*), detectaram um grande *overhead* embutido no recálculo de pesos quando são utilizadas medições NWS. Por este motivo, o parâmetro R, que determina o número de recálculos, foi definido de forma que fossem realizados apenas dois recálculos, um no início da execução e ou outro na metade. Essa medida foi tomada, pois o alto *overhead* apresentado poderia prejudicar a análise gráfica. Ainda assim, o *overhead* pôde ser notado, como será verificado na análise experimental.

Os experimentos foram realizados com duas configurações diferentes de rede no *grid*. Na primeira, a comunicação inter-*cluster*, isto é, entre máquinas localizadas em

cluster distintos, utilizou a latência real de comunicação entre os *clusters* (entre 1ms e 4ms). A outra configuração utilizou a ferramenta TC (*Traffic Control*), apresentada no Capítulo 4, para emular uma rede com canais de comunicação mais lentos entre os *clusters*. Para isto, o TC foi configurado de modo que a latência inter-*cluster* fosse de 250 ms. Este valor foi utilizado com base nas latências observadas entre as máquinas das duas universidades (UFF e PUC-Rio) e máquinas de universidades dos Estados Unidos da América (como por exemplo: Universidade de San Diego, Universidade da Califórnia e Universidade de Berkeley). Com a utilização do comando *ping* do Linux (com envio de pacotes de 64 bytes) foram observados valores médios de latência de 250ms. Em ambas as configurações, a latência de rede intra-*cluster* foi a real.

Primeiramente, as aplicações foram executadas com o conjunto inicial de tarefas localizado no *cluster*UFF e posteriormente no *cluster*PUC. Nas estratégias do tipo mestre-escravo, o processo mestre foi executado na máquina do *cluster* que possui o conjunto inicial de tarefas (na estratégia MEH o mestre global foi executado nesta máquina e o submestre em uma máquina do outro *cluster*). O conjunto de estratégias foi executado cinco vezes e os tempos fornecidos são uma média aritmética desses tempos. Nas Tabelas A.13 e A.14, localizadas no apêndice A, podem ser conferidos para cada estratégia: o tempo de execução, o IDC, o desvio padrão das execuções e o número de tarefas executadas por cada cluster.

Nos gráficos das Figuras 5.26 e 5.27, são mostrados os resultados obtidos com a aplicação dos *termions*. Os gráficos da Figuras 5.28 e 5.29, exibem o resultados da aplicação *Ray Tracing*. Por fim, nos gráficos das Figuras 5.30 e 5.31, podem ser observados os resultados obtidos com a aplicação sintética. Para cada aplicação são exibidos dois gráficos: o primeiro contém os resultados obtidos quando as tarefas estão localizadas no *cluster*UFF e o segundo quando estão no *cluster*PUC.

É importante ressaltar que não foram exibidos nos gráficos os resultados das estratégias Est, MERB1 e MERB2, pois estas estratégias apresentaram um desempenho muito ruim e prejudicariam a análise das demais estratégias por alterarem significativamente a escala dos gráficos. Também não foram exibidas as execuções SMT (sem o mestre trabalhando) das estratégias do tipo mestre-escravo porque, mesmo com uma diferença bem pequena, as execuções CMT obtiveram, na maioria dos casos, um desempenho melhor. Todos os resultados, incluindo estes que não foram exibidos nos

gráficos, podem ser conferidos nas Tabelas A.13 e A.14, no apêndice A.

Uma avaliação mais geral pode ser feita sobre as três aplicações executadas. Observa-se nos gráficos (exibidos no final desta subseção), uma grande diferença no comportamento entre as execuções com a latência inter-*cluster* real (entre 1 ms e 4 ms) e 250 ms quando se modificou a localização do conjunto inicial de tarefas.

Inicialmente, todas as aplicações foram executadas com o conjunto inicial de tarefas localizado no *cluster*UFF. Após a coleta dos resultados, observou-se que as estratégias não apresentaram grande diferença de desempenho quando a latência inter-*cluster* passou da real para 250 ms. Por este motivo, os mesmos experimentos foram realizados com o conjunto inicial de tarefas localizado no *cluster*PUC, que possui uma baixa capacidade de processamento (cerca de 6,5 vezes mais lento do que o *cluster*UFF). Os resultados obtidos mostraram grande diferença na execução de uma estratégia com essas duas configurações de latência, principalmente nas aplicações *Ray Tracing* e Sintética.

Os resultados obtidos com a aplicação dos *termions*, quando o conjunto inicial de tarefas estava no *cluster*PUC e a latência inter-*cluster* foi de 250 ms (gráfico da Figura 5.27) possibilitam as seguintes considerações: (a) apenas nas estratégias MEBF1 e MEBF8, o tempo de execução foi maior do que o obtido no *grid* com latência inter-*cluster* real, o que era o esperado. Na MEBF1 o motivo foi o alto *overhead* de comunicação gerado pelo envio unitário de tarefas para as máquinas do *cluster*UFF através de um canal lento e, na MEBF8, pelo alto índice de desbalanceamento (alto IDC – 26,7%) gerado por uma máquina do *cluster*PUC que, possivelmente, tenha recebido o último bloco de tarefas; (b) nas demais estratégias, percebe-se que, mesmo com o atraso no envio de tarefas para o *cluster*UFF, forçando assim o *cluster*PUC a executar mais tarefas, os experimentos no *grid* com a latência inter-*cluster* de 250 ms obtiveram um desempenho melhor (os ganhos foram em média de 7% com percentuais variando entre 0,6% e 15%). O motivo deste mau desempenho com a latência mais baixa pode ser explicado pelo baixo potencial de processamento do mestre (localizado no *cluster*PUC) e, conseqüentemente, pela sua capacidade em atender prontamente os seus escravos. Nota-se a ocorrência de um "gargalo" no mestre, o que acabou sub-utilizando as máquinas do *cluster*PUC, uma vez que o número excessivo de requisições feitas por máquinas do *cluster*UFF, que executam tarefas mais rapidamente, atrasaram o envio de tarefas localmente. Para comprovar esta afirmação, basta observar o número

de tarefas executadas pelo *cluster*PUC na Tabela A.12. O *cluster*PUC executou, em média, 57% mais tarefas quando o mestre estava localizado no *cluster*UFF (gráfico da Figura 5.26) e a latência inter-*cluster* era real. Essa diferença caiu para 3,7% quando esta latência foi de 250 ms. Conclui-se, então, que o *overhead* de envio de tarefas para o *cluster*UFF acabou amenizando o "gargalo" no mestre (localizado no *cluster*PUC), o qual foi capaz de atender mais prontamente as requisições dos escravos locais, elevando assim a capacidade de processamento do *cluster*PUC.

Este comportamento também foi verificado na aplicação *Ray Tracing* e pode ser explicado da mesma forma. No entanto, todas as estratégias com o mestre no *cluster*PUC e com latência inter-*cluster* de 250 ms apresentaram um desempenho melhor do que o observado na aplicação dos *termions* (em média 26,61% mais rápido do que as execuções no *grid* com latência real, com percentuais variando entre 8% e 54,2%). Nas execuções no *grid* com latência inter-*cluster* real e com o mestre localizado no *cluster*UFF, o *cluster*PUC executou, em média, 37,8% mais tarefas do que quando o mestre estava no próprio *cluster* (vide os gráficos das Figuras 5.28 e 5.29 e a Tabela A.13). As mesmas execuções no *grid* com latência de 250 ms mostraram uma diferença de apenas 1,7%. Novamente, se observou um "gargalo" no mestre quando este estava localizado no *cluster*PUC e que este "gargalo" foi atenuado quando a latência inter-*cluster* foi maior (250 ms).

Na aplicação Sintética, o comportamento foi diferente (vide os gráficos das Figuras 5.30 e 5.31 e a Tabela A.14), pois quando o mestre estava localizado no *cluster*PUC o desempenho foi, na maioria da vezes, melhor no *grid* com latência real (com exceção das estratégias EstNWS, MERB4 e MERB5). Os ganhos foram, em média, de 25,45% com percentuais variando entre 2,2% e 48%. Neste caso, o maior número de tarefas e o maior tamanho delas (1kb) foi determinante para este comportamento. O envio de tarefas de tamanho maior para as máquinas do *cluster*UFF embutiram um maior *overhead* no tempo total de computação e foram mais significativos do que um possível "gargalo" no mestre. De fato não foram observadas grandes diferenças no total de tarefas executadas pelo *cluster*PUC nas quatro variações de execuções (latência inter-*cluster* real – com mestre no *cluster*UFF ou na *cluster*PUC e latência inter-*cluster* de 250 ms – com mestre no *cluster*UFF ou no *cluster*PUC).

O desempenho das estratégias de balanceamento de carga será agora avaliado para

as três aplicações executadas. A partir de uma análise dos resultados obtidos com a aplicação dos *termions* pode-se fazer as seguintes considerações sobre as estratégias de balanceamento de carga executadas (vide os gráficos das Figuras 5.26 e 5.27 e a Tabela A.12):

- Quando o conjunto inicial de tarefas estava localizado no *cluster*PUC, apenas a estratégia MEH obteve tempos de execução menores do que quando as tarefas estavam no *cluster*UFF. Todas as demais estratégias obtiveram tempos maiores e o *overhead* de comunicação também foi um fator determinante para esses resultados.

- A estratégia EstNWS apresentou o pior desempenho dentre as demais estratégias devido ao *overhead* embutido pelo NWS no momento da recuperação das previsões de disponibilidade de CPU. Este *overhead* acabou ocasionado grandes desbalanceamentos (IDC nas execuções com o conjunto inicial de tarefas localizado no *cluster*UFF: 34,12% no *grid* com latência inter-*cluster* real e 25,82% com latência de 250 ms; e IDC nas execuções com o conjunto inicial de tarefas localizado no *cluster*PUC: 20,79% no *grid* com latência inter-*cluster* real e 21,83% com latência de 250 ms) .

- Com o mestre localizado no *cluster*UFF, os tempos de execução das estratégias MEBF foram aumentando à medida que o tamanho do bloco fixo também foi aumentado. Este desempenho pode ser explicado observando-se o aumento dos IDC nestas estratégias nas duas variações de latência inter-*cluster* (real e com 250 ms). Estas estratégias apresentaram os seguintes IDC: (a) MEBF1 – IDC igual a 3,14% nas execuções com latência inter-*cluster* real e 3,08% com 250 ms; (b) MEBF2 – IDC igual a 5,03% nas execuções com latência inter-*cluster* real e 6,12% com 250 ms; (c) MEBF4 – IDC igual a 8,59% nas execuções com latência inter-*cluster* real e 7,21% com 250 ms; (d) MEBF8 – IDC igual a 14,43% nas execuções com latência inter-*cluster* real e 13,52% com 250 ms. O mesmo aconteceu com o mestre localizado no *cluster*PUC, onde o aumento do tamanho do bloco fixo também gerou desbalanceamentos bastantes significativos. Os IDC das estratégias MEBF para as duas configurações de latência foram: (a) MEBF1 – IDC igual a 5,57% nas execuções com latência inter-*cluster* real e 5,57% com

250 ms; (b) MEBF2 – IDC igual a 6,88% nas execuções com latência inter-*cluster* real e 8,17% com 250 ms; (c) MEBF4 – IDC igual a 11,17% nas execuções com latência inter-*cluster* real e 13,58% com 250 ms; (d) MEBF8 – IDC igual a 12,93% nas execuções com latência inter-*cluster* real e 26,7% com 250 ms.

- As estratégias MERB4, MERB5 e MERB6b, com o mestre localizado no *cluster*UFF, não obtiveram um bom desempenho e apresentaram tempos de execução bem semelhantes. Estas três estratégias são do tipo *Weighted Factoring* e efetuam recálculo de peso cada vez que o lote vigente de tarefas fica vazio. Cada recálculo utiliza medições NWS como parâmetro auxiliar na definição do peso de cada processador. Mesmo com a limitação de dois recálculos de peso, um no início e outro no meio da execução, estas estratégias não se mostraram eficientes, permitindo-se concluir que a coleta de medições NWS gerou um *overhead* muito alto. Pelo mesmo motivo, estas três estratégias também obtiveram os piores desempenhos quando o mestre estava localizado no *cluster*PUC, com tempos de execução tão altos quanto o da estratégia EstNWS. A localização do mestre em um processador mais lento pode ter gerado um *overhead* ainda maior na coleta de medições NWS.

- Quando o mestre estava localizado no *cluster*UFF, as estratégias MEBF1, MERB3 e MERB6a obtiveram o melhor desempenho nas duas variações de latência. A MERB3, que é do tipo *Weighted Factoring*, efetua um único cálculo de peso no início da execução utilizando medições NWS. Este único cálculo parece não ter gerado *overhead* significativo. A MERB6a utiliza apenas o *feedback* de execução (envio de TMCT – Tempo Médio de Computação de uma Tarefa – pelos escravos) para recalcular os pesos, o que demonstrou eficiência. Na MEBF1, mesmo quando o canal de comunicação inter-*cluster* foi de 250 ms, o desempenho não foi prejudicado, pois a alta capacidade de processamento do *cluster*UFF fez com que suas máquinas executassem mais tarefas, diminuindo o número de envios de tarefas para o *cluster*PUC. Quando o mestre foi colocado no *cluster*PUC, pode-se notar que, a MEBF1 teve uma piora de desempenho. Este comportamento pode ser explicado por um provável "gargalo" no mestre, que não foi capaz de atender prontamente muitas requisições de blocos de tarefas unitários. Quando o mestre

estava localizado no *cluster*PUC, além das estratégias MERB3 e MERB6a, o MEH também apresentou um excelente desempenho. Ao diminuir o *overhead* de comunicação, com o envio de blocos grandes para o submestre, e sem ocasionar grandes desbalanceamentos (IDC de 4,18% e 5,52%), o MEH se mostrou bastante eficiente.

- A estratégia MEH não foi tão eficiente quando o mestre global estava localizado no *cluster*UFF. O envio do último bloco de tarefas para o submestre do *cluster*PUC gerou desbalanceamentos, deixando os escravos do mestre global ociosos, enquanto que os escravos deste submestre ficaram sobrecarregados. Nesta aplicação, isto pode ter sido mais relevante porque o tempo de computação de cada *término* é relativamente alto. Estas conclusões podem ser confirmadas através da observação dos IDC desta estratégia quando o mestre global estava no *cluster*UFF (IDC de aproximadamente 6% nas duas variações de latência).

Na renderização da imagem *5balls*, utilizando a aplicação *Ray Tracing*, podem ser feitas as seguintes observações (vide os gráficos das Figuras 5.28 e 5.29 e a Tabela A.13):

- Diferentemente do que foi observado na aplicação dos *termions*, quando o conjunto inicial de tarefas estava localizado no *cluster*UFF, a estratégia EstNWS não obteve o pior desempenho. Embora não tenha apresentado um bom desempenho, confirmado através do alto IDC (em média 30%), o EstNWS conseguiu ter um comportamento equivalente ao da estratégia MERB5 e superior ao da MEH. Quando o conjunto inicial de tarefas estava no *cluster*PUC, o desempenho do EstNWS piorou, mas foi superior ao das estratégias MERB4, MERB5 e MERB6b.

- Assim como aconteceu na aplicação dos termions, as estratégias MEBF1, MERB3 e MERB6a foram as mais eficientes quando o mestre estava localizado no *cluster*UFF. Quando o mestre estava localizado no *cluster*PUC, as estratégias MERB3, MERB6a e MEH, novamente, foram as mais eficientes e, diferente do que aconteceu na aplicação dos *termions*, a estratégia MEBF1 apresentou um desempenho eficiente e semelhante ao destas estratégias.

- A estratégia MEH repetiu o mau desempenho apresentado na aplicação do *termions* com o mestre localizado no *cluster*UFF. Da mesma forma, o envio do último bloco de tarefas para o submestre do *cluster*PUC pode ter gerado desbalanceamentos, deixando os escravos do mestre global ociosos, enquanto que os escravos deste submestre ficaram sobrecarregados.

- Independentemente da localização do mestre, as estratégias do tipo MEBF apresentaram altos índices de desbalanceamento quando se aumentou o tamanho do bloco fixo. Estes desbalanceamentos foram bem mais acentuados quando o mestre estava localizado no *cluster*PUC. Nesta configuração, estas estratégias apresentaram os seguintes IDC: (a) MEBF1 – IDC igual a 18,2% nas execuções com latência inter-*cluster* real e 15,07% com 250 ms; (b) MEBF2 – IDC igual a 31,5% nas execuções com latência inter-*cluster* real e 7,54% com 250 ms; (c) MEBF4 – IDC igual a 47,19% nas execuções com latência inter-*cluster* real e 39,53% com 250 ms; (d) MEBF8 – IDC igual a 45,59% nas execuções com latência inter-*cluster* real e 42,58% com 250 ms.

- Nesta aplicação, as estratégias MERB4, MERB5 e MERB6b também apresentaram os piores desempenhos, ainda pior do que aquele observado na aplicação dos *termions*. Os motivos do comportamento destas estratégias podem ser explicados da mesma forma que o da aplicação dos *termions* (explicação no quarto item de avaliação da aplicação do *termions*). O caráter heterogêneo das tarefas desta aplicação pode ter prejudicado ainda mais o desempenho destas estratégias quando o mestre estava localizado no *cluster*PUC.

Analisando os resultados da aplicação sintética obtêm-se as seguintes conclusões (vide os gráficos das Figuras 5.30 e 5.31 e a Tabela A.14):

- Com o conjunto inicial de tarefas localizado no *cluster*UFF, a maioria das estratégias apresentaram um desempenho bastante semelhante. A exceção foi as estratégias EstNWS e MERB5. A segunda, principalmente, na execução com latência inter-*cluster* de 250 ms.

- A execução da estratégia EstNWS com o conjunto inicial de tarefas localizado no *cluster*UFF foi 8,46% mais eficiente quando a latência inter-*cluster* foi real.

Quando o conjunto inicial de tarefas estava no *cluster*PUC, ocorreu o contrário e a execução com latência inter-*cluster* de 250 ms apresentou um ganho de 34,94%. Como nesta estratégia foram feitos apenas $P - 1$ envios de tarefas, não se pode atribuir esta mudança de comportamento ao fator comunicação. Com as tarefas no *cluster*PUC, observou-se uma melhor distribuição de tarefas entre os processadores nas execuções com latência inter-*cluster* de 250ms. Isto pode ser confirmado comparando os IDC nas duas variações de latência (IDC igual a 7,78% com latência inter-*cluster* de 250ms e 24,26% com latência real). A partir destes valores pode-se concluir que as previsões de disponibilidade de CPU, fornecida pelo NWS, foram mais eficientes na execução com latência inter-*cluster* de 250ms.

- Quando o mestre estava localizado no *cluster*PUC e a latência inter-*cluster* foi real, as estratégias MEBF apresentaram um melhor desempenho do que as estratégias MERB (exceção para a MEBF1). Já nas execuções com latência de 250 ms, ocorreu o contrário e as estratégias MERB se mostraram mais eficientes (exceção para a MERB6b). Ambas as estratégias mencionadas como exceção apresentaram um mau desempenho, independentemente da variação de latência. Na MEBF1, o mau desempenho pode ser explicado, novamente, pelo maior número de requisições de blocos de tarefas unitários. Esta maior quantidade de requisições gerou um "gargalo" no mestre, principalmente nas execuções com latência real. Embora este "gargalo" tenha sido atenuado nas execuções com latência inter-*cluster* de 250ms, o *overhead* de comunicação foi muito alto devido ao tamanho maior das tarefas nesta aplicação (1kb). Assim, o fator comunicação foi determinante para o péssimo desempenho desta estratégia nesta configuração de latência. O mau desempenho da estratégia MERB6b pode ser explicado por dois fatores: o *overhead* na coleta de medições de rede do NWS e *overhead* com os cálculos dos pesos, que envolvem o *feedback* de execução de cada escravo e as medições NWS coletadas.

- Independentemente da localização do conjunto inicial de tarefas, a estratégia MERB6a apresentou o melhor desempenho nos dois casos. A utilização de uma política de redução de bloco e a utilização de um peso determinado pelo *feedback* de execução dos escravos parecem ter sido o suficiente para diminuir o *overhead*

de comunicação, sem gerar grandes desbalanceamentos.

- A estratégia MEH obteve um excelente desempenho nas execuções quando o mestre global estava localizado no *cluster*PUC e com latência inter-*cluster* real, com desempenho semelhante ao das estratégias MEBF. No entanto, quando a latência inter-*cluster* foi de 250 ms, o desempenho foi 89,5% pior. Observando o número de tarefas executadas por cada *cluster*, nas duas variações de latência inter-*cluster*, conclui-se que o *overhead* de comunicação foi alto e bastante relevante quando a latência foi de 250 ms. Mesmo com menor capacidade de processamento, o *cluster*PUC teve que executar cerca de 82,24% mais tarefas do que quando a latência inter-*cluster* foi a real.

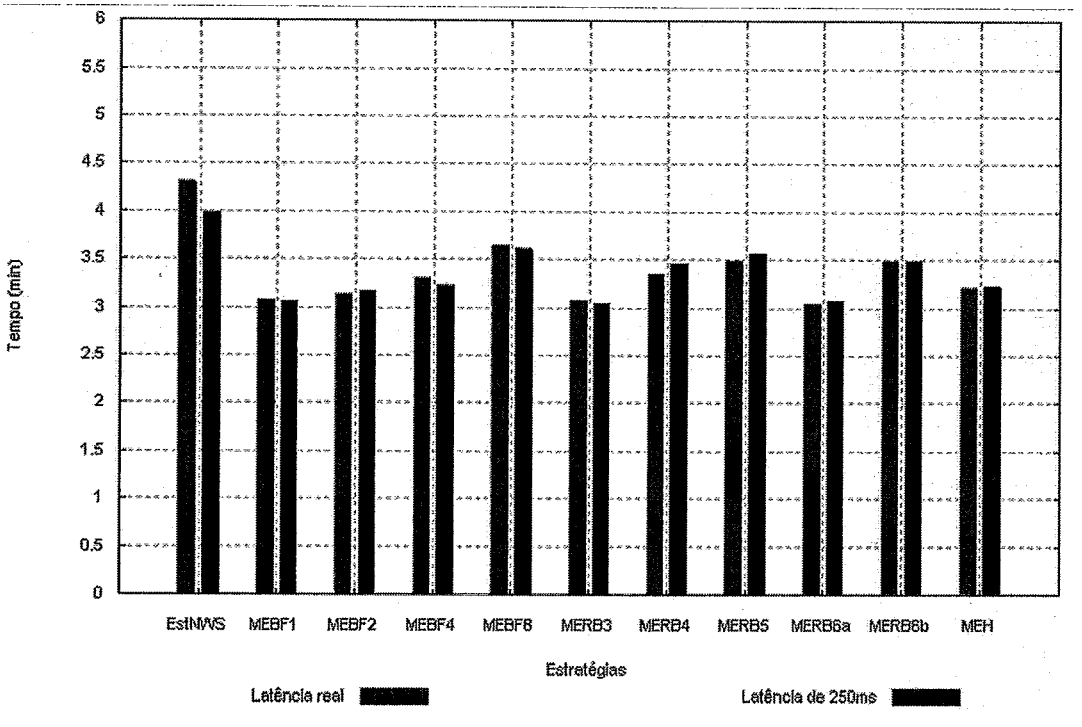

Figura 5.26: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* executada no *grid* (com 2000 *termions*) e com o conjunto inicial de tarefas localizado no *cluster*UFF

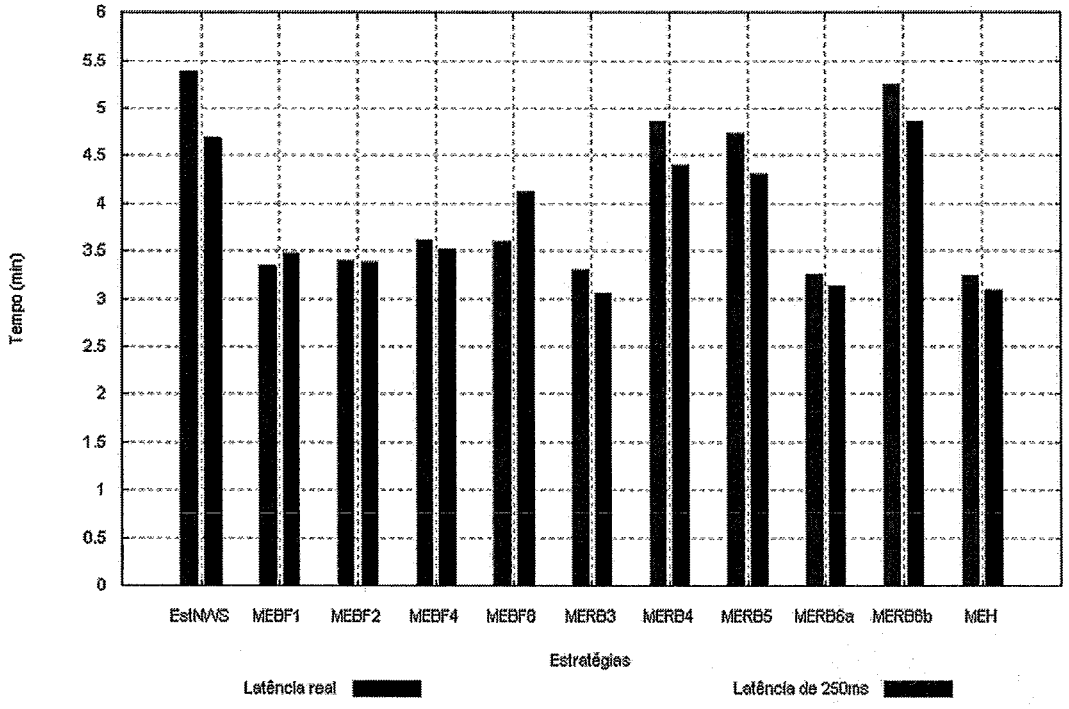

Figura 5.27: Tempos das estratégias de balanceamento de carga – Aplicação dos *termions* (com 2000 *termions*) executada no *grid* e com o conjunto inicial de tarefas localizado no *cluster*PUC

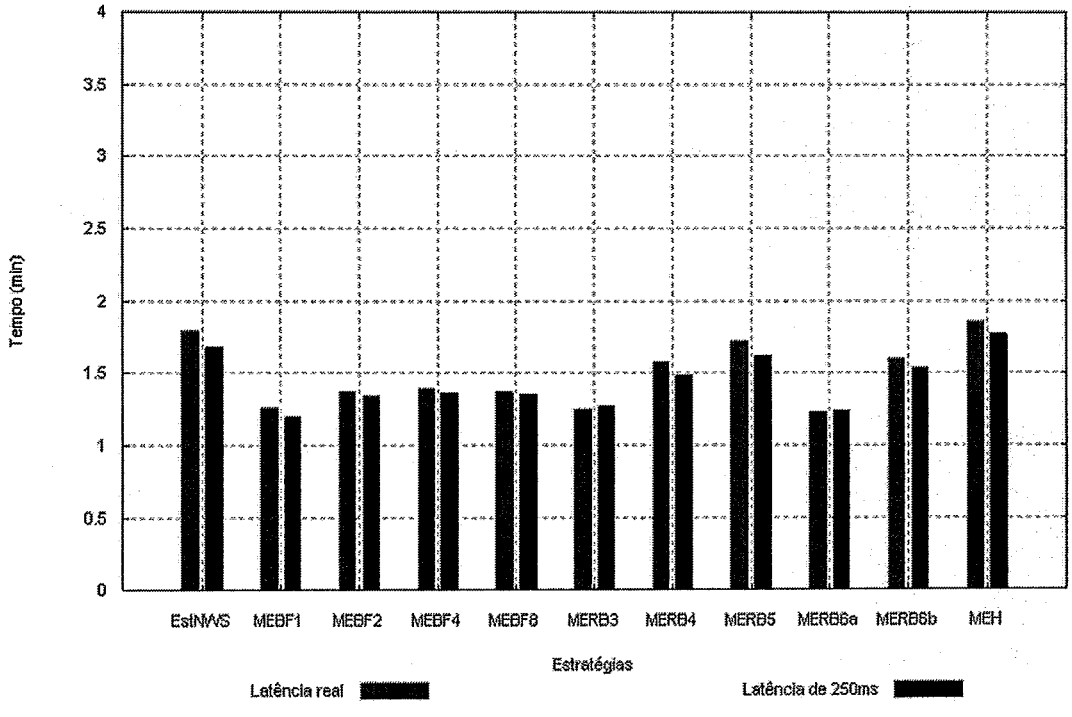

Figura 5.28: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* (com a imagem *5balls*) executada no *grid* e com o conjunto inicial de tarefas localizado no *cluster*UFF

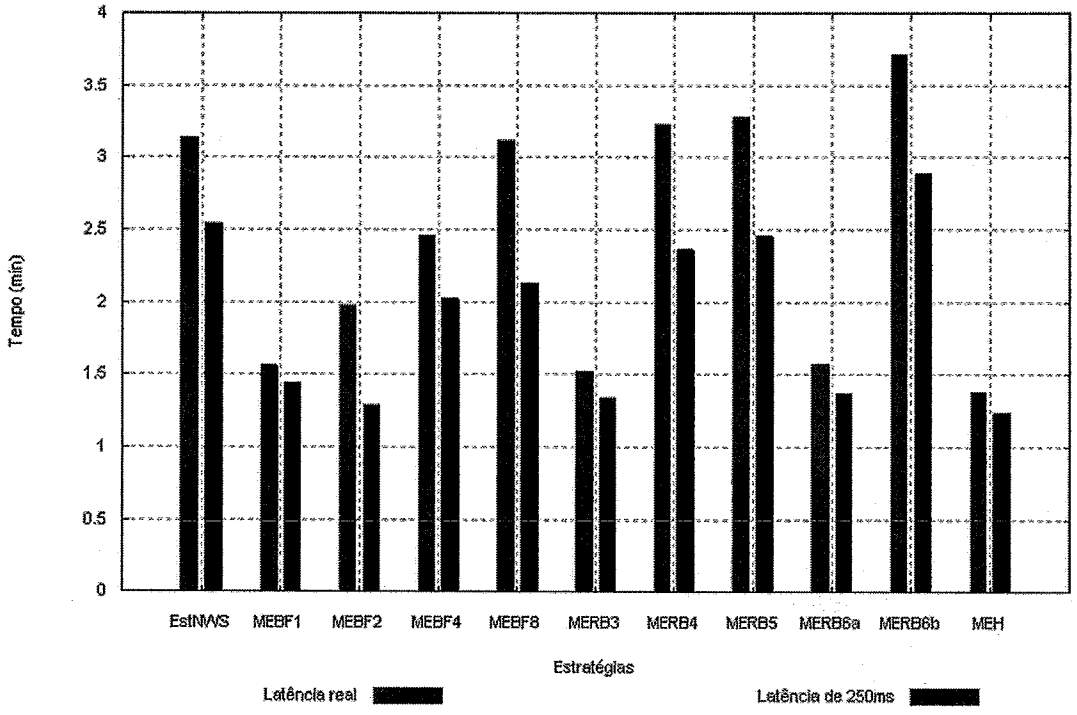

Figura 5.29: Tempos das estratégias de balanceamento de carga – Aplicação *Ray Tracing* (com a imagem *5balls*) executada no *grid* e com o conjunto inicial de tarefas localizado no *cluster*PUC

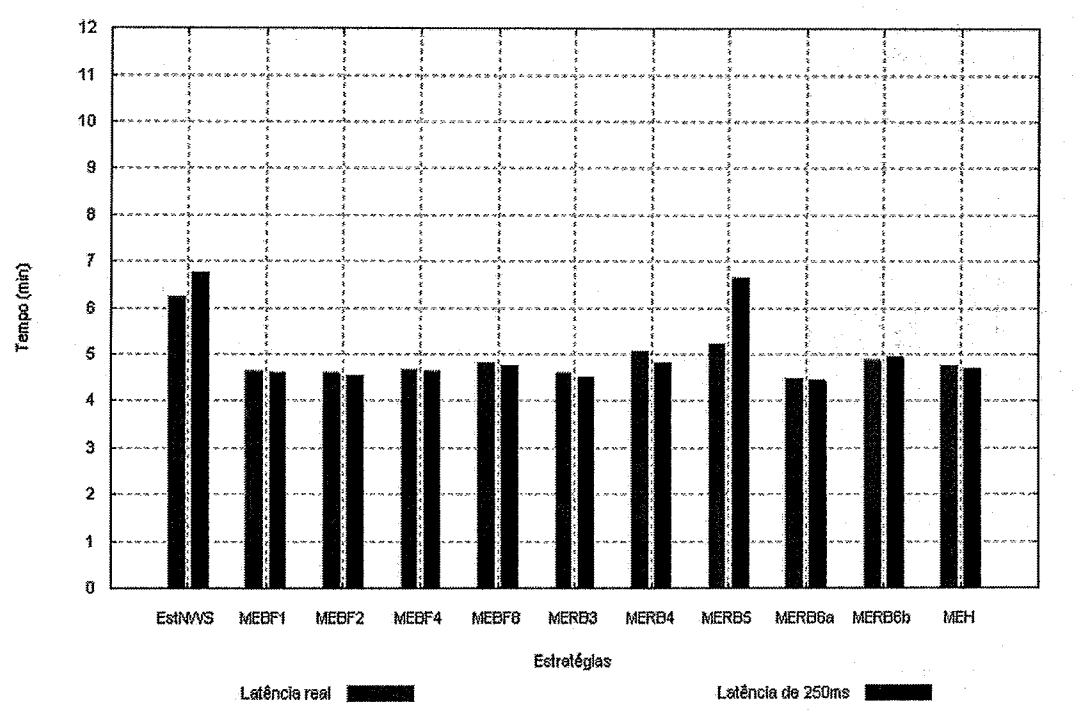

Figura 5.30: Tempos das estratégias de balanceamento de carga – Aplicação Sintética (com 20000 tarefas de tamanho 1kb) executada no *grid* e com o conjunto inicial de tarefas localizado no *cluster*UFF

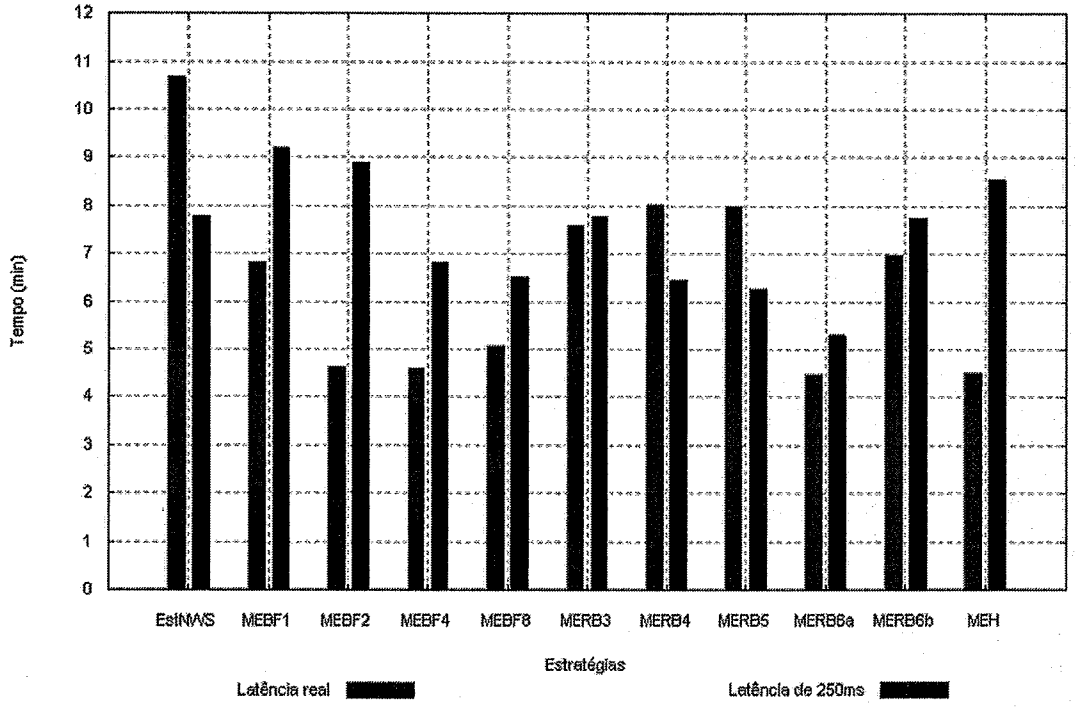

Figura 5.31: Tempos das estratégias de balanceamento de carga – Aplicação Sintética (com 20000 tarefas de tamanho 1kb) executada no *grid* e com o conjunto inicial de tarefas localizado no *cluster*PUC

Capítulo 6

Conclusões

Esta dissertação de mestrado teve por objetivo analisar o desempenho de algumas estratégias de balanceamento de carga, particularmente as que utilizam a abordagem mestre-escravo. A partir do experimentos computacionais mostrados no Capítulo 5, que foram realizados com diferentes aplicações SPMD e em ambientes paralelos bastante diversificados, foi possível chegar a diversas conclusões. Neste capítulo, serão apresentadas estas conclusões e as propostas de trabalhos futuros.

A necessidade da utilização de estratégias dinâmicas foi perfeitamente justificada após a análise dos resultados obtidos com as estratégias estáticas. O estático simples se mostrou bastante sensível aos fatores heterogeneidade de processadores e carga externa. Embora a estratégia estática com NWS tenha amenizado o impacto gerado por estes fatores, ainda apresentou altos índices de desbalanceamentos, principalmente em ambientes que apresentam variação de carga externa ou que possuam recursos computacionais com um alto grau de heterogeneidade. Nestes casos, a divisão de tarefas entre os processadores não foi adequada, pois o *clock* de CPU e a previsão de disponibilidade do NWS, calculadas apenas uma vez no início da execução, não foram suficientes para determinar a capacidade real de um processador.

A realização dos experimentos com diferentes aplicações SPMD mostrou que o desempenho de uma estratégia de balanceamento de carga também pode ser influenciado pelas características da aplicação executada. Fatores como, uniformidade do tempo de computação das tarefas, tamanho das tarefas e o número de tarefas executadas foram considerados neste trabalho e podem influenciar, significativamente, o desem-

penho das estratégias. O primeiro fator está relacionado com o caráter homogêneo ou heterogêneo de uma aplicação, o qual é definido pelo tempo de computação das suas tarefas. Quando as tarefas necessitam do mesmo tempo para serem computadas, a aplicação é dita homogênea e, caso contrário, heterogênea. A variação do tamanho das tarefas (em bytes) constatou que tarefas de tamanho grande tendem a aumentar o *overhead* de comunicação. Já a variação do número de tarefas pode amenizar ou intensificar os efeitos produzidos pelos outros dois fatores citados.

A variação dos ambientes de execução evidenciou que o desempenho de uma estratégia de balanceamento de carga está intimamente ligado às características da arquitetura paralela adotada e da sua condição no momento da execução de uma aplicação. A presença de carga de externa nos experimentos realizados no *cluster* com processadores homogêneos e a heterogeneidade dos processadores e da rede nos experimentos executados no *grid*, possibilitaram essa conclusão. As ferramentas *Playload* e TC foram fundamentais para reproduzir estas características. A primeira simulando um ambiente de concorrência pelo recurso CPU e a segunda, canais de comunicação mais lentos do que o real.

As diferentes implementações das estratégias do tipo mestre-escravo se mostraram, na maioria das vezes, alternativas interessantes de algoritmos dinâmicos de balanceamento de carga. O problema da definição do tamanho ideal do bloco de tarefas enviadas pelo mestre aos seus escravos foi constatado após a análise dos resultados computacionais. Observou-se que as estratégias com blocos de tamanho fixo (MEBF) pequenos podem ser prejudicadas pelo aumento do *overhead* de comunicação, principalmente quando as tarefas possuírem um tamanho grande. Em contrapartida, blocos de tamanho grande tenderam a prejudicar o desempenho da aplicação devido ao desbalanceamento provocado após o envio dos últimos blocos.

As estratégias com redução de bloco (MERB) implementadas obtiveram desempenhos bastante variados. As duas estratégias propostas inicialmente para sistemas de memória compartilhada, MERB1 e MERB2, não apresentaram desempenhos satisfatórios. A primeira não obteve um bom desempenho no *cluster* e no *grid*, já a segunda obteve um desempenho ruim apenas nas execuções do *grid*. A partir destas observações pode-se concluir que estas estratégias com redução de bloco, que não consideram a capacidade de cada escravo e utilizam apenas uma fórmula para definir o

tamanho do bloco, podem não ser eficientes em ambientes de memória distribuída e com alto grau de heterogeneidade. As estratégias MERB do tipo *weighted factoring* apresentaram bons desempenhos, mas o comportamento geral dessas estratégias foi bem variado. No *cluster*, esta variação pode ser observada nos ambientes de execução, onde as cargas externas variavam com maior freqüência e no, *grid*, a simples heterogeneidade dos escravos e dos canais de comunicação proporcionaram a ocorrência de diferentes resultados.

Um dos fatores que mais contribuiu para a variação dos resultados das estratégias MERB no *grid* foi o *overhead* gerado pela coleta de resultados do NWS nas estratégias que utilizam os resultados monitorados por esta ferramenta. Embora o *overhead* gerado pela monitoração dos recursos não tenha sido percebido de forma significante, foi constatado um *overhead* bem alto no momento da coleta de medições do NWS. Este foi verificado no momento que o mestre recuperava os valores de monitorações de CPU e rede. Observou-se que, quando o conjunto inicial de tarefas estava localizado no *cluster*PUC, que possui processadores mais lentos, o mestre não conseguiu capturar rapidamente todas as medições necessárias para as decisões de escalonamento. Este problema seria ainda agravado com o aumento do número de escravos, uma vez que seriam necessários mais acessos aos arquivos que armazenam as medições do NWS. Além disso, quanto mais recálculos de pesos dos escravos durante a execução da aplicação, maior o impacto no tempo total de computação.

Embora o *overhead* citado no parágrafo anterior não esteja ligado a concorrência dos processos NWS com o processo da aplicação SPMD, notou-se, na realização dos experimentos, uma certa instabilidade da ferramenta NWS durante a coleta de medições. Erros de execução, freqüentemente, ocorreram nas estratégias MERB quando eram feitos muitos recálculos de peso. Este também foi um dos motivos para a redução do número de recálculos nas estratégias deste tipo. Por este motivo, pode-se concluir que o NWS não se mostrou muito eficiente para aplicações que necessitem de atualização constante dos dados sobre a situação de um recurso monitorado.

A localização do conjunto inicial de tarefas nos experimentos realizados no *grid* e, conseqüentemente, do mestre nas estratégias do tipo mestre-escravo, mostrou ser um fator relevante para o desempenho das estratégias de balanceamento de carga. A grande diferença entre a capacidade de processamento das máquinas do *cluster*UFF e as

maquinas do *cluster*PUC, somada a variação de latência inter-*cluster*, permitiram essas conclusões. O gargalo no mestre, quando o mesmo estava localizado no *cluster* mais lento (*cluster*PUC), foi amenizado quando a latência inter-*cluster* foi mais alta, pois o tempo gasto com comunicação diminuiu a sobrecarga de processamento no mestre. Em contrapartida, quando a latência inter-*cluster* foi real, o desempenho das estratégias do tipo mestre-escravo que estabelecem comunicação excessiva com o mestre foi fortemente prejudicado.

Como propostas de trabalhos futuros podem-se apontar:

- A estratégia MERB6a, que obteve um desempenho bastante satisfatório e que utiliza o tempo médio de computação de uma tarefa (TMCT) para definir o peso um processador escravo, pode sofrer uma pequena modificação na forma como se calcula o peso de um escravo. Nesta estratégia, o peso é definido apenas pelo TMCT mais atual, não levando em consideração os tempos de comunicação que incluem o tempo de envio das tarefas e de recepção dos resultados. Já havia sido proposta neste trabalho uma variação desta estratégia onde o peso de cada escravo é a soma do seu TMCT com o valor da latência de envio das tarefas para este (a MERB6b). No entanto, a partir da análise experimental, observou-se que a coleta desta latência (fornecida pelo NWS) embutiu um *overhead* bastante significativo no tempo total de computação. A modificação proposta seria considerar o peso como a diferença entre o tempo imediatamente antes do envio de um bloco de tarefas para um escravo e o tempo em que o mestre recebeu a próxima requisição deste mesmo escravo dividido pelo número total de tarefas do bloco executado. Neste tempo estariam inclusos, além do tempo de computação, os tempos de comunicação.

- A estratégia mestre-escravo hierárquico pode ser modificada para que a distribuição de tarefas dentro de cada *cluster* seja configurável, permitindo que os mestres locais utilizem uma das estratégias do tipo mestre-escravo já mostradas neste trabalho. Sendo assim, um *cluster* poderia utilizar a estratégia MERB6a, o outro a MEBF4 e inter-*cluster* os submestres receberem tarefas do mestre global via estratégia MERB3, por exemplo.

- Uma análise mais ampla poderia ser feita utilizando-se um *grid* formado por

mais *clusters* e com latências inter-*cluster* mais variadas. Além da variação das latências, a heterogeneidade de uma arquitetura paralela como esta pode resultar em análises e conclusões ainda mais representativas.

- Todos os experimentos realizados neste trabalho utilizaram processadores do tipo Pentium. Testes inicias foram feitos em processadores do tipo AMD Atlon, onde foi constatado que este tipo de arquitetura executa operações com ponto flutuante com maior rapidez. Por este motivo, algumas das estratégias implementadas e avaliadas neste trabalho, que consideram apenas a leitura do clock de CPU (a partir do arquivo de sistema */proc/cpuinfo* do Unix), podem não ser eficientes neste tipo de arquitetura. Sendo assim, outras alternativas de estratégias de balancemanento de carga que considerem outros tipos de informação do sistema, podem ser propostas.

Referências Bibliográficas

[1] AINDA, K., FUTAKATA, Y., e HARA S., "High-Performance Parallel and Distributed Computing for the BMI Eigenvalue Problem", In: *Proceedings of the 16th International Parallel and Distributed Processing Symposium*, IPDPS, pp. 71-78, 2002.

[2] ALMESBERGER, W., *Linux Network Traffic Control: Implementation overview - Implementation Details*, Technical Report SSC / 1998 / 037, EPFL, Disponível em: ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz, Novembro/1998.

[3] ARAÚJO, R. G., GATTASS, M. e DREUX, M., "Refinamento Progressivo da Cena com Traçado de Raios Distribuído", In: *Anais do IX Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens (SIBGRAPI 96)*, pp. 1-8, Caxambu, MG, Brasil, Outubro/1996.

[4] ARGOLO, G., OLIVEIRA, A., MARTINS, S. et al. "Avaliação de Estratégias de Balanceamento de Carga do tipo Mestre-Escravo em um Grid Computacional", In: *Segundo Workshop de Grades Computacionais e Aplicações, Petrópolis. Anais do Segundo Workshop de Grades Computacionais e Aplicações*, 2004.

[5] ARGOLO, G., IGLESIAS, P., OLIVEIRA, A., et al., "Evaluating a Scientific SPMD Application on a Computational Grid with Different Load Balancing Techniques", In: *International Symposium and School on Advanced Distributed Systems*, ISSADS, Guadalarara 2005.

[6] BAL, H., BURGER, M., KIELMANN, T., "TOPOMON: A Monitoring Tool for Grid Network Topology", In: *Proceedings of the International Conference on Computational Science-Part II*, pp. 558-567, Abril/2002.

[7] BENOIT, A., COLE, M., GILMORE, S., HILLSTON, J., "Scheduling Skeleton-Based Grid Applications Using PEPA and NWS", In: *Proceedings Computer Journal*, vol. 48, pp. 369-378, Oxford University Press, Great Britain, 2005.

[8] BERMAN, F., FIGUEIRA, S., WOLSKI, R. et al. "Application Level Scheduling on Distributed Heterogeneous Networks", *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*, pp. 39, Pennsylvania.

[9] BERMAN, F., SHAO, G. e WOLSKI, R. "Master/Slave Computing on Grid", In: *Proceeding of the 9th Heterogeneous Computing Workshop*, pp. 3-16, Cancun, México, 2000.

[10] BERMAN, F., DAIL, H., CASANOVA, H., *A Modular Scheduling Approach for Grid Application Development Environments*, UCSD CSE Technical Report CS20020708, 2002.

[11] BLAHA, M., RUMBAUGH, J., PREMERLANI, W., et al. *Object Oriented Modeling and Design*, Prentice-Hall, 1991.

[12] BOERES, C. e REBELLO, V.E.F., "Projeto EasyGrid: Um Framework para a habilitação automáica de aplicações MPI em Grids Computacionais (e a Iniciativa GridRio).", In: *Anais do I Workshop em Grade Computacional e Aplicações*, Programa de verão do LNCC, Petrópolis, RJ, Brazil, 2003.

[13] BREHM J., WORLEY, P.H e MADHUKAR, M., *Performance Modeling for SPMD message passing programs, Concurrency: Practice and Experience*, vol. 10, pp. 333-357, Dez/1998.

[14] BOOCH, G e RUMBAUGH, J., *United Method*, Rational Software Corporation, Santa Clara, 1996.

[15] CAO, J., JARVIS, S.A., SPOONER, D.P., et. al., "Agent-based Grid Load Balancing using Performance-driven Task Scheduling", In: *Proceedings of 17th International Parallel and Distributed Processing Symposium* , IPDPS, pp. 49b, CD-ROM/Abstracts, 2003.

[16] CARPENTER B., FOX G., LI X., WEN Y., ZHANG G., "A High level SPMD Programming Model: HPspmd and its Java Language Binding", *Proceedings of the International Conference on Parallel ande Distributed Processing Techniques and Applications*, PDPTA'98, pp. 83-90, Las Vegas, 1998.

[17] CASAVANT, T. L. e KUHL, J. G., "A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems", In: *IEEE Transactions of Software Engineering*, Vol. 14, pp. 141-154, 1988.

[18] CHAPIN, S.J., KATRAMATOS, D., KARPOVICH, J.F., et al., "The Legion Resource Management System", In: *Proceedings of the 5 Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP'99), in conjunction with the International Parallel and Distributed Processing Symposium (IPDPS'99)*, San Juan, Porto Rico, Abril/1999.

[19] COOK, R.L., PORTER, T. e CARPENTER, L., "Distributed Ray Tracing", In: *ACM SIGGRAPH Computer Graphics*, Vol. 18(4), pp 165-174, Julho/1984.

[20] DAVID, R., GENAUD, S., GIERSCH, A., et al., "Source Code Transformations Strategies to Load-Balance Grid Applications", In: *Proceedings of the Third International Workshop on Grid Computing*, pp. 82-87, Novembro/2002.

[21] DIJKSTRA, E. W., FEIJEN, W. H. J., GASTEREN, A. J. M., "Derivation of a Termination Detection Algorithm for a Distributed Computation", *Information Processing Letters*,vol. 16, pp. 217-219, 1983.

[22] DINDA, P. e HALLARON, D. O., "Realistic CPU Workloads Through Host Load Trace Playback", In: *Proceedings of 5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers*, Lecture Notes in Computer Science, pp. 246-259, Rochester, New York, Maio/2000.

[23] FOSTER, I. e KESSELMAN C, "Globus: A metacomputing infrastructure toolkit.", In: *The International Journal of Supercomputer Applications and High Performance Computing*, Vol 11, pp. 115-128, 1997.

[24] FOSTER, I. e KESSELMAN, C., *The Grid: blueprint for a new Computing Infrastructure*, Morgan-Kaufman, 1999.

[25] FRANKLIN, M.A e GOVINDAN, V., "A General Matrix Iterative Model for Dynamic Load Balancing", *Parallel Computing*, vol. 22, pp. 969-989, 1996.

[26] FURTADO, A. N., LUQUE, E., REBOUÇAS, A. C. M., et al., "Efficient Algorithm Execution in a Collection of HNOWS", *Proceedings of the ACIS - International Conference on Computer Science, Software Engineering, Information Technology, e-Business and Applications. Michigan: International Association for Computer and Information Science*, pp. 41-47, 2002.

[27] FURUICH, M., TAKI, K., ICHIYOSHI N., "A Multi-Level Load Balancing Schema for Or-Parallel Exhaustive Search Programs on Multi-Psi", *Proceedings of the Second ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pp. 50-59, 1990.

[28] GAMMA, E., HELM, R., JOHNSON, R., et al. *Design pattern - Elements of Reusable Object Oriented Software*, Addison-Wesley, 1994.

[29] GOUX, J. P., KULKARNI, S., YODER, M. et al. "An Enabling Framework for Master-Worker Applications on the Computational Grid", *Proceedings of the Ninth IEEE International Symposium on High Performance Distributed Computing*, (HPDC'00), IEEE Computer Society Press, pp. 43-50, Pittsburgh, Pennsylvania, Aug/2000.

[30] GROPP, W., LUSK, E., SKJELLUM, A., *Using MPI: Portable Parallel Programming with the Message Passing Interface.*, 2^a edição, Cambridge, USA: MIT Press, 1994.

[31] HEYMANN, E., SENAR, M.A., LUQUE, E., et al. "Adaptive Scheduling for Master-Worker Applications on the Computational Grid", In: *Proceedings of the First IEEE/ACM International Workshop on Grid Computing*, pp. 214-227, 2000.

[32] HODGES, J. e MORGAN, R., *Lightweight Directory Access Protocol (v3): Technical Specification*, Request for Comments 3377, Setembro/2002.

[33] HUMMEL, S. F., SCHONBERG, E. e FLYNN, L.E., "Factoring: A Metod for Scheduling Parallel Loops", In: *Communication of the ACM*, Vol. 35, N.8, pp. 90-101, Agosto/1992.

[34] HUMMEL, S. F., SCHMIDT, J., Uma, R.N., e WEIN, J., "Load-Sharing in Heterogeneous Systems via Weighted Factoring", In: *Proceedings of the 8th Annual ACM Symposium on Parallel Algorithms and Architectures*, SPA, pp. 318-328, Pádua, Itália, Junho/1996.

[35] JIN, H., SHI, X., QIANG, W., ZOU, D., "An adaptive meta-scheduler for data-intensive applications", In: *International Journal of Grid and Utility Computing*,IJGUC, pp. 32-37, 2005.

[36] KARYPIS, G., KUMAR, V., GUPTA, A., et al. *Introduction to Parallel Computing: Design and Analysis of Algorithms*, Benjamin/Cummings, 1994.

[37] LIFSCHITZ, S., PLASTINO, A., e RIBEIRO, C.C., "Exploring Load balancing in Parallel Processing of Recursive Queries", *Proceedings of the III Euro-Par Conference, Lecture Notes in Computer Science*, LNCS 1300, pp. 1125-1129, Passau, Springer-Verlag, 1997.

[38] MATTSON, T. G., Scientific Computation, *Parallel and Distributed Computing Handbook*, A. Y. Zomaya, pp. 981-1002, McGraw-Hill, 1996.

[39] MING WU , XIAN-HE SUN, "Memory Conscious Task Partition and Scheduling in Grid Environments", In: *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing*, pp. 138-145, Novembro/2004.

[40] Página Web contendo o manual do usuário do NWS - *Network Weather Service User's Guide* http://nws.cs.ucsb.edu/users-guide.html (última consulta realizada em Janeiro/2008).

[41] Página Web contendo informações sobre o *Playload* http://www.cs.northwestern.edu/pdinda/LoadTraces/playload/ (última consulta realizada em Março/2008).

[42] Página Web contendo os *Traces* do *Playload* utilizados neste trabalho. http://cs.uchicago.edu/lyang/load (última consulta realizada em Maio/2008).

[43] Página Web contendo informações sobre o MPI (*Message Passing Interface*). http://www.mpi-forum.org (última consulta realizada em Maio/2008).

[44] Página Web contendo informações sobre o *middleware* Globus Toolkit - *Globus Alliance, Globus Toolkit Release Manuals*, http://www.globus.org/toolkit/ (última consulta realizada em Junho/2008).

[45] PAIVA, A. C., RODRIGUEZ, N.L.R. e GATTASS, M., *Técnicas Paralelas de Rendering*, Relatório Técnico, Departamento de Informática - PUC/Rio, Rio de Janeiro, RJ, 1998.

[46] PLASTINO, A., RIBEIRO, C.C. e RODRIGUEZ, N., "A tool for SPMD application development with support for load balancing", In: *Proceedings of the ParCo Parallel Computing Conference*, pp. 639-646, Imperial College Press, 2000.

[47] PLASTINO, A., *Balanceamento de Carga de Aplicações Paralelas SPMD*, Tese de Doutorado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Brasil, 2000.

[48] PLASTINO, A., COSTA, R., THOME, V., et al. "Exploring Load Balancing in a Scientific SPMD Parallel Application", In: *paper of the 4th International Workshop on High Performance Scientific and Engineering Computing with Applications (HPSECA), realizado em conjunto com 31st International Conference on Parallel Processing (ICPP)*, pp. 419, 2002.

[49] PLASTINO A., RIBEIRO, C.C. e RODRIGUEZ, N., "Development SPMD Applications with Load Balance", *Parallel Computing*, vol. 29, pp. 743-766, 2003.

[50] PLASTINO, A., THOMÉ V., VIANNA, D. et al. "Load Balancing in SPMD Applications: Concepts and Experiments", In: *High Performance Scientific and Engineering Computing: Hardware/Software Support (L.T. Yang and Y. Pan, editors)*, Kluwer Academic Publishers, pp. 95-107, 2004.

[51] POLYCHRONOPOULOS, C. D. e KUCK, D. J., "Guided Self-Scheduling: A Pratical Scheduling Scheme for Parallel Supercomputers", In: *IEEE Transaction on Computers*, Vol. C-36, N.12, pp. 1425-1439, Dezembro/1987.

[52] REEVES, A.P. e WILLEBEEK-LEMAIR, M.A., "Strategies for Dynamic Load Balancing on Highly Parallel Computers", In: *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, pp. 979-993, 1993.

[53] SHAO, G., BERMAN, F., WOLSKI, R., "Using Effective Network Views to Promote Distributed Application Performance", In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, 1999.

[54] SILVEIRA, O. T. *Dispersão Térmica em Meios Porosos Periódicos. Um Estudo Numérico*, Tese de Doutorado, Instituto Politécnico - Universidade do Estado do Rio de Janeiro, Brasil, 2001.

[55] QUINN, M.J. *Parallel Computing: Theory and Practice*, 2^a edição, McGraw-Hill College, 1994.

[56] WHITTED, T., "An Improvement Illumination Model for Shaded Display", In: *Communications of the ACM*, Vol.23(6), pp. 343-349, Junho/1980.

[57] WOLSKI, R., SPRING, N., HAYES. J., "The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing.", In: *Journal of Future Generation Computing Systems*, vol. 15, pp. 757-768, Outubro/1999.

[58] WOLSKI, R., "Experiences with predicting resource performance on-line in computational grid settings", In: *ACM SIGMETRICS Performance Evaluation Review*, vol 30, Number 4, pp. 41-49, Março/2003.

[59] WOLSKI, R., OBERTELLI, G., ALLEN, M., NURMI, D. e BREVIK, J., "Predicting Grid Resource Performance On-line", In: *Handbook of Innovative Computing: Models, Enabling Technologies, and Applications*, Springer-Verlag, 2005.

[60] YANG, C.-T., SHIN, P.-C. CHEN, S.-Y. SHIH, W.-C. , "An Efficient Network Information Model Using NWS for Grid Computing Environments", In: *Proceedings of the 4th International Conference on Grid and Cooperative Computing*, GCC, pp. 287-299, Beijing, China, Nov/Dez 2005.

[61] YANG, L., JENNIFER, M. S., FOSTER, I.T., "Conservative Scheduling: Using Predicted Variance to Improve Scheduling Decisions in Dynamic Environments", In: *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pp. 31, Novembro/2003.

[62] ZAKI, M.J., LI, W. e PARTHASARATHY, S., "Customized Dynamic Load Balancing for a Network of Workstations", In: *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing*, HPDC'96, pp. 282-291, Syracuse, New York, Agosto/1996.

Apêndice A

Tabelas

Neste apêndice, apresentam-se as tabelas que contêm os resultados dos experimentos computacionais realizados e alguns dados estatísticos calculados sobre estes. Nas Tabelas A.2 a A.11, podem ser conferidos os resultados das execuções realizadas no *cluster* e, nas Tabelas A.12 a A.14, os resultados no *grid*. Uma legenda para identificar os significados das colunas dessas tabelas é mostrada na Tabela A.1.

Coluna	Significado
P	Número total de processadores.
BC	Estratégia de balanceamento de carga.
SMT	Tempo total (em minutos) de execução de uma estratégia do tipo mestre-escravo quando o mestre não trabalhou.
CMT	Tempo total (em minutos) de execução de uma estratégia estática ou de uma estratégia do tipo mestre-escravo quando o mestre trabalhou.
DP	Desvio padrão das várias execuções de uma mesma estratégia.
IDC	Índice de desbalanceamento de carga de uma estratégia.
UFF	Número de tarefas executadas pelos processadores do clusterUFF nas execuções de uma estratégia no *grid*.
PUC	Número de tarefas executadas pelos processadores do clusterPUC nas execuções de uma estratégia no *grid*.

Tabela A.1: Legenda para as colunas das tabelas de resultados

		250 *termions*						500 *termions*						1000 *termions*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	3,02	0,03	1,2	-	-	-	6,02	0,04	0,98	-	-	-	12,46	0,83	3,22
	EstNWS	-	-	-	3,05	0,01	1,05	-	-	-	6	0,03	0,27	-	-	-	12,06	0,12	1,09
	MEBF1	4,18	0,05	0,65	3,22	0,05	2,16	8,27	0,07	0,14	6,28	0,07	0,69	16,64	0,14	0,06	12,53	0,06	0,46
	MEBF2	4,09	0,03	0,87	3,15	0,14	2,54	8,15	0,07	0,2	6,19	0,08	1,78	16,2	0,04	0,18	12,87	0,99	0,62
	MEBF4	4,01	0,02	0,53	3,17	0,04	3,9	8,14	0,15	1,05	6,12	0,03	1,75	16	0,03	0,26	12,17	0,1	0,8
	MEBF8	4,12	0,01	3,61	3,32	0,06	7,8	8,07	0,01	1,78	6,2	0,15	2,59	16,24	0,15	0,94	12,21	0,08	1,69
	MERB1	4,07	0,14	2,13	3,1	0,06	2,53	9,15	1,95	8,89	6,14	0,05	1,63	15,93	0,07	0,66	13,47	2,21	2,92
	MERB2	4,02	0,09	0,66	3,04	0,05	1,54	7,96	0,06	0,64	6	0,06	0,49	15,93	0,15	0,17	13,37	2,14	0,83
	MERB3	3,98	0,02	0,54	3,06	0,03	1,25	7,91	0,02	0,21	6,11	0,09	1,84	15,79	0,01	0,09	12,88	1,52	0,72
	MERB4	4,02	0,06	0,6	3,07	0,03	1,9	7,9	0,02	0,21	6,02	0,07	0,61	15,81	0	0,13	11,97	0,06	0,38
	MERB5	4	0	0,56	3,03	0,05	0,85	7,91	0,04	0,2	6,04	0,01	1,21	15,81	0	0,12	11,99	0,03	0,63
	MERB6	3,98	0,02	0,55	3,11	0,02	2,59	7,97	0,05	0,46	6	0,02	0,58	15,85	0,05	0,16	11,99	0,01	0,58
8	Est	-	-	-	1,6	0,06	7,07	-	-	-	3,05	0,01	3,02	-	-	-	6,06	0,04	1,52
	EstNWS	-	-	-	1,56	0,01	2,2	-	-	-	3,06	0	2,08	-	-	-	6,03	0,02	0,69
	MEBF1	1,79	0	1,29	1,59	0,02	2,12	3,54	0,04	0,46	3,17	0,05	2,06	7,08	0,01	0,64	6,37	0,04	1,55
	MEBF2	1,81	0,03	3,45	1,59	0,03	4,02	3,48	0,02	1,15	3,11	0	1,65	7,02	0,03	0,81	6,2	0,07	1,39
	MEBF4	1,81	0,06	4,17	1,7	0,09	10,27	3,48	0,01	1,95	3,15	0,04	3,74	6,97	0,02	1,53	6,19	0,09	2,33
	MEBF8	1,89	0	9,84	1,77	0,07	13,57	3,55	0,01	4,35	3,42	0,08	10,51	6,96	0	2,29	6,34	0,08	4,55
	MERB1	1,77	0,04	3,14	1,64	0,08	7,32	3,5	0,06	2,74	3,13	0,05	3,19	6,89	0,04	0,9	6,09	0,01	1,35
	MERB2	1,75	0,01	2,66	1,58	0,03	4,57	3,44	0	1,76	3,06	0,01	1,79	6,86	0,02	0,96	6,07	0,07	0,78
	MERB3	1,74	0,03	1,13	1,59	0,05	5,24	3,41	0	0,86	3,1	0,03	2,14	6,88	0,05	0,43	6,08	0,03	1,5
	MERB4	1,73	0,01	1,33	1,58	0,03	4,11	3,41	0	0,82	3,12	0,02	2,92	6,82	0,01	0,38	6,08	0,06	1,25
	MERB5	1,74	0,01	1,46	1,62	0,01	5,78	3,43	0,01	0,9	3,05	0,05	1,59	6,83	0,01	0,33	6,09	0,05	1,5
	MERB6	1,73	0,01	1,25	1,57	0,02	3,55	3,42	0	1,2	3,04	0,02	1,09	6,81	0,01	0,42	6,05	0,01	0,95
16	Est	-	-	-	0,82	0	7,8	-	-	-	1,62	0	7,77	-	-	-	3,22	0,01	5,84
	EstNWS	-	-	-	0,86	0	7,34	-	-	-	1,66	0	6,31	-	-	-	3,27	0,02	5,83
	MEBF1	0,87	0	3,54	0,87	0,05	8,82	1,69	0	1,42	1,66	0,03	5,21	3,44	0,01	0,79	3,25	0,01	1,69
	MEBF2	0,9	0	7,48	0,82	0	4,18	1,68	0	3,35	1,64	0,08	6,57	3,35	0,01	1,64	3,21	0,05	3,18
	MEBF4	0,93	0	12,69	0,8	0	5,2	1,75	0,02	7,78	1,81	0,03	15,45	3,35	0,02	2,94	3,26	0,07	5,4
	MEBF8	1,08	0	25,24	1,18	0	34,12	1,86	0,01	14,11	1,86	0,25	18,79	3,44	0,01	7,43	3,41	0,18	9,88
	MERB1	0,87	0,02	6,74	0,85	0,01	9,48	1,68	0	4,94	1,71	0	12,15	3,34	0,03	3,31	3,18	0,05	4,02
	MERB2	0,88	0,01	8,45	0,83	0,05	7,49	1,69	0,02	5,43	1,62	0	6,64	3,28	0,04	1,69	3,15	0,02	2,83
	MERB3	0,84	0	3,64	0,84	0,02	7,34	1,64	0	1,99	1,59	0,04	4,45	3,26	0,02	0,97	3,15	0,01	2,73
	MERB4	0,86	0,02	3,81	0,86	0,05	7,12	1,65	0,01	2,11	1,58	0,03	2,54	3,29	0,05	0,95	3,16	0,06	2,38
	MERB5	0,86	0	3,79	0,88	0,02	7,21	1,65	0	1,84	1,65	0,01	5,73	3,34	0,03	0,94	3,18	0,03	1,97
	MERB6	0,85	0,01	3,49	0,82	0,02	5,86	1,63	0	1,59	1,59	0,01	5,1	3,24	0,01	0,77	3,11	0,04	1,94

Tabela A.2: Resultados da aplicação dos *termions* no ambiente SC

		250 *termions*						500 *termions*						1000 *termions*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	11,81	0,03	25,03	-	-	-	23,54	0,06	24,89	-	-	-	47,38	0,13	36,5
	EstNWS	-	-	-	8,5	0,21	24,84	-	-	-	15,41	3,19	21,83	-	-	-	34,41	0,02	25,4
	MEBF1	11,37	0,13	0,47	6,05	0,04	1,39	22,56	0,26	0,3	11,83	0,04	0,39	44,95	0,06	0,16	23,72	0,01	0,42
	MEBF2	11,16	0,08	0,7	5,98	0,09	1,97	22,19	0,04	0,57	11,7	0,04	0,74	44,39	0,04	0,27	23,41	0,08	0,61
	MEBF4	11,13	0,04	1,21	6,13	0,07	4,36	22,86	0,53	1,43	11,77	0	2,07	44,21	0,01	0,44	23,14	0,13	0,58
	MEBF8	11,47	0,03	4,32	6	0,09	3,08	22,3	0,02	2,31	12,23	0,32	4,93	44,65	0,02	1,72	23,82	0,44	2,85
	MERB1	11,96	0,01	6,59	6,98	0,17	11,97	23,47	0,06	5,82	13,56	0	10,51	46,99	0,18	5,89	26,99	0,03	10
	MERB2	11,15	0,1	0,88	6,1	0,03	4,68	22,35	0,33	0,71	11,87	0,01	3,25	43,79	0,02	0,2	23,47	0,03	2,12
	MERB3	11,02	0,03	0,54	6,08	0,02	3,52	22,22	0,28	0,46	11,88	0,02	3,22	43,83	0,21	0,25	23,46	0,02	2,15
	MERB4	11,12	0,1	0,88	5,91	0,09	1,38	21,87	0,02	0,31	11,48	0,03	0,57	43,77	0,02	0,17	22,91	0,03	0,19
	MERB5	10,98	0,03	0,47	5,84	0,01	1,39	21,87	0,01	0,36	11,51	0,02	0,57	44,03	0,44	0,14	22,95	0,04	0,25
	MERB6	11,09	0,18	1,21	5,84	0,02	1,24	21,92	0,08	0,39	11,49	0,02	0,52	43,81	0,1	0,2	22,91	0,11	0,35
8	Est	-	-	-	5,93	0,01	32,53	-	-	-	11,91	0	32,54	-	-	-	23,89	0,58	31,5
	EstNWS	-	-	-	3,04	0,06	6,8	-	-	-	6,11	0,09	6,97	-	-	-	12,12	0,34	5,54
	MEBF1	3,98	0,05	2,06	3,05	0,04	2,68	7,78	0,02	0,85	6,03	0,01	1,44	15,57	0,01	0,55	11,92	0,08	0,73
	MEBF2	3,99	0,15	4,3	3,04	0,03	3,99	7,71	0,01	1,63	6,06	0,06	3,48	15,53	0,26	0,96	11,78	0,08	1,49
	MEBF4	4,06	0,08	5,94	3,14	0,1	7,2	7,96	0,07	4,66	6,19	0,08	5,15	15,49	0,08	1,93	11,91	0,11	2,9
	MEBF8	4,52	0	14,72	3,55	0,02	15,85	8,01	0,07	5,8	6,13	0,15	4,6	15,56	0,22	2,86	12,19	0,05	5,05
	MERB1	5,11	0	23,5	4,75	0	34,87	9,94	0,02	22	9,32	0,08	33,83	20,13	0,8	22,2	18,6	0,06	34,2
	MERB2	3,88	0	2,49	3,23	0,01	9,29	7,81	0,08	3,97	6,5	0	9,54	15,42	0,08	1,42	12,93	0,33	9,93
	MERB3	3,89	0	2,49	3,23	0	9,2	7,59	0,03	1,26	6,48	0,04	9,47	15,22	0,09	0,9	12,51	0,01	7,54
	MERB4	3,88	0,05	2,19	3	0,02	3,65	7,58	0,03	1,13	5,89	0,03	1,56	15,12	0,06	0,59	11,53	0,02	0,58
	MERB5	3,88	0,02	1,89	3,03	0,02	3,76	7,58	0,03	1,14	5,86	0,01	1,42	15,25	0,19	0,59	11,57	0,05	0,77
	MERB6	3,9	0,08	2,7	2,99	0,06	3,4	7,58	0,05	1,03	5,84	0,02	1,44	15,15	0,11	0,62	11,52	0,02	0,8
16	Est	-	-	-	3,24	0,02	38,79	-	-	-	6,44	0,06	39,19	-	-	-	12,58	0,06	37,7
	EstNWS	-	-	-	1,65	0,02	13,96	-	-	-	3,2	0,02	11,71	-	-	-	6,35	0,11	11
	MEBF1	1,79	0,03	5,05	1,6	0	6,19	3,51	0,01	3,58	3,1	0,01	2,93	6,84	0,04	1,35	6,08	0,05	1,68
	MEBF2	1,97	0,07	13,08	1,69	0,01	11	3,55	0,05	5,24	3,15	0,07	6,09	6,88	0,07	3,16	6,04	0,06	2,86
	MEBF4	2,2	0,07	21,49	1,87	0,13	17,87	3,91	0,09	14,65	3,28	0,09	9,86	6,88	0,22	3,9	6,23	0,02	5,47
	MEBF8	2,46	0,05	29,96	2,14	0,2	30,43	4,3	0,03	21,53	3,59	0,03	16,29	7,86	0,06	14,4	6,22	0,21	5,93
	MERB1	2,96	0,08	39,61	2,76	0,06	42,7	6,09	0,02	42,23	5,58	0,07	43,92	11,75	0,12	41,1	10,75	0,05	42,5
	MERB2	1,98	0,08	15,3	1,57	0,01	5,58	3,61	0,07	7,81	3,06	0,04	4,18	6,99	0,06	5,09	6	0,02	3,24
	MERB3	1,92	0,05	12,52	1,57	0,02	4,26	3,53	0,07	4,54	3,05	0,04	4,06	6,81	0,08	3,39	5,92	0,06	1,76
	MERB4	1,81	0,04	6,65	1,59	0,06	6,55	3,41	0,03	3,29	3,02	0,05	3,23	6,7	0,04	1,83	5,89	0,04	1,72
	MERB5	1,77	0,04	5,06	1,62	0,02	7,96	3,42	0,03	3,3	3,04	0,04	3,58	6,71	0,05	1,72	5,93	0,01	2,02
	MERB6	1,78	0,02	5,67	1,58	0,03	6,15	3,52	0,17	6,51	3,05	0,02	4,49	6,66	0,02	2,08	5,85	0,03	2,28

Tabela A.3: Resultados da aplicação dos *termions* no ambiente CC-0.1.2.3

		250 *termions*						500 *termions*						1000 *termions*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	6,67	0,91	18,51	-	-	-	13,76	1,83	19,18	-	-	-	26,28	1,18	17,3
	EstNWS	-	-	-	5,15	0,67	16,32	-	-	-	9,95	1,14	12,63	-	-	-	20,85	3,91	16,4
	MEBF1	6,88	0,73	0,49	4,92	0,33	1,13	14,05	1,41	0,36	10,12	0,46	0,56	27,05	2,86	0,13	19,68	1,58	0,27
	MEBF2	7,14	0,68	1,32	4,91	0,31	2,01	14,47	1,33	0,47	9,95	0,6	1,03	26,58	2,76	0,27	18,96	1,44	0,5
	MEBF4	7,14	0,82	1,28	4,93	0,35	2,31	13,75	1,41	1,15	10,02	0,63	2,03	28,09	4,48	5,39	19	1,47	0,89
	MEBF8	7,13	0,91	1,83	5,02	0,4	3,87	13,76	1,61	1,53	9,9	0,74	2,34	27,36	4,25	1,09	19,22	1,39	1,9
	MERB1	7,18	1,07	2,45	5,24	0,41	8,3	13,42	1,66	1,06	10,27	0,34	6,79	27,51	4,36	1,94	20,16	0,34	6,54
	MERB2	7,09	0,96	1,27	4,8	0,44	1,39	13,99	1,74	0,57	9,67	0,75	1,03	26,21	3,99	0,37	18,67	1,19	0,94
	MERB3	7,14	0,85	0,58	4,82	0,38	1,74	14,2	1,64	0,27	9,55	0,65	0,7	25,72	3,38	0,16	18,37	1,48	0,92
	MERB4	7,06	0,78	0,59	4,85	0,36	1,61	14,08	1,69	0,27	9,52	0,71	0,97	25,5	3,16	0,14	18,7	1,73	0,76
	MERB5	6,93	0,73	0,57	4,84	0,34	1,14	13,82	1,94	0,46	9,57	0,72	0,89	25,47	3,23	0,31	18,97	1,85	0,32
	MERB6	6,69	0,71	0,76	4,84	0,32	1,34	13,78	1,89	0,42	9,63	0,62	0,92	25,11	3,53	0,49	19,17	1,71	0,42
8	Est	-	-	-	3,86	0,66	28,53	-	-	-	6,98	0,67	26,42	-	-	-	14,11	2	30,7
	EstNWS	-	-	-	4,04	0,51	40,32	-	-	-	5,29	0,63	17,16	-	-	-	10,11	1,87	15,5
	MEBF1	2,96	0,2	2,23	2,67	0,14	2,9	5,56	0,48	0,99	4,26	0,19	1,48	10,68	1,07	0,52	9,09	0,61	0,78
	MEBF2	2,96	0,18	3,54	2,72	0,17	5,03	5,55	0,51	1,86	4,28	0,19	2,88	10,48	0,99	0,93	9,12	0,67	1,32
	MEBF4	2,95	0,14	4,39	2,74	0,16	6,55	5,52	0,35	3,44	4,35	0,21	4,56	10,57	1,12	1,95	9,21	0,67	2,58
	MEBF8	3,17	0,18	10,35	3	0,31	14,4	5,71	0,33	5,8	4,47	0,19	7,63	10,56	1,21	3,18	9,41	0,56	4,82
	MERB1	4,15	1,03	29,01	4,98	0,74	45,34	7,06	0,72	22,24	5,78	0,91	28,36	13,38	0,85	21,8	11,94	1,08	24,9
	MERB2	2,96	0,23	4,79	2,59	0,66	5,94	5,52	0,45	2,74	4,37	0,91	7,01	10,56	1,18	1,72	8,74	0,59	1,95
	MERB3	2,94	0,35	4,31	2,59	0,67	6,08	5,49	0,46	2,21	4,29	0,58	5,65	10,33	0,97	1,41	8,62	0,61	1,13
	MERB4	2,87	0,11	2,3	2,61	0,17	7,86	5,38	0,38	0,94	4,1	0,16	1,41	10,2	0,96	0,82	8,69	0,66	0,94
	MERB5	2,92	0,21	3,24	2,67	0,12	9,17	5,38	0,43	0,95	4,12	0,17	2,04	10,26	1,05	0,47	8,71	0,59	0,92
	MERB6	2,85	0,2	2,01	2,62	0,15	3,1	5,34	0,45	1,15	4,09	0,17	1,59	10,16	1,06	0,54	8,7	0,62	1,01
16	Est	-	-	-	1,98	0,36	36,66	-	-	-	4	0,62	36,5	-	-	-	7,19	0,86	30,5
	EstNWS	-	-	-	1,35	0,12	18,87	-	-	-	2,78	0,36	20,41	-	-	-	5,75	0,8	23,5
	MEBF1	1,29	0,07	5	1,29	0,05	6,79	2,56	0,13	2,74	2,42	0,14	3,21	5,06	0,25	1,27	4,79	0,22	1,83
	MEBF2	1,36	0,08	9,89	1,29	0,06	9,37	2,6	0,1	4,39	2,45	0,13	6,45	5,1	0,27	2,74	4,76	0,19	2,86
	MEBF4	1,44	0,12	15,52	1,4	0,06	15,94	2,69	0,12	8,33	2,54	0,15	9,77	5,14	0,28	4,13	4,84	0,22	5,21
	MEBF8	1,66	0,08	26,33	1,67	0,06	28,31	2,92	0,24	15,37	2,68	0,13	13,55	5,29	0,24	8,05	5,08	0,23	9,73
	MERB1	1,85	0,32	32,67	1,63	0,24	27,2	3,36	0,3	26,08	3,04	0,22	23,96	7,11	1,29	29,8	6,26	0,52	26,1
	MERB2	1,36	0,08	10,4	1,26	0,05	6,8	2,63	0,13	6,59	2,36	0,11	4,03	5,12	0,23	3,96	4,72	0,16	2,7
	MERB3	1,29	0,07	5,75	1,27	0,04	7,5	2,53	0,09	3,19	2,36	0,11	4,02	5,05	0,31	2,87	4,71	0,17	2,01
	MERB4	1,29	0,07	4,87	1,26	0,04	6,38	2,53	0,1	2,63	2,35	0,1	3,53	4,99	0,24	1,42	4,7	0,18	1,8
	MERB5	1,31	0,07	5,33	1,27	0,06	6,47	2,55	0,09	2,72	2,38	0,1	3,79	5,01	0,27	1,39	4,73	0,21	1,77
	MERB6	1,27	0,06	4,78	1,26	0,05	6,89	2,51	0,1	2,82	2,35	0,11	3,76	4,96	0,27	1,43	4,63	0,22	1,58

Tabela A.4: Resultados da aplicação dos *termions* no ambiente CC-LL.LH.HL.HH

		250 *termions*						500 *termions*						1000 *termions*					
P	BC	SMT	DP	IDC	CMT	DP	IDC	SMT	DP	IDC	CMT	DP	IDC	SMT	DP	IDC	CMT	DP	IDC
4	Est	-	-	-	6,88	1,17	15,76	-	-	-	13,47	0,72	14,85	-	-	-	27,47	3,21	17,3
	EstNWS	-	-	-	5,6	1,51	17,67	-	-	-	11,5	3,11	21,4	-	-	-	22,75	6,91	20,9
	MEBF1	7,22	0,96	0,46	4,72	0,28	0,95	14,93	2,22	0,27	9,75	0,83	0,6	28,7	4,27	0,13	19,98	1,89	0,28
	MEBF2	7,12	0,87	1,11	4,71	0,31	2,07	14,44	2,27	0,45	9,59	0,85	0,88	28,05	4,21	0,33	19,54	2,07	0,53
	MEBF4	7,11	1,06	1,31	4,69	0,27	2,1	14,3	2,11	1,03	9,62	0,83	1,85	27,81	4,1	0,39	19,13	2,18	0,96
	MEBF8	7,14	0,95	2,19	4,9	0,48	5,17	14,52	2,4	1,56	9,63	0,88	2,38	27,97	4,13	1,11	19,3	2,28	1,92
	MERB1	7,07	0,91	2,19	5,5	1,57	15,04	14,96	2,78	2,87	10,84	1,36	12,21	27,97	4,51	1,7	23,76	7,22	19,4
	MERB2	6,85	0,81	1,12	4,58	0,24	1,46	14,72	2,16	0,51	9,39	0,78	1,02	27,64	4,14	0,36	18,88	2,03	0,55
	MERB3	6,81	0,78	0,45	4,61	0,23	1,6	14,26	1,96	0,32	9,36	0,78	0,77	27,43	3,88	0,15	19,1	2,28	0,32
	MERB4	6,84	0,85	0,66	4,56	0,25	1,55	14,1	2,33	1,15	9,35	0,8	0,71	28,45	4,15	0,36	18,99	2,47	0,38
	MERB5	6,92	1,21	1,32	4,54	0,24	1,47	14,01	2,13	0,3	9,38	0,81	0,91	27,76	4,37	1,41	18,92	2,57	0,35
	MERB6	7,03	1,09	0,65	4,53	0,28	1,03	14,05	2,21	0,26	9,38	0,83	0,82	27,07	3,81	0,16	18,83	2,49	0,83
8	Est	-	-	-	3,31	0,05	22,22	-	-	-	6,51	0,08	22,01	-	-	-	13,19	0,13	23,2
	EstNWS	-	-	-	2,58	0,07	13,73	-	-	-	5,04	0,53	14,51	-	-	-	9,86	0,56	11,8
	MEBF1	2,87	0,05	2,2	2,4	0,07	3,85	5,49	0,19	1,12	4,68	0,16	1,62	10,89	0,43	0,45	9,08	0,33	0,76
	MEBF2	2,89	0,08	3,37	2,42	0,08	5,59	5,49	0,22	1,88	4,63	0,15	2,65	10,82	0,42	0,97	9	0,36	1,33
	MEBF4	2,92	0,06	5,06	2,44	0,11	7,28	5,53	0,22	3,74	4,73	0,17	4,84	10,78	0,41	2,09	9,04	0,34	2,62
	MEBF8	3,02	0,06	9,81	2,65	0,05	12,37	5,65	0,26	5,93	4,9	0,22	7,92	10,91	0,44	3,87	9,2	0,3	4,22
	MERB1	2,85	0,11	4,12	2,42	0,15	6,58	5,5	0,36	3,91	4,62	0,1	3,74	10,69	0,36	2,32	9,12	0,69	4,2
	MERB2	2,8	0,05	2,83	2,34	0,07	3,76	5,41	0,19	2,61	4,51	0,13	1,72	10,66	0,45	1,1	8,86	0,33	1,71
	MERB3	2,78	0,05	2,22	2,33	0,08	3,71	5,34	0,2	0,96	4,52	0,15	1,62	10,53	0,38	0,47	8,71	0,32	0,82
	MERB4	2,78	0,06	2,02	2,33	0,09	3,64	5,3	0,19	1,04	4,5	0,16	1,48	10,5	0,39	0,45	8,7	0,29	0,81
	MERB5	2,78	0,05	2,18	2,32	0,08	3,15	5,32	0,22	0,95	4,52	0,14	1,6	10,46	0,38	0,48	8,73	0,33	0,79
	MERB6	2,78	0,07	2,01	2,31	0,08	3,3	5,29	0,19	0,86	4,51	0,17	1,83	10,46	0,38	0,44	8,72	0,32	0,92
16	Est	-	-	-	3,22	0,07	56,88	-	-	-	5,98	0,03	55,93	-	-	-	12,51	0,15	56,5
	EstNWS	-	-	-	2,22	0,11	44,74	-	-	-	3,88	0,33	41,55	-	-	-	8,3	0,42	43,7
	MEBF1	1,46	0,06	8,46	1,28	0,04	6,54	2,57	0,05	3,88	2,59	0,04	3,31	5,3	0,21	2,23	4,84	0,16	2,27
	MEBF2	1,48	0,09	11,23	1,31	0,04	10,01	2,57	0,05	5,13	2,6	0,04	5,66	5,2	0,22	3,2	4,8	0,19	3,8
	MEBF4	1,74	0,07	24,14	1,36	0,06	13,21	2,69	0,06	9,17	2,69	0,05	9,17	5,23	0,25	4,91	5,01	0,13	8,28
	MEBF8	1,8	0,02	24,38	1,69	0,07	28,44	3,04	0,15	19,48	2,77	0,1	11,73	5,51	0,31	9,53	5,15	0,16	10,6
	MERB1	2,46	0,03	45,2	1,43	0,22	17,86	4,69	0,18	46,66	3,19	0,38	23,46	9,51	0,29	45,8	8,52	0,27	44,6
	MERB2	1,96	0,03	32,15	1,25	0,05	6,72	3,68	0,07	32,65	2,51	0,04	3,69	7,5	0,21	32,2	6,72	0,18	30,9
	MERB3	1,97	0,05	32,14	1,25	0,04	6,05	3,68	0,07	32,97	2,53	0,05	4,21	7,53	0,24	32,1	6,69	0,16	30,3
	MERB4	1,38	0,02	4,97	1,26	0,04	6,15	2,52	0,08	4,01	2,54	0,06	4,15	5,15	0,18	2,19	4,68	0,15	2,29
	MERB5	1,39	0,03	5,67	1,28	0,06	7,09	2,57	0,07	4,68	2,57	0,05	4,15	5,19	0,24	2,81	4,81	0,33	3,78
	MERB6	1,37	0,03	5,02	1,26	0,07	7,28	2,49	0,06	3,48	2,51	0,04	3,55	5,1	0,17	1,87	4,65	0,17	2,24

Tabela A.5: Resultados da aplicação dos *termions* no ambiente CC-LL.LL.HL.HL

		250 *termions*						500 *termions*						1000 *termions*					
P	BC	SMT	DP	IDC	CMT	DP	IDC	SMT	DP	IDC	CMT	DP	IDC	SMT	DP	IDC	CMT	DP	IDC
4	Est	-	-	-	6,61	1,13	21,39	-	-	-	14,48	2,75	24,63	-	-	-	25,02	2,55	18,5
	EstNWS	-	-	-	6,2	1,12	31,57	-	-	-	12,06	2,08	27,04	-	-	-	22,56	5,59	23,6
	MEBF1	6,81	0,62	0,52	5,12	0,39	1,12	13,07	1,38	0,29	10,12	1,01	0,73	26,24	1,48	0,13	19,82	1,3	0,34
	MEBF2	6,6	0,57	1,42	5,08	0,32	2,3	14,14	1,76	0,55	10,32	0,59	1,35	24,53	2,02	0,26	19,81	1,35	0,61
	MEBF4	6,41	0,52	1,27	5,19	0,3	2,73	13,77	1,92	0,93	10,49	0,68	2,41	25,56	2,54	0,53	19,49	1,43	0,82
	MEBF8	6,45	0,58	2,84	5,31	0,4	5,04	13,35	1,24	1,46	10,8	1,05	3,78	27,18	2,08	1,45	19,44	1,68	2,06
	MERB1	6,49	0,96	2,8	5,46	0,73	8,14	12,92	0,99	1,33	12,82	2,4	15,26	27,35	3,76	3,45	19,27	1,31	2,99
	MERB2	6,29	0,97	1,2	5,01	0,28	1,59	12,53	1,88	1,18	11,69	1,81	9,96	26,71	4,1	0,7	19,07	1,75	0,62
	MERB3	6,37	0,78	0,74	5,1	0,25	1,3	12,87	1,94	0,51	11,53	1,79	9,26	26,52	3,71	0,14	19,17	2,15	0,35
	MERB4	6,57	0,53	0,77	4,95	0,34	1,21	13,03	1,63	0,28	10,23	0,9	0,86	25,54	1,52	0,19	19,5	2,17	0,68
	MERB5	6,71	0,5	0,59	5	0,29	1,36	13,34	1,06	0,26	10,01	0,89	1,04	27,04	6,13	0,15	19,37	2,2	0,4
	MERB6	6,39	0,76	0,57	5,09	0,29	1,43	13,4	1,05	0,34	10,01	0,82	1,25	25,7	2,92	0,3	19,01	1,66	0,34
8	Est	-	-	-	3,62	0,64	32,88	-	-	-	7,67	1,3	34,7	-	-	-	15,52	2,66	33,1
	EstNWS	-	-	-	3,16	0,2	32,99	-	-	-	6,02	0,58	29,29	-	-	-	11,7	2,08	24,4
	MEBF1	2,68	0,21	2,13	2,42	0,22	3,96	5,35	0,4	1,09	4,68	0,5	1,73	10,91	0,96	0,51	9,26	0,76	0,8
	MEBF2	2,67	0,18	3,65	2,41	0,23	5,56	5,3	0,55	2,11	4,65	0,46	2,63	10,85	0,79	0,96	9,24	0,82	1,57
	MEBF4	2,76	0,14	6,57	2,48	0,26	7,68	5,35	0,58	3,68	4,77	0,45	4,91	10,84	0,78	2,04	9,22	0,73	2,51
	MEBF8	2,93	0,34	11,92	2,65	0,34	13,51	5,56	0,51	5,76	4,92	0,36	7,46	11,14	0,87	3,2	9,44	0,68	4,62
	MERB1	2,9	0,39	11,19	2,6	0,43	12,36	5,6	0,67	6,73	5,4	0,76	15,52	11,75	1,63	9,06	10,08	1,46	11,6
	MERB2	2,68	0,17	3,66	2,38	0,26	4,64	5,24	0,42	2,15	4,77	0,29	4,45	10,61	0,99	2,22	9,33	0,73	4,84
	MERB3	2,64	0,17	2,39	2,38	0,23	4,97	5,11	0,3	1,15	4,72	0,33	3,01	10,42	0,96	1,3	9,43	0,6	3,5
	MERB4	2,66	0,13	2,74	2,35	0,22	3,34	5,13	0,28	1,03	4,73	0,36	1,65	10,44	0,87	1,25	9,33	0,77	1,08
	MERB5	2,67	0,12	1,92	2,35	0,19	3,4	5,14	0,35	0,97	4,75	0,39	1,88	10,58	0,9	0,56	9,3	0,8	0,81
	MERB6	2,67	0,18	2,46	2,36	0,19	3,28	5,27	0,32	1,17	4,71	0,41	1,83	10,69	1,03	0,69	9,18	0,73	0,77
16	Est	-	-	-	2,76	0,42	48,17	-	-	-	5,44	0,75	48,71	-	-	-	12,13	2,3	52,8
	EstNWS	-	-	-	2,15	0,24	42,72	-	-	-	4,6	0,42	47,04	-	-	-	8,63	0,73	43,5
	MEBF1	1,45	0,07	5,79	1,34	0,08	7,05	2,78	0,14	3,16	2,55	0,16	3,96	5,41	0,34	2,04	5,03	0,28	2,06
	MEBF2	1,5	0,07	10,4	1,36	0,08	9,75	2,79	0,15	5,28	2,58	0,2	7,1	5,31	0,25	3,69	5,03	0,24	4,11
	MEBF4	1,67	0,12	19,11	1,48	0,15	17,5	2,91	0,15	10,19	2,72	0,25	11,56	5,52	0,31	7,09	5,16	0,2	7,43
	MEBF8	1,83	0,07	25,52	1,69	0,11	26,79	3,18	0,25	17,77	3,06	0,31	21,06	5,88	0,5	12,4	5,28	0,38	10
	MERB1	2,08	0,49	34,65	1,55	0,21	21,19	3,72	0,79	28,92	4,8	1,73	48,38	9,19	1,59	42,5	8,06	1,35	40
	MERB2	1,62	0,17	17,11	1,31	0,07	7,73	2,88	0,21	9,3	2,77	0,43	13,83	7,41	1,19	29,5	6,3	1,17	24,3
	MERB3	1,55	0,13	13,76	1,34	0,08	8,8	2,85	0,26	8,06	2,71	0,36	12,05	7,33	1,15	28,9	6,28	1,07	24,3
	MERB4	1,43	0,07	6,22	1,33	0,07	7,61	2,76	0,12	3,99	2,48	0,13	4,02	5,3	0,37	2,57	4,86	0,28	3,48
	MERB5	1,43	0,05	6,12	1,36	0,08	8,14	2,77	0,13	3,55	2,52	0,15	3,95	5,39	0,43	3,38	4,97	0,32	3,79
	MERB6	1,43	0,06	7,11	1,33	0,08	8,6	2,82	0,27	6,73	2,71	0,3	11,94	5,54	0,95	7,53	5,66	0,73	15,09

Tabela A.6: Resultados da aplicação dos *termions* no ambiente CC-LH.LH.HH.HH

		imagem *5balls*						imagem *room*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	2,19	0,02	9,29	-	-	-	1,56	0	39,31
	EstNWS	-	-	-	2,21	0,03	11,11	-	-	-	1,59	0,06	41,48
	MEBF1	3,07	0,03	0,14	2,24	0,03	2,31	1,64	0,06	0,16	1,24	0,02	1,38
	MEBF2	2,9	0,02	0,65	2,15	0,05	2,27	1,54	0	0,34	1,2	0,01	4,27
	MEBF4	2,81	0,03	1,27	2,12	0,04	2,51	1,47	0,01	0,67	1,15	0,05	2,5
	MEBF8	2,77	0,03	0,94	2,11	0,03	2,19	1,46	0,01	2	1,19	0,06	6,5
	MERB1	2,92	0,04	7,17	2,2	0,04	7,42	1,43	0,01	1,01	1,13	0,02	4
	MERB2	2,74	0,01	0,41	2,09	0,04	1,97	1,42	0,01	0,42	1,11	0,04	2,16
	MERB3	2,72	0,04	0,52	2,09	0,03	2,16	1,43	0,02	0,39	1,12	0,03	2,48
	MERB4	2,71	0,02	0,41	2,1	0,04	2,02	1,43	0,01	0,24	1,12	0,03	2,91
	MERB5	2,71	0,01	0,41	2,08	0,04	1,61	1,43	0,01	0,21	1,11	0,04	1,72
	MERB6	2,7	0,02	0,3	2,07	0,03	1,58	1,44	0,01	0,17	1,11	0,02	1,61
8	Est	-	-	-	1,09	0	9,12	-	-	-	0,83	0,01	38,19
	EstNWS	-	-	-	1,1	0,01	9,39	-	-	-	0,83	0,03	39,53
	MEBF1	1,22	0	0,84	1,15	0,02	3,67	0,7	0,01	0,64	0,7	0,01	6,81
	MEBF2	1,2	0	1,98	1,14	0,04	5,06	0,67	0,01	1,07	0,66	0,01	8,29
	MEBF4	1,2	0	3,56	1,16	0,06	7,7	0,66	0,02	3,03	0,65	0,04	10,32
	MEBF8	1,2	0	4,33	1,18	0,06	11,03	0,65	0,01	4,94	0,64	0,03	10,48
	MERB1	1,25	0	8,61	1,12	0,03	5,94	0,63	0	3,38	0,59	0,02	5,37
	MERB2	1,16	0	1,6	1,11	0,04	5	0,63	0,01	1,93	0,61	0,02	7,2
	MERB3	1,16	0	1,5	1,11	0,02	4,89	0,62	0	0,48	0,61	0,01	8,54
	MERB4	1,17	0	1,5	1,12	0,03	4,87	0,63	0,01	0,86	0,61	0,03	6,92
	MERB5	1,17	0	0,67	1,13	0,05	4,92	0,65	0,02	1,29	0,61	0,02	4,71
	MERB6	1,15	0	0,66	1,11	0,03	5,41	0,63	0,01	1,22	0,61	0,03	7,51
16	Est	-	-	-	0,59	0,01	10,15	-	-	-	0,45	0,03	37,72
	EstNWS	-	-	-	0,63	0,02	11	-	-	-	0,47	0,02	36,74
	MEBF1	0,62	0,02	2,74	0,58	0,01	2,87	0,51	0	1,81	0,57	0,01	10,26
	MEBF2	0,6	0,01	4,3	0,59	0,01	8,93	0,33	0	3,37	0,39	0,02	17,78
	MEBF4	0,6	0,01	6,66	0,58	0,01	8,06	0,33	0,01	7,87	0,39	0,02	25,03
	MEBF8	0,59	0	5,45	0,68	0,09	21,88	0,35	0,01	14,64	0,45	0,02	35,65
	MERB1	0,6	0,01	7,98	0,58	0,01	9,47	0,31	0,01	3,83	0,35	0,01	18,45
	MERB2	0,58	0	4,13	0,57	0,04	6,07	0,32	0	6,11	0,33	0,03	10,83
	MERB3	0,57	0	2,58	0,6	0,01	11,23	0,31	0,01	1,75	0,34	0,02	14,14
	MERB4	0,59	0,01	3,14	0,61	0,02	8,28	0,33	0,02	2,07	0,35	0,02	12,15
	MERB5	0,66	0,03	3,6	0,62	0,03	7,42	0,41	0,01	2,63	0,43	0,02	12,11
	MERB6	0,58	0,01	2,28	0,6	0,05	9,05	0,32	0,01	2,17	0,33	0,02	7,03

Tabela A.7: Resultados da aplicação *Ray Tracing* no ambiente SC

		imagem *5balls*						imagem *room*					
P	BC	SMT	DP	IDC	CMT	DP	IDC	SMT	DP	IDC	CMT	DP	IDC
4	Est	-	-	-	8,06	0,01	49,74	-	-	-	4,48	0,02	56,07
	EstNWS	-	-	-	4,28	0,09	12,24	-	-	-	2,65	0,07	41,24
	MEBF1	7,82	0,01	0,33	4,11	0,01	0,76	4,5	0,07	0,19	2,3	0,03	0,37
	MEBF2	7,62	0,02	0,67	3,99	0,02	0,85	4,22	0,02	0,24	2,17	0,02	0,97
	MEBF4	7,57	0,05	1,34	4,01	0,06	2,3	4,14	0,02	0,97	2,18	0,07	2,81
	MEBF8	7,54	0,01	1,55	4,05	0,02	3,1	4,21	0,09	2,98	2,24	0,04	6,52
	MERB1	7,44	0,02	0,86	4,53	0,02	9,4	4,18	0,16	2,95	2,18	0,04	4,27
	MERB2	7,41	0,03	0,36	4,12	0,01	5,4	4,11	0,03	2,05	2,12	0,04	1,71
	MERB3	7,47	0,04	1,09	4,11	0,01	5,07	4,02	0,01	0,14	2,16	0,14	0,86
	MERB4	7,41	0,01	0,28	3,94	0,03	1,28	4,05	0,07	0,36	2,14	0,15	0,73
	MERB5	7,4	0,02	0,26	3,92	0,01	0,74	4,01	0,05	0,35	2,22	0,32	1,99
	MERB6	7,4	0,02	0,32	3,9	0,02	0,7	4,01	0,05	0,46	2,24	0,39	2,35
8	Est	-	-	-	4,17	0,03	43,53	-	-	-	3,07	0,03	62,87
	EstNWS	-	-	-	2,19	0,01	8,7	-	-	-	1,56	0,09	40,41
	MEBF1	2,72	0,04	1,09	2,1	0,01	1,98	1,52	0,01	0,99	1,18	0,04	2,63
	MEBF2	2,73	0,04	2,79	2,07	0,01	2,38	1,49	0,03	2,5	1,16	0,03	4,28
	MEBF4	2,7	0,02	3,82	2,08	0,02	4,49	1,52	0,02	6,91	1,15	0,01	5,99
	MEBF8	2,79	0,02	6,62	2,26	0,01	10,39	1,48	0,05	6,1	1,15	0,03	7,32
	MERB1	3,46	0	22,39	3,27	0,02	34,29	1,49	0,01	7,94	1,23	0,01	13,12
	MERB2	2,69	0,01	4,07	2,22	0,01	10,24	1,42	0,02	3,32	1,1	0,08	3,68
	MERB3	2,66	0,01	3,33	2,23	0	10,29	1,42	0,03	2,89	1,1	0,04	3,23
	MERB4	2,6	0,02	1,26	2,02	0,01	1,84	1,39	0,02	1,36	1,09	0,02	2,62
	MERB5	2,62	0,01	1,56	2,03	0,01	2,23	1,4	0,02	0,87	1,1	0,02	2,93
	MERB6	2,58	0,02	1,28	2,01	0,02	2,32	1,38	0,01	0,9	1,1	0,02	3,08
16	Est	-	-	-	2,24	0,04	44,93	-	-	-	1,58	0,08	61,34
	EstNWS	-	-	-	1,13	0,03	9,52	-	-	-	0,87	0,04	39,38
	MEBF1	1,21	0,02	2,5	1,09	0,02	3,9	0,69	0,04	4,3	0,64	0,03	5,19
	MEBF2	1,23	0,01	5,9	1,12	0,02	7,82	0,68	0,02	7,22	0,61	0,02	6,66
	MEBF4	1,31	0,04	13,19	1,12	0,04	8,97	0,76	0,03	14,75	0,68	0,04	18,34
	MEBF8	1,38	0,14	16,45	1,23	0,02	15,85	0,86	0,05	25,11	0,76	0,06	26,31
	MERB1	2,11	0,03	43,62	1,92	0,04	45	0,74	0,04	15,67	0,71	0,04	21,82
	MERB2	1,29	0,05	11,89	1,11	0,02	9,26	0,71	0,03	11,57	0,6	0,02	8,25
	MERB3	1,21	0,04	6,07	1,14	0,01	11	0,65	0,02	4,37	0,59	0,02	5,81
	MERB4	1,17	0,03	3,4	1,03	0,03	3,68	0,64	0,02	3,28	0,6	0,01	5,64
	MERB5	1,2	0,03	3,37	1,1	0,03	5,3	0,69	0,01	3,04	0,63	0,01	6,24
	MERB6	1,17	0,02	3,98	1,15	0,05	13,19	0,65	0,01	3,57	0,6	0,02	6,63

Tabela A.8: Resultados da aplicação *Ray Tracing* no ambiente CC-0.1.2.3

		imagem *5balls*						imagem *room*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	5,12	2,14	21,47	-	-	-	2,57	1,07	30,36
	EstNWS	-	-	-	4,49	2,26	18,8	-	-	-	2,46	0,76	43,63
	MEBF1	5,38	1,86	0,21	3,52	1,15	0,95	2,94	1,25	0,14	1,78	0,67	0,8
	MEBF2	5,45	1,8	0,57	3,44	1,17	1,4	2,85	1,32	0,32	1,7	0,65	1,12
	MEBF4	5,43	1,68	1,08	3,47	1,17	2,29	2,87	1,26	1,25	1,68	0,65	2,5
	MEBF8	5,45	1,72	1,04	3,45	1,19	2,43	2,89	1,27	2,45	1,71	0,64	4,11
	MERB1	6,09	3,21	11,2	4,33	2,84	20,17	2,87	1,28	3,28	1,8	0,96	9,39
	MERB2	5,31	1,83	0,85	3,4	1,19	1,99	2,87	1,29	1,26	1,64	0,63	2
	MERB3	5,29	1,88	0,68	3,42	1,26	2,41	2,83	1,28	0,66	1,64	0,65	1,39
	MERB4	5,27	1,84	0,54	3,33	1,14	1	2,81	1,3	0,7	1,63	0,65	1,27
	MERB5	5,27	1,83	0,73	3,33	1,11	0,81	2,79	1,31	0,75	1,64	0,66	1,28
	MERB6	5,06	1,36	0,57	3,34	1,08	1,15	2,74	1,27	0,35	1,65	0,65	1,74
8	Est	-	-	-	2,48	0,71	32,2	-	-	-	1,74	0,46	49,84
	EstNWS	-	-	-	1,85	0,43	9,21	-	-	-	1,46	0,38	47,23
	MEBF1	1,98	0,5	1,16	1,68	0,44	3,48	1,17	0,29	0,68	1	0,23	3,34
	MEBF2	1,97	0,47	2,01	1,63	0,44	3,56	1,13	0,3	2,08	0,95	0,22	4,74
	MEBF4	1,95	0,48	2,89	1,64	0,46	5,23	1,12	0,29	3,75	0,95	0,23	6,98
	MEBF8	1,98	0,51	5,29	1,67	0,44	7,38	1,16	0,34	8,36	1	0,22	11,68
	MERB1	2,41	0,55	21,84	2,13	0,54	26,35	1,12	0,28	5,43	0,93	0,24	6,46
	MERB2	1,89	0,49	2,26	1,62	0,41	3,98	1,09	0,27	3,13	0,91	0,22	5,46
	MERB3	1,89	0,5	2,35	1,61	0,41	3,77	1,07	0,26	1,84	0,9	0,22	4,45
	MERB4	1,88	0,49	1,23	1,59	0,41	2,94	1,07	0,25	0,75	0,92	0,23	5,2
	MERB5	1,89	0,5	1,41	1,61	0,43	3,19	1,08	0,25	0,92	0,92	0,24	4,13
	MERB6	1,88	0,47	1,37	1,6	0,4	3,48	1,07	0,26	1,18	0,9	0,19	4,29
16	Est	-	-	-	1,39	0,32	34,87	-	-	-	0,83	0,2	43,56
	EstNWS	-	-	-	1	0,21	13,91	-	-	-	0,77	0,19	39,21
	MEBF1	0,93	0,21	2,67	0,89	0,2	5,01	0,62	0,15	2,45	0,64	0,14	8,71
	MEBF2	0,91	0,21	4,41	0,87	0,19	6,74	0,54	0,13	5,38	0,52	0,12	11,77
	MEBF4	0,93	0,22	8,21	0,9	0,19	10,76	0,57	0,13	11,28	0,53	0,12	14,78
	MEBF8	0,95	0,21	10,98	0,97	0,22	16,33	0,63	0,16	21,35	0,59	0,12	23,79
	MERB1	1,37	0,3	36,72	1,17	0,26	30,51	0,55	0,13	10,14	0,51	0,13	12,64
	MERB2	0,9	0,2	5,45	0,87	0,21	8,32	0,55	0,14	9,46	0,5	0,11	9,1
	MERB3	0,89	0,21	3,85	0,87	0,22	8,15	0,51	0,13	3,23	0,51	0,12	11,41
	MERB4	0,9	0,21	2,85	0,91	0,19	10,93	0,55	0,12	5,96	0,52	0,11	10,81
	MERB5	0,95	0,22	3,26	0,92	0,18	8,88	0,59	0,13	4,4	0,57	0,14	9,43
	MERB6	0,87	0,2	2,84	0,88	0,19	8,91	0,52	0,12	3,63	0,51	0,11	10,32

Tabela A.9: Resultados da aplicação *Ray Tracing* no ambiente CC-LL.LH.HL.HH

		imagem *5balls*						imagem *room*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	4,68	1,3	28,73	-	-	-	2,37	0,46	33,27
	EstNWS	-	-	-	4,03	2,17	21,19	-	-	-	2,42	0,43	45,77
	MEBF1	5,04	1,81	0,24	3,3	0,84	0,72	2,78	0,83	0,13	1,91	0,43	1,3
	MEBF2	5,04	1,79	0,63	3,29	0,87	1,83	2,68	0,81	0,4	1,83	0,42	1,62
	MEBF4	4,98	1,82	0,95	3,25	0,87	2,09	2,67	0,85	1,16	1,81	0,43	2,78
	MEBF8	4,97	1,79	1,16	3,25	0,84	1,93	2,65	0,81	2,04	1,81	0,44	3,23
	MERB1	5,22	2,05	5,9	3,71	1,68	13,41	2,62	0,79	1,65	1,77	0,41	2,27
	MERB2	4,98	1,82	1,53	3,25	0,87	3,17	2,64	0,83	0,88	1,74	0,42	1,57
	MERB3	4,88	1,67	1,52	3,24	0,85	2,9	2,59	0,81	0,49	1,75	0,4	1,68
	MERB4	4,9	1,98	2,16	3,19	0,84	1,7	2,61	0,87	1,7	1,73	0,41	1,15
	MERB5	4,84	1,71	0,46	3,18	0,81	1,52	2,57	0,82	0,4	1,75	0,41	1,49
	MERB6	4,81	1,7	0,95	3,31	1,03	5,3	2,59	0,81	0,29	1,75	0,41	1,82
8	Est	-	-	-	2,31	0,51	30,27	-	-	-	1,69	0,38	51,95
	EstNWS	-	-	-	1,82	0,55	16,4	-	-	-	1,35	0,32	42,54
	MEBF1	1,92	0,47	1,15	1,63	0,41	3,05	1,1	0,25	0,74	0,97	0,22	3
	MEBF2	1,89	0,48	1,89	1,61	0,4	4,26	1,05	0,24	1,8	0,94	0,21	5,58
	MEBF4	1,87	0,48	2,97	1,58	0,39	4,16	1,04	0,22	3,74	0,94	0,22	7,98
	MEBF8	1,9	0,48	5,25	1,61	0,47	6,17	1,1	0,25	10,17	0,97	0,2	11,5
	MERB1	1,91	0,48	5,55	1,58	0,4	5,14	1,02	0,22	4,13	0,9	0,21	5,8
	MERB2	1,84	0,46	2,53	1,56	0,38	3,71	1,01	0,22	2,74	0,89	0,19	4,99
	MERB3	1,84	0,48	2,04	1,57	0,4	3,51	1	0,22	1,4	0,89	0,19	5,09
	MERB4	1,83	0,45	1,2	1,56	0,39	3,06	0,99	0,21	0,6	0,9	0,2	4,76
	MERB5	1,83	0,46	1,07	1,59	0,43	3,29	1,01	0,22	0,91	0,91	0,22	4,73
	MERB6	1,82	0,45	1,38	1,56	0,39	3,12	0,99	0,22	1,09	0,88	0,2	3,92
16	Est	-	-	-	1,35	0,28	31,81	-	-	-	0,92	0,21	48,43
	EstNWS	-	-	-	1,54	0,37	47,13	-	-	-	0,71	0,16	31,57
	MEBF1	0,98	0,22	3,21	0,91	0,22	4,46	0,61	0,14	2,34	0,62	0,15	7,73
	MEBF2	0,96	0,21	4,26	0,89	0,2	6,89	0,53	0,12	5,84	0,53	0,12	11,09
	MEBF4	0,97	0,23	7,49	0,91	0,22	9,52	0,55	0,13	10,97	0,53	0,12	14,17
	MEBF8	1,01	0,24	11,43	0,98	0,25	15,22	0,61	0,15	20,8	0,57	0,17	21,87
	MERB1	1,19	0,21	24,63	1,09	0,34	25,02	0,52	0,12	6,54	0,51	0,13	12,48
	MERB2	0,99	0,23	9,39	0,87	0,22	6,95	0,53	0,12	9,06	0,5	0,12	10,32
	MERB3	0,94	0,21	3,6	0,87	0,21	6,64	0,5	0,11	2,91	0,5	0,12	10,09
	MERB4	0,96	0,21	4,53	0,9	0,22	7,95	0,52	0,12	4,43	0,52	0,12	11
	MERB5	1	0,22	4,17	0,94	0,23	7,35	0,6	0,16	2,74	0,55	0,14	8,9
	MERB6	0,93	0,21	3,2	0,88	0,21	7,42	0,51	0,11	3,57	0,51	0,13	10,59

Tabela A.10: Resultados da aplicação *Ray Tracing* no ambiente CC-LL.LL.HL.HL

		imagem *5balls*						imagem *room*					
P	**BC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**	**SMT**	**DP**	**IDC**	**CMT**	**DP**	**IDC**
4	Est	-	-	-	4,88	1,77	26,46	-	-	-	2,62	0,77	25,54
	EstNWS	-	-	-	4,25	1,8	15,97	-	-	-	3,13	0,93	56,85
	MEBF1	5	2,4	0,27	3,73	0,67	1,19	2,76	1,12	0,12	2,01	0,71	1,05
	MEBF2	4,89	1,84	0,56	3,69	0,63	1,52	2,77	1,09	0,37	1,92	0,72	1,68
	MEBF4	4,85	1,95	0,98	3,72	0,64	2,48	2,71	1,08	1,01	1,94	0,71	3,13
	MEBF8	4,91	2,24	1,3	3,74	0,71	3,14	2,7	1,1	1,85	2	0,65	7,36
	MERB1	5,48	3,53	10,35	4,58	1,13	18,13	2,71	0,98	3,09	2,06	0,57	9,47
	MERB2	4,88	1,69	1,28	4,25	0,97	10,46	2,69	1,02	1,32	1,89	0,59	3
	MERB3	4,84	1,89	1,3	4,24	0,98	9,91	2,65	1,05	0,58	1,87	0,59	2,7
	MERB4	4,85	2,23	0,93	3,85	0,7	2,24	2,66	1,11	0,43	1,84	0,64	1,88
	MERB5	4,81	2,36	1,54	3,84	0,66	2,74	2,77	1,56	3,38	1,85	0,62	2,02
	MERB6	4,8	2,1	0,77	3,69	0,69	1,26	2,69	1,24	1,44	1,85	0,67	1,49
8	Est	-	-	-	2,54	0,61	26,9	-	-	-	1,59	0,48	48,87
	EstNWS	-	-	-	1,79	0,44	12,48	-	-	-	1,34	0,41	47,49
	MEBF1	1,84	0,46	1,17	1,64	0,37	3,08	0,98	0,27	0,73	0,94	0,25	3,2
	MEBF2	1,81	0,45	2,16	1,61	0,38	4,17	0,94	0,26	2,18	0,89	0,24	4,2
	MEBF4	1,8	0,45	3,38	1,57	0,37	4,76	0,94	0,25	4,87	0,9	0,23	7,26
	MEBF8	1,85	0,45	6,05	1,63	0,41	6,95	0,98	0,28	9,58	0,96	0,26	12,94
	MERB1	2	0,55	12,39	1,76	0,61	14,55	0,92	0,25	4,48	0,88	0,22	6,67
	MERB2	1,82	0,38	4,86	1,59	0,46	5,65	0,91	0,25	3,9	0,87	0,23	5,62
	MERB3	1,81	0,38	4,04	1,59	0,47	5,23	0,9	0,25	2,02	0,86	0,21	4,74
	MERB4	1,76	0,4	1,54	1,57	0,36	4,24	0,9	0,24	0,87	0,85	0,22	3,61
	MERB5	1,77	0,39	1,27	1,59	0,39	4,37	0,91	0,24	0,82	0,87	0,23	3,52
	MERB6	1,76	0,41	1,43	1,55	0,35	3,7	0,9	0,24	1,47	0,86	0,24	3,63
16	Est	-	-	-	2,26	0,53	60,88	-	-	-	1,44	0,33	67,79
	EstNWS	-	-	-	1,57	0,42	48,89	-	-	-	0,82	0,17	41,39
	MEBF1	0,96	0,23	3,29	0,9	0,23	5,43	0,61	0,14	3,03	0,61	0,14	7,33
	MEBF2	0,97	0,24	6,44	0,88	0,23	7,19	0,55	0,14	8,49	0,54	0,14	12,77
	MEBF4	1,01	0,24	11,77	0,9	0,23	10,41	0,58	0,14	14,19	0,56	0,14	16,85
	MEBF8	1,15	0,28	21,66	1,05	0,3	22,22	0,67	0,19	25,08	0,63	0,16	25,99
	MERB1	2,14	0,49	55,85	1,41	0,62	41,32	0,57	0,13	12,9	0,52	0,13	12,98
	MERB2	1,21	0,28	24,98	0,97	0,23	16,65	0,57	0,15	12,86	0,51	0,12	10,53
	MERB3	1,21	0,27	25,21	0,98	0,25	17,67	0,53	0,12	5,88	0,52	0,11	11,37
	MERB4	0,94	0,21	4,04	0,87	0,22	7,13	0,54	0,12	5,8	0,52	0,12	10,25
	MERB5	0,98	0,23	4,4	0,91	0,21	6,56	0,58	0,13	4,6	0,58	0,13	9,54
	MERB6	0,93	0,23	4,7	0,87	0,23	7,38	0,53	0,11	6,4	0,51	0,12	10,12

Tabela A.11: Resultados da aplicação *Ray Tracing* no ambiente CC-LH.LH.HH.HH

Conjunto inicial de tarefas localizado no *cluster*UFF																				
	Com latência inter-*cluster* real										Com latência inter-*cluster* de 250 ms									
BC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC
Est	-	-	-	-	-	8,21	0,55	42,55	1005	995	-	-	-	-	-	7,69	0,10	40,34	1005	995
EstNWS	-	-	-	-	-	4,30	0,18	34,12	1784	216	-	-	-	-	-	3,98	0,05	25,82	1728	272
MEBF1	3,22	0,02	2,81	1578	422	3,07	0,00	3,14	1597	403	3,21	0,03	2,94	1582	418	3,06	0,02	3,08	1604	396
MEBF2	3,32	0,02	4,96	1573	427	3,13	0,01	5,03	1597	403	3,29	0,01	5,05	1573	427	3,15	0,02	6,12	1595	405
MEBF4	3,48	0,02	9,02	1555	445	3,30	0,10	8,59	1579	421	3,47	0,05	9,38	1548	452	3,23	0,06	7,21	1579	421
MEBF8	3,68	0,04	11,62	1523	477	3,65	0,03	14,43	1539	461	3,58	0,06	10,37	1520	480	3,60	0,09	13,52	1528	472
MERB1	4,19	0,03	18,42	1475	525	4,11	0,00	20,05	1490	510	4,06	0,03	17,00	1475	525	4,00	0,02	19,29	1490	510
MERB2	5,00	0,07	32,93	1493	507	4,82	0,13	33,39	1514	486	4,86	0,01	31,34	1489	511	4,65	0,07	31,7	1509	491
MERB3	3,23	0,02	2,93	1577	423	3,07	0,02	2,79	1598	402	3,22	0,01	2,81	1571	429	3,04	0,02	2,85	1595	405
MERB4	3,55	0,14	2,69	1575	425	3,35	0,09	2,40	1600	400	3,53	0,07	2,69	1572	428	3,45	0,07	2,61	1591	409
MERB5	3,59	0,04	4,59	1624	376	3,48	0,10	4,19	1640	360	3,70	0,06	4,23	1617	383	3,56	0,09	4,41	1634	366
MERB6a	3,20	0,03	2,81	1572	428	3,04	0,02	3,18	1592	408	3,22	0,02	3,02	1571	429	3,07	0,08	2,74	1600	400
MERB6b	3,56	0,07	2,93	1615	385	3,48	0,05	4,60	1622	378	3,58	0,02	2,50	1599	401	3,48	0,07	3,75	1612	388
MERB6b-1	3,49	0,04	2,84	1611	389	3,44	0,01	4,96	1631	369	3,60	0,08	3,51	1596	404	3,48	0,04	4,17	1615	385
MEH	3,36	0,11	3,94	1580	420	3,21	0,03	6,09	1583	417	3,31	0,03	3,32	1580	420	3,22	0,14	6	1567	433

Conjunto inicial de tarefas localizado no *cluster*PUC																				
	Com latência inter-*cluster* real										Com latência inter-*cluster* de 250 ms									
BC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC
Est	-	-	-	-	-	13,82	0,03	47,73	1005	995	-	-	-	-	-	13,78	0,02	63,79	1005	995
EstNWS	-	-	-	-	-	5,38	0,55	20,79	168	1832	-	-	-	-	-	4,68	0,28	21,83	248	1752
MEBF1	3,35	0,00	5,30	227	1773	3,34	0,01	5,57	257	1743	3,48	0,01	4,73	410	1590	3,47	0,02	5,93	436	1564
MEBF2	3,33	0,01	4,48	224	1776	3,39	0,10	6,88	256	1744	3,32	0,06	5,79	389	1611	3,37	0,11	8,17	414	1586
MEBF4	3,67	0,00	13,56	224	1776	3,61	0,09	11,17	255	1745	3,26	0,01	5,91	380	1620	3,52	0,11	13,58	416	1584
MEBF8	3,49	0,28	8,82	224	1776	3,60	0,19	12,93	256	1744	3,24	0,04	7,15	381	1619	4,11	0,38	26,70	413	1587
MERB1	7,44	0,27	37,02	490	1510	7,40	0,56	38,18	510	1490	7,28	0,03	51,38	490	1510	7,05	0,01	50,66	510	1490
MERB2	14,23	0,01	55,06	782	1218	13,16	0,02	52,49	805	1195	14,20	0,02	67,12	782	1218	13,14	0,01	64,44	805	1195
MERB3	3,32	0,01	5,58	221	1779	3,31	0,02	6,31	250	1750	3,13	0,00	5,20	355	1645	3,06	0,07	5,01	376	1624
MERB4	4,70	0,12	32,09	162	1187	4,85	0,06	4,86	238	1762	4,22	0,08	3,38	337	1663	4,40	0,12	5,30	355	1645
MERB5	4,81	0,12	17,94	211	1309	4,73	0,19	4,66	235	1765	4,27	0,06	4,21	337	1663	4,30	0,09	4,79	357	1643
MERB6a	3,31	0,02	5,23	224	1776	3,26	0,02	4,49	254	1746	3,09	0,05	4,34	363	1637	3,12	0,02	6,63	389	1611
MERB6b	4,98	0,04	4,08	259	1741	5,24	0,06	4,42	281	1719	4,39	0,03	4,17	368	1632	4,86	0,05	5,63	389	1611
MERB6b-1	5,05	0,16	4,11	258	1742	5,11	0,11	4,79	281	1719	4,47	0,08	3,78	371	1629	4,77	0,06	4,04	391	1609
MEH	3,49	0,02	3,89	237	1763	3,23	0,01	4,18	252	1748	3,27	0,04	4,48	385	1615	3,08	0,03	5,52	389	1611

Tabela A.12: Resultados da aplicação dos *termions* no *grid*

Conjunto inicial de tarefas localizado no *cluster*UFF																				
	Com latência inter-*cluster* real										Com latência inter-*cluster* de 250 ms									
BC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC
Est	-	-	-	-	-	2,88	0,04	44,82	255	245	-	-	-	-	-	2,80	0,03	44,69	255	245
EstNWS	-	-	-	-	-	1,79	0,05	30,44	445	55	-	-	-	-	-	1,68	0,04	27,88	433	67
MEBF1	1,31	0,03	8,16	381	119	1,26	0,03	9,03	390	110	1,31	0,01	7,22	380	120	1,20	0,02	4,07	390	110
MEBF2	1,38	0,02	10,64	380	120	1,36	0,02	12,80	380	120	1,38	0,04	11,45	380	120	1,34	0,02	11,83	380	120
MEBF4	1,38	0,01	10,39	380	120	1,39	0,04	13,55	380	120	1,35	0,01	8,68	380	120	1,36	0,02	11,80	380	120
MEBF8	1,39	0,01	13,22	380	120	1,36	0,01	13,72	380	120	1,40	0,02	14,19	380	120	1,35	0,04	13,43	380	120
MERB1	1,52	0,03	17,29	365	135	1,54	0,03	21,25	365	135	1,50	0,01	16,82	365	135	1,49	0,02	18,81	365	135
MERB2	1,70	0,01	26,65	368	132	1,71	0,03	29,78	372	128	1,71	0,04	27,86	368	132	1,64	0,01	27,26	372	128
MERB3	1,29	0,03	6,88	378	122	1,25	0,02	7,60	384	116	1,32	0,03	7,35	377	123	1,26	0,04	7,37	382	118
MERB4	1,55	0,04	4,26	379	121	1,57	0,07	6,77	380	120	1,51	0,03	5,80	378	122	1,48	0,04	7,09	382	118
MERB5	1,65	0,10	10,35	388	112	1,72	0,05	13,18	394	106	1,66	0,05	4,96	386	114	1,62	0,04	5,44	395	105
MERB6a	1,31	0,03	8,99	389	111	1,23	0,06	7,13	394	106	1,33	0,01	9,57	388	112	1,24	0,03	7,57	394	106
MERB6b	1,69	0,17	11,89	383	117	1,59	0,03	9,91	385	115	1,53	0,03	4,08	382	118	1,54	0,04	7,64	383	117
MERB6b-1	1,59	0,04	10,95	385	115	1,52	0,08	10,03	389	111	1,54	0,03	4,09	383	117	1,53	0,01	8,00	382	118
MEH	1,45	0,02	12,21	380	120	1,85	0,02	27,67	350	150	1,40	0,02	10,71	380	120	1,76	0,05	25,08	350	150
Conjunto inicial de tarefas localizado no *cluster*PUC																				
	Com latência inter-*cluster* real										Com latência inter-*cluster* de 250 ms									
BC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC
Est	-	-	-	-	-	2,95	0,07	42,62	255	245	-	-	-	-	-	2,82	0,02	42,82	255	245
EstNWS	-	-	-	-	-	3,13	0,10	15,27	43	457	-	-	-	-	-	2,54	0,02	18,42	58	442
MEBF1	1,49	0,14	12,87	82	418	1,56	0,02	18,20	87	413	1,37	0,05	9,85	114	386	1,44	0,12	15,07	119	381
MEBF2	1,74	0,30	20,95	91	409	1,97	0,14	31,50	93	407	1,31	0,07	8,91	115	385	1,28	0,00	7,54	118	382
MEBF4	1,40	0,02	8,75	69	431	2,46	0,54	47,19	77	423	1,34	0,02	9,95	112	388	2,02	0,04	39,53	120	380
MEBF8	2,69	0,02	36,83	112	388	3,12	0,09	45,59	120	380	1,32	0,00	9,39	112	388	2,13	0,01	42,58	120	380
MERB1	3,05	0,02	34,79	126	374	3,64	0,13	44,15	135	365	1,98	0,81	24,74	126	374	2,51	0,01	44,65	135	365
MERB2	5,99	0,06	55,30	201	299	7,81	0,05	64,98	208	292	3,89	1,65	51,59	201	299	5,77	0,06	66,57	208	292
MERB3	1,46	0,11	9,80	59	441	1,53	0,10	15,12	66	434	1,26	0,02	6,06	113	387	1,34	0,04	13,25	117	383
MERB4	2,99	0,10	5,80	69	431	3,23	0,05	9,71	75	425	2,27	0,03	3,47	97	403	2,36	0,04	5,10	106	394
MERB5	3,11	0,04	7,63	68	432	3,28	0,20	10,21	74	426	2,29	0,01	3,32	97	403	2,45	0,10	8,13	106	394
MERB6a	1,49	0,13	12,86	64	436	1,57	0,01	17,87	69	431	1,30	0,01	10,73	107	393	1,37	0,16	14,69	112	388
MERB6b	3,03	0,08	2,44	95	405	3,71	0,04	9,35	103	397	2,48	0,05	4,85	122	378	2,89	0,02	3,79	125	375
MERB6b-1	3,02	0,09	2,12	95	405	3,67	0,22	4,30	103	397	2,47	0,04	5,12	121	379	3,00	0,16	7,19	128	372
MEH	1,58	0,02	11,12	71	429	1,37	0,07	6,97	69	431	1,29	0,02	5,81	114	386	1,24	0,01	8,18	111	389

Tabela A.13: Resultados da aplicação *Ray Tracing* no *grid*

Conjunto inicial de tarefas localizado no *cluster*UFF																				
	Com latência inter-*cluster* real										Com latência inter-*cluster* de 250 ms									
BC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC
Est	-	-	-	-	-	22,57	0,06	46,59	10005	9995	-	-	-	-	-	22,56	0,11	46,93	10005	9995
EstNWS	-	-	-	-	-	6,23	0,11	17,54	17771	2229	-	-	-	-	-	6,76	0,09	20,50	17306	2694
MEBF1	4,88	0,03	0,63	18552	1448	4,62	0,03	0,82	18632	1368	4,87	0,03	0,87	18625	1375	4,60	0,03	1,13	18710	1290
MEBF2	4,85	0,01	1,38	18565	1435	4,59	0,00	1,57	18649	1351	4,82	0,00	1,29	18579	1421	4,55	0,00	1,41	18667	1333
MEBF4	4,87	0,00	2,69	18572	1428	4,66	0,06	3,23	18668	1332	4,87	0,01	2,72	18584	1416	4,62	0,01	3,32	18657	1343
MEBF8	5,02	0,04	5,03	18555	1445	4,80	0,05	6,07	18632	1368	4,94	0,08	4,39	18555	1445	4,76	0,01	5,24	18600	1400
MERB1	17,55	0,02	38,77	14825	5175	17,02	0,02	40,29	14990	5010	17,61	0,09	39,98	14825	5175	16,79	0,03	40,56	14990	5010
MERB2	21,18	0,46	50,53	14985	5015	20,12	0,36	51,05	15197	4803	20,60	0,04	49,75	14985	5015	19,53	0,10	50,26	15197	4803
MERB3	4,93	0,06	3,66	18572	1428	4,60	0,02	3,23	18658	1342	4,82	0,05	3,13	18569	1431	4,51	0,02	2,45	18654	1346
MERB4	5,19	0,13	3,38	18589	1411	5,05	0,01	5,11	18657	1343	5,14	0,08	2,75	18560	1440	4,81	0,01	1,92	18652	1348
MERB5	5,49	0,26	7,38	18739	1261	5,21	0,09	6,27	18801	1199	5,71	0,62	8,92	18698	1302	6,63	0,61	24,50	18760	1240
MERB6a	4,77	0,00	1,00	18581	1419	4,48	0,01	1,27	18674	1326	4,72	0,01	0,72	18567	1433	4,43	0,01	0,95	18662	1338
MERB6b	5,03	0,10	1,04	18728	1272	4,86	0,04	1,60	18792	1208	5,23	0,16	3,30	18700	1300	4,92	0,09	2,47	18764	1236
MERB6b-1	5,27	0,25	4,08	18715	1285	4,94	0,09	1,57	18791	1209	5,12	0,03	1,36	18701	1299	4,93	0,07	3,26	18772	1228
MEH	4,99	0,02	1,15	18620	1380	4,75	0,02	2,14	18600	1400	5,00	0,02	1,71	18650	1350	4,68	0,02	1,79	18600	1400
Conjunto inicial de tarefas localizado no *cluster*PUC																				
	Com latência inter-*cluster* real										Com latência inter-*cluster* de 250 ms									
BC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC	SMT	DP	IDC	UFF	PUC	CMT	DP	IDC	UFF	PUC
Est	-	-	-	-	-	38,92	0,01	67,78	10005	9995	-	-	-	-	-	32,70	10,71	61,03	10005	9995
EstNWS	-	-	-	-	-	10,66	0,45	24,26	1639	18361	-	-	-	-	-	7,79	1,67	7,24	2590	17410
MEBF1	6,76	0,04	0,84	1801	18199	6,81	0,02	1,22	1950	18050	9,24	0,16	0,45	2580	17420	9,20	0,15	0,96	2760	17240
MEBF2	4,50	0,03	1,17	1221	18779	4,62	0,04	3,13	1296	18704	8,94	0,16	1,03	2499	17501	8,89	0,18	1,44	2671	17329
MEBF4	4,61	0,01	3,67	1224	18776	4,60	0,01	4,01	1320	18680	6,81	0,18	2,21	1900	18100	6,80	0,07	2,76	2029	17971
MEBF8	4,94	0,00	9,71	1264	18736	5,06	0,22	9,79	1397	18603	6,29	0,02	2,47	1771	18229	6,51	0,21	6,30	1888	18112
MERB1	28,20	3,45	61,58	4830	15170	28,13	1,89	60,93	5010	14990	30,20	0,09	58,21	4830	15170	29,20	0,01	56,48	5010	14990
MERB2	60,32	0,03	71,62	7768	12232	56,16	0,37	69,47	7979	12021	49,10	9,68	63,62	7768	12232	56,62	0,16	65,36	7979	12021
MERB3	7,36	0,90	33,84	1308	18692	7,60	0,36	34,23	1397	18603	7,85	0,00	29,59	1429	18571	7,77	0,01	29,64	1527	18473
MERB4	7,43	0,91	14,68	744	19256	8,02	0,31	21,35	890	19110	6,11	0,02	0,68	1490	18510	6,44	0,16	4,61	1562	18438
MERB5	7,55	0,43	15,21	742	19258	7,97	0,73	19,09	858	19142	6,16	0,13	1,38	1484	18516	6,26	0,11	2,02	1569	18431
MERB6a	4,32	0,01	0,90	1130	18870	4,47	0,26	3,57	1220	18780	5,27	0,00	3,53	1068	18932	5,29	0,01	3,75	1157	18843
MERB6b	6,60	0,11	1,00	732	19268	6,97	0,21	5,07	802	19198	6,47	0,06	1,19	1534	18466	7,74	0,07	14,71	1620	18380
MERB6b-1	6,75	0,13	2,24	748	19252	6,91	0,16	3,44	803	19197	6,49	0,04	1,12	1528	18472	7,45	0,29	11,77	1638	18362
MEH	4,76	0,06	0,80	1203	18797	4,50	0,03	1,35	1250	18750	9,45	0,06	0,43	2325	17675	8,53	0,01	0,93	2278	17722

Tabela A.14: Resultados da aplicação *Sintética* no *grid*