Avaliação de métricas para determinar o grau de heterogeneidade de sistemas computacionais

Fábio Hitoshi Ide

Orientador: ***Prof. Dr. Marcos José Santana***

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências de Computação e Matemática Computacional.

USP – São Carlos

Março de 2008

Avaliação de métricas para determinar o grau de heterogeneidade de sistemas computacionais

Fábio Hitoshi Ide

Agradecimentos

A Deus, pela proteção em todos os momentos.

Aos professores Marcos Santana e Regina Santana pela orientação, pela oportunidade e pela confiança depositada.

Ao professor Paulo pelo acompanhamento/participação constante desde o início do mestrado.

Agradecimentos especiais aos meus pais, Osvaldo e Emiko, pelo incentivo desde o início.

Aos colegas do grupo LaSDPC, amigos nos momentos de estudo e também de descontração.

Às nossas famílias pelo incentivo, que foi essencial para concluirmos essa etapa.

À Universidade de São Paulo, pela ótima estrutura oferecida para a realização do mestrado.

Ao CNPq, pelo apoio financeiro.

RESUMO

Este trabalho avalia de maneira detalhada diferentes parâmetros para a definição de níveis de homogeneidade e heterogeneidade em sistemas computacionais distribuídos. O objetivo é analisar a eficiência da métrica GH – Grau de Heterogeneidade – em relação a diferentes perspectivas. Métricas encontradas na literatura e *benchmarks* de código aberto (reconhecidos pela comunidade científica) são utilizados para quantificar a heterogeneidade do sistema computacional. A métrica GH também é empregada no AMIGO, um ambiente de escalonamento real, para analisar a sua utilização em algoritmos de escalonamento de processos. Os principais resultados obtidos neste trabalho são: a comprovação da estabilidade da métrica GH para determinar o grau de heterogeneidade de plataformas computacionais distribuídas, o uso da métrica GH com sucesso em um ambiente de escalonamento real e o desenvolvimento de um algoritmo de escalonamento adaptativo. Sub-produtos deste trabalho são: um levantamento dos principais *benchmarks* com código aberto e livre disponíveis na literatura, os quais podem ser utilizados em trabalhos futuros no grupo de pesquisa e a continuidade do desenvolvimento do ambiente de escalonamento AMIGO.

ABSTRACT

This work evaluates several parameters in a detailed way for the definition of homogeneity and heterogeneity levels in distributed computing systems. The objective is to analise the GH metric efficiency (heterogeneity degree) according to different perspectives. Metrics found in the literature and open source benchmarks (recognized by the scientific community) are both used to quantify the heterogeneity of the computational system. The GH metric is also employed in the AMIGO, a real scheduling environment, in order to evaluate its use in process scheduling algorithms. The main results obtained in this work are: the verification of the GH metric stability for determining the heterogeneity degree of distributed computing platforms; the use of the GH metric with success in a real scheduling environment and the development of an adaptive scheduling algorithm. By-products of this work are: the highlighting of the main benchmarks with open source available in the literature, which can be used in future works by the research group and the continuity of the development of the AMIGO.

Sumário

ÍNDICE DE FIGURA

ÍNDICE DE TABELA

1 INTRODUÇÃO

1.1 CONTEXTUALIZAÇÃO

A computação paralela surgiu com a necessidade de aumentar a potência computacional do sistema, oferecendo um melhor desempenho a aplicações específicas (Tanenbaum, 1995) (Tanenbaum, 2002). Através do uso de vários processadores trabalhando em conjunto, problemas maiores podem ser solucionados em um tempo de processamento menor, quando comparado este com o mesmo tempo das execuções feitas em máquinas seqüenciais.

Nas últimas décadas várias mudanças ocorreram na área de computação, como por exemplo a maior conectividade dos recursos, as quais permitiram solucionar muitos problemas de modo eficiente a um custo baixo (Tanenbaum, 1995) (Tanenbaum, 2002).

A motivação para a conexão dos diferentes recursos dos sistemas computacionais não é única, variando desde o simples compartilhamento de discos e impressoras à união de uma grande quantidade de processadores. Outros aspectos motivadores são: viabilizar um aumento na potência computacional disponível, melhorar a tolerância à falhas e possibilitar maior mobilidade aos usuários (Tanenbaum, 1995) (Tanenbaum, 2002).

Com o avanço da computação paralela foram criadas muitas maneiras de conectar os recursos do sistema, criando diferentes arquiteturas, cada uma delas apresentando características específicas. Assim, foram propostas algumas taxonomias para agrupar características comuns. As mais conhecidas e populares são a de Flynn (Flynn, 1972, Flynn & Rudd, 1996) e a de Duncan (Duncan, 1990).

Um problema para a disseminação da computação paralela é o alto custo dos equipamentos e a implantação do sistema. Por outro lado, o aumento de desempenho dos computadores pessoais e estações de trabalho, a diminuição do custo dessas máquinas e o surgimento de redes mais eficientes, favoreceram a utilização dos sistemas computacionais distribuídos como plataforma à computação paralela (Tanenbaum, 1995) (Tanenbaum, 2002). De fato, os sistemas computacionais distribuídos têm apresentado vantagens aprimorando-se a fim de fornecer bom desempenho a um custo baixo.

Desde a década de 80, vários trabalhos foram publicados com o objetivo de explorar a potência dos sistemas computacionais distribuídos aliado aos conceitos de computação paralela. Em vista disso, a convergência da área da computação paralela e dos sistemas distribuídos trouxe algumas vantagens, como por exemplo, no que se refere à implementação

da computação paralela, proporcionando redução de custos e utilização adequada dos recursos. Com isso, foi possível a união do custo baixo oferecido pelos sistemas computacionais distribuídos, ao alto desempenho fornecido pelo processamento paralelo, dando origem à computação paralela distribuída (Tanenbaum, 1995) (Tanenbaum, 2002) (Radulescu et. al., 2000) (Bajaj et. al., 2004) (Boyer et. al., 2005).

As vantagens obtidas com a computação paralela distribuída são relevantes, mas alguns novos problemas surgem dessa nova abordagem. Muitas pesquisas estão sendo desenvolvidas levando em consideração os problemas provenientes da utilização da computação paralela distribuída. As pesquisas em andamento abordam problemas em relação aos meios de interconexão, aos protocolos de comunicação, ao escalonamento de processos, entre outros (Kamthe et. al., 2005). Dentre esses temas, o escalonamento de processos destaca-se pela relevância, pois tem uma grande influência no desempenho final do sistema.

Vários objetivos podem ser atingidos a partir de políticas de escalonamento de processos, como por exemplo, o balanceamento de carga, que é responsável por fornecer um equilíbrio no uso dos recursos do sistema, mantendo estabilidade e obtendo melhor desempenho (Lee et. al., 2002).

Índices de carga têm sido utilizados pelas políticas de escalonamento, visando quantificar a carga de trabalho em cada elemento de processamento do sistema. Esses índices considerados na literatura (Ferrari & Zhou, 1987; Kunz, 1991; Mehra, 1993; Schnor et. al., 1996; Xu & Lau, 1997; Dantas & Zaluska, 1998) (Grosu et. al., 2002) (Junwei et. al., 2003) (Bahi et. al., 2003) (Guermouche et. al., 2005) (Baker et. al., 2005) são aplicados tanto em sistemas computacionais homogêneos quanto heterogêneos, descartando assim os vários níveis de heterogeneidade nos sistemas computacionais distribuídos.

A complexidade em se tratar a heterogeneidade dos sistemas, aponta uma lacuna bastante interessante para o desenvolvimento de pesquisas nessa área. Com o preenchimento dessa lacuna, índices de carga mais adequados e, conseqüentemente, melhores níveis de desempenho podem ser atingidos, utilizando-se soluções específicas para sistemas computacionais heterogêneos.

A heterogeneidade do ponto de vista de computação de alto desempenho, inclui modelagem da heterogeneidade, estimativa de desempenho, confiabilidade, escalonamento, balanceamento de carga (Lee et. al., 2002) (Kamthe et. al., 2005).

1.2 MOTIVAÇÃO E OBJETIVOS

O projeto de pesquisa desenvolvido em (Branco, 2005) propõe uma abordagem para determinar se um dado sistema computacional distribuído pode ou não ser tratado como homogêneo. Para isso, foi definido o grau de heterogeneidade do sistema indicando se o sistema está ou não se comportando de modo heterogêneo. A heterogeneidade é analisada levando-se em consideração a dimensão tempo, os aspectos de arquitetura e configuração.

Com relação aos experimentos feitos em (Branco, 2005), realizaram-se estudos de caso para quantificar a heterogeneidade do sistema baseando-se na potência computacional das máquinas. Em (Branco, 2005), entende-se por potência computacional como sendo a velocidade da máquina. O grau de heterogeneidade proposto em (Branco, 2005) faz uso de uma máquina virtual, cujo desempenho equivale à média aritmética das velocidades obtidas nas máquinas pertencentes ao sistema. Outra abstração utilizada em (Branco, 2005) é o desvio padrão absoluto, cujo valor representa a diferença entre as potências computacionais das máquinas existentes no sistema em relação à máquina virtual.

A métrica proposta em (Branco, 2005) é uma métrica geral e flexível. A métrica pode assumir diversos parâmetros, como por exemplo: processador, memória ou rede de comunicação. Além disso, a métrica é independente da unidade de grandeza adotada, seja *MIPS*, *MFPLOS*, bytes, bytes/segundos ou outra. Considerando esse contexto, os resultados apresentados em (Branco, 2005) utilizam como parâmetro apenas um valor genérico para representar a "velocidade do sistema". Esse valor não permite analisar o comportamento da métrica proposta frente às diferentes perspectivas de heterogeneidade vindas do sistema computacional. Espera-se que a heterogeneidade do sistema computacional distribuído também varie em função dos parâmetros escolhidos para representar a heterogeneidade. Espera-se igualmente que a métrica proposta em (Branco, 2005) seja capaz de representar essa variação no grau de heterogeneidade quando forem usados diferentes parâmetros para determinar a velocidade do sistema.

Do ponto de vista do escalonamento de processos, e conseqüentemente da aplicação distribuída, ambas geram demandas específicas sobre os sistemas computacionais. Assim, espera-se que a heterogeneidade apontada em (Branco, 2005) possa ser flexível o bastante para se adequar à essa demanda gerada.

O objetivo deste trabalho é, portanto, realizar um estudo elaborado do grau de heterogeneidade proposto em (Branco, 2005). O estudo visa quantificar a heterogeneidade do sistema levando-se em consideração pontos de vista específicos, isto é, a partir da métrica

proposta em (Branco, 2005) mostrar a viabilidade de novos parâmetros que levem em conta diferentes dispositivos. Os grupos de parâmetros analisados são: processador, memória e rede de comunicação. Dentro de cada grupo, são analisadas características específicas, como por exemplo: unidade de ponto flutuante, espaço de memória disponível, latência de acesso à memória e tempo de acesso à rede de comunicação.

Além disso, é considerado um ambiente de escalonamento adaptativo que aborda o grau de heterogeneidade obtido pelos novos parâmetros.

1.3 ESTRUTURAÇÃO

Este documento apresenta a seguinte organização. O Capítulo 2 apresenta o estudo dos conceitos de homogeneidade e heterogeneidade.

No Capítulo 3 é apresentado o estudo de métricas para a obtenção do grau de heterogeneidade e homogeneidade. O Capítulo 4 apresenta os *benchmarks* encontrados na literatura e a análise dos mesmos.

No Capítulo 5 é apresentado o uso da métrica GH em um ambiente de escalonamento adaptativo. O Capítulo 6 apresenta a conclusão e trabalhos futuros. E finalmente são apresentadas as Referências e o Apêndice A.

2 CONCEITOS DE HETEROGENEIDADE E HOMOGENEIDADE

2.1 CONSIDERAÇÕES INICIAIS

Diferentes referências são encontradas na literatura apresentando estudos que abordam sistemas computacionais homogêneos e heterogêneos. Na maior parte dos casos, definições sobre homogeneidade e heterogeneidade não apresentam um grau de detalhamento adequado (Khokhar et. al., 1993; Ambrosious et. al., 1996; Braun et. al., 1998; Ekemecic, Tartaja & Milutinovic, 1996; Braun, 1999; Chen et. al., 1993; Potter, 1993; Beitz et. al., 2000; Amir et. al., 2000; Abdelzaher & Shin, 2000; Beaumont et. al., 2003).

Para realizar a classificação de um sistema computacional em termos de homogeneidade e heterogeneidade devem ser utilizadas métricas capazes de traduzir com precisão as características arquiteturais, configuracionais e temporais (Branco, 2005).

2.2 HISTÓRICO

O termo homogeneidade tem sido definido de várias maneiras e em áreas de pesquisa diversificadas, tais como a física, matemática, engenharia, computação, entre outras (Khokhar et. al., 1993; Ambrosious et. al., 1996; Braun et. al., 1998; Ekemecic, Tartaja & Milutinovic, 1996; Beitz et. al., 2000; Amir et. al., 2000; Abdelzaher & Shin, 2000; Beaumont et. al., 2003). Apesar da diversidade das definições, observa-se uma certa convergência no que se refere à semântica do termo.

Por muitos anos, a computação executada em sistemas computacionais homogêneos possibilitou desempenho adequado para um grande número de aplicações. A computação homogênea dentro da área de computação, é a que faz uso de uma ou mais máquinas de um mesmo tipo, permitindo que diversas aplicações sejam atendidas de maneira adequada (Khokhar et. al., 1993).

O termo “Sistema Computacional Homogêneo” pode ser visto como um conjunto de máquinas que possuem as mesmas características, incluindo características da arquitetura, dos diversos componentes do sistema e o comportamento temporal das máquinas (Branco, 2005).

Dessa forma, um sistema computacional heterogêneo é aquele que faz uso de diferentes tipos de processadores, velocidades dos processadores, tamanhos de memória, número de processadores (mesmo em máquinas essencialmente paralelas), componentes de processamento, e\ou paradigmas de conectividade que tentam otimizar o desempenho, entre outros. Sendo assim, diferentes tipos de processadores e componentes de processamento

podem envolver processadores vetoriais, *SIMD* e/ou *MIMD*, conexões ponto a ponto, anel, ou até mesmo uma mistura desses meios de comunicação (Radulescu et. al., 2000) (Bajaj et. al., 2004) (Hagras et. al., 2005) (Boyer et. al., 2005) (Branco, 2005).

Quando uma aplicação é inicializada em um sistema computacional heterogêneo, espera-se poder executá-la de uma maneira melhor, em relação a um sistema computacional homogêneo, desde que se conheçam quais máquinas do sistema têm melhores condições para processá-la (Radulescu et. al., 2000) (Bajaj et. al., 2004) (Boyer et. al., 2005). Pode haver uma adequação da carga aos componentes do sistema uma vez que não se considera a equivalência dos processadores, a uniformidade das arquiteturas, fatores estes que contribuem para a simplificação na obtenção de muitas métricas como os índices de carga (Amir et. al., 2000; Abdelzaher & Shin, 2000).

Um exemplo de um sistema computacional heterogêneo pode ser visualizado na Fig. 2.1.

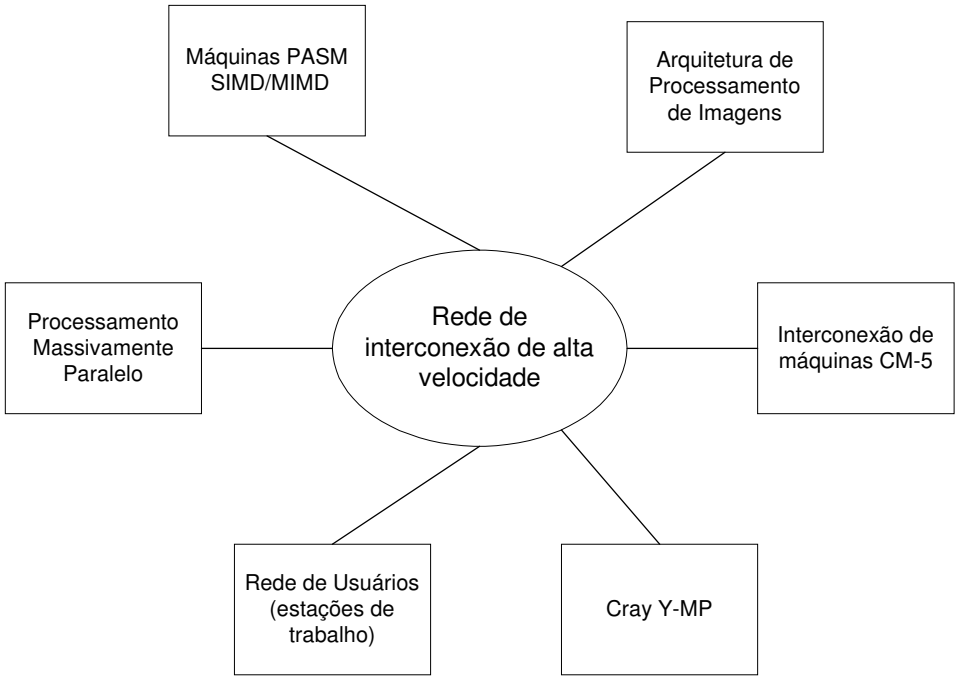

Figura 2. 1 – Um exemplo de um ambiente heterogêneo (Branco, 2005).

Dessa maneira, tendo-se um sistema computacional heterogêneo, as máquinas que possuem diferentes potências computacionais podem ser ordenadas pela "capacidade computacional" (o quanto dela está sendo usado), através de métricas que levem em consideração a heterogeneidade do sistema (Branco, 2005).

A idéia de computação heterogênea busca adequar as tarefas que estão chegando para serem executadas nas melhores máquinas disponíveis no sistema em função da capacidade e carga atual. Partindo de uma tarefa global conhecida como meta-tarefa que poderá ser subdividida em outras tarefas de acordo com as características específicas de cada porção do código, para que cada pedaço possa ser executado pela máquina que melhor convier (Lee et. al., 2002) (Bajaj et. al., 2004) (Boyer et. al., 2005) (Branco, 2005).

A Fig. 2.2 ilustra a idéia de se dividir uma meta-tarefa em tarefas menores que possam ser apropriadas às máquinas que deverão executá-las de uma melhor forma.

Embora o esquema da Fig. 2.2 seja geral, Branco (Branco, 2005) considerou que a fase de análise do código não é realizada para as tarefas que chegam ao sistema, de modo que todos os níveis de paralelismo são explícitos.

Além disso, uma aplicação pode ser representada por um *DAG – Direct Acyclic Graph* (Radulescu et. al., 2000) (Bajaj et. al., 2004) (Hagras et. al., 2005) (Boyer et. al., 2005) (Liu et. al., 2005) (Kamthe et. al., 2005), ou seja, os nós do *DAG* representam tarefas e são rotulados com o custo computacional, isto é, o tempo de execução esperado e os arcos direcionados representam dependência intertarefas, em outras palavras, precedência de tarefa e são rotulados com o custo da comunicação, isto é, o tempo de comunicação esperado entre tarefas. Os rótulos nos nós e nas arestas do *DAG* são considerados determinísticos, pois não variam com o tempo.

2.3 HETEROGENEIDADE POSITIVA OU NEGATIVA

A heterogeneidade pode ser definida através do seguinte aspecto: heterogeneidade positiva ou negativa (Branco, 2005).

A heterogeneidade é considerada positiva quando, por exemplo, dado um conjunto de máquinas com velocidades ou potências computacionais idênticas é adicionado a ele um novo computador com velocidade maior que as máquinas pertencentes ao sistema, neste caso, esse computador ajuda a melhorar o desempenho do sistema.

Considerando a mesma situação descrita anteriormente, mas ao invés de adicionar uma máquina mais rápida é adicionado um computador mais lento em relação às máquinas do conjunto, esse novo computador tende a prejudicar o sistema. Dessa maneira a heterogeneidade pode ser vista como negativa, ou seja, esse computador poderia ser suprimido para não influenciar no desempenho do sistema (Branco et. al., 2003c).

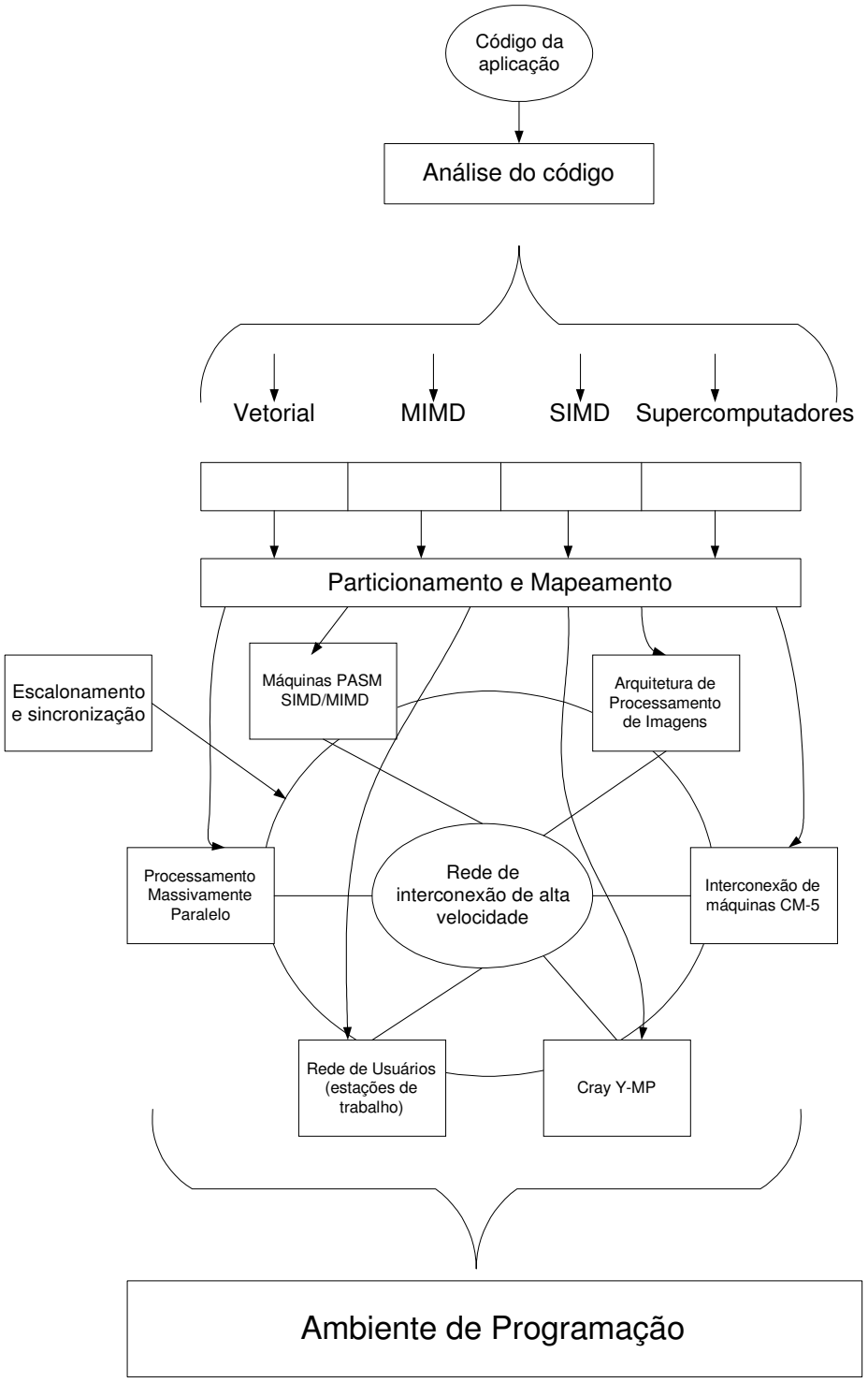

Figura 2. 2 – Representação da partição da meta-tarefa em subtarefas a serem alocadas em máquinas que possam executá-las da melhor maneira possível (Freund & Conwell, 1990).

A heterogeneidade positiva e negativa acima é vista do ponto de vista intuitivo, mas quando essa heterogeneidade depende das características da carga de trabalho, ou seja, se as

aplicações submetidas ao sistema são dependentes uma das outras, a adição de novas máquinas influência no desempenho do sistema. Entretanto, quando as aplicações são independentes, a adição de novos computadores pode não influenciar no sistema.

2.4 HETEROGENEIDADE CONFIGURACIONAL, ARQUITETURAL E TEMPORAL OU DINÂMICA

A heterogeneidade configuracional e arquitetural, ou seja, a heterogeneidade espacial (Lee et. al., 2002) (Kamthe et. al., 2005) pode ser definida através de algumas características principais que dependem da configuração física do sistema computacional distribuído:

- Sistemas computacionais homogêneos tanto em capacidade quanto em compatibilidade de recursos, isto é, todas as máquinas têm a mesma arquitetura e configuração (Branco, 2005).
- Sistemas computacionais homogêneos em termos de compatibilidade, mas heterogêneos em capacidade de recursos, ou seja, todas as máquinas têm a mesma arquitetura, mas configurações diferentes (Branco, 2005).
- Sistemas computacionais heterogêneos que diferem tanto em capacidade quanto em compatibilidade de recursos, em outras palavras, as máquinas são diferentes do ponto de vista de arquitetura e configuração (Branco, 2005).

Além dessas diferenças com relação à heterogeneidade configuracional e arquitetural apresentadas, pode-se afirmar também que existe uma heterogeneidade temporal ou dinâmica. Em um determinado instante de tempo, máquinas arquitetural e configuracionalmente homogêneas em conjunto com as cargas do sistema e os recursos disponíveis, tornam o sistema temporal ou dinamicamente heterogêneas (Branco et. al., 2003a; Branco et. al., 2003b) (Lee et. al., 2002) (Kamthe et. al., 2005).

Analogamente, ambientes heterogêneos podem apresentar uma homogeneidade temporal: em um dado instante de tempo existe a possibilidade de todas as máquinas pertencentes ao sistema juntamente com as cargas do sistema e as capacidades de recursos apresentarem uma homogeneidade temporal.

Em resumo, quando um processo executando em uma máquina é combinado com as características da mesma, produz um conjunto de recursos disponíveis perante as demais máquinas do sistema. Desse modo, quando as máquinas possuem características inicialmente iguais, essas capacidades restantes podem ser diferentes (Branco, 2005).

Além disso, os efeitos da heterogeneidade temporal e espacial podem ser analisadas pelas estratégias de balanceamento de carga (Lee et. al., 2002) e outras políticas de

escalonamento de processos (Kamthe et. al., 2005), a fim de minimizar o tempo médio de execução das tarefas.

Lee (Lee et. al., 2002), propôs uma estratégia de balanceamento de carga para minimizar o tempo médio de execução de uma tarefa em um ambiente computacional espacialmente heterogêneo. A estratégia leva em conta o desvio padrão da potência computacional disponível, além da potência computacional média disponível em cada computador. As tarefas são atribuídas conforme a potência computacional média das máquinas, ou seja, máquinas com potências computacionais mais altas recebem mais tarefas do que as máquinas com potências computacionais mais baixas.

Kamthe (Kamthe et. al., 2005) propôs um escalonamento de *DAG*s estocásticos em um sistema computacional distribuído. Entende-se por *DAG* estocástico, cujos pesos dos nós e das arestas são valores não determinísticos e variam com o tempo. O escalonamento considera o desvio padrão dos rótulos dos nós e das arestas, isto é, a heterogeneidade temporal é utilizada para minimizar o tempo médio de execução de um *DAG* estocástico. Em outras palavras, quando um *DAG* é executado muitas vezes, deverá apresentar variações no tempo médio da tarefa em cada execução em um sistema temporalmente heterogêneo.

2.5 CONSIDERAÇÕES FINAIS

Este capítulo apresentou os conceitos de sistemas computacionais homogêneos e heterogêneos encontrados na literatura, além de abordar definições de heterogeneidade, tais como, heterogeneidade positiva ou negativa, heterogeneidade arquitetural e configuracional e heterogeneidade temporal ou dinâmica.

O uso da heterogeneidade temporal poderá facilitar a confecção de métricas de desempenho que façam uso de características dinâmicas do sistema. Desse modo, todo sistema computacional distribuído considerado heterogêneo pode ter momentos de homogeneidade (Branco, 2005).

Além disso, observa-se que o uso de computação baseada em sistemas computacionais distribuídos tem crescido ao longo das últimas décadas, uma vez que a composição dos sistemas modernos tem variado constantemente com a inserção de novas máquinas, muitas delas heterogêneas.

3 MÉTRICAS PARA OBTENÇÃO DO GRAU DE HETEROGENEIDADE E HOMOGENEIDADE

3.1 CONSIDERAÇÕES INICIAIS

Este capítulo apresenta algumas métricas encontradas na literatura para determinar adequadamente o grau de heterogeneidade de um sistema computacional distribuído (Zhang & Yan, 1995) (Grosu, 1996) (Branco, 2005). Além disso, o capítulo mostra estudos de casos que analisam o comportamento das métricas propostas por (Zhang & Yan, 1995) (Grosu, 1996) (Branco, 2005).

3.2 MÉTRICAS PARA OBTENÇÃO DO GRAU DE HETEROGENEIDADE E HOMOGENEIDADE

O uso de sistemas computacionais heterogêneos oferece uma boa oportunidade para a obtenção de melhor desempenho das aplicações através da atribuição das tarefas ou processos aos processadores, pois o desempenho da aplicação varia em função da plataforma (*hardware* e *software* básico) utilizada (Radulescu et. al., 2000) (Bajaj et. al., 2004) (Branco, 2005) (Hagras et. al., 2005) (Boyer et. al., 2005).

As métricas de heterogeneidade podem ser utilizadas de várias maneiras, por exemplo, os escalonadores de processos podem usar essas métricas para calcular índices de desempenho confiáveis, permitindo melhor decisão de escalonamento (Branco, 2005). Em (Branco, 2005), entende-se por índice de desempenho como sendo uma métrica capaz de fornecer uma imagem da capacidade de trabalho, ou melhor, constitui uma grandeza ilustrando claramente o que pode ser esperado, em termos de desempenho, do elemento em análise.

O grau de heterogeneidade pode ser melhor caracterizado através de métricas de desempenho. Tais métricas, no contexto de sistemas homogêneos, são: *speedup* (Ferrari & Zhou, 1987), eficiência (Mehra, 1993) e tempo de resposta (Zhang & Yan, 1995).

Zhang e Yan (Zhang & Yan, 1995) propuseram alguns modelos e métricas para sistemas computacionais heterogêneos, no qual o sistema pode ser representado por um grafo (M, C), onde $M=\{M_1, M_2, M_3, M_4, M_5, ..., M_n\}$ é considerado um conjunto de máquinas heterogêneas, cada uma possuindo potência computacional que pode ser medida a partir da velocidade da *CPU*, disco e capacidade de memória, e C sendo a rede de comunicação ligando essas máquinas.

Zhang e Yan (Zhang & Yan, 1995) propuseram duas métricas para avaliar a potência computacional existente entre um conjunto de máquinas. Em ambas as medidas foi estipulada uma máquina como sendo a máquina padrão, a partir da qual todas as outras máquinas são comparadas. Neste caso os autores escolheram a máquina mais rápida do sistema.

$$W_i\ (AP) = \frac{S_i\ (AP)}{\max_{i=1}^{n}\{S_i\ (AP)\}} \qquad \textbf{Equação 3.1}$$

onde i=1, ..., n e $S_i(A)$ é a velocidade da máquina M_i para executar a aplicação AP. A velocidade, por exemplo, pode ser definida a partir de um número de operações básicas por unidade de tempo – *MIPS* e a potência computacional, $W_i(AP)$, de cada máquina é representada pela velocidade em relação à máquina padrão, isto é, em relação à máquina mais rápida do sistema.

A segunda métrica proposta é:

$$W_i(AP) = \frac{\min_{i=1}^{n}\{T(AP, M_i)\}}{T(AP, M_i)} \qquad \textbf{Equação 3.2}$$

onde i=1, ..., n e $T(AP, M_i)$ é o tempo de execução da aplicação AP na máquina M_i.

Grosu (Grosu, 1996) ampliou o conceito dessas métricas de modo a escolher a máquina padrão como sendo a máquina mais lenta do sistema.

$$W_i(AP) = \frac{\min_{i=1}^{n}\{S_i\ (AP)\}}{S_i\ (AP)} \qquad \textbf{Equação 3.3}$$

onde i=1, ..., n e $S_i(AP)$ é a velocidade da máquina M_i para executar a aplicação AP e a potência computacional,$W_i(AP)$, de cada máquina é representada pela velocidade em relação à máquina padrão, isto é, em relação à máquina mais lenta do sistema. Deste modo, Grosu (Grosu, 1996) define:

$$W_i\ (AP) = \frac{T(AP, M_i)}{\max_{i=1}^{n}\{T(AP.M_i)\}} \qquad \textbf{Equação 3.4}$$

onde i=1, ..., n e $T(AP, M_i)$ é o tempo de execução da aplicação AP na máquina M_i.

As Eqs. 3.1 e 3.2 formam as bases para o cálculo da potência computacional dos computadores, levando em consideração que a máquina padrão é a estação de trabalho mais rápida do sistema, sendo chamada W_i^f (f - fast). Por outro lado, as Eqs. 3.3 e 3.4 são utilizadas

para o cálculo da potência computacional das máquinas, levando em consideração que a máquina padrão é a máquina mais lenta do sistema, sendo representada por W_i^s (s - slow).

Zhang e Yan (Zhang & Yan, 1995) propuseram quatro modos de quantificar a heterogeneidade de um sistema baseado nos valores de W. O primeiro e o segundo caso fazem uso do desvio padrão H_1, que pode ser calculado baseado na potência computacional em relação à máquina mais rápida ou mais lenta do sistema,

$$H_1 = \sqrt{\frac{\sum_{i=1}^{n} (W_{med} - W_i)^2}{n}} \qquad \textbf{Equação 3.5}$$

ou do desvio padrão absoluto, chamado H_2, também calculado baseado na potência computacional em relação à máquina mais rápida ou mais lenta do sistema,

$$H_2 = \frac{\sum_{i=1}^{n} |W_{med} - W_i|}{n} \qquad \textbf{Equação 3.6}$$

onde $W_{med} = \frac{\sum_{i=1}^{n} W_i}{n}$.

Em ambos, os valores de H_1 e H_2 são observados e analisados de maneira uniforme, fazendo uso da média para obter tanto o desvio padrão quanto o desvio padrão absoluto. Segundo Zhang e Yan (Zhang & Yan, 1995), essas métricas são inadequadas quando existe uma diferença razoável entre as potências computacionais das máquinas, uma vez que tanto H_1 quanto H_2 não reflete o efeito causado pela presença de máquinas mais rápidas ou mais lentas.

Baseando-se no modelo de sistema computacional heterogêneo proposto por Zhang e Yan (Zhang & Yan, 1995), Xiao (Xiao et. al., 2000) estendeu as Eqs. 3.1 e 3.5, propondo duas métricas: uma para avaliar a potência computacional existente entre um conjunto de máquinas e a outra para quantificar a heterogeneidade do sistema levando-se em consideração o tamanho da memória:

$$W_{men}(j) = \frac{RAM_j}{\max_{i=1}^{k} RAM_i} \qquad \textbf{Equação 3.7}$$

onde RAM_j é a quantidade de espaço de memória disponível do usuário no computador j para j=1, ..., k. k representa o número total de computadores no sistema.

$$H_{men} = \sqrt{\frac{\sum_{i=1}^{k} (W_{med} - W_{men}(j))^2}{k}}$$ **Equação 3.8**

onde $W_{med} = \frac{\sum_{j=1}^{k} W_{men}(j)}{k}$, é a média da potência computacional do sistema em relação a memória. Valores altos nas Eqs. 3.5 e 3.8 em um sistema distribuído correspondem à alta variação nas capacidades de CPU e memória entre computadores diferentes. Um sistema homogêneo é caracterizado por valor zero nas Eqs. 3.5 e 3.8.

Levando em consideração as restrições das métricas H_1 e H_2, Zhang e Yan (Zhang & Yan, 1995) propuseram uma terceira métrica, H_3, calculada a partir da potência computacional das máquinas em relação à máquina mais rápida do sistema:

$$H_3 = \frac{\sum_{i=1}^{n} (1 - W_i^f(A))}{n}$$ **Equação 3.9**

De modo similar, Grosu (Grosu, 1996) define H_4 baseado na potência computacional das máquinas em relação à máquina mais lenta do sistema:

$$H_4 = \frac{\sum_{i=1}^{n} (1 - W_i^s(A))}{n}$$ **Equação 3.10**

Em H_3, a potência computacional da máquina mais rápida do sistema é igual a 1, enquanto que em H_4, a máquina mais lenta é que possui potência computacional igual a 1. Desse modo, H_3 representa a distância, em termos de potência computacional, entre cada máquina e a máquina mais rápida do sistema, e H_4 calcula a mesma distância entre cada máquina e a máquina mais lenta do sistema.

Baseando-se no modelo computacional heterogêneo proposto por Zhang e Yan (Zhang & Yan, 1995), Al-Jaroodi (Al-Jaroodi et. al., 2003) propôs a seguinte alteração: cada máquina do sistema contém um ou mais processadores e cada máquina pode executar simultaneamente vários processos. Em vista disso, Al-Jaroodi (Al-Jaroodi et. al., 2003) estenderam as Eqs. 3.2 e 3.9 e propôs uma métrica para quantificar a heterogeneidade do sistema, levando em consideração o número de processadores utilizados em cada máquina:

$$H = \frac{\sum_{i=1}^{n} (m_i * (1 - W_i^f(A)))}{n}$$ **Equação 3.9**

onde m_i é o número de processadores ou tarefas usados na máquina M_i. Multiplicando-se o número de processadores m_i pela potência computacional W_i da máquina M_i, pode-se representar a potência computacional esperada da máquina usando os m_i processadores, melhor do que em um único processador.

Baseado em experimentos, Grosu (Grosu, 1996) determina que a métrica H_4 é mais apropriada do que H_3. Mas, alguns estudos de caso realizados em (Branco, 2005) revelaram que essa afirmação não é correta para todas as situações como pode ser observado nas Tabs. 3.1 a 3.3 e correspondentes Figs. 3.1 a 3.3.

A Tab. 3.1 mostra a configuração inicial 1 no qual a maioria das máquinas são homogêneas até atingir a configuração final 9 onde as máquinas são heterogêneas.

Tabela 3. 1 – Potência computacional das máquinas estão próximas de 1 e são gradualmente decrementadas

*	$W_{(M1)}$	$W_{(M2)}$	$W_{(M3)}$	$W_{(M4)}$	$W_{(M5)}$	$W_{(M6)}$	$W_{(M7)}$	$W_{(M8)}$	$W_{(M9)}$	$W_{(M10)}$	H_1	H_2	H_3	H_4
1	1.00	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.03	0.01	0.09	0.01
2	1.00	0.75	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.05	0.02	0.10	0.15
3	1.00	0.75	0.65	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.09	0.06	0.13	0.24
4	1.00	0.75	0.65	0.55	0.90	0.90	0.90	0.90	0.90	0.90	0.13	0.11	0.16	0.32
5	1.00	0.75	0.65	0.55	0.40	0.90	0.90	0.90	0.90	0.90	0.18	0.15	0.21	0.45
6	1.00	0.75	0.65	0.55	0.40	0.35	0.90	0.90	0.90	0.90	0.21	0.19	0.27	0.45
7	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.90	0.90	0.90	0.25	0.22	0.33	0.54
8	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.90	0.90	0.28	0.25	0.41	0.64
9	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.10	0.90	0.29	0.26	0.49	0.67

* Configuração

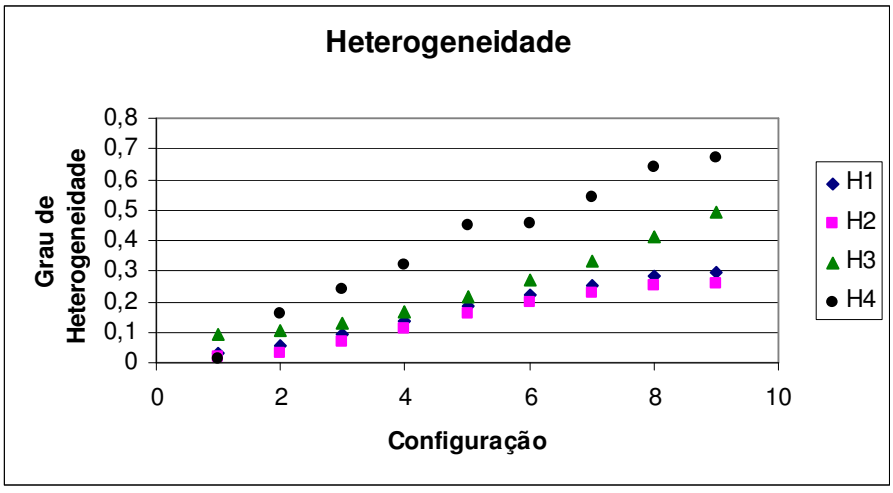

Figura 3. 1 – Grau de Heterogeneidade quando todas as potências computacionais das máquinas estão próximas de 1

A Fig. 3.1 é equivalente a Tab 3.1, ela mostra que todas as métricas são satisfatórias, isto é, quando a maioria das potências computacionais das máquinas estão próximas a 1, o

grau de heterogeneidade é baixo, indicando que o sistema é mais homogêneo do que heterogêneo e a medida que o sistema se torna mais heterogêneo do que homogêneo, o grau de heterogeneidade aumenta, conforme aumenta a variação das potências computacionais (Branco, 2005).

A Tab. 3.2 também mostra a configuração inicial 1 onde a maioria das máquinas são homogêneas até atingir a configuração final 9 na qual as máquinas são heterogêneas.

Tabela 3. 2 – Todas as potências computacionais das máquinas são iniciadas distantes do valor da potência computacional da máquina mais rápida e são gradualmente incrementadas, mas mantendo uma distância da máquina mais rápida

*	$W_{(M1)}$	$W_{(M2)}$	$W_{(M3)}$	$W_{(M4)}$	$W_{(M5)}$	$W_{(M6)}$	$W_{(M7)}$	$W_{(M8)}$	$W_{(M9)}$	$W_{(M10)}$	H_1	H_2	H_3	H_4
1	1.00	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.09	0.45	0.05
2	1.00	0.75	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.16	0.12	0.42	0.08
3	1.00	0.75	0.65	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.12	0.41	0.10
4	1.00	0.75	0.65	0.55	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.12	0.40	0.11
5	1.00	0.75	0.65	0.55	0.40	0.50	0.50	0.50	0.50	0.50	0.16	0.12	0.41	0.27
6	1.00	0.75	0.65	0.55	0.40	0.35	0.50	0.50	0.50	0.50	0.17	0.13	0.43	0.33
7	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.50	0.50	0.50	0.20	0.15	0.45	0.47
8	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.50	0.50	0.23	0.18	0.49	0.61
9	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.10	0.50	0.26	0.22	0.53	0.66

* Configuração

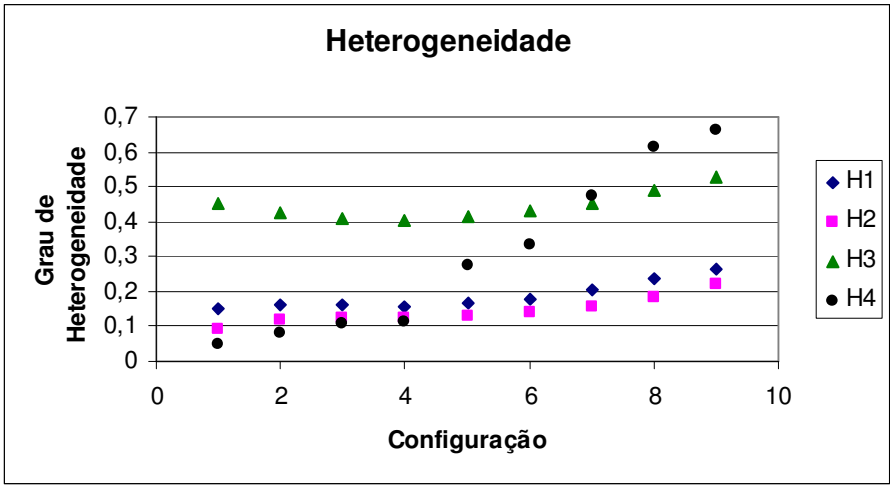

Figura 3. 2 – Grau de Heterogeneidade quando todas as potências computacionais das máquinas estão distantes de 1

A Fig. 3.2 corresponde a Tab. 3.2, neste experimento, a métrica H_3 não representa o quão heterogêneo é o sistema, por outro lado, a métrica H_4 produz bons resultados. Tomando como base somente esses resultados, poder-se-ia chegar à conclusão que a métrica H_4 é a que melhor caracteriza a heterogeneidade de um sistema. Esta inferência é equivocada, uma vez que mostra apenas uma visão parcial e os resultados não podem ser generalizados (Branco, 2005).

A Tab. 3.3 mostra duas situações similares, as configurações 1 e 2 e as configurações 3 e 4.

Tabela 3. 3 – Quatro casos diferentes são apresentados nesta tabela: 1) uma máquina com potência computacional alta e todas as outras com potência computacional baixa; 2) uma máquina com potência computacional baixa e todas as outras com potência computacional alta; 3) duas configurações na qual metade das máquinas possui potência computacional abaixo da média e metade das máquinas possui potência computacional acima da média.

*	$W_{(M1)}$	$W_{(M2)}$	$W_{(M3)}$	$W_{(M4)}$	$W_{(M5)}$	$W_{(M6)}$	$W_{(M7)}$	$W_{(M8)}$	$W_{(M9)}$	$W_{(M10)}$	H_1	H_2	H_3	H_4
1	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.15	0.09	0.05	0.45
2	1.00	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.09	0.45	0.05
3	1.00	1.00	1.00	1.00	1.00	0.50	0.50	0.50	0.50	0.50	0.25	0.25	0.25	0.25
4	1.00	1.00	1.00	1.00	1.00	0.30	0.30	0.30	0.30	0.30	0.35	0.35	0.35	0.35

* Configuração

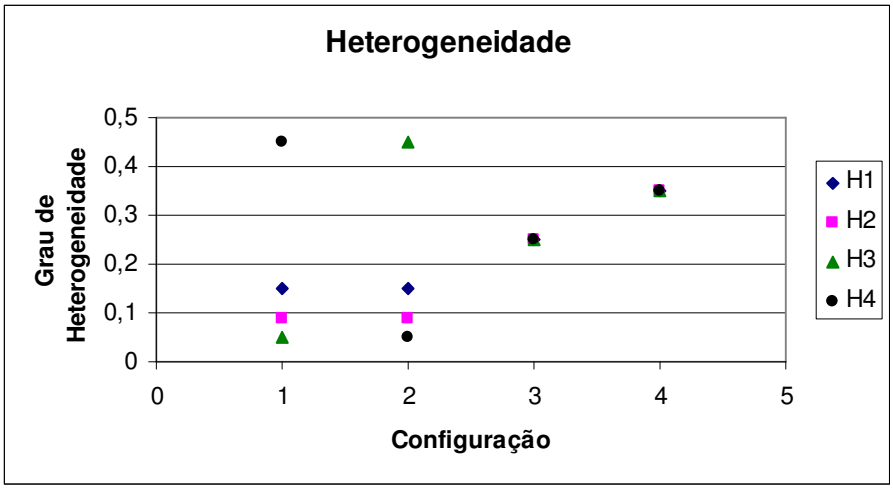

Figura 3. 3 – Grau de Heterogeneidade na qual 4 situações são impostas

A Fig. 3.3 equivale a Tab. 3.3, ela mostra que existem dois casos: 1) 9 máquinas com potência computacional igual a 1 e uma máquina com potência computacional igual a 0,5; 2) 9 máquinas com potência computacional igual a 0,5 e 1 máquina com potência computacional igual a 1, nas quais os graus de heterogeneidade em H_3 e H_4 deveriam ser similares, mas são contraditórios. Ou seja, as configurações 1 e 2, intuitivamente, são mais homogêneas do que heterogêneas.

Por esta razão, Branco (Branco, 2005) buscou averiguar a incompatibilidade dos resultados apresentados pelas métricas H_3 e H_4, e realizou uma análise matemática dessas métricas. A análise demonstrou que ambas as equações fazem referência ao mesmo tipo de medida e que essas medidas não são complementares ou contraditórias (Branco, 2005).

3.2.1 Estudos de Caso

Para averiguar a veracidade dos resultados analisados anteriormente, quatro estudos de caso foram realizados. O primeiro estudo apresenta um sistema composto por 10 máquinas onde 9 têm velocidades altas e idênticas e uma tem velocidade baixa. No segundo estudo, é considerado um sistema com 9 máquinas de velocidades baixas e uma máquina de velocidade alta. O terceiro e o quarto estudos apresentam uma análise de 10 máquinas em um sistema computacional heterogêneo. Os resultados desses estudos são apresentados nas Tabs. 3.4, 3.5, 3.6 e 3.7 (Branco, 2005).

Considerando que H_3 e H_4 produzem resultados distintos quando aplicados ao mesmo sistema, foi averiguada a necessidade de uma heurística que considere a média das velocidades das máquinas. Por exemplo, se mais de 50% das velocidades das máquinas estiverem abaixo da média, então a métrica H_4 deve ser escolhida para calcular o grau de heterogeneidade do sistema, senão, se mais de 50% das velocidades das máquinas estão acima da média, a métrica H_3 deve ser escolhida para calcular o grau de heterogeneidade.

Baseando-se nessa heurística e partindo do cálculo da média das velocidades, 650 no primeiro estudo de caso (Tab. 3.4), e verificando que este valor se aproxima do valor da máquina mais rápida, 700, o grau de heterogeneidade deve ser calculado pela métrica H_3 e o valor da potência computacional das máquinas deve ser calculado em relação à máquina mais rápida do sistema. Isso vem ao encontro dos resultados obtidos intuitivamente, já que 9 das 10 máquinas são idênticas e dessa maneira, o sistema deve estar mais próximo de ser homogêneo do que heterogêneo (Branco, 2005).

Tabela 3. 4 – Primeiro estudo de caso com 9 máquinas rápidas e 1 máquina lenta

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade(Si)	700.00	700.00	700.00	700.00	700.00	700.00	700.00	700.00	700.00	200.00	650.00
Carga (wi)fast	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.28	–
Carga(wi)slow	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	1.00	–
Média Fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	–	Média slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	–	H4_slow	–	–
0.92	0.21	0.12		0.35	0.21	0.12	0.07		0.64		

No segundo estudo (Tab 3.5), considerando a média das velocidades das máquinas, 250, e observando que essa média está mais próxima do valor da máquina mais lenta do sistema, 200, o cálculo da heterogeneidade deve ser efetuado a partir da métrica H_4, e os valores das potências computacionais deverão ser considerados em relação à máquina mais lenta do sistema. Neste caso, os resultados obtidos vêm ao encontro dos resultados intuitivos,

uma vez que 9 das 10 máquinas são idênticas e o sistema deverá ser considerado mais homogêneo do que heterogêneo (Branco, 2005).

Tabela 3. 5 – Segundo estudo de caso com 9 máquinas lentas e 1 máquina rápida

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	700.00	250.00
Carga (wi)fast	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	0.28	1.00	_
Carga (wi)slow	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.28	_
Média fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	_	Média Slow	Desvio Padrão Slow	Desvio Padrão Absoluto Slow	H3_fast	_	H4_slow	_	_
0.35	0.21	0.12		0.92	0.21	0.12	0.64		0.07		

No terceiro estudo (Tab 3.6), as velocidades são definidas objetivando um alto grau de heterogeneidade em um sistema de 10 máquinas, que a partir da heurística e intuitivamente pode-se ver que a melhor métrica é H_4 (Branco, 2005).

Tabela 3. 6 – Terceiro estudo de caso com 10 máquinas de velocidades diferentes e com um alto grau de heterogeneidade

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	100.00	200.00	300.00	400.00	500.00	600.00	700.00	800.00	900.00	1000.00	550.00
Carga (wi) fast	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	1.00	_
Carga (wi) slow	1.00	0.50	0.33	0.25	0.20	0.16	0.14	0.12	0.11	0.10	_
Média fast	Desvio Padrão fast	Desvio Padrão Absoluto Fast	_	Média slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
0.55	0.28	0.25		0.29	0.26	0.18	0.45		0.70		

No quarto estudo (Tab 3.7), na qual as velocidades das máquinas são definidas aleatoriamente em um sistema de 10 máquinas, pode ser concluído intuitivamente que o cálculo do grau de heterogeneidade de um sistema é obtido a partir de H_4.

Tabela 3. 7 – Quarto estudo de caso com 10 máquinas diferentes na qual as velocidades são aleatórias

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	540.00	760.00	210.00	115.00	700.00	150.00	930.00	300.00	425.00	66.00	419.60
Carga (wi) fast	0.58	0.81	0.22	0.12	0.75	0.16	1.00	0.32	0.45	0.07	_
Carga (wi) slow	0.12	0.08	0.31	0.57	0.09	0.44	0.07	0.22	0.15	1.00	_
Média fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	_	Média slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
0.45	0.30	0.27		0.30	0.27	0.20	0.54		0.69		

No entanto, usar somente a média como parâmetro para escolha da métrica a ser utilizada para obter o grau de heterogeneidade do sistema não é suficiente, pois, como observado no terceiro e no quarto estudos de caso, o número de máquinas com velocidades

acima e abaixo da média das velocidades é o mesmo e a escolha de uma ou de outra métrica não estaria levando em consideração as distâncias das máquinas em relação à máquina padrão, o que deixa evidente a necessidade da utilização ou proposição de nova métrica (Branco, 2005).

3.2.2 Modelando a Heterogeneidade

Segundo Zhang e Yan (Zhang & Yan, 1995), uma expressão que quantifique a heterogeneidade de um sistema deve levar em consideração não somente a variação da capacidade das máquinas, mas também o efeito causado no sistema pela presença de máquinas mais lentas e mais rápidas.

O estudo de equações que avaliem o grau de heterogeneidade de um sistema foi alvo de investigação em (Branco, 2005), uma vez que as métricas anteriormente apresentadas e avaliadas não apresentaram resultados satisfatórios.

Para suprir as deficiências apresentadas pelas métricas analisadas na seção 3.2, Branco (Branco, 2005) propôs uma métrica que quantifica a heterogeneidade de modo a refletir não somente a variação da potência computacional, mas também os efeitos dinâmicos do sistema. Isto é, inspecionou qual o impacto da presença de máquinas mais rápidas e mais lentas no sistema e qual o impacto em se retirar essas máquinas do sistema a fim de torná-lo homogêneo.

Alguns estudos de casos foram realizados e levaram em consideração as equações propostas por Zhang e Yan (Zhang & Yan, 1995) e Grosu (Grosu, 1996) fazendo uso, respectivamente, da máquina média e da máquina mediana como sendo a máquina padrão ou máquina de referência para o cálculo da potência computacional, W. Essa adoção não proporcionou resultados adequados.

No caso da adoção da média, os valores de H_3 são sempre iguais a zero para qualquer conjunto de dados, e o valor de H_4 apresenta valores negativos, não indicando adequadamente o grau de heterogeneidade do sistema.

A Tab 3.8 (Branco, 2005) apresenta uma configuração na qual apenas uma máquina difere das demais, entretanto, as métricas H_3 e H_4 mostram que o sistema é totalmente homogêneo, sendo que não é verdade.

Tabela 3. 8 – Primeiro estudo de caso com 9 máquinas rápidas e 1 máquina lenta (fazendo uso da média como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade(Si)	700.00	700.00	700.00	700.00	700.00	700.00	700.00	700.00	700.00	200.00	650.00
Carga (wi)fast	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	1.07	0.30	_
Carga(wi)slow	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.92	3.25	_
Média Fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	_	Média slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
1.00	0.23	0.13		1.16	0.69	0.41	0.00		-0.16		

A Tab 3.9 (Branco, 2005) também apresenta uma configuração na qual apenas uma máquina difere das demais, entretanto, as métricas H_3 e H_4 mostram que o sistema é totalmente homogêneo, sendo que não é verdade.

Tabela 3. 9 – Segundo estudo de caso com 9 máquinas lentas e 1 máquina rápida (fazendo uso da média como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	700.00	250.00
Carga (wi)fast	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	0.80	2.80	_
Carga (wi)slow	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25	1.25	0.35	_
Média fast	Desvio PadrãoFast	Desvio Padrão Absoluto fast	_	Média Slow	Desvio Padrão Slow	Desvio Padrão Absoluto Slow	H3_fast	_	H4_slow	_	_
1.00	0.60	0.36		1.16	0.26	0.16	0.00		-0.16		

A Tab 3.10 (Branco, 2005) apresenta uma configuração heterogênea e as métricas H_3 e H_4 mostram que o sistema é totalmente homogêneo, sendo que não é verdade.

Tabela 3. 10 – Terceiro estudo de caso com 10 máquinas de velocidades diferentes e com um alto grau de heterogeneidade (fazendo uso da média como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	100.00	200.00	300.00	400.00	500.00	600.00	700.00	800.00	900.00	1000.00	550.00
Carga (wi) fast	0.18	0.36	0.54	0.72	0.90	1.09	1.27	1.45	1.63	1.81	_
Carga (wi) slow	5.50	2.75	1.83	1.37	1.10	0.91	0.78	0.68	0.61	0.55	_
Média fast	Desvio Padrão fast	Desvio Padrão Absoluto Fast	_	Média Slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
1.00	0.52	0.45		1.61	1.44	1.03	0.00		-0.61		

A Tab 3.11 (Branco, 2005) também apresenta uma configuração heterogênea e as métricas H_3 e H_4 mostram que o sistema é totalmente homogêneo, sendo que não é verdade.

Tabela 3. 11 – Quarto estudo de caso com 10 máquinas diferentes na qual as velocidades são aleatórias (fazendo uso da média como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	540.00	760.00	210.00	115.00	700.00	150.00	930.00	300.00	425.00	66.00	419.60
Carga (wi) fast	1.28	1.81	0.50	0.27	1.66	0.35	2.21	0.71	1.01	0.15	_
Carga (wi) slow	0.77	0.55	1.99	3.64	0.59	2.79	0.45	1.39	0.98	6.35	_
Média fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	_	Média s-low	Desvio Pa-drão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
1.00	0.68	0.59		1.95	1.77	1.32	0.00		-0.95		

A mediana, quando adotada como máquina padrão propiciou resultados semelhantes aos da média, isto é, H_3 e H_4 apresentaram valores próximos de zero.

A Tab 3.12 (Branco, 2005) apresenta a mesma configuração da Tab. 3.8 e H_3 obtém um valor coerente, mas H_4 não, indicando que o sistema é totalmente homogêneo.

Tabela 3. 12 – Primeiro estudo de caso com 9 máquinas rápidas e 1 máquina lenta (fazendo uso da mediana como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade(Si)	700.00	700.00	700.00	700.00	700.00	700.00	700.00	700.00	700.00	200.00	650.00
Carga (wi)fast	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.28	_
Carga(wi)slow	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	3.50	_
Média Fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	_	Média slow	Desvio Padrão slow	Desvio Pa-drão Abso-luto slow	H3_fast	_	H4_slow	_	_
0.92	0.21	0.12		1.25	0.75	0.45	0.07		-0.25		

A Tab 3.13 (Branco, 2005) também apresenta a mesma configuração da Tab. 3.9 e H_4 obtém um valor coerente, mas H_3 não, indicando que o sistema é totalmente homogêneo.

Tabela 3. 13 – Segundo estudo de caso com 9 máquinas lentas e 1 máquina rápida (fazendo uso da mediana como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	200.00	700.00	250.00
Carga (wi)fast	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	.,50	_
Carga (wi)slow	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.28	_
Média fast	Desvio Pa-drãoFast	Desvio Padrão Absoluto fast	_	Média Slow	Desvio Pa-drão Slow	Desvio Padrão Absoluto Slow	H3_fast	_	H4_slow	_	_
1.25	0.75	0.45		0.92	0.21	0.12	-0.25		0.07		

A Tab 3.14 (Branco, 2005) apresenta a mesma configuração da Tab. 3.10 e os valores de H_3 e H_4 são inadequados, indicando que o sistema é totalmente heterogêneo.

Tabela 3. 14 – Terceiro estudo de caso com 10 máquinas de velocidades diferentes e com um alto grau de heterogeneidade (fazendo uso da mediana como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	100.00	200.00	300.00	400.00	500.00	600.00	700.00	800.00	900.00	1000.00	550.00
Carga (wi) fast	0.18	0.36	0.54	0.72	0.90	1.09	1.27	1.45	1.63	1.81	_
Carga (wi) slow	5.50	2.75	1.83	1.37	1.10	0.91	0.78	0.68	0.61	0.55	_
Média fast	Desvio Padrão fast	Desvio Padrão Absoluto Fast	_	Média slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
1.00	0.52	0.45		1.61	1.44	1.03	0.00		-0.61		

A Tab 3.15 (Branco, 2005) também apresenta a mesma configuração da Tab. 3.11 e os valores de H_3 e H_4 são inadequados, indicando que o sistema é totalmente heterogêneo.

Tabela 3. 15 – Quarto estudo de caso com 10 máquinas diferentes na qual as velocidades são aleatórias (fazendo uso da mediana como máquina de referência)

	Máquina1	Máquina2	Máquina3	Máquina4	Máquina5	Máquina6	Máquina7	Máquina8	Máquina9	Máquina10	Média
Velocidade (Si)	540.00	760.00	210.00	115.00	700.00	150.00	930.00	300.00	425.00	66.00	419.60
Carga (wi) fast	1.48	2.09	0.57	0.31	1.93	0.41	2.56	0.82	1.17	0.18	_
Carga (wi) slow	0.67	0.48	1.73	3.15	0.52	2.42	0.39	1.21	0.85	5.50	_
Média fast	Desvio Padrão Fast	Desvio Padrão Absoluto fast	_	Média slow	Desvio Padrão slow	Desvio Padrão Absoluto slow	H3_fast	_	H4_slow	_	_
1.57	0.78	0.69		1.69	1.53	1.14	-0.15		-0.69		

Como os estudos mostraram-se inadequados, Branco (Branco, 2005) optou por utilizar outra métrica. A métrica a ser utilizada é simples e busca averiguar a dispersão das máquinas em torno de um padrão, como efetuado em Zhang e Yan (Zhang & Yan, 1995) e Grosu (Grosu, 1996), mas procurando-se uma referência mais adequada que substitua a máquina mais rápida ou a mais lenta. Essa nova referência leva em consideração as distâncias das máquinas que compõem o sistema para a nova máquina padrão.

3.2.3 Modelo para obtenção do grau de heterogeneidade

A métrica proposta por Branco (Branco, 2005) considera uma máquina virtual cuja velocidade é a média das velocidades observadas no sistema. O desvio padrão absoluto é utilizado para averiguar a dispersão das máquinas. Neste caso, entende-se por desvio padrão absoluto como sendo a distância entre as velocidades computacionais das diferentes máquinas que compõem o sistema para a máquina virtual.

O grau de heterogeneidade de um dado sistema pode ser obtido a partir de:

$$GH = \frac{\sum_{i=1}^{n} \left| X_i - \overline{X} \right|}{n.\overline{X}} \qquad \textbf{Equação 3.10}$$

A velocidade média é ajustada dinamicamente para cada conjunto, contemplando as variações ocorridas quando são inseridas mais máquinas, sejam elas com velocidades altas ou baixas (Branco et. al., 2003a; Branco et. al., 2003c).

Isto é exatamente o que não ocorre com as métricas anteriores, já que consideram sempre a máquina de velocidade mais alta ou mais baixa, restringindo a flexibilidade da máquina virtual, conseqüentemente, o ajuste correto do grau de heterogeneidade.

As Tabs. 3.16 a 3.18 e Figs. 3.4 a 3.6 apresentam os mesmos estudos de caso já apresentados anteriormente, mas acrescentados da métrica proposta por Branco (Branco, 2005).

A Tab. 3.16 mostra a configuração inicial 1 no qual a maioria das máquinas são homogêneas até atingir a configuração final 9 onde as máquinas são heterogêneas.

Tabela 3. 16 – Potência computacional das máquinas é iniciada próximas de 1 e gradualmente decrementada.

*	$W_{(M1)}$	$W_{(M2)}$	$W_{(M3)}$	$W_{(M4)}$	$W_{(M5)}$	$W_{(M6)}$	$W_{(M7)}$	$W_{(M8)}$	$W_{(M9)}$	$W_{(M10)}$	H_1	H_2	H_3	H_4	GH
1	1.00	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.03	0.01	0.09	0.01	0.01
2	1.00	0.75	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.05	0.02	0.10	0.15	0.03
3	1.00	0.75	0.65	0.90	0.90	0.90	0.90	0.90	0.90	0.90	0.09	0.06	0.13	0.24	0.07
4	1.00	0.75	0.65	0.55	0.90	0.90	0.90	0.90	0.90	0.90	0.13	0.11	0.16	0.32	0.13
5	1.00	0.75	0.65	0.55	0.40	0.90	0.90	0.90	0.90	0.90	0.18	0.15	0.21	0.45	0.20
6	1.00	0.75	0.65	0.55	0.40	0.35	0.90	0.90	0.90	0.90	0.21	0.19	0.27	0.45	0.26
7	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.90	0.90	0.90	0.25	0.22	0.33	0.54	0.33
8	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.90	0.90	0.28	0.25	0.41	0.64	0.42
9	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.10	0.90	0.29	0.26	0.49	0.67	0.50

* Configuração

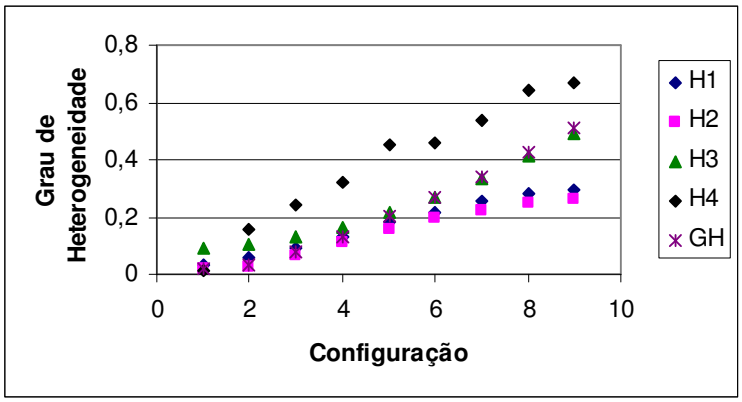

Figura 3. 4 – Grau de Heterogeneidade quando todas as potências estão próximas de 1.

A Fig. 3.4 é equivalente a Tab. 16, ela mostra que todas as métricas são satisfatórias, isto é, quando a maioria das potências computacionais das máquinas estão próximas a 1, o grau de heterogeneidade é baixo, indicando que o sistema é mais homogêneo do que heterogêneo e a medida que o sistema se torna mais heterogêneo do que homogêneo, o grau

de heterogeneidade aumenta conforme aumenta a variação das potências computacionais (Branco, 2005).

A Tab. 3.17 também mostra a configuração inicial 1 onde a maioria das máquinas são homogêneas até atingir a configuração final 9 na qual as máquinas são heterogêneas.

Tabela 3. 17 – Todas as potências computacionais das máquinas são iniciadas distantes do valor da potência computacional da máquina mais rápida e são gradualmente incrementadas, mas ainda mantendo uma distância da máquina mais rápida.

*	$W_{(M1)}$	$W_{(M2)}$	$W_{(M3)}$	$W_{(M4)}$	$W_{(M5)}$	$W_{(M6)}$	$W_{(M7)}$	$W_{(M8)}$	$W_{(M9)}$	$W_{(M10)}$	H_1	H_2	H_3	H_4	GH
1	1.00	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.09	0.45	0.05	0.16
2	1.00	0.75	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.16	0.12	0.42	0.08	0.20
3	1.00	0.75	0.65	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.12	0.41	0.10	0.21
4	1.00	0.75	0.65	0.55	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.12	0.40	0.11	0.20
5	1.00	0.75	0.65	0.55	0.40	0.50	0.50	0.50	0.50	0.50	0.16	0.12	0.41	0.27	0.22
6	1.00	0.75	0.65	0.55	0.40	0.35	0.50	0.50	0.50	0.50	0.17	0.13	0.43	0.33	0.24
7	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.50	0.50	0.50	0.20	0.15	0.45	0.47	0.28
8	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.50	0.50	0.23	0.18	0.49	0.61	0.35
9	1.00	0.75	0.65	0.55	0.40	0.35	0.25	0.15	0.10	0.50	0.26	0.22	0.53	0.66	0.46

* Configuração

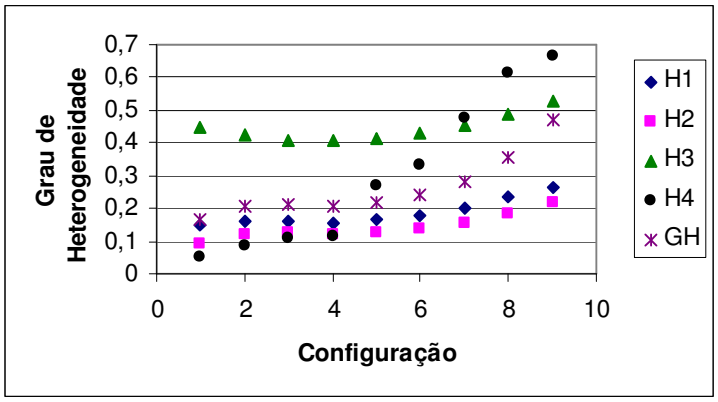

Figura 3. 5 – Grau de Heterogeneidade quando todas as potências computacionais das máquinas estão distantes de 1.

A Fig. 3.5 equivale a Tab. 3.17, ela mostra que o grau de heterogeneidade, GH, também se comporta de forma adequada no caso em que as máquinas distam de 1, representando a heterogeneidade do sistema.

A Tab. 3.18 mostra duas situações similares, as configurações 1 e 2 e as configurações 3 e 4.

Tabela 3. 18 – Quatro casos diferentes são apresentados nesta tabela: 1) uma máquina com potência computacional alta e todas as outras com potência computacional baixa; 2) uma máquina com potência computacional baixa e todas as outras com potência computacional alta; 3) duas configurações na qual metade das máquinas possui potência computacional abaixo da média e metade das máquinas possui potência computacional acima da média.

*	$W_{(M1)}$	$W_{(M2)}$	$W_{(M3)}$	$W_{(M4)}$	$W_{(M5)}$	$W_{(M6)}$	$W_{(M7)}$	$W_{(M8)}$	$W_{(M9)}$	$W_{(M10)}$	H_1	H_2	H_3	H_4	GH
1	0.50	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.15	0.09	0.05	0.45	0.09
2	1.00	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.15	0.09	0.45	0.05	0.16
3	1.00	1.00	1.00	1.00	1.00	0.50	0.50	0.50	0.50	0.50	0.25	0.25	0.25	0.25	0.33
4	1.00	1.00	1.00	1.00	1.00	0.30	0.30	0.30	0.30	0.30	0.35	0.35	0.35	0.35	0.53

* Configuração

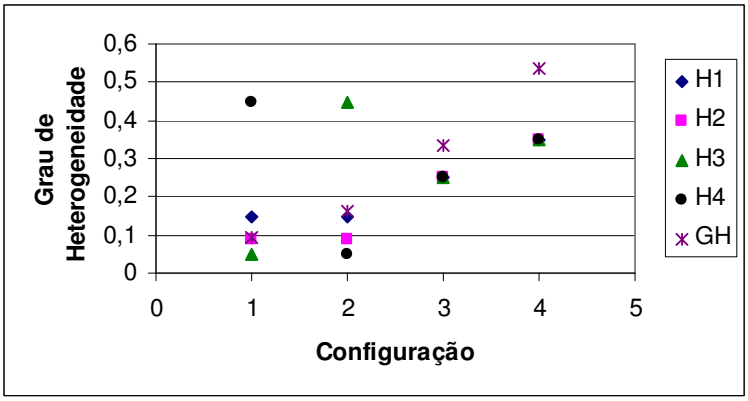

Figura 3. 6 – Grau de heterogeneidade no qual quatro situações distintas são impostas.

Intuitivamente, a opção pelos extremos, a máquina mais rápida ou mais lenta não apresentou resultados satisfatórios, então decidiu-se verificar o comportamento da métrica levando-se em consideração uma máquina virtual.

Em oposição aos graus H_3 e H_4, o grau de heterogeneidade, GH, produz bons resultados em todos os estudos de caso realizados.

3.2.4 Comportamento do Modelo

Para representar o comportamento da métrica GH, foram executados diferentes testes. A primeira configuração é composta por três máquinas com velocidades distintas e iguais a 10, 100 e 1000, e uma a uma são acrescentadas máquinas idênticas à máquina de maior velocidade até atingir a configuração 8000 composta de uma máquina de 10, uma de 100 e 7998 de 1000. Inicialmente, na primeira configuração o grau de heterogeneidade é alto (1,13), e a medida que se insere mais máquinas idênticas à máquina mais rápida, o grau de heterogeneidade tende a zero, conforme mostrado na Fig. 3.7.

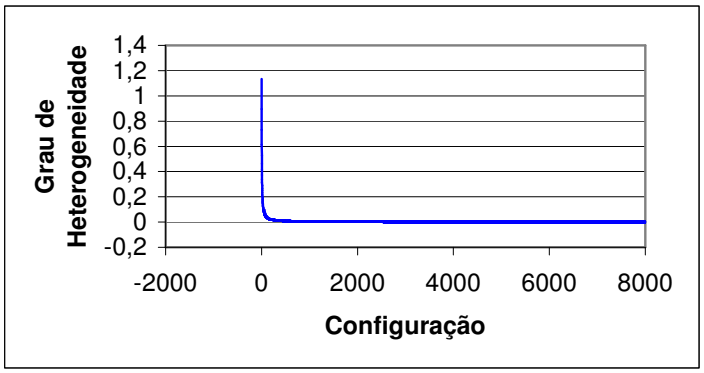

Figura 3. 7 – Comportamento do grau de heterogeneidade quando inseridas máquinas idênticas a máquina mais rápida do sistema (velocidades iniciais iguais a 10, 100, 1000).

O comportamento do grau de heterogeneidade é coerente, pois a medida que máquinas semelhantes e com alta velocidade são inseridas, o grau de heterogeneidade do sistema cai.

No segundo estudo, a configuração é de três máquinas sendo as velocidades de 10, 100, 1000, e uma a uma são inseridas máquinas idênticas à máquina mais lenta do sistema. O grau de heterogeneidade inicial é igual ao anterior (1,13), entretanto, o comportamento é diferente quando ocorre a inserção gradativa das máquinas, ilustrando o impacto da heterogeneidade do sistema levando em consideração as grandes diferenças de velocidade existentes entre as máquinas, como ilustrado na Fig. 3.8.

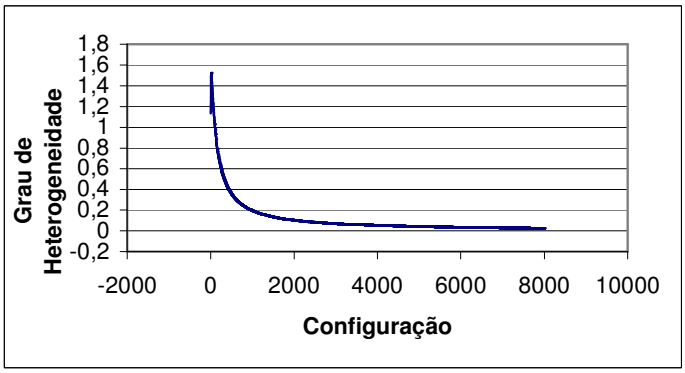

Figura 3. 8 – Comportamento do grau de heterogeneidade quando inseridas máquinas idênticas a máquina mais lenta do sistema (velocidades iniciais iguais a 10, 100, 1000).

Quando é inserida máquina idêntica à máquina mais lenta do sistema, o comportamento do grau de heterogeneidade mostra que até um certo número de máquinas adicionadas, o impacto de retirá-las do sistema a fim de que o sistema se torne homogêneo é pequeno.

Ainda mostra que a partir de um certo número de máquinas, mesmo que essas máquinas sejam idênticas à máquina mais lenta do sistema, o impacto de retirá-las é grande,

uma vez que a soma das potências computacionais dessas máquinas mais lentas se sobrepõem às máquinas mais rápidas.

Repetindo os dois estudos anteriores, alterando apenas a configuração inicial das velocidades das máquinas para 10, 10000 e 100000, tem-se para a inserção de máquinas idênticas à mais rápida e, depois, máquinas idênticas à mais lenta, comportamentos similares aos apresentados nas Figs. 3.7 e 3.8, alterando-se apenas os valores dos índices e o número de máquinas necessárias.

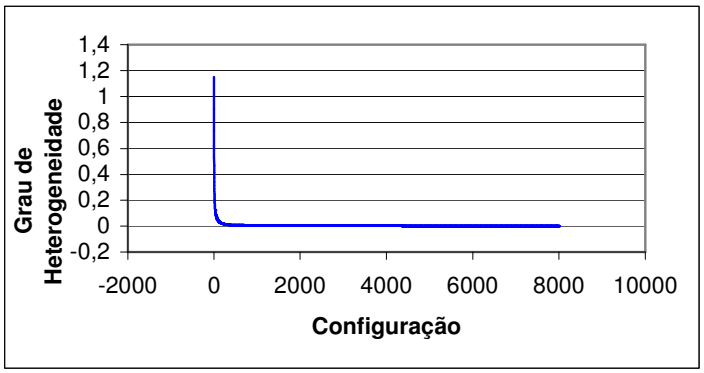

Figura 3. 9 – Comportamento do grau de heterogeneidade quando inseridas máquinas idênticas a máquina mais rápida do sistema (velocidades iniciais iguais a 10, 10000, 100000).

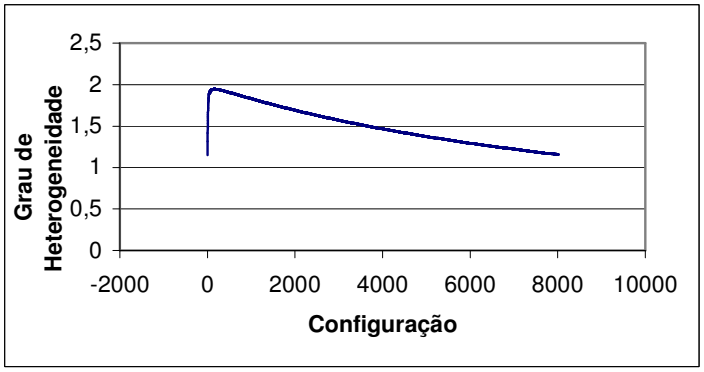

Figura 3. 10 – Comportamento do grau de heterogeneidade quando inseridas máquinas idênticas a máquina mais lenta do sistema (velocidades iniciais iguais a 10, 10000, 100000).

Outro estudo de caso buscou averiguar o comportamento do grau de heterogeneidade obtido quando o sistema é iniciado com uma determinada capacidade de processamento, em termos de velocidade total, e essa capacidade de processamento é mantida, alterando-se a configuração das máquinas.

A primeira configuração é composta de nenhuma máquina de velocidade igual a 10 e 10 máquinas de velocidade igual a 100, de modo a ter uma velocidade total de 1000. Mudanças sucessivas são efetuadas nas configurações de modo a ter 10 máquinas de 10 e 9 máquinas de 100, até atingir 100 máquinas de 10 e nenhuma máquina de 100.

O comportamento é apresentado na Fig. 3.11:

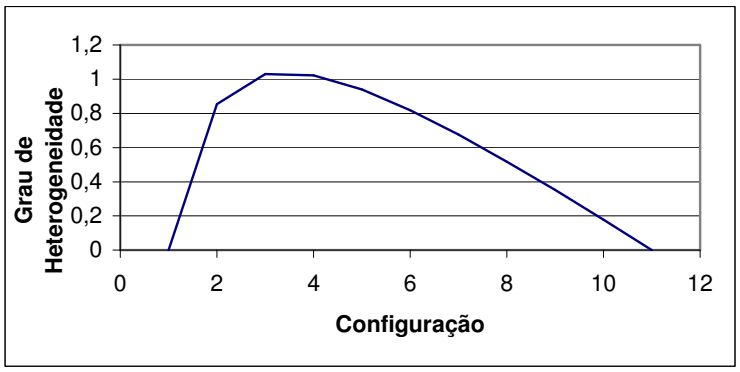

Figura 3. 11 – Comportamento do grau de heterogeneidade quando inseridas máquinas de baixa e alta velocidades no sistema de modo a manter a mesma velocidade total do sistema apenas alterando a quantidade de máquinas de alta e baixa velocidade (configuração inicial igual a 0 máquinas de velocidade 10 e 10 máquinas de velocidades iguais a 100).

A Fig. 3.11 mostra que o grau de heterogeneidade varia de acordo com a presença de máquinas com velocidades idênticas ou com velocidades distintas, refletindo o quanto essas máquinas distam da máquina virtual e qual a influência do sistema ser constituído de um número maior de máquinas lentas ou rápidas.

A Fig. 3.12 representa o mesmo experimento anterior realizado só que com nenhuma máquina de velocidade igual a 10 e 100 máquinas de velocidades iguais a 100, e assim por diante, até se ter 1000 máquinas com velocidades iguais a 10 e nenhuma máquina com velocidade igual a 100.

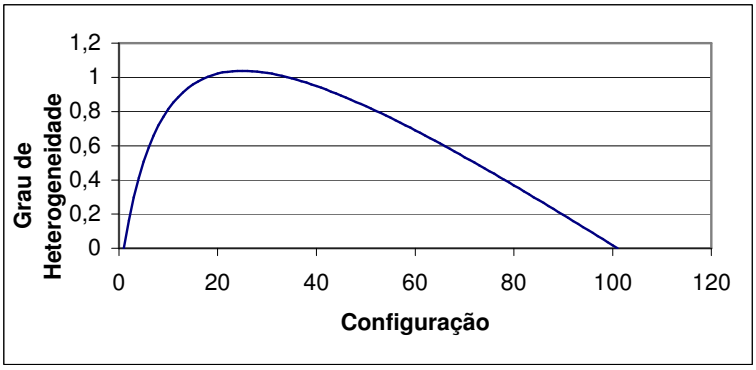

Figura 3. 12 – Comportamento do grau de heterogeneidade quando inseridas máquinas de baixa e alta velocidades no sistema de modo a manter a mesma velocidade total do sistema apenas alterando a quantidade de máquinas de alta e baixa velocidade (configuração inicial igual a 0 máquinas de velocidade 10 e 100 máquinas de velocidades iguais a 100).

Dois estudos adicionais foram executados, tomando como configurações iniciais, no primeiro caso, duas máquinas, uma com velocidade 1 e outra com velocidade 1000, e no segundo caso, uma máquina com velocidade 1 e outra com velocidade 10000. A partir dessa configuração inicial dividiu-se a velocidade da máquina mais rápida pelo número de

máquinas existentes, e esse valor é inserido como uma nova máquina; a cada passo mais uma máquina é inserida.

Esses experimentos são apresentados nas Figs. 3.13 e 3.14:

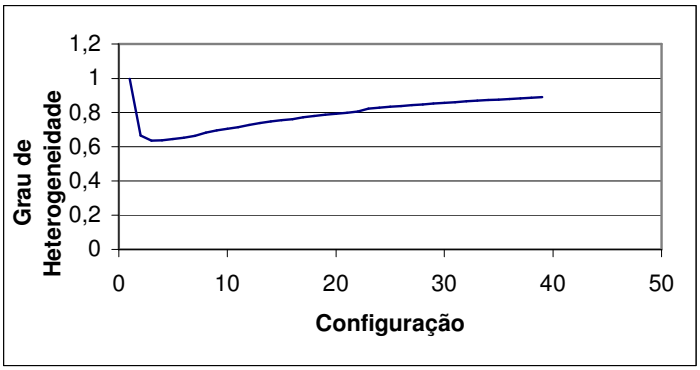

Figura 3. 13 – Comportamento do grau de heterogeneidade quando inseridas máquinas com velocidades intermediárias no sistema (configuração inicial sendo de 1 máquina de velocidade 1 e 1 máquina de velocidade 1000).

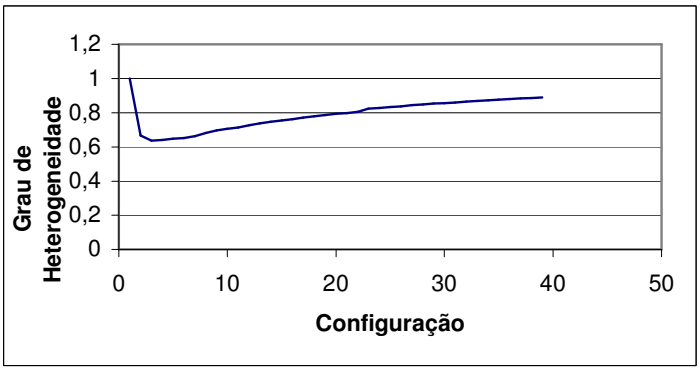

Figura 3. 14 – Comportamento do grau de heterogeneidade quando inseridas máquinas com velocidades intermediárias no sistema (configuração inicial sendo de 1 máquina de velocidade 1 e 1 máquina de velocidade 10000).

Com os dois experimentos acima, consegue-se observar que a média das máquinas em qualquer das situações permanece a mesma, isto é, o grau de heterogeneidade é praticamente o mesmo com configurações diferentes e número de máquinas diferentes. Além disso, nota-se que a adição de máquinas intermediárias até certo ponto gera uma heterogeneidade positiva, mas a partir da inserção de um determinado número de máquinas com potência computacional intermediária a heterogeneidade passa a ser negativa.

3.3 CONSIDERAÇÕES FINAIS

Este capítulo descreve as principais métricas para obtenção do grau de heterogeneidade existentes na literatura, tais como H_1, H_2, H_3, H_4 e GH, buscando avaliar as

diferenças e apresentar as restrições entre elas. Destaca-se a métrica GH, que tem como objetivo averiguar a dispersão das máquinas em torno de uma máquina virtual.

Além disso, os efeitos causados pela escolha adequada de uma métrica para obtenção do grau de heterogeneidade de um sistema computacional distribuído foram examinados. Foi mostrado que, dado o crescente interesse em sistemas distribuídos e a evolução desses sistemas levar à existência de alta heterogeneidade, investigações sobre métricas apropriadas à heterogeneidade devem ser consideradas.

Resultados de pesquisas empíricas foram apresentados em forma de estudos de caso que investigaram a utilidade do uso do grau de heterogeneidade. Apesar dos estudos de caso para validação da métrica focarem como parâmetros a velocidade das máquinas, a métrica GH pode assumir diversos parâmetros como memória, rede de comunicação, entre outros.

4 *BENCHMARKS* E ANÁLISE DE DESEMPENHO

4.1 CONSIDERAÇÕES INICIAIS

Neste capítulo são apresentadas algumas motivações para o uso de *benchmarks* ao se analisar o grau de heterogeneidade de plataformas distribuídas e, na seqüência, são apresentados os *benchmarks* utilizados para instanciar o desempenho dos computadores sob a perspectiva de: processador, memória e rede. No final do capítulo são apresentados os resultados obtidos com os *benchmarks* e também com a métrica GH descrita no capítulo anterior, a partir dessas diferentes perspectivas.

4.2 AVALIAÇÃO DE DESEMPENHO E O USO DE BENCHMARKS

A avaliação de desempenho de sistemas computacionais é um desafio. A grande quantidade e complexidade dos *softwares* modernos, em conjunto com as diferentes técnicas para otimizar o desempenho do *hardware*, tornam a avaliação de desempenho não tão simples de ser efetuada (Patterson et. al., 2005).

As características técnicas existentes em um manual do sistema computacional e as informações disponibilizadas pelo sistema operacional em tempo de execução (como as existentes no */proc* do Linux) nem sempre são suficientes para determinar a velocidade que uma determinada aplicação vai executar na plataforma. Isso ocorre porque diferentes aplicações geram diferentes demandas sobre o sistema computacional, acarretando que diferentes aspectos do sistema computacional irão se sobrepor para determinar o desempenho global do sistema, quando consideradas diferentes aplicações.

De fato, para diferentes execuções deverão ser utilizadas diferentes métricas de desempenho que avaliem não apenas a heterogeneidade configuracional entre diferentes computadores, mas efetivamente determinem o quão bem foi (ou será) a execução de uma aplicação em um computador, este com melhor ou pior configuração dentro de um conjunto de computadores.

Para exemplificar essa questão foi desenvolvido pequeno estudo utilizando 5 computadores com configurações diferentes, todos estes instalados no Laboratório de Sistemas Distribuídos e Programação Concorrente – LaSDPC do ICMC/USP. Os computadores estão descritos na Tab. 4.1 e usam o Sistema Operacional Linux, distribuição Slackware, versão 10.2, *kernel* 2.4.31.

As configurações dessas 5 máquinas apresentadas na Tab. 4.1 foram obtidas por meio do pseudo-sistema de arquivos */proc* (Woorsluys, 2007). As informações coletadas foram: freqüência de velocidade da CPU (cpu MHz), tamanho da *cache* (*cache size*), tamanho da memória RAM total disponível (MemTotal), tamanho total da área de *swap* (SwapTotal) e placa de rede.

Configuração das Máquinas						
	Jaiminho (Pentium II)	Lasdpc06 (Pentium III)	Lasdpc14 (AMD Duron)	Lasdpc53 (AMD Athlon XP)	Lasdpc54 (Pentium IV)	GH
cpu MHz	400,91	451,05	1200,07	1666,73	2017,99	0,50
tamanho da *cache* (KB)	512,00	512,00	64,00	256,00	512,00	0,45
MemTotal (KB)	191312,00	126536,00	25618,00	256188,00	256192,00	0,21
SwapTotal (KB)	200804,00	248996,00	425680,00	522072,00	265064,00	0,34
MemTotal + SwapTotal (KB)	392116,00	375532,00	681868,00	778260,00	521256,00	0,26
Placa de rede (Mbps)	Ethernet 10/100	Ethernet 10/100	Ethernet 10/100	Ethernet 10/100	Ethernet 10/100	0,00

Tabela 4. 1 - Informação dos computadores obtida através do pseudo-sistema de arquivos */proc* e a métrica GH para cada métrica coletada no */proc*.

Analisando-se as informações coletadas do */proc*, as ordens decrescentes da potência computacional das máquinas do ponto de vista da freqüência da velocidade da CPU é: Lasdpc54, Lasdpc53, Lasdpc14, Lasdpc06 e Jaiminho; do ponto de vista do tamanho da *cache* é: Jaiminho/Lasdpc06/Lasdpc54 são equivalentes, Lasdpc53 e Lasdpc14; do ponto de vista do tamanho da memória RAM total disponível é: Lasdpc54, Lasdpc53/Lasdpc14 são equivalentes, Jaiminho e Lasdpc06; do ponto de vista do tamanho total da área de *swap* é: Lasdpc53, Lasdpc14, Lasdpc54, Jaiminho e Lasdpc06; do ponto de vista do tamanho da memória total (tamanho da memória RAM total disponível + tamanho total da área de *swap*) é: Lasdpc53, Lasdpc14, Lasdpc54, Jaiminho e Lasdpc06; do ponto de vista de placa de rede todas as máquinas estão equipadas com o mesmo dispositivo de *hardware* e tem a mesma potência computacional.

A métrica GH foi calculada, tendo como base as 6 diferentes métricas coletadas a partir do */proc*. A última coluna da Tab. 4.1 e a Fig. 4.1 mostram os resultados obtidos com a GH com essas diferentes perspectivas. Analisando-se os resultados podem ser destacados os seguintes pontos:

- sob a perspectiva do processador a métrica GH exibe um alto grau de heterogeneidade para o conjunto de computadores analisado, caracterizando que o sistema é mais heterogêneo do que homogêneo devido à pouca capacidade apresentada pelas máquinas Jaiminho e Lasdpc06.

- considerando apenas o tamanho da *cache*, a métrica GH obteve um grau de heterogeneidade também bastante alto devido à grande diferença de quantidade de *cache* apresentada pelo computador Lasdpc14.

- analisando-se o tamanho da memória RAM total disponível em particular, a métrica GH obteve o menor valor do grau de heterogeneidade devido à pouca diferença entre os valores apresentados pelos computadores.

- tomando-se como base apenas o tamanho total da área de *swap*, a métrica GH volta a obter um grau de heterogeneidade um pouco mais alto do que o anterior devido aos altos valores apresentados pelos computadores Lasdpc14 e Lasdpc53.

- a métrica GH apresenta um grau de heterogeneidade com valor intermediário levando-se em consideração a quantidade de memória RAM total disponível mais a quantidade total da área de *swap*.

O gráfico da Fig. 4.1 não mostra o grau de heterogeneidade quando consideradas as placas de rede do conjunto de computadores em questão, visto que todas elas são iguais e o GH = 0.

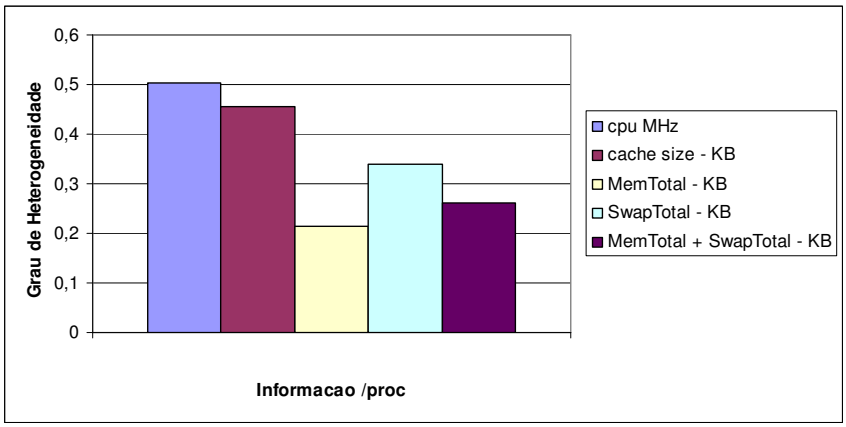

Figura 4. 1 - Grau de heterogeneidade baseada nas informações coletadas do pseudo-sistema de arquivos *proc*.

Os resultados obtidos com a métrica GH para esse grupo de computadores exemplificam a dificuldade de se determinar o grau de heterogeneidade de uma plataforma considerando apenas a configuração dos mesmos, neste caso obtida pelo */proc*. Outro fator a ser reforçado é a necessidade de se verificar o quanto que essas diferenças na configuração podem, de fato, ajudar a estimar as diferenças de desempenho esperadas quando as aplicações reais dos usuários forem executadas.

Dado o contexto deste trabalho de mestrado e a necessidade de se estimar a diferença efetiva de desempenho entre os computadores, optou-se por obter métricas de desempenho mais voltadas às aplicações finais dos usuários.

De um modo geral, o desempenho dos computadores pode ser definido de várias maneiras. Alguns exemplos são: freqüência usada pela CPU, tempo de resposta, *throughput* de aplicações finalizadas, quantidade de memória disponível e a quantidade de instruções executadas por unidade de tempo, tais como MIPS (Milhões de Instruções por Segundo) e MFLOPS (Milhões de Instruções de Ponto Flutuante por Segundo). Dentre essas métricas, as relacionadas com o tempo são vistas por alguns autores como mais eficientes para representar o desempenho de um sistema computacional como um todo (Patterson, et. al, 2005). A justificativa para tanto é que tais métricas tendem a encapsular as diferenças arquiteturais de *hardware* e *software* básico (sistema operacional e compiladores), quando o objetivo é analisar o desempenho final de aplicações dos usuários. No entanto, as métricas relacionadas ao tempo podem também ser afetadas por otimizações de *hardware* e/ou compiladores, quando as aplicações executadas são demasiadamente pequenas, muito específicas e concentram-se em poucos recursos computacionais.

O tempo na computação pode ter diferentes significados. O tempo de resposta ou tempo decorrido refere-se ao tempo total para completar a execução, incluindo o tempo para acesso à memória, à E/S, às ações do sistema operacional e tudo mais o que for necessário à execução da aplicação. Não incomum, o tempo decorrido inclui o tempo usado por outros processos, dado o compartilhamento do computador. Nesses casos, pode-se usar o tempo de processador, afim de destacar o tempo usado pela CPU apenas na execução das instruções de um processo, sem incluir aqui o tempo com E/S nem o tempo de outros processos na CPU. O tempo de processador pode, por sua vez, ser dividido em tempo de sistema e tempo de usuário. O tempo de sistema equivale ao tempo gasto na execução de rotinas do sistema operacional em benefício deste processo. O tempo de usuário refere-se ao tempo utilizado apenas na execução de instruções do próprio programa.

Os tempos utilizados neste trabalho, quando não citado explicitamente o contrário, referem-se a tempos de resposta.

A escolha pela métrica "tempo de resposta" para avaliar o desempenho de um computador e conseqüentemente o grau de heterogeneidade da plataforma como um todo, gera o problema de se escolher qual será a aplicação a ser executada para gerar a demanda.

Não há uma resposta única para tal pergunta, no entanto, alguns direcionamentos podem fornecer um norte a ser seguido.

Quando um usuário final executa sempre o mesmo programa e deseja avaliar uma nova máquina para este programa, ele deve executar o seu programa diretamente na mesma, afim de avaliar se o tempo de resposta obtido é o desejado. Esse programa pode ser chamado de carga de trabalho (ou *workload*).

No entanto, muitos usuários não dispõem de uma carga de trabalho adequada para execução. Nesses casos, esses usuários usam outras aplicações as quais, espera-se, gerem uma demanda semelhante àquela gerada futuramente pela aplicação real que será executada no computador. Essas aplicações, conhecidas como *benchmarks*, são programas desenvolvidos especialmente para avaliar o desempenho de um sistema computacional. Os *benchmarks* formam, portanto, uma carga de trabalho sintética com o objetivo de prever o desempenho futuro de uma carga de trabalho real, esta última utilizada por determinados grupos de usuários. O uso de *benchmarks* facilita a comparação de diferentes sistemas computacionais, pois favorece a compilação, a execução e a instanciação do código fonte nessas diferentes plataformas. Outra atividade facilitada com o uso dos *benchmarks* é a reprodução dos resultados obtidos, desde que documentados adequadamente.

Uma das preocupações constantes com *benchmarks* é o uso de otimizações pontuais tanto na arquitetura quanto no compilador, com o objetivo de "forjar" bons resultados, isto é, permitindo que o sistema computacional tenha um desempenho ótimo apenas quando executando o pequeno trecho de código do *benchmark*. Para evitar tais situações, muitos *benchmarks* são baseados em aplicações reais, estas representativas para certos grupos de usuários. Assim, espera-se que os resultados obtidos com o *benchmark* em uma determinada plataforma possam também ser obtidos com as aplicações reais no futuro. Outra maneira de se evitar a otimização específica de um *benchmark*, sem a possibilidade da devida analogia com as aplicações reais, é o uso de *benchmarks* não muito pequenos e/ou demasiadamente específicos, cujo código possa ser facilmente detectado por compiladores e/ou arquiteturas.

Considerando o exposto, optou-se neste trabalho por utilizar os resultados de *benchmarks* conhecidos na literatura. Tais resultados foram usados como argumento de entrada para a métrica GH proposta por Branco (Branco, 2005), a qual determina o grau de heterogeneidade de plataformas distribuídas. Com o uso de diferentes *benchmarks*, pôde-se analisar o grau de heterogeneidade gerado na plataforma distribuída sob diferentes perspectivas e, assim, verificar a eficiência da métrica GH.

Para tanto, os *benchmarks* analisados neste trabalho foram agrupados em: *benchmarks* de processador, *benchmarks* de memória e *benchmark* de rede. Algumas características nortearam a escolha dos *benchmarks* apresentados a seguir. Optou-se por *benchmarks* que possuem código aberto, código livre, usando linguagens de alto nível como C, disponibilidade na Internet, documentação das suas finalidades / características / instalação, facilidade de portabilidade e uso pela comunidade da computação em geral.

As próximas seções irão apresentar os *benchmarks* estudados e executados, descrevendo suas principais características. Após, serão feitas algumas considerações sobre os resultados obtidos nos experimentos realizados com os mesmos.

4.3 *BENCHMARKS* DE PROCESSADOR

Os *benchmarks* apresentados nesta seção são utilizados para analisar o desempenho de computadores do ponto de vista de processador. Eles possuem código aberto, são gratuitos e estão disponíveis na linguagem C (Netlib, 2007).

4.3.1 *Benchmark* Whetstone

O Whetstone é um *benchmark* escrito na linguagem C que permite testar o desempenho do processador em operações de ponto-flutuante. A versão original foi desenvolvida em 1976 (publicada inicialmente na linguagem ALGOL 60 e posteriormente codificado também em FORTRAN). Ele faz parte de vários *benchmarks* atuais, cujo resultado indica o número de vezes por segundo no qual o processador é capaz de executar o programa. O desempenho do processador nesse teste é um bom indicativo do seu desempenho em aplicativos científicos. O Whetstone é um programa com poucas linhas de código, composto de vários módulos. Cada módulo manipula um tipo de dado, como por exemplo, identificadores simples e elementos de vetor. Os módulos exploram diferentes características da linguagem de programação e exploram um número variado de iterações através de laços do tipo "*FOR*" (Conte et. al., 1991a) (Clark et. al., 1982) (Rus et. al., 2003) (Kri et. al., 2004) (Whetstone, 2007).

As principais vantagens desse *benchmark* são o seu tamanho reduzido e a simplicidade do código, além de explorar bastante as operações em ponto-flutuante. Portanto, serve como comparativo para pequenas aplicações científicas em computadores de pequeno e médio porte. Ele também apresenta algumas desvantagens: devido ao tamanho pequeno de seus módulos, o sistema de memória fora da *cache* não é testado; compiladores podem facilmente

otimizar o Whetstone (Saavedra et. al., 1996) (Weiderman et. al., 1990) (Shivam et. al., 2006).

O Whetstone oferece como saída duas métricas: tempo em segundos e MIPS. As duas métricas são inversamente proporcionais, isto é, tempo em segundos - quanto maior o valor pior o desempenho; MIPS - quanto maior o valor melhor o desempenho.

4.3.2 *Benchmark* Dhrystone

O Dhrystone foi publicado pela primeira vez em 1984 na linguagem ADA e posteriormente foi codificado na linguagem C. Ele é utilizado para medir desempenho de processadores. Pretendia-se originalmente com o Dhrystone, criar um *benchmark* pequeno e representativo para programação de sistemas (no caso de operações aritméticas). O código do Dhrystone é dominado por aritmética simples, operações com *strings*, decisões lógicas e acessos de memória com intenção de refletir as atividades da CPU nas aplicações de computação de propósito mais geral. O resultado do Dhrystone é determinado através do cálculo do tempo médio que um processador leva para executar as muitas iterações de um laço que, por sua vez, contém uma seqüência fixa de instruções que o compõem. Ele tem um número de atributos que o levaram a ser amplamente usado no passado para medir desempenho de processadores: é compacto, tem alta disponibilidade no domínio público e é simples de usar (Conte et. al., 1991a) (Salminem et. al., 2005) (Robinson et. al., 2007).

O Dhrystone compara o desempenho do processador ao de uma "máquina de referência", o que é considerado por muitos como uma vantagem em relação à utilização direta da métrica MIPS. A justificativa para tanto é que usar uma máquina de referência efetivamente compensa as diferenças na complexidade das instruções, onde comparar os números de MIPS de uma arquitetura RISC com os de uma CISC não é considerado válido por muitos pesquisadores. Inicialmente a indústria adotou o VAX 11/780 como uma máquina de referência de 1 MIPS. O VAX 11/780 alcança 1757 D/S (dhrystone por segundo). O resultado do *benchmark* é calculado mensurando o número de D/S para o sistema, e dividindo-se este número por 1757 (da máquina de referência). Então, 80 MIPS "significa 80 MIPS VAX Dhrystone", o que por sua vez significa dizer que esta máquina é 80 vezes mais rápida que a máquina de referência VAX 11/780. Uma taxa de DMIPS/MHz leva essa normalização ainda mais adiante, permitindo uma comparação de desempenho de processador para taxas diferentes de *clock* (Weicker et. al., 1984) (Benso et. al., 2003) (Rodrigues et. al., 2004).

Entretanto, algumas das vantagens aparentes do Dhrystone também são fraquezas significativas dele. Os números do Dhrystone refletem na verdade o desempenho do compilador da linguagem C e suas bibliotecas, provavelmente mais do que o desempenho do próprio processador. Além disso, seu projeto foi baseado na análise de vários outros programas escritos em diferentes linguagens e por diferentes autores, porém voltados à programação de sistemas operacionais e compiladores. Esta é uma característica bastante relevante, pois diferentes classes de aplicações enfatizam diferentes tipos de operações. Por exemplo, considere as aplicações numéricas que utilizam muitos vetores e aritmética de ponto flutuante; já as aplicações comerciais e programação de sistemas utilizam predominantemente atividades de entrada/saída, ponteiros, sentenças "*IF*", chamadas de procedimento, além de conter menos laços e expressões numéricas mais simples. Além disso, o *benchmark* apresenta tamanho de código pequeno (100 comandos, 1 a 1.5 kB de código), o sistema de memória fora da *cache* não é testado - compiladores podem facilmente otimizar o código (usá-lo somente para experimentos controlados) (Conte et. al., 1991b) (Pisharath et. al., 2003).

De acordo com a freqüência das operações nos diferentes programas analisados, foram construídas algumas tabelas que deram origem ao Dhrystone. Em seu código original, em Ada, ele contém 100 sentenças entre o início e o fim do contador de tempo, balanceadas em relação a tipos de sentenças, tipos de dados (operandos) e sua localidade (global, local, parâmetro, constante). O programa contém uma distribuição de 51% de atribuições, 32% de sentenças de controle e 17% de chamadas de funções e procedimentos. O corpo do programa contém 12 procedimentos e, durante um laço (isto é, 1 Dhrystone), as 100 sentenças que compõem o laço são executadas. O maior tempo gasto está nas operações de atribuição e comparação de *strings* (Vandierendonck et. al., 2004) (Thid et. al., 2006) (Kenny et. al., 2005).

O Dhrystone oferece como saída duas métricas: tempo em microsegundos e dhrystone por segundo. As duas métricas são inversamente proporcionais, isto é, tempo em microsegundos - quanto maior o valor pior o desempenho; dhrystone por segundo - quanto maior o valor melhor o desempenho.

4.3.3 *Benchmark* Linpack

É um dos mais famosos *benchmarks* (tem maior número de resultados reportados). Originalmente, o Linpack era um pacote de sub-rotinas com a finalidade de resolver sistemas de equações lineares algébricas. Jack Dongarra, da Universidade do Tennessee (antigo Laboratório Nacional Argonne), publicou-o em 1976 sem intenção de torná-lo um *benchmark*.

Ele contém dois conjuntos de rotinas: um para decomposição de matrizes e outro para resolver o sistema de equações lineares baseados em decomposição. Dentre as várias sub-rotinas contidas no pacote, a mais utilizada e a que consome a maior parte do tempo de execução do programa é a "saxpy" na versão de precisão simples e a "daxpy" na versão de precisão dupla - fazendo o laço interno para operações freqüentes da matriz (valores em ponto flutuantes): y(i)=y(i)+a*x(i). Ele contém somente 15 linhas de código em linguagem de baixo nível e trabalha com vetores de apenas uma dimensão. As rotinas de mais alto nível chamam-na várias vezes e operam com vetores bi-dimensionais (Conte et. al., 1991a) (Cameron et. al., 2005) (Meeker, 2005) (Constantinescu, 2005) (Garg et. al., 2006) (Head et. al., 2006) (Thiruvathukal et. al., 2007).

Este *benchmark* baseia-se em um subpacote de rotinas para operações básicas de álgebra linear, o BLAS (*Basic Linear Algebra Subroutine* - Subrotinas de Álgebra Linear Básica). A versão FORTRAN é chamada FORTRAN BLAS, a versão Assembly - Coded BLAS (esta já não é mais utilizada) e existe também a versão disponível na linguagem C. Existem versões diferentes para o Linpack, cada versão diferencia-se no tamanho das matrizes (a mais utilizada é 100x100, há também versões para tamanhos de 300x300 e 1000x1000); a precisão pode ser dupla ou simples; e em relação aos tipos de laços (*rolled/unrolled)*, estes podem ser otimizados ou não. Os resultados dos testes são reportados em MFLOPS (*Millions of Floating Point Operations per Second*), GFLOPS (*Billions of Floating Point Operations per Second*) ou até TFLOPS (*Trillions of Floating Point Operations per Second*) e tempo total em segundos. As métricas MFLOPS/GFLOPS/TFLOPS e tempo total em segundos são inversamente proporcionais, isto é, MFLOPS/GFLOPS/TFLOPS - quanto maior o valor melhor o desempenho enquanto que tempo total em segundos - quanto maior o valor pior o desempenho. Sua aplicação é visível em máquinas que utilizam softwares para cálculos científicos e de engenharia, visto que as operações mais utilizadas nesses tipos de aplicações são em ponto flutuante (Teresco et. al., 2005) (Kishimoto et. al., 2005) (Parks, 2005) (Dongarra, 2006) (Barton et. al., 2006) (Pheatt et. al., 2007) (Harbaugh et. al., 1984).

4.4 *BENCHMARKS* DE MEMÓRIA

Os *benchmarks* apresentados nesta seção são utilizados para analisar o desempenho de computadores do ponto de vista de memória. Eles possuem código aberto, são gratuitos e estão disponíveis na linguagem C (Stream, 2007) (Llcbench, 2007).

4.4.1 *Benchmark* Stream

O *benchmark* Stream está disponível nas versões FORTRAN e C e os resultados são usados pelos principais vendedores em computação de alto desempenho. Ele é um *benchmark* simples que avalia a capacidade de banda de memória disponível (em MB/s – MBytes/segundo) correspondente a vetores simples. Ele é especificamente projetado para trabalhar com conjuntos de dados muito maiores do que a *cache* disponível no sistema e, por isso, seus resultados são mais indicativos para aplicações que utilizam vetores de tamanhos significativos (Cantonnet et. al., 2003) (Mckee, 2004) (Cantonnet et. al., 2005) (Zhang et. al., 2005) (Teller et. al., 2005) (Funk et. al., 2005) (Hur et. al., 2006) (Rui et. al., 2007).

O Stream oferece como saída duas métricas: taxa em MBytes/segundo e tempo médio em segundos. As duas métricas são inversamente proporcionais, isto é, taxa em MBytes/segundo – quanto maior o valor melhor o desempenho; tempo médio em segundos – quanto maior o valor pior o desempenho.

4.4.2 *Benchmark* Cachebench

O Cachebench é um *benchmark* com o objetivo de estimar o desempenho da hierarquia de memória local de uma máquina. Ele é implementado em C e desempenha um conjunto de operações de leitura, escrita, leitura/modificação/escrita, *memset()* e *memcopy()*, sempre variando o tamanho do vetor e expondo o desempenho da *cache* (potencialmente multi-nível). As operações em cada vetor executam uma quantidade de tempo (configurável - padrão 2 segundos) e, no final, informam a capacidade de banda média (MB/s). Este *benchmark* produz uma métrica similar ao *benchmark* Stream, embora execute por um tempo mais longo, visto que ele obtém medidas de tamanho de memória diferentes (tempos de execução são tipicamente na ordem de 5 minutos). Ele tem como foco o desempenho de *caches* de memória, fornecendo subsídios para avaliar diferentes níveis de *cache* para CPUs. O Cachebench oferece como saída a métrica taxa em MBytes/segundo, isto é, quanto maior o valor melhor o desempenho (Ding et. al., 2000) (Vetter et. al., 2002) (Kerbyson et. al., 2004) (Bhatia, et. al., 2006) (Chen et. al., 2006).

4.5 *BENCHMARK* DE REDE

O *benchmark* apresentado nesta seção é utilizado para analisar o desempenho de computadores do ponto de vista de rede. Ele possui código aberto, é gratuito e está disponível na linguagem C (Netperf, 2007).

4.5.1 *Benchmark* Netperf

É um *benchmark* de rede simples desenvolvido pela *Hewlett-Packard* (Netperf, 2007). Ele é útil para verificar rapidamente a capacidade de banda máxima de uma conexão TCP e UDP entre duas máquinas. Ele tem dois componentes: *netserver* e *netperf*. *Netserver* é executado em uma máquina e espera por conexões do *netperf*. *Netperf* conecta-se ao *netserver*, realiza um *handshake* curto, envia o dado para o servidor na taxa mais alta possível por 10 segundos. No fim dos 10 segundos, o *netperf* calcula a média da capacidade de banda alcançada naquele período de tempo. O laço interno no *netperf* é muito curto, basicamente preenchendo um *buffer* e executando a chamada de sistema *write*. Este *benchmark* tem muito pouco tempo de usuário e gasta a maioria do seu tempo no *kernel*, manipulando a pilha do protocolo TCP/IP ou esperando as transações DMA completarem. Ele tem 4 possíveis configurações: *TCP STREAM*, *UDP STREAM*, *TCP RR* e *UDP RR*. As versões *TCP STREAM* e *UDP STREAM* são projetadas para testar capacidade de banda - o *netperf* abre uma conexão para a máquina *netserver* e envia o dado tão rápido quanto possível. As outras versões *TCP RR* e *UDP RR* realizam um teste de latência requisição/resposta - nesse teste, o cliente envia um pacote TCP/UDP do tamanho de um MTU (*Maximum Transmission Unit* - 1500 Bytes) para o servidor e espera por uma resposta do mesmo tamanho. Quando a resposta retorna, uma outra mensagem é enviada imediatamente. A latência *round-trip* pode ser calculada como a inversa do número de requisições por segundo (Hui et. al., 2004) (Underwood et. al., 2004) (Binkert et. al., 2005) (Chang et. al., 2005) (Gotsis et. al., 2005) (Minohara et. al., 2005) (Bai et. al, 2007) (Binkert et. al., 2006) (Huang et. al, 2005) (Pope et. al., 2007).

O Netperf oferece como saída duas métricas: vazão em 10^6 MBytes/segundo e tempo decorrido em segundos. As duas métricas são inversamente proporcionais, isto é, vazão em 10^6 MBytes/segundo – quanto maior o valor melhor o desempenho; tempo decorrido em segundos – quanto maior o valor pior o desempenho.

4.6 ANÁLISE DOS RESULTADOS GERADOS PELOS *BENCHMARKS*

Para coletar as informações produzidas pelos *benchmarks* para o processador (Whetstone, Dhrystone e Linpack), para a memória (Cachebench e Stream) e para a rede (Netperf) foram utilizados os mesmos 5 computadores já citados anteriormente e descritos na Tab. 4.1. Os compiladores utilizados nos experimentos foram: *gcc 3.3.6*.

Cada um dos *benchmarks* foi executado 30 vezes e a média aritmética dessas execuções está sendo utilizada como parâmetro de entrada na métrica GH para calcular o grau

de heterogeneidade do sistema em questão. Para cada valor médio apresentado também são destacados a variância e o desvio padrão.

As Tabs. 4.2 a 4.8 apresentam as potências computacionais de cada computador (a média aritmética, variância e desvio padrão das 30 execuções) e o grau de heterogeneidade (GH), levando-se em consideração os *benchmarks* Whetstone, Dhrystone e Linpack. As Tabs. 4.9 a 4.12 estão relacionadas aos *benchmarks* Cachebench e Stream. Já as Tabs. 4.13 e 4.14 estão associadas ao *benchmark* Netperf.

4.6.1 *Benchmark* Whetstone

A Tab. 4.2 apresenta os resultados para o *benchmark* Whetstone, o qual apresenta como resultado de execução duas métricas para análise de desempenho: MIPS e tempo em segundos. As métricas apresentadas na Tab. 4.2 estão em escalas diferentes, ou seja, a métrica tempo em segundos sugere quanto menor o tempo melhor o desempenho, já a métrica MIPS com valor maior indica melhor desempenho. A Tab. 4.3 apresenta as duas métricas na mesma proporção/escala, onde obteve-se o inverso do tempo em segundos.

Benckmark whetstone				
Computador	Unidade	Média	Variância	Desvio padrão
Jaiminho	MIPS	89,40	0,00	0,00
	tempo - segundos	1119,00	0,00	0,00
Lasdpc06	MIPS	100,46	0,00	0,04
	tempo - segundos	995,40	0,24	0,49
Lasdpc14	MIPS	377,42	6,23	2,49
	tempo - segundos	264,96	3,13	1,77
Ladspc53	MIPS	525,15	16,54	4,06
	tempo - segundos	190,43	2,18	1,47
Lasdpc54	MIPS	79,09	0,00	0,02
	tempo - segundos	1264,96	0,17	0,41

Tabela 4. 2 - Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Whetstone.

Benckmark Whetstone						
Unidade	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	Lasdpc54	GH
MIPS	89,4000	100,4600	377,4200	525,1500	79,0900	0,7400
tempo - seg	0,0008	0,0010	0,0037	0,0052	0,0007	0,7400

Tabela 4. 3 - Potência computacional das máquinas utilizando *benchmark* Whetstone e a métrica GH.

Levando-se em consideração a configuração dos computadores exibida na Tab. 4.1, esperava-se que as máquinas Lasdpc53 e Lasdpc54 tivessem uma potência computacional equivalente e superior as demais, o que acabou não acontecendo no caso da Lasdpc54. Os computadores que têm as melhores potências computacionais em ordem decrescente são: Lasdpc53, Lasdpc14, Lasdpc06, Jaiminho e Lasdpc54.

Apesar do resultado inesperado para a máquina Lasdpc54, a comparação entre as métricas tempo em segundos e MIPS no Whetstone mostram que as métricas apresentaram resultados coerentes entre si. Acredita-se que a perda de desempenho da Lasdpc54 em relação às demais máquinas deve-se a pouca capacidade de processamento envolvendo aritmética de ponto flutuante dessa máquina. Partindo-se do pressuposto que o resultado do benchmark esteja correto, tal resultado exemplifica a importância de se prever o desempenho dos computadores com base no tempo de execução de aplicações voltadas às demandas geradas pelos usuários.

O grau de heterogeneidade apresentado na Tab. 4.3 para ambas as métricas (MIPS e tempo) são equivalentes (quanto maior melhor).

4.6.2 *Benchmark* Dhrystone

A Tab. 4.4 apresenta os resultados para o *benchmark* Dhrystone. Ele possui duas opções de execução (com e sem o uso de variáveis do tipo *register*) e oferece como resultado para cada uma dessas execuções duas métricas para análise de desempenho: microsegundos e dhrystone por segundo. Assim como ocorreu com o *benchmark* Whetstone, essas duas escalas são inversamente proporcionais. A Tab. 4.5 apresenta as duas métricas na mesma escala, quanto maior o valor melhor o desempenho.

As ordens decrescentes da potência computacional das máquinas considerando o uso de registradores do tipo *register* para a métrica microsegundos é: Lasdpc54/Lasdpc53 são equivalentes, Lasdpc14, Lasdpc06 e Jaiminho; e para a métrica dhrystone por segundo é: Lasdpc53, Lasdpc54, Lasdpc14, Ladspc06 e Jaiminho.

E as ordens decrescentes considerando que não são usados registradores do tipo *register* para a métrica microsegundos é: Lasdpc53, Lasdpc54, Lasdpc14, Lasdpc06 e Jaiminho; e para a métrica dhrystone por segundo é: Lasdpc53, Lasdpc54, Lasdpc14, Lasdpc06 e Jaiminho.

Os resultados obtidos com o *benchmark* Dhrystone apresentam uma leve diferença entre os computadores Lasdpc53 e Lasdpc54. É preciso estar atento para esse fato, pois se levar em consideração a opção utilização de registradores do tipo *register* e as duas métricas (microsegundos e dhrystone/segundo), a máquina Lasdpc54 é na melhor das hipóteses igual a Lasdpc53. Ao fazer uso da opção, não utilização de registradores do tipo *register*, tanto a métrica microsegundos quanto a dhrystone por segundo chegam à conclusão que a Lasdpc53 tem uma potência computacional levemente superior à Lasdpc54. Além disso, ambas as

opções concordam que as outras 3 máquinas (Jaiminho, Lasdpc06 e Lasdpc14) apresentam um desempenho semelhante independente da métrica adotada.

Benchmark Dhrystone					
Computa-dor	Tipo de execução	Unidade	Média	Variância	Desvio padrão
Jaiminho	com register	microsegundos	3,10	0,00	0,00
		dhrystone/seg	320660,20	4869,19	69,77
	sem register	microsegundos	3,20	0,00	0,00
		dhrystone/seg	315347,96	1602,05	40,02
Lasdpc06	com register	microsegundos	2,80	0,00	0,00
		dhrystone/seg	356527,60	63198,10	251,39
	sem register	microsegundos	2,80	0,00	0,00
		dhrystone/seg	355965,00	81237,34	285,02
Lasdpc14	com register	microsegundos	1,00	0,00	0,00
		dhrystone/seg	991966,30	1290749,00	1136,11
	sem register	microsegundos	1,00	0,00	0,00
		dhrystone/seg	995919,90	3202222,00	1789,47
Lasdpc53	com register	microsegundos	0,70	0,00	0,00
		dhrystone/seg	1380713,00	8508701,00	2916,96
	sem register	microsegundos	0,70	0,00	0,00
		dhrystone/seg	1384624,00	43039736,00	6560,46
Lasdpc54	com register	microsegundos	0,70	0,00	0,00
		dhrystone/seg	1356669,00	1016188,00	1008,06
	sem register	microsegundos	0,80	0,00	0,00
		dhrystone/seg	1276217,00	1223919,00	1106,30

Tabela 4. 4 - Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Dhrystone.

Benchmark Dhrystone							
Tipo de execução	Unidade	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	Lasdpc54	GH
com register	microsegundos	0,32	0,35	1,00	1,42	1,42	0,50
	dhrystone/seg	320660,20	356527,60	991966,30	1380713,00	1356669,00	0,49
sem *register*	microsegundos	0,31	0,35	1,00	1,42	1,25	0,49
	dhrystone/seg	315348,00	355965,00	995919,90	1384624,00	1276217,00	0,48

Tabela 4. 5 - Potência computacional das máquinas utilizando *benchmark* Dhrystone e a métrica GH.

Novamente, apesar da Tab 4.1 mostrar que a máquina Lasdpc54 possui uma freqüência superior à Lasdpc53, essa diferença não é verificada no desempenho final do *benchmark*. Nota-se também que o desempenho da máquina Lasdpc54 teve uma grande variação em relação às demais, quando se analisando os dois *benchmarks* vistos até agora: Whetstone e Dhrystone. Atribui-se essa variação às diferentes demandas geradas pelos dois *benchmarks*: operações de ponto-flutuante no Whetstone e operações de aritmética simples, *strings* e acessos à memória no Dhrystone.

O grau de heterogeneidade apresentado na Tab. 4.5 indica que as duas opções disponíveis (utilização de registradores do tipo *register* e a não utilização desses registradores) com as duas métricas possuem uma leve variação, porém, foram considerados equivalentes neste trabalho.

4.6.3 *Benchmark* Linpack

As Tabs. 4.6 e 4.7 apresentam os resultados obtidos com o *benchmark* Linpack. Ele possui dois tipos de execução (precisão dupla - ddp e precisão simples – dsp). Para cada opção estão disponibilizados dois tipos de versão (*roll* – sem otimização e *unroll* – com otimização). As 4 combinações oferecem como resultado de execução duas métricas para análise de desempenho: tempo de execução em segundos (geração da matriz em segundos + tempo total em segundos) e MFLOPS (milhões de operações de ponto flutuante por segundo). As métricas apresentadas nas Tabs. 4.6 e 4.7 estão em escalas diferentes e a Tab. 4.8 apresenta as duas métricas na mesma proporção/escala (quanto maior melhor).

As ordens decrescentes da potência computacional das máquinas do ponto de vista da execução ddp e da opção *roll* em relação à métrica tempo de execução em segundos é: Lasdpc53, Lasdpc54, Lasdpc14, Lasdpc06 e Jaiminho; em relação a métrica MFLOPS é: Lasdpc54, Lasdpc53, Lasdpc14, Lasdpc06 e Jaiminho. E do ponto de vista da opção *unroll* em relação à métrica tempo de execução em segundos é: Lasdpc54, Lasdpc53, Lasdpc14, Lasdpc06 e Jaiminho; em relação à métrica MFLOPS é: Lasdpc53, Lasdpc54, Lasdpc14, Lasdpc06 e Jaiminho.

E as ordens decrescentes do ponto de vista da execução dsp e da opção *roll* em relação à métrica tempo de execução em segundos é: Lasdpc53, Lasdpc14, Lasdpc54, Lasdpc06 e Jaiminho; em relação à métrica MFLOPS é: Lasdpc53, Lasdpc14, Lasdpc54, Lasdpc06 e Jaiminho. E do ponto de vista da opção *unroll* em relação à métrica tempo de execução em segundos é: Lasdpc53, Lasdpc14, Lasdpc54, Lasdpc06 e Jaiminho; em relação à métrica MFLOPS é: Lasdpc53, Lasdpc14, Lasdpc54, Lasdpc06 e Jaiminho.

Benchmark Linpack						
Computa-dor	Tipo de execução		Unidade	Média	Variância	Desvio padrão
Jaiminho	ddp	roll	geração matriz – seg	0,0028	0,0000	0,0000
			tempo total – seg	0,0168	0,0000	0,0000
			Mflops	40,7010	0,0500	0,1600
		unroll	geração matriz – seg	0,0028	0,0000	0,0000
			tempo total – seg	0,0151	0,0000	0,0000
			mflops	45,1788	0,0000	0,0100
	dsp	roll	geração matriz – seg	0,0028	0,0000	0,0000
			tempo total – seg	0,0156	0,0000	0,0000
			mflops	43,9020	0,0000	0,0000
		unroll	geração matriz – seg	0,0028	0,0000	0,0000
			tempo total – seg	0,0140	0,0000	0,0000
			mflops	48,7480	0,0000	0,0100
Lasdpc06	ddp	roll	geração matriz – seg	0,0025	0,0000	0,0000
			tempo total – seg	0,0150	0,0000	0,0000
			mflops	45,6498	0,0000	0,0300
		unroll	geração matriz – seg	0,0025	0,0000	0,0000
			tempo total – seg	0,0135	0,0000	0,0000
			mflops	50,6540	0,0000	0,0100
	dsp	roll	geração matriz – seg	0,0025	0,0000	0,0000
			tempo total – seg	0,0138	0,0000	0,0000
			mflops	49,4308	0,0000	0,0100
		unroll	geração matriz – seg	0,0025	0,0000	0,0000
			tempo total – seg	0,0124	0,0000	0,0000
			mflops	55,1026	0,0000	0,0100
Lasdpc14	ddp	roll	geração matriz – seg	0,00079	0,0000	0,0000
			tempo total – seg	0,0046	0,0000	0,0000
			mflops	146,5430	0,0500	0,2000
		unroll	geração matriz – seg	0,0007	0,0000	0,0000
			tempo total – seg	0,0041	0,0000	0,0000
			mflops	163,8240	0,0000	0,0700
	dsp	roll	geração matriz – seg	0,0008	0,0000	0,0000
			tempo total – seg	0,0042	0,0000	0,0000
			mflops	161,2480	0,0100	0,1000
		unroll	geração matriz – seg	0,0008	0,0000	0,0000
			tempo total – seg	0,0039	0,0000	0,0000
			mflops	175,4522	0,1600	0,4000
Lasdpc53	ddp	roll	geração matriz – seg	0,0005	0,0000	0,0000
			tempo total – seg	0,0033	0,0000	0,0000
			mflops	202,1392	0,1000	0,3200
		unroll	geração matriz – seg	0,0005	0,0000	0,0000
			tempo total – seg	0,0030	0,0000	0,0000
			mflops	226,3602	0,1200	0,3500
	dsp	roll	geração matriz – seg	0,0005	0,0000	0,0000
			tempo total – seg	0,0033	0,0000	0,0000
			mflops	203,8328	0,0100	0,1100
		unroll	geração matriz – seg	0,0005	0,0000	0,0000
			tempo total – seg	0,0031	0,0000	0,0000
			mflops	214,8332	0,2100	0,4500

Tabela 4. 6 - Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Linpack.

Benchmark Linpack						
Computa-dor	Tipo de execução		Unidade	Média	Variância	Desvio padrão
Lasdpc54	ddp	roll	geração matriz – seg	0,0008	0,0000	0,0000
			tempo total – seg	0,0032	0,0000	0,0000
			mflops	210,3855	0,3800	0,6100
		unroll	geração matriz – seg	0,0008	0,0000	0,0000
			tempo total – seg	0,0030	0,0000	0,0000
			mflops	225,1452	0,2100	0,4600
	dsp	roll	geração matriz – seg	0,0008	0,0000	0,0000
			tempo total – seg	0,0048	0,0000	0,0000
			mflops	140,1617	0,1900	0,4300
		unroll	geração matriz – seg	0,0008	0,0000	0,0000
			tempo total – seg	0,0054	0,0000	0,0000
			mflops	126,4887	0,3400	0,5800

Tabela 4. 7 – Continuação da média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Linpack.

Benchmark Linpack								
Tipo de execução		Unidade	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	Lasdpc54	GH
Ddp	roll	tempo total - seg	50,64	56,84	182,61	252,70	244,91	0,51
		Mflops	40,70	45,64	146,54	202,13	210,38	0,54
	unroll	tempo total - seg	55,35	62,10	200,76	277,52	258,50	0,50
		Mflops	45,17	50,65	163,82	226,36	225,14	0,52
Dsp	roll	tempo total - seg	54,05	60,82	196,58	252,17	174,76	0,38
		Mflops	43,90	49,43	161,24	203,83	140,16	0,38
	unroll	tempo total - seg	59,00	66,67	210,97	263,78	160,15	0,33
		Mflops	48,74	55,10	175,45	214,83	126,48	0,32

Tabela 4. 8 - Potência computacional das máquinas utilizando *benchmark* Linpack e a métrica GH.

Analisando-se os dados referentes à Tab. 4.8, pode-se observar uma divergência entre as máquinas Lasdpc53 e Lasdpc54 com a execução do *benchmark* utilizando precisão dupla com e sem otimização. Na execução sem otimização, se for levado em consideração a métrica tempo de execução em segundos, o computador Lasdpc53 tem a melhor potência computacional, mas do ponto de vista da métrica MFLOPS o resultado é o oposto. Acredita-se que essa diferença ocorreu em função das diferentes implementações em hardware das unidades de ponto flutuante nesses dois processadores, o que faz variar o número e/ou peso das operações feitas por unidade de tempo. Em função disso, julga-se melhor analisar o desempenho desses processadores por meio do tempo de execução (Patterson et. al., 2005). Na execução com otimização (*unroll*), a máquina Ladpc53 manteve o seu desempenho superior à Lasdpc54 nas duas métricas.

Nos resultados utilizando precisão simples sem e com otimização, todos os resultados mostraram que o computador Lasdpc53 tem a melhor potência computacional. No entanto, a máquina Lasdpc54 obteve um desempenho inferior à Lasdpc14, ficando na terceira colocação.

Novamente, acredita-se que as diferentes demandas geradas nessas execuções foram as responsáveis pela diferença verificada.

O grau de heterogeneidade apresentado na Tab. 4.8 mostra que os valores obtidos utilizando precisão dupla com e sem otimização estão próximos. O mesmo pode ser dito com relação à utilização de precisão simples com e sem otimização. Mas, existe uma diferença na métrica GH entre os resultados com precisão simples e dupla, mostrando que o sistema é mais heterogêneo quando consideradas operações com precisão dupla. Isso exemplifica a necessidade de se ter cautela quando são comparados dados de avaliação de desempenho, sendo essencial especificar corretamente a demanda gerada pela aplicação final do usuário.

4.6.4 Um resumo dos resultados obtidos com os *benchmarks* de processador

As Figs. 4.2 e 4.3 apresentam um resumo do grau de heterogeneidade utilizando os *benchmarks* de processador (Whetstone, Dhrystone e Linpack). A partir dessas figuras, a métrica GH teve um comportamento estável e satisfatório frente aos diferentes *benchmarks* utilizados. A escolha do *benchmark*, como era esperado, é essencial e significativa para o resultado final da métrica GH. A escolha do *benchmark* correto deve ser guiada pela demanda gerada na aplicação, em relação às unidades funcionais da CPU.

Os resultados dos *benchmarks* mostram também a importância de não se considerar apenas a freqüência do processador como métrica para se prever um desempenho futuro das aplicações. Obviamente os diferentes aspectos arquiteturais dos processadores influenciam muito o desempenho final para o usuário, fato que pode ser observado nas variações obtidas com os diferentes *benchmarks* já executados.

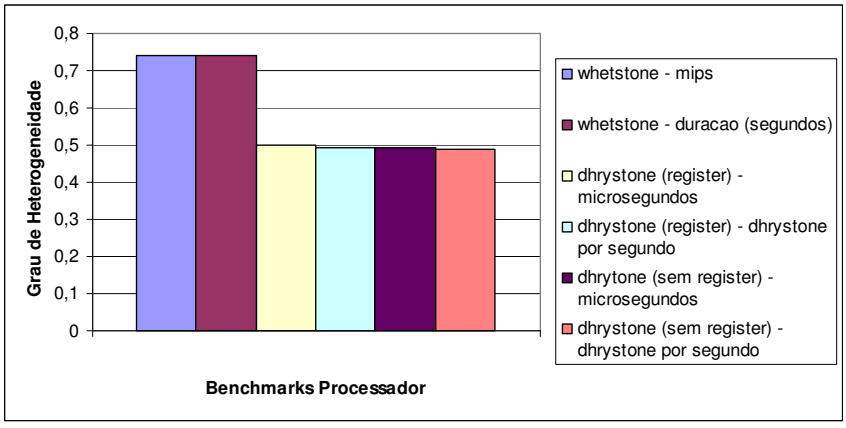

Figura 4. 2 - Resumo do grau de heterogeneidade utilizando *benchmarks* de processador.

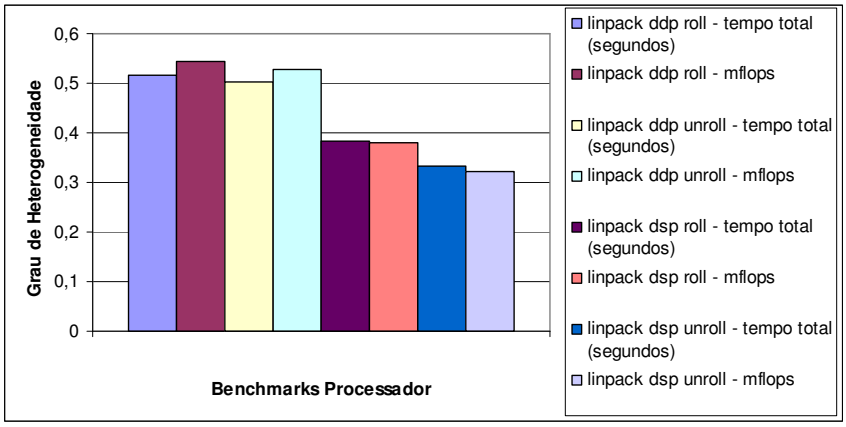

Figura 4. 3 - Continuação do resumo do grau de heterogeneidade utilizando *benchmarks* de processador.

4.6.5 *Benchmark* Cachebench

A Tab. 4.9 apresenta os resultados obtidos com o *benchmark* Cachebench, tendo como saída a taxa/vazão (MBytes/seg). O Cachebench, ao realizar a análise do desempenho da hierarquia de memória, classificou as máquinas na seguinte ordem decrescente: Lasdpc54, Lasdpc53, Lasdpc14, Lasdpc06 e Jaiminho.

Considerando os objetivos do Cachebench, estimar o desempenho da hierarquia de memória local incluindo vários níveis de memória *cache*, observa-se que houve uma variação na relação de desempenho existente entre as máquinas quando são comparadas as Tabs.4.1 e 4.10. Apesar de apresentar menos *cache*, a máquina Lasdpc14 apresentou um desempenho superior às máquinas Lasdpc06 e Jaiminho. Acredita-se que isso tenha ocorrido em função da maior quantidade de memória RAM existente na Lasdpc14. Em relação às máquinas Lasdpc54, Lasdpc53 e Lasdpc14, apesar de todas apresentarem a mesma quantidade de memória RAM, houve uma diferença relevante de desempenho. Acredita-se que dois fatores foram responsáveis por essa diferença: o impacto do desempenho da CPU no acesso à memória e a quantidade de memória *cache* existente.

Benchmark Cachebench					
Computador	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	Lasdpc54
Unidade	taxa (mbytes/seg)	taxa (mbytes/seg)	Taxa (mbytes/seg)	taxa (mbytes/seg)	taxa (mbytes/seg)
Média	984,22	1105,03	3509,58	5054,50	5400,31
Variância	5,44	60,83	204,81	1279,32	35285122,00
Desvio padrão	0,18	0,44	0,53	0,51	9,61

Tabela 4. 9 - Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Cache-bench.

Benchmark Cachebench						
	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	Lasdpc54	GH
taxa - mbytes/seg	984,22	1105,03	3509,58	5054,50	5400,31	0,53

Tabela 4. 10 - Potência computacional das máquinas utilizando *benchmark* Cachebench e a métrica GH.

A Tab. 4.10 apresenta um grau de heterogeneidade satisfatório levando-se em consideração a característica do *benchmark.*

4.6.6 *Benchmark* Stream

A Tab. 4.11 apresenta os resultados obtidos com o *benchmark* Stream, sendo as suas saídas as métricas: taxa/vazão (MBytes/segundo) e tempo médio em segundos. As métricas apresentadas na Tab. 4.11 são inversamente proporcionais. A Tab. 4.12 apresenta as métricas na mesma escala (quanto maior melhor).

Os resultados obtidos com o *benchmark* Stream são similares ao *benchmark* Cachebench. A ordem decrescente das máquinas em relação ao seu desempenho é: Lasdpc54, Lasdpc53, Lasdpc14, Lasdpc06 e Jaiminho.

Benchmark Stream				
Computador	Unidade	Média	Variância	Desvio padrão
Jaiminho	taxa – mb/s	126,55	0,20	0,43
	tempo médio - seg	0,31	0,00	0,00
Lasdpc06	taxa – mb/s	130,54	0,05	0,20
	tempo médio - seg	0,30	0,00	0,00
Lasdpc14	taxa – mb/s	547,02	0,21	0,41
	tempo médio - seg	0,07	0,00	0,00
Lasdpc53	taxa – mb/s	655,06	0,16	0,39
	tempo médio - seg	0,06	0,00	0,00
Lasdpc54	taxa – mb/s	981,36	0,24	0,48
	tempo médio - seg	0,04	0,00	0,00

Tabela 4. 11 - Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Stream.

Benchmark Stream						
Unidade	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	Lasdpc54	GH
taxa - mb/seg	126,55	130,54	547,02	655,06	981,36	0,58
tempo médio – seg	3,19	3,29	13,76	16,52	24,74	0,58

Tabela 4. 12 - Potência computacional das máquinas utilizando *benchmark* Stream e a métrica GH.

O grau de heterogeneidade apresentado na Tab. 4.12 indica que as duas métricas disponíveis mostram valores equivalentes, ou seja, pode-se escolher qualquer uma para representar a heterogeneidade do sistema.

4.6.7 Um resumo dos resultados obtidos com os *benchmarks* de memória

A Fig. 4.4 apresenta um resumo do grau de heterogeneidade utilizando os *benchmarks* de memória (Cachebench e Stream). A partir dessa figura, a métrica GH teve um comportamento estável e satisfatório frente aos dois *benchmarks* utilizados. A diferença

verificada na GH de 0,05 entre os *benchmarks* Cachebench e Stream não foi considerada relevante.

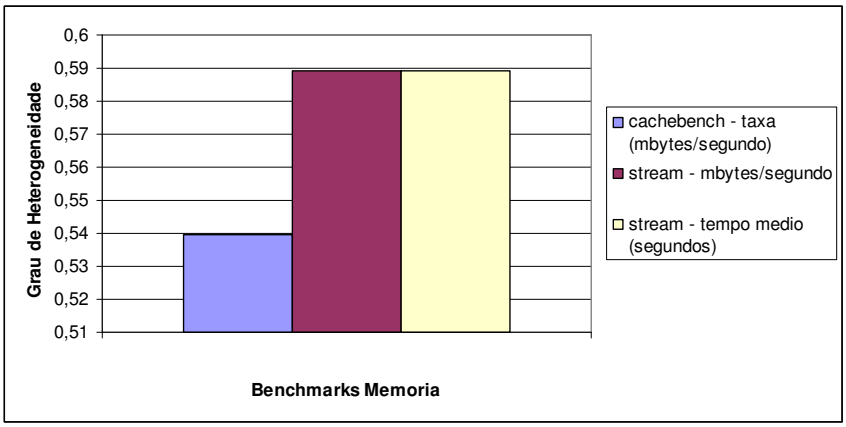

Figura 4. 4 - Resumo do grau de heterogeneidade utilizando *benchmarks* de memória.

4.6.8 *Benchmark* Netperf

Para realizar os experimentos de comunicação pela rede com o *benchmark* Netperf, foi preciso escolher uma máquina que desempenhasse a função de servidora. O computador escolhido foi o Lasdpc54 baseando-se nas informações coletadas do */proc*, ou seja, a Lasdpc54 apresenta uma das melhores configurações de *hardware* dentro do conjunto de máquinas disponíveis. Os testes foram realizados com acesso exclusivo à rede do laboratório (a rede foi isolada).

A Tab. 4.13 apresenta os resultados obtidos com o *benchmark* Netperf, o qual oferece 4 opções de execução (tcp - stream, udp - stream, tcp – rr e udp- rr). Cada uma das opções disponibiliza duas métricas para análise de desempenho: tempo decorrido em segundos e vazão (10^6 bits/segundo). As métricas apresentadas na Tab. 4.13 estão em escalas inversamente proporcionais, enquanto que a Tab. 4.14 apresenta os resultados na mesma escala (quanto maior melhor). A métrica tempo decorrido em segundos não conseguiu diferenciar as máquinas, pois o tempo de execução manteve-se em torno de 10s para todos os computadores. Segundo o tempo decorrido, a plataforma é homogênea.

Considerando que as versões TCP Stream e UDP Stream têm por objetivo avaliar a capacidade de banda do sistema computacional distribuído, observa-se que os valores obtidos para a taxa de transmissão (vazão) com tais métricas conseguiram diferenciar o desempenho das máquinas. Tendo-se como comparação a versão TCP Stream, a relação de desempenho (melhor para pior) ficou da seguinte forma: Lasdpc53, Lasdpc06, Lasdpc14 e Jaiminho. Considerando-se a versão UDP Stream a ordem decrescente estabelecida foi: Lasdpc53,

Jaiminho, Lasdpc14 e Lasdpc06. Nesta última versão a diferença não foi considerada relevante, dada a proximidade das médias.

Benchmark Netperf					
Computador	Tipo de execução	Unidade	Média	Variância	Desvio padrão
Jaiminho	tcp – stream	tempo decorrido - seg	10,0084	0,0064	0,0472
		vazão – 10^6 bits/seg	88,2231	40,5960	5,7186
	udp – stream	tempo decorrido - seg	9,9953	0,0000	0,0049
		vazão – 10^6 bits/seg	95,4941	0,0001	0,0107
	tcp - rr	tempo decorrido - seg	9,9973	0,0000	0,0044
		vazão – 10^6 bits/seg	1396,5190	29708,8100	172,3624
	udp - rr	tempo decorrido - seg	9,9956	0,0000	0,0050
		vazão – 10^6 bits/seg	64,8830	3959,3170	62,9231
Lasdpc06	tcp – stream	tempo decorrido - seg	10,0181	0,0209	0,0963
		vazão – 10^6 bits/seg	92,1191	58,8825	6,6350
	udp – stream	tempo decorrido - seg	9,9952	0,0000	0,0050
		vazão – 10^6 bits/seg	95,2954	0,0275	0,1087
	tcp - rr	tempo decorrido - seg	9,9953	0,0000	0,0050
		vazão – 10^6 bits/seg	5115,5630	85751,1500	292,8330
	udp - rr	tempo decorrido - seg	9,9946	0,0000	0,0050
		vazão – 10^6 bits/seg	5753,2780	94519,5400	307,4403
Lasdpc14	tcp – stream	tempo decorrido - seg	10,0908	0,3894	0,3173
		vazão – 10^6 bits/seg	91,0111	77,5210	7,6395
	udp – stream	tempo decorrido - seg	9,9952	0,0000	0,0050
		vazão – 10^6 bits/seg	95,3381	0,0011	0,0231
	tcp - rr	tempo decorrido - seg	9,9963	0,0000	0,0049
		vazão – 10^6 bits/seg	5401,0360	18136,3800	134,6714
	udp - rr	tempo decorrido - seg	9,9950	0,0000	0,0050
		vazão – 10^6 bits/seg	5559,4940	331866,5000	576,0785
Lasdpc53	tcp – stream	tempo decorrido - seg	10,0050	0,0000	0,0030
		vazão – 10^6 bits/seg	94,1517	0,0012	0,0249
	udp – stream	tempo decorrido - seg	9,9951	0,0000	0,0050
		vazão – 10^6 bits/seg	95,4997	0,0015	0,0295
	tcp - rr	tempo decorrido - seg	9,9953	0,0000	0,0050
		vazão – 10^6 bits/seg	14686,9	92,46065	9,6156
	udp - rr	tempo decorrido - seg	9,9953	0,0000	0,0050
		vazão – 10^6 bits/seg	15988,1600	237,7931	15,4205

Tabela 4. 13 - Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Net-perf.

As versões TCP-rr e UDP-rr também apresentaram resultados distintos para a taxa de transmissão (vazão) sendo as máquinas assim classificadas (da melhor para a pior): Lasdpc53, Lasdpc14, Lasdpc06 e Jaiminho para TCP-rr; e Lasdpc53, Lasdpc06, Lasdpc14 e Jaiminho para UDP-rr.

Benchmark Netperf						
Tipo de execução	Unidade	Jaiminho	Lasdpc06	Lasdpc14	Lasdpc53	GH
tcp – stream	tempo decorrido – seg	0,0999	0,0998	0,0991	0,0999	0,0029
	vazão – 10^6 bits/seg	88,2231	92,1191	91,0111	94,1517	0,0192
udp – stream	tempo decorrido - seg	0,1000	0,1000	0,1000	0,1000	0,0000
	vazão – 10^6 bits/seg	95,4941	95,2954	95,3381	95,4997	0,0009
tcp – rr	tempo decorrido – seg	0,1000	0,1000	0,1000	0,1000	0,0000
	vazão – 10^6 bits/seg	1396,5190	5115,5630	5401,0360	14686,9000	0,6042
udp – rr	tempo decorrido – seg	0,1000	0,1000	0,1000	0,1000	0,0000
	vazão – 10^6 bits/seg	64,8830	5753,2780	5559,4940	15988,1600	0,6684

Tabela 4. 14 - Potência computacional das máquinas utilizando *benchmark* Netperf e a métrica GH.

Quando comparados os dois grupos, TCP-Stream/UDP-Stream com TCP-rr/UDP-rr observa-se uma grande diferença no resultado obtido com a métrica GH. O primeiro grupo indica que a plataforma é homogênea e o segundo indica heterogeneidade. Essa diferença espelha as diferentes demandas geradas pelos dois grupos, tanto sobre a rede quanto sobre o preparo das mensagens a serem enviadas. No primeiro grupo (TCP-Stream/UDP-Stream) é aberta uma conexão entre as máquinas e então são enviadas mensagens tão rápido quanto possível. Assim, acredita-se que o tempo dominante seja o tempo de transmissão pela rede, visto que o tempo de preparar a mensagem para a transmissão é minimizado ao máximo. Já para o segundo grupo (TCP-rr/UDP-rr) diferentes mensagens são montadas do tamanho de um MTU, impondo uma parcela de tempo maior para o preparo de tais mensagens no computador. Considerando essa análise, conclui-se que os resultados foram condizentes com o esperado, dada a diferença existente nas demais configurações da máquina, principalmente aquelas relativas ao desempenho do processador.

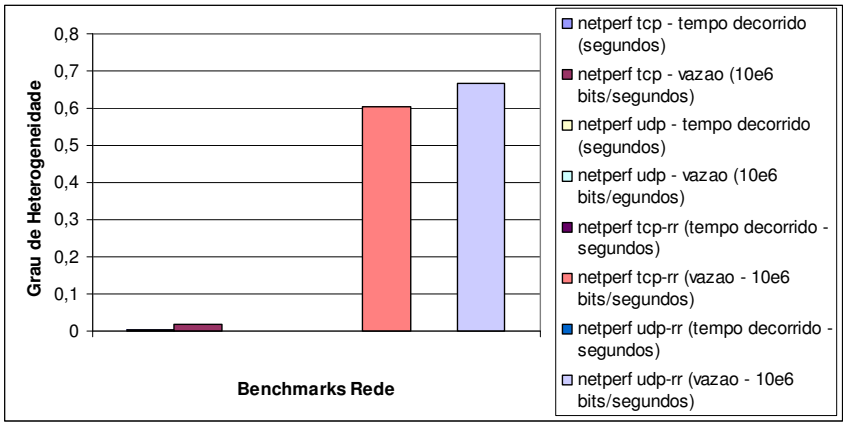

Figura 4. 5 - Resumo do grau de heterogeneidade utilizando *benchmark* Netperf.

A Fig. 4.5 apresenta um resumo do grau de heterogeneidade utilizando o *benchmark* Netperf. A partir dessa figura observa-se que a métrica GH teve um comportamento satisfatório e que, novamente, a escolha dos tipos de execução do *benchmark* é significativa

para o resultado final da métrica GH. Deve-se escolher o tipo de execução mais apropriado à demanda esperada.

4.7 CONSIDERAÇÕES FINAIS

Este capítulo descreveu os *benchmarks* encontrados na literatura e que foram utilizados para analisar o desempenho da métrica GH através de diferentes perspectivas: processador (Whetstone, Dhrystone, e Linpack), memória (Cachebench e Stream) e rede (Netperf).

Além da descrição dos *benchmarks*, foram apresentados também os resultados obtidos com os mesmos em um experimento realizado com 5 computadores instalados no Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC) do ICMC/USP. Os resultados mostraram que as mesmas máquinas podem apresentar grandes variações de desempenho e, conseqüentemente, no grau de heterogeneidade da plataforma distribuída que tais máquinas formam.

O uso de *benchmarks* corretos, embora suscetível às otimizações pelo hardware e pelo compilador, permite uma análise de desempenho desassociada das informações de configuração, tais como freqüência de processador e quantidade de memória. Usando-se a métrica tempo de maneira idônea, pode-se comparar o desempenho de diferentes máquinas e, assim, obter uma previsão do desempenho de aplicações com características semelhantes àquelas descritas pelo *benchmark* utilizado.

A métrica GH apresentou um ótimo comportamento satisfatório sempre indo ao encontro do resultado do *benchmark* utilizado. A escolha correta do *benchmark*, como era esperado inicialmente, é essencial para que o resultado final da métrica GH reflita adequadamente o grau de heterogeneidade da plataforma, frente à demanda gerada pela aplicação.

5 O USO DA MÉTRICA GH EM UM AMBIENTE DE ESCALONAMENTO ADAPTATIVO

5.1 CONSIDERAÇÕES INICIAIS

Este capítulo apresenta um estudo de caso onde a métrica GH é inserida em um ambiente de escalonamento real. O objetivo é que a informação sobre o grau de heterogeneidade da plataforma seja utilizada durante a atividade de escalonamento dos processos concorrentes pertencentes a uma aplicação paralela. Espera-se verificar com este estudo de caso, o impacto no desempenho final da aplicação, quando a GH é usada em uma política de escalonamento.

Afim de se agregar as informações advindas com a GH em uma política de escalonamento, este trabalho de mestrado propõe o uso da GH no AMIGO, que permitirá a construção de políticas de escalonamento adaptativas. Tais políticas alteram o sua funcionalidade em tempo de execução (Casavant et. al., 1988). Dessa maneira, o grau de heterogeneidade da plataforma indicado pela GH será usado para indicar quando a troca de uma política de escalonamento deve ser realizada pelo AMIGO.

Para o desenvolvimento deste estudo de caso foram utilizados os seguintes recursos: um *cluster* Beowulf formado por 10 máquinas, o ambiente de escalonamento flexível e dinâmico AMIGO com a sua política de escalonamento distribuída DPWP, o ambiente de passagem de mensagem PVM com a sua política de escalonamento *round-robin* e uma aplicação paralela que soluciona um sistema linear através do método iterativo de Gauss-Jacobi, também conhecido como Jacobi-Richardson.

Este capítulo apresenta uma breve descrição da estrutura do AMIGO, a política de escalonamento DPWP e o ambiente de passagem de mensagem PVM. Em seguida, são descritas as alterações realizadas no ambiente de escalonamento AMIGO afim de viabilizar o cálculo e o uso da métrica GH. Na seqüência são apresentadas a aplicação paralela Jacobi, a plataforma computacional usada no estudo de caso e os cenários de execução propostos. Finalizando o capítulo, os resultados obtidos são apresentados e discutidos.

5.2 ESTRUTURA DO AMIGO

O AMIGO é composto por duas camadas principais: uma superior e outra inferior como pode ser observado na Fig. 5.1. Essa divisão na sua estrutura interna permite que a atividade de escalonar processos (feita pela camada inferior) seja independente da

configuração e do monitoramento (ambos feitos pela camada superior). Essa divisão também permite uma maior modularidade ao ambiente, assim como facilita a portabilidade do mesmo para outras plataformas. Apenas a camada inferior é mais dependente da plataforma que está sendo usada. A camada superior, embora também seja dependente de alguns fatores (como linguagem de programação e sistema operacional), pode ser executada em outras plataformas. Já a camada inferior, utiliza todas as primitivas necessárias para a comunicação, obtenção da carga computacional, execução de novos processos, etc. Essas atividades estão relacionadas à plataforma utilizada e precisam ser adaptadas às novas plataformas (Souza et al., 2001).

Outra vantagem da divisão feita é que para utilizar o AMIGO, uma vez configurado o escalonamento, não é necessário executar a camada superior do ambiente. O usuário apenas executa a sua aplicação com o PVM ou com o MPI, e o escalonamento correto será empregado pela camada inferior de maneira transparente ao usuário e independente da camada superior (Souza et al., 2001).

A camada superior é composta por uma interface gráfica, pela qual o usuário tem acesso às várias opções do ambiente, como por exemplo: inserir as classes de *software* - grupo de aplicações a serem escalonadas; inserir o *hardware* utilizado; inserir *benchmarks* usados para determinar a heterogeneidade existente na plataforma; inserir as métricas utilizadas para monitorar o escalonamento feito; e inserir e determinar as políticas de escalonamento. Na camada inferior encontram-se os algoritmos disponíveis para a realização do escalonamento, assim como a interface necessária ao relacionamento com o ambiente de passagem de mensagem utilizado (Souza et al., 2001).

Informações mais detalhadas sobre as características e funcionamento do AMIGO podem ser encontradas em Souza (Souza et al., 1996) (Souza et al., 1999a) (Souza et al., 1999b) (Souza et al., 1999c) (Souza et al., 2000) (Souza et al., 2001).

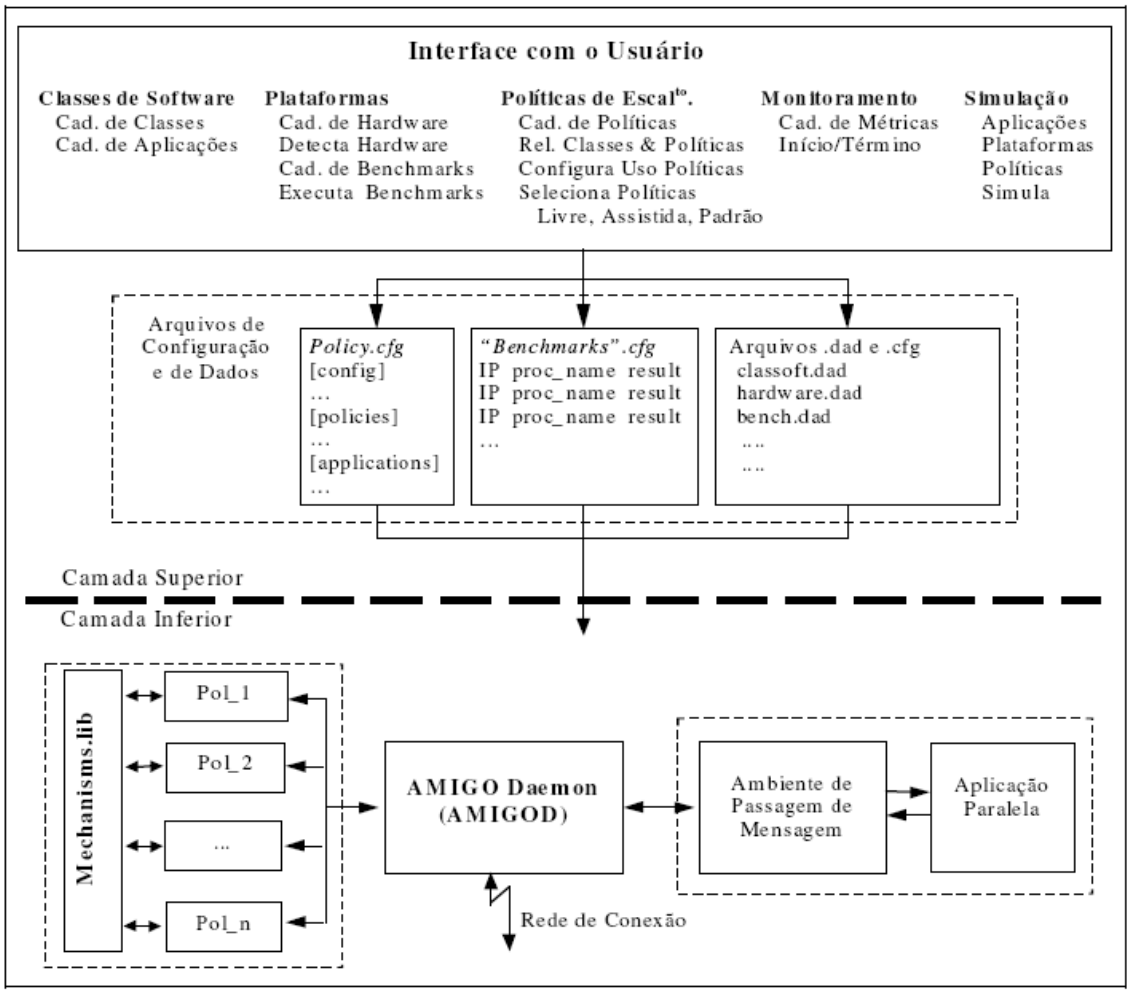

Figura 5. 1 – Esquema da estrutura do AMIGO (Souza et al., 2001).

5.3 POLÍTICA DE ESCALONAMENTO

Uma das principais características do AMIGO é permitir que diferentes políticas possam ser implementadas sob o mesmo. Essas políticas compõem um módulo independente e podem apresentar características e objetivos diversos (Souza et al., 2001).

A separação das políticas de escalonamento em um módulo à parte apresenta a desvantagem de aumentar a comunicação entre processos existentes na camada inferior, mas possui a vantagem de viabilizar a flexibilidade e a característica dinâmica do AMIGO. Sendo a política um módulo à parte, a mesma pode ser substituída por outra de maneira transparente à aplicação e ao ambiente de passagem de mensagem, sendo que o AMIGOD se encarrega de encapsular essa mudança (Souza et al., 2001).

As duas principais características das políticas implementadas são:

- a independência do ambiente de passagem de mensagem adotado, porque interagem somente com o AMIGOD (Souza et al., 2001).

- e a tendência por algoritmos específicos, uma vez que o objetivo é ter políticas pequenas e mais eficientes em determinadas situações (Souza et al., 2001).

Quando uma política está em funcionamento, ela é executada em cada processador onde o AMIGOD também está sendo executado. Como o AMIGOD não implementa nenhuma política de escalonamento é necessário que em cada processador haja uma cópia da mesma para que possam ser efetivadas as decisões sobre o escalonamento. Embora o AMIGOD seja distribuído, as políticas de escalonamento não necessitam ser distribuídas, podendo ser centralizadas. Nesse caso, um processador mestre (do ponto de vista da política) decide como o escalonamento será realizado e os outros processadores escravos apenas coletam informações sobre a carga computacional (Souza et al., 2001).

Para construir uma nova política de escalonamento para o AMIGO, o projeto da mesma deve considerar e se adaptar:

- às regras de inicialização e finalização dos módulos da camada inferior (Souza et al., 2001).

- ao protocolo de comunicação entre os módulos (Souza et al., 2001).

- e aos arquivos de configuração disponíveis através da camada superior e que contém informações como *hardware*, *benchmarks*, etc (Souza et al., 2001).

A primeira política de escalonamento especificada e implementada para o AMIGO é a DPWP (*Dynamical Policy Without Preemption*). Os principais objetivos da DPWP são o balanceamento das cargas nos processadores e a redução do tempo de execução dos processos (Souza et al., 2001).

Dentre as características da DPWP destacam-se:

- ser dinâmica, utilizando informações em tempo de execução mais atualizadas e precisas, ao mesmo tempo em que tenta minimizar as mensagens enviadas entre as políticas (Souza et al., 2001).

- ser voltada para aplicações *cpu-bound* (Souza et al., 2001).

- não possuir migração de processos, e ser direcionada às plataformas com pouca variação das cargas computacionais em curtos intervalos de tempo (Souza et al., 2001).

- ser objetiva e específica a um grupo de características, apresentando dessa forma um código pequeno e eficiente (Souza et al., 2001).

A DPWP também contempla a heterogeneidade da plataforma computacional, possibilitando que as decisões possam considerar as diferenças existentes entre os processadores disponíveis. Dessa forma, além de considerar o índice de carga escolhido (número de processos prontos na fila), a DPWP também normaliza os valores obtidos com a potência computacional de cada processador. Informações mais detalhadas sobre as características e funcionamento da DPWP podem ser encontradas em Araújo (Araújo, 1999) (Araújo et al., 1999a) (Araújo et al., 1999b).

5.4 O AMBIENTE DE PASSAGEM DE MENSAGEM

Os ambientes de passagem de mensagem compõem um dos módulos da camada inferior. Eles atuam como um cliente do AMIGO, solicitando-lhes os processadores necessários às aplicações. Esses ambientes (PVM e MPI) sofreram pequenas, mas significativas alterações para que pudessem interagir com o ambiente (Souza et al., 2001).

As mudanças necessárias foram:

- na inclusão e remoção de novos processadores na máquina paralela virtual, onde novos AMIGODs devem ser iniciados e finalizados (Souza et al., 2001);

- no escalonamento de novos processos nos processadores (Souza et al., 2001);

- e na migração de processos que já estão em execução (Souza et al., 2001).

O escalonamento de novos processos (no caso do PVM) é iniciado com a execução da *pvm_spawn()*, a qual irá solicitar a execução de novos processos sobre os processadores disponíveis. Conforme pode-se observar na Fig. 5.2, a *pvm_spawn()* na biblioteca LIBPVM apenas prepara uma mensagem a ser enviada ao PVMD local, o qual recebe a solicitação pela *tm_spawn()*. A *tm_spawn()*, com base nos parâmetros enviados pela *pvm_spawn()*, determina quais processadores farão parte do escalonamento, inserindo-os em uma estrutura de dados própria e depois chama a rotina *assign_tasks()* no PVMD local (Souza et al., 2001).

A rotina *assign_tasks()* verifica o número de processos a escalonar e encontra o mínimo entre o número de processos e o número de processadores determinados pela *tm_spawn()*. Isso é feito para saber se será necessário atribuir mais de um processo para o mesmo processador. Determinando quantos processos vão para cada processador da relação, o PVMD local executa os processos que devem ser executados localmente e envia mensagens para os PVMDs remotos solicitando-lhes que executem os processos que faltam, através de uma política de escalonamento *round-robin* (Souza et al., 2001).

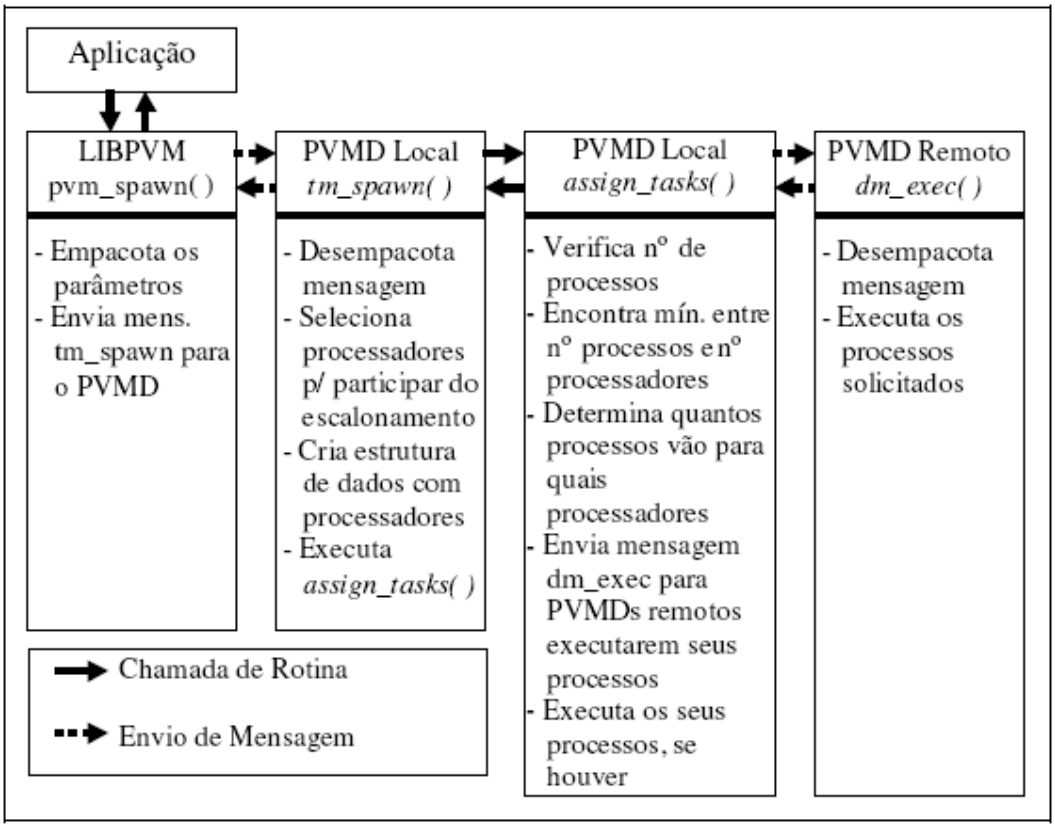

Figura 5. 2 – Esquema gráfico do escalonamento original no PVM (Souza et al., 2001).

A Fig. 5.3 ilustra a alteração feita no código do PVMD com a inclusão da rotina *GetHostsFromAMIGOD()*. Essa rotina, chamada pela *tm_spawn()*, determina quais processadores farão parte do escalonamento em substituição à parte do código da *tm_spawn()* responsável por isso (*tm_spawn()* na Fig. 5.2). A *GetHostsFromAMIGOD()* envia uma mensagem para o AMIGOD contendo o nome do processo a ser escalonado, o número de instâncias a criar desse processo e a arquitetura desejada. O AMIGOD ao receber essa mensagem determina qual política de escalonamento deve receber essa solicitação e repassa essa mensagem para a mesma. As mensagens retornadas da política e do AMIGOD contêm uma relação com os processadores receptores dos novos processos e a ordem em que os mesmos devem ser utilizados no escalonamento (Souza et al., 2001).

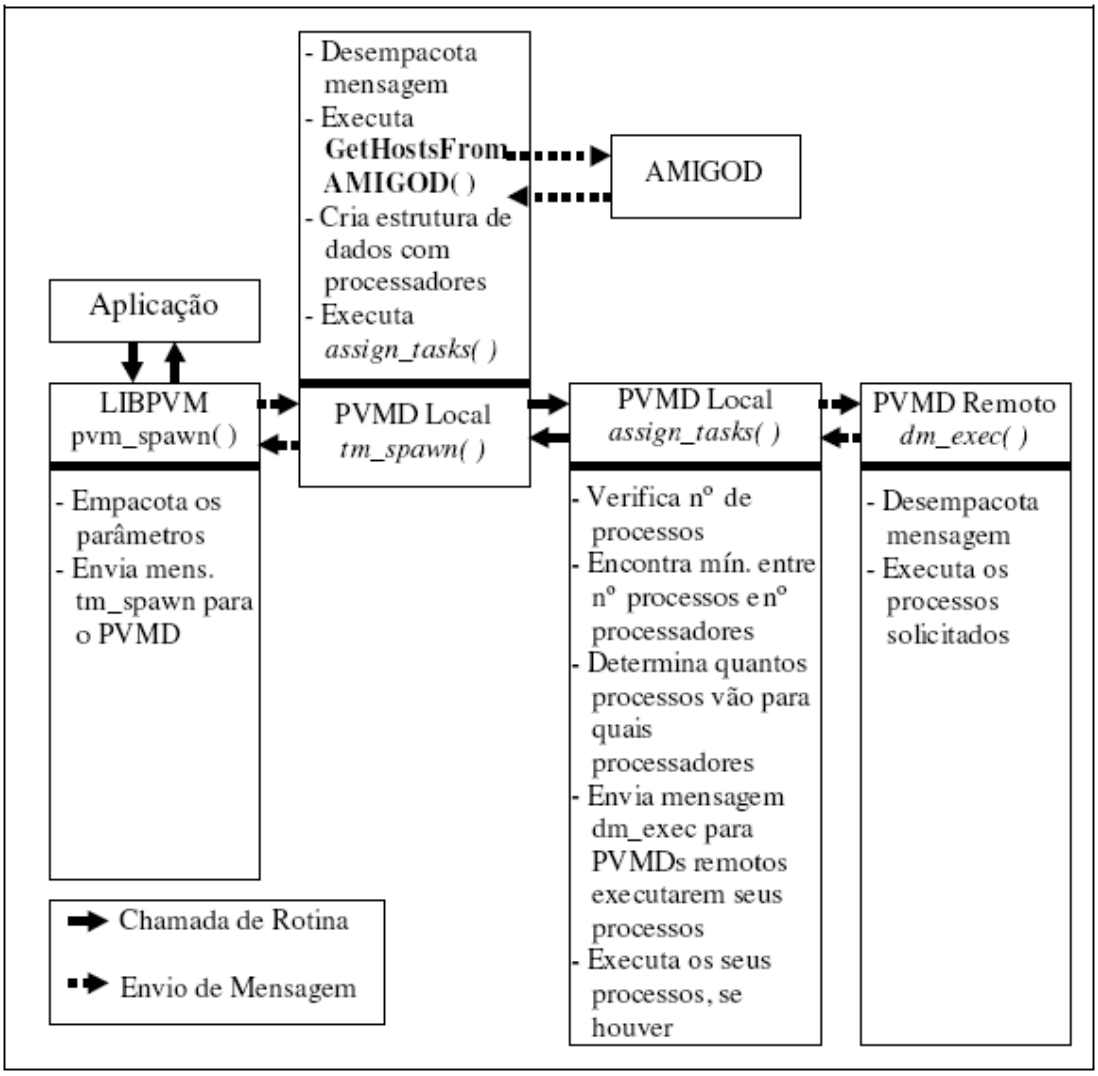

Figura 5. 3 – Esquema gráfico do escalonamento no PVM com o AMIGO (Souza et al., 2001).

As alterações feitas no ambiente de passagem de mensagem para o AMIGO são sutis, mas decisivas para alterar a política de escalonamento usada. As rotinas inseridas atuam como um elo de ligação entre o ambiente de passagem de mensagem e o AMIGOD. Assim, para interagir com o AMIGOD é utilizada uma estrutura de dados própria do AMIGO e, ao interagir com o ambiente de passagem de mensagem, convertem-se os dados para a estrutura de dados adequada ao mesmo (Souza et al., 2001).

Considerando essas características, pode-se afirmar que adaptar o AMIGO para outro ambiente de passagem de mensagem ou outra ferramenta de *software* que necessite realizar escalonamento (como uma linguagem de programação ou um sistema operacional), é necessário mudar somente as rotinas pertencentes ao AMIGO e incluídas no código do PVM.

As rotinas do AMIGOD e das políticas de escalonamento se adaptarão à mudança porque o protocolo de comunicação entre os módulos permanecerá o mesmo (Souza et al., 2001).

Informações mais detalhadas sobre as características e funcionamento do AMIGO podem ser encontradas em Souza (Souza et al., 1996) (Souza et al., 1999a) (Souza et al., 1999b) (Souza et al., 1999c) (Souza et al., 2000) (Souza et al., 2001).

5.5 INSERINDO O GRAU DE HETEROGENEIDADE DA PLATAFORMA NO AMIGO

O grau de heterogeneidade da plataforma instanciado pela métrica GH tem por objetivo representar as diferenças (ou similaridades) no desempenho dos computadores que compõem uma plataforma paralela/distribuída. Afim de utilizar essa informação em um ambiente de escalonamento real, a métrica GH foi inserida no AMIGO, permitindo assim a realização de um estudo de caso como prova de conceito.

Além de agregar mais qualidade à atividade de escalonar processos com o uso da métrica GH no AMIGO, essa inclusão do grau de heterogeneidade foi pautada pelos seguintes itens: 1) garantir flexibilidade ao uso da GH com diferentes políticas de escalonamento; 2) permitir portabilidade entre os ambientes de passagem de mensagens, como o PVM e o MPI; 3) garantir transparência total para a aplicação paralela/distribuída; e 4) ter uma instrusão no código tão pequena quanto possível afim de viabilizar os aspectos citados acima.

Baseando-se nesses itens, as principais alterações realizadas foram o desenvolvimento de uma nova função chamada CalculaGH( ) e a inserção desta nova função na *tm_spawn( ),* no caso do PVM (Fig. 5.4). Esse procedimento altera de maneira sutil, porém, decisiva o fluxo de execução do algoritmo de escalonamento empregado. Dada a arquitetura do AMIGO, essa alteração pode ser facilmente empregada também no MPI, visto que este também utiliza uma versão adaptada na função *GetHostsFromAMIGOD()* para solicitar a relação de processadores que estão aptos ao escalonamento.

Com essa mudança na *tm_spawn()*, em linhas gerais, o fluxo de execução do AMIGO será o seguinte: caso a plataforma seja considerada mais homogênea que heterogênea o ambiente de escalonamento optará pela política *round-robin* existente no próprio PVM, evitando assim toda sobrecarga de comunicação imposta pelo AMIGO. Caso a plataforma seja mais heterogênea que homogênea, o escalonamento dos processos será realizado pela DPWP, considerando as suas diferenças de desempenho. Espera-se com essa heurística propiciar um ambiente de escalonamento adaptativo que agregue maior simplicidade à

atividade de escalonar processos quando possível e maior qualidade (porém com maior sobrecarga de comunicação) quando as características da plataforma exigirem. Como resultado final espera-se uma diminuição do tempo de execução da aplicação paralela executada tanto em plataformas homogêneas quanto em plataformas heterogêneas. Uma variável de ambiente chamada GH_PADRAO foi definida e seu conteúdo é utilizado como um limite (*threshold*) para indicar quando uma política de escalonamento deve considerar a plataforma homogênea ou heterogênea.

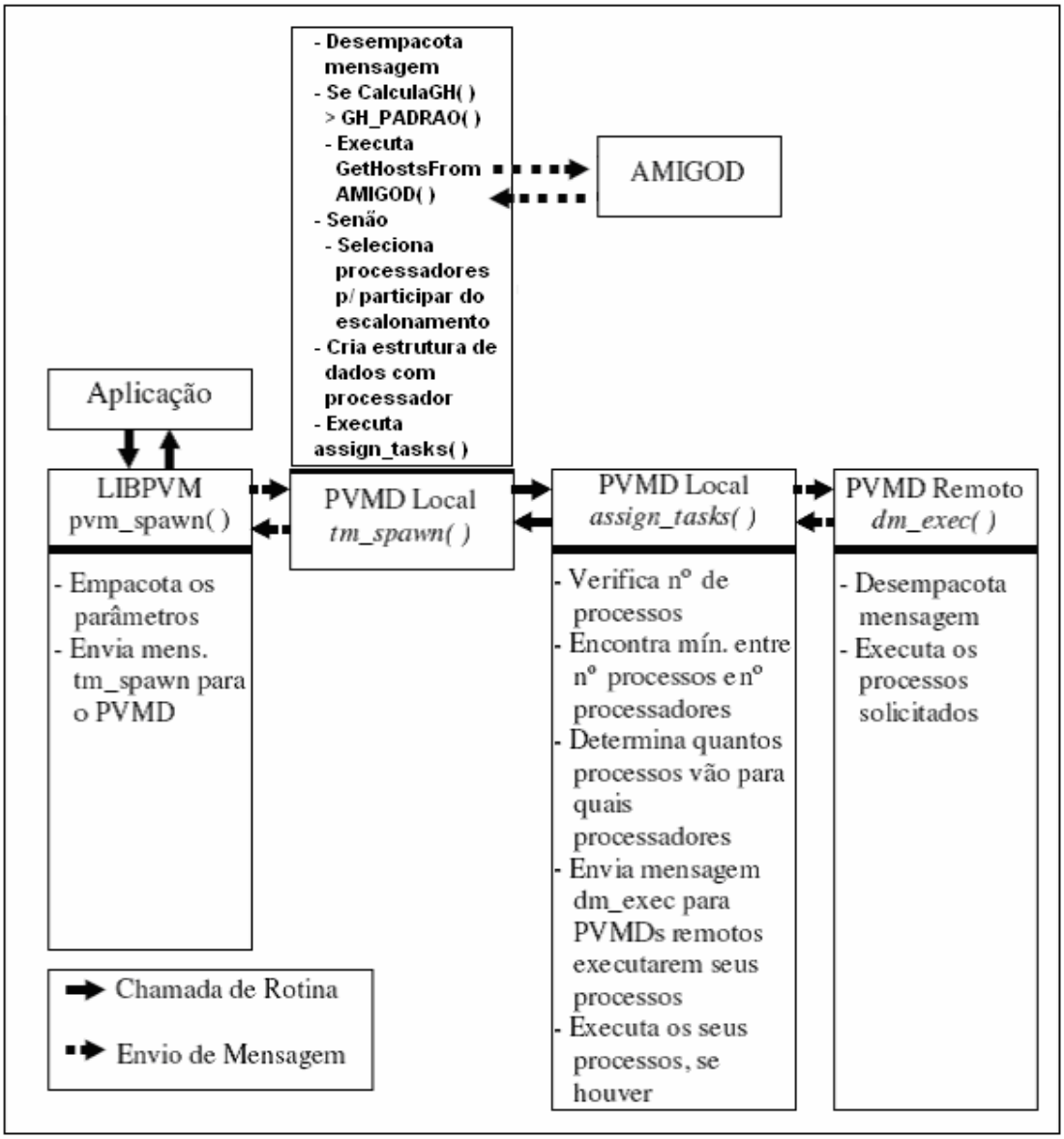

Figura 5. 4 – Esquema gráfico do escalonamento do PVM com o AMIGO levando-se em consideração a métrica GH.

A função CalculaGH() calcula o grau de heterogeneidade da plataforma tendo como entrada as potências computacionais dos computadores disponíveis. A CalculaGH() desempenha as seguintes atividades: obtém as potências computacionais das máquinas pertencentes ao *cluster*; calcula a métrica GH usando a Eq. 3.10 e as potências computacionais; e retorna o grau de heterogeneidade da plataforma.

A obtenção das potências computacionais no AMIGO é realizada por meio de um arquivo de configuração contendo para cada linha o nome da máquina e a sua potência computacional já normalizada. A Fig. 5.5 ilustra o formato desse arquivo onde pode-se observar valores de 1.0 a infinito. O valor 1.0 representa a máquina com a menor potência computacional. Este valor de potência computacional é usado como entrada para o cálculo da GH.

```
Nome da máquina=potência computacional
Máquina1=1.0
Máquina2=1.2
Máquina3=1.4
...
MáquinaN=infinito
```

Figura 5. 5 – Arquivo de configuração contendo o nome da máquina e a respectiva potência computacional.

Conforme visto anteriormente nessa dissertação, as potências computacionais podem ser calculadas de diferentes maneiras, inclusive a partir de *benchmarks*. Neste experimento as potências foram estipuladas a partir do *benchmark* Linpack, cuja implementação apresenta características semelhantes às da aplicação Jacobi utilizada nos experimentos.

5.6 APLICAÇÃO GAUSS-JACOBI

Gauss-Jacobi é um método iterativo para a solução do sistema linear **Ax=b**, onde: **A** é uma matriz **n x n** de coeficientes; **x** é um vetor **n x 1** das variáveis; e **b** é um vetor **n x 1** dos termos constantes (Ruggiero et. al., 1998).

O método Gauss-Jacobi pode ser deduzido intuitivamente do seguinte sistema:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned}$$

Isolando-se o vetor de soluções $\mathbf{x_k}$ obtém-se:

$$x_1 = (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11}$$
$$x_2 = (b_2 - a_{21}x_1 - a_{23}x_3)/a_{22}$$
$$x_3 = (b_3 - a_{31}x_1 - a_{32}x_2)/a_{33}$$

Nesse caso, considerando-se que os elementos $\mathbf{a_{ii}}$ são diferentes de zero e que $\mathbf{x_k}$ é uma aproximação para a solução do sistema **Ax=b**, pode-se fazer a seguinte generalização:

$$x_1^{k+1} = (b_1 - a_{12}x_2^k - a_{13}x_3^k)/a_{11}$$
$$x_2^{k+1} = (b_2 - a_{21}x_1^k - a_{23}x_3^k)/a_{22}$$
$$x_3^{k+1} = (b_3 - a_{31}x_1^k - a_{32}x_2^k)/a_{33}$$

Dessa maneira chega-se a seguinte formulação:

$$x_i^{k+1} = (b_i - \sum_{j=1}^{i-1} a_{ij}x_j^k - \sum_{j=i+1}^{n} a_{ij}x_j^k)/a_{ii}$$

para i = 1,. . . ,n.

Esse método gera uma seqüência de soluções aproximadas até o momento que o resíduo atende a uma tolerância inicialmente imposta. O resíduo é verificado ao final de cada iteração e corresponde a $\left\| x_n^{k+1} - x_n^k \right\|$. Outro critério de parada pode ser um número máximo de iterações.

Há também a necessidade de se verificar a convergência do sistema, ou seja, o critério de convergência. Tal critério pode ser realizado por meio do critério das linhas, sendo este o utilizado neste trabalho (Ruggiero et. al., 1998).

5.6.1 Algoritmo Jacobi Paralelo

A aplicação paralela Jacobi descrita nesta seção implementa o método iterativo Gauss-Jacobi descrito anteriormente. A aplicação paralela foi desenvolvida na linguagem C, usando o ambiente de passagem de mensagem PVM no paradigma mestre-escravo. O código fonte da aplicação encontra-se no Apêndice A desta dissertação.

A execução começa com a execução do processo mestre, o qual é responsável pela geração dos **p** processos escravos. A ordem da matriz (**n**) e o número de processos escravos a serem gerados (**p**) são informados como argumentos de entrada pelo usuário. Cada processo escravo receberá uma faixa de dados com **n/p** elementos, sendo que o último processo, eventualmente, poderá receber elementos a mais em virtude de uma faixa com divisão não exata de **n/p** elementos.

O processo mestre envia aos **p** escravos a ordem **n** usada no sistema mais a quantidade **p** de processos escravos existentes e determina a quantidade de valores a enviar aos escravos. O mestre gera de maneira pseudo-aleatória os dados da matriz A e do vetor B e, na seqüência envia as respectivas faixas de valores da matriz A e do vetor B aos escravos correspondentes. O mestre recebe os testes de convergência vindos dos escravos e envia as mensagens aos escravos suspendendo ou autorizando o cálculo da matriz A e do vetor B.

Os processos escravos recebem a primeira mensagem do mestre informando a ordem **n** e a quantidade total de escravos **p**. Os escravos recebem as suas respectivas faixas de valores contento os dados da matriz A e do vetor B.

Os escravos determinam em paralelo a convergência do sistema considerando apenas a sua faixa de valores. Após esse cálculo, os escravos enviam ao mestre seus resultados parciais sobre a convergência.

O mestre recebe os resultados parciais da convergência vindos dos escravos e decide pela suspensão ou não da execução em função do resultado sobre a convergência final do sistema. Essa permissão ou suspensão da continuidade da execução é enviada para os escravos através de uma nova mensagem. É importante salientar aqui que, neste estudo de caso, a geração pseudo-aleatória dos dados sempre garante a convergência do sistema. Independentemente disso, optou-se por sempre executar o critério de convergência.

Após receberem a mensagem contendo a autorização para iniciarem o processo iterativo, todos os escravos trocam mensagens entre si informando os elementos de B que, até esse momento, estavam divididos respectivamente entre os escravos.

Enquanto não for atingido o critério de parada, os escravos iterativamente calculam em paralelo os seus respectivos valores para o novo vetor **x**, sendo que a cada passo é enviada

uma mensagem ao mestre para que ele agrupe os valores parciais de **x** e verifique o critério de parada.

Após a última iteração o processo mestre terá o vetor **x** final compreendendo a solução do sistema.

5.7 *CLUSTER* BEOWULF UTILIZADO NO ESTUDO DE CASO

Para as execuções realizadas neste trabalho, foram utilizadas 10 máquinas denominadas Colossus, Tempestade, Demolidor, Venon, Vampira, Gambit, Elektra, Anjo, Justiceiro e Noturno. Todas fazem parte de um *cluster* Beowulf localizado no ICMC/USP em São Carlos. Os nós do *cluster* apresentam a seguinte configuração: Processador Intel Pentium 4 – 64 Bits - 3.4GHz, 4GB de memória, 160 GB de disco compartilhado e placa de rede Gigabit Ethernet. O *cluster* utiliza uma rede interna de 1 GB e o sistema operacional Linux, distribuição OpenSuse 10.0.

Como pode-se observar pela configuração do *cluster* utilizado, este caracteriza-se como uma plataforma distribuída homogênea. Afim de se avaliar o comportamento da GH no AMIGO com a aplicação Jacobi, não apenas em um ambiente homogêneo, foram considerados três cenários distintos: um *cluster* homogêneo, um *cluster* parcialmente heterogêneo e outro heterogêneo. Essa heterogeneidade no *cluster* foi obtida inserindo-se carga de trabalho em alguns nós, para que houvesse uma diferença de desempenho fixa entre os mesmos.

As cargas extras foram geradas utilizando-se algumas instâncias dos seguintes *benchmarks*: Linpack, Dhrystone e Whetstone. A geração da carga extra: de 10% utilizou o Linpack ddp – *unroll*; a de 20% usou o Linpack ddp - *unroll* e o Linpack ddp – *roll*; a de 30% utilizou o Linpack ddp – *unroll*, o Linpack ddp – *roll* e o Linpack dsp – *unroll*; a de 40% usou o Linpack ddp – *unroll*, o Linpack ddp – *roll*, o Linpack dsp – *unroll* e o Linpack dsp – *roll*; a de 60% utilizou o Linpack ddp – *unroll*, o Linpack ddp – *roll*, o Linpack dsp – *unroll*, o Linpack dsp – *roll*, Dhrystone sem *register* e Dhrystone com *register*; e a de 80% usou o Linpack ddp – *unroll*, o Linpack ddp – *roll*, o Linpack dsp – *unroll*, o Linpack dsp – *roll*, Dhrystone sem *register*, Dhrystone com *register* e Whetstone.

Dessa forma, o cenário homogêneo é composto pelos 10 computadores sem qualquer carga extra. No cenário parcialmente heterogêneo, as máquinas estão agrupadas em pares, ou seja, Colossus/Tempestade - sem carga extra, Demolidor/Venon – cada uma tem 20% de

carga extra, Vampira/Gambit – 40% de carga extra cada uma, Elektra/Gambit – cada uma com 10% de carga extra e Justiceiro/Noturno – 30% de carga extra cada. No cenário heterogêneo, as máquinas também estão agrupadas aos pares: Colossus/Tempestade - sem carga extra, Demolidor/Venon – cada uma tem 20% de carga extra, Vampira/Gambit – 40% de carga extra cada uma, Elektra/Gambit – cada uma com 60% de carga extra e Justiceiro/Noturno – 80% de carga extra cada.

Para cada um dos 3 cenários foram coletadas e registradas as informações sobre a potência computacional de cada máquina utilizando o *benchmark* Linpack. A opção pelo Linpack deve-se às suas características de implementação (operações com vetores de ponto flutuante). Em cada um dos 3 cenários é utilizado tanto a política de escalonamento r*ound-robin* pertencente ao PVM quanto a DPWP pertencente ao AMIGO.

5.7.1 Cenário Homogêneo

As Tabs. 5.1 a 5.3 apresentam as potências computacionais de cada máquina (média aritmética, variância e desvio padrão das 30 execuções realizadas pelo *benchmark* Linpack), o grau de heterogeneidade (GH) obtido e o tempo de execução da aplicação Jacobi no cenário homogêneo (média aritmética, variância e desvio padrão das 30 execuções).

A Tab. 5.1 apresenta os valores obtidos com o *benchmark* Linpack utilizando a métrica tempo total em segundos no ambiente homogêneo, os resultados estão na mesma proporção/escala dos valores apresentados na Tab. 4.8, ou seja, quanto maior o valor melhor o desempenho. A Tab. 5.2 mostra o grau de heterogeneidade do ambiente em questão e a Tab. 5.3 apresenta os resultados da execução da aplicação Jacobi utilizando tanto a política de escalonamento r*ound-robin* quanto a DPWP. A Tab. 5.3 disponibiliza o resultado da execução da aplicação Jacobi em segundos, sendo quanto menor o tempo melhor o desempenho da aplicação.

Benchmark Linpack			
Computador	Unidade – Tempo total em segundos (Média)	Variância	Desvio padrão
Colossus	1981,8780	0,2686	0,5183
Tempestade	1982,7290	0,2194	0,4684
Demolidor	1981,8260	0,1995	0,4467
Venon	1983,0470	0,2624	0,5122
Vampira	1981,8290	0,2487	0,4987
Gambit	1982,9940	0,1733	0,4164
Elektra	1981,8870	0,1889	0,4346
Anjo	1982,8050	0,3061	0,5532
Justiceiro	1981,9840	0,1939	0,4404
Noturno	1982,8820	0,3035	0,5509

Tabela 5. 1 – Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Linpack no ambiente homogêneo.

Benchmark Linpack	
Computador	Unidade – Tempo total em segundos (Média)
Colossus	1981,8780
Tempestade	1982,7290
Demolidor	1981,8260
Venon	1983,0470
Vampira	1981,8290
Gambit	1982,9940
Elektra	1981,8870
Anjo	1982,8050
Justiceiro	1981,9840
Noturno	1982,8820
GH	0,0002

Tabela 5. 2 - Potência computacional das máquinas utilizando *benchmark* Linpack e a métrica GH no ambiente homogêneo.

Benchmark Linpack – GH=0,0002			
	Tempo total em segundos (Média)	Variância	Desvio padrão
Aplicação Jacobi com Round-Robin	5,10	0,09	0,30
Aplicação Jacobi com DPWP	7,83	8,90	2,98

Tabela 5. 3 – Média aritmética, variância e desvio padrão das 30 execuções da aplicação Jacobi no ambiente homogêneo.

Nesse ambiente homogêneo, o valor do grau de heterogeneidade obtido com a execução do *benchmark* Linpack foi próximo a zero e a aplicação Jacobi executada com a política de escalonamento *round-robin* obteve melhor desempenho do que a aplicação Jacobi com a política DPWP. Os resultados mostram que a perda de desempenho observada com a execução via DPWP foi de 53,5% para a média das execuções realizadas. A DPWP gastou mais tempo para encontrar quais computadores deveriam receber mais processos (balanceamento da carga) enquanto que a política de escalonamento r*ound-robin* distribuiu a quantidade de processos igualmente entre os processadores realizando essa mesma operação

de escalonamento de uma maneira mais simples. Visto que a plataforma utilizada é homogênea e não houve interferência externa de outras aplicações concorrendo aos recursos disponíveis, o uso da DPWP mostra-se ineficiente. Neste caso a GH mostrou-se eficaz em impedir o uso desnecessário de toda a estrutura envolvida com a DPWP, visto que a política *round-robin* é eficiente para realizar uma boa distribuição dos processos.

5.7.2 Cenário Parcialmente Heterogêneo

As Tabs. 5.4 a 5.6 apresentam as potências computacionais de cada máquina (média aritmética, variância e desvio padrão das 30 execuções realizadas pelo *benchmark* Linpack), o grau de heterogeneidade (GH) obtido e o tempo de execução da aplicação Jacobi no cenário parcialmente heterogêneo (média aritmética, variância e desvio padrão das 30 execuções).

A Tab. 5.4 apresenta os valores obtidos com o *benchmark* Linpack utilizando a métrica tempo total em segundos no ambiente parcialmente heterogêneo. Os resultados estão na mesma proporção/escala dos valores apresentados na Tab. 5.1, ou seja, quanto maior o valor melhor o desempenho. A Tab. 5.5 mostra o grau de heterogeneidade do ambiente em questão e a Tab. 5.6 apresenta os resultados da execução da aplicação Jacobi utilizando tanto a política de escalonamento r*ound-robin* quanto a DPWP. A Tab. 5.6 disponibiliza o resultado da execução da aplicação Jacobi em segundos, onde quanto menor o tempo melhor o desempenho da aplicação.

Benchmark Linpack			
Computador	Unidade – Tempo total em segundos (Média)	Variância	Desvio padrão
Colossus	1981,87	0,26	0,51
Tempestade	1982,72	0,21	0,46
Demolidor	1461,55	0,47	0,68
Venon	1462,84	0,45	0,67
Vampira	931,52	0,78	0,88
Gambit	932,62	0,74	0,86
Elektra	1708,68	0,36	0,60
Anjo	1708,80	0,34	0,58
Justiceiro	1102,45	0,54	0,73
Noturno	1102,61	0,53	0,73

Tabela 5. 4 – Média aritmética, variância e desvio padrão das 30 execuções utilizando *benchmark* Linpack no ambiente parcialmente heterogêneo.

Benchmark Linpack	
Computador	Unidade – Tempo total em segundos (Média)
Colossus	1981,87
Tempestade	1982,72
Demolidor	1461,55
Venon	1462,84
Vampira	931,52
Gambit	932,62
Elektra	1708,68
Anjo	1708,80
Justiceiro	1102,45
Noturno	1102,61
GH	0,22

Tabela 5. 5 - Potência computacional das máquinas utilizando *benchmark* Linpack e a métrica GH no ambiente parcialmente heterogêneo.

Benchmark Linpack – GH=0,22			
	Média	Variância	Desvio padrão
Aplicação Jacobi com Round-Robin	17,63	0,79	0,88
Aplicação Jacobi com DPWP	11,56	30,46	5,51

Tabela 5. 6 – Média aritmética, variância e desvio padrão das 30 execuções da aplicação Jacobi no ambiente parcialmente heterogêneo.

Esperava-se que o desempenho da aplicação Jacobi no ambiente AMIGO/DPWP fosse melhor do que no ambiente AMIGO/*round-robin* o que acabou sendo confirmado através da Tab. 5.6. Neste caso, a perda de desempenho ao se utilizar a política r*ound-robin* foi de 52,5%. Isso ocorreu em função da heterogeneidade presente na plataforma, fato considerado apenas pela política DPWP. Novamente neste caso a métrica GH foi fundamental para a escolha da política de escalonamento apropriada.

5.7.3 Cenário Heterogêneo

As Tabs. 5.7 a 5.9 apresentam as potências computacionais de cada máquina (média aritmética, variância e desvio padrão das 30 execuções realizadas pelo *benchmark* Linpack), o grau de heterogeneidade (GH) obtido e o tempo de execução da aplicação Jacobi no cenário heterogêneo (média aritmética, variância e desvio padrão das 30 execuções).

A Tab. 5.7 apresenta os valores obtidos com o *benchmark* Linpack utilizando a métrica tempo total em segundos no ambiente heterogêneo, os resultados estão na mesma proporção/escala dos valores apresentados na Tab. 5.4, ou seja, quanto maior o valor melhor o desempenho. A Tab. 5.8 mostra o grau de heterogeneidade do ambiente em questão e a Tab. 5.9 apresenta os resultados da execução da aplicação Jacobi utilizando tanto a política de

escalonamento r*ound-robin* quanto a DPWP. A Tab. 5.9 disponibiliza o resultado da execução da aplicação Jacobi em segundos, quanto menor o tempo melhor o desempenho da aplicação.

Benchmark Linpack			
Computador	Unidade – Tempo total em segundos (Média)	Variância	Desvio padrão
Colossus	1981,87	0,26	0,51
Tempestade	1982,72	0,21	0,46
Demolidor	1461,55	0,47	0,68
Venon	1462,84	0,45	0,67
Vampira	931,52	0,78	0,88
Gambit	932,62	0,74	0,86
Elektra	509,02	1,04	1,02
Anjo	509,10	1,09	1,04
Justiceiro	202,28	1,45	1,20
Noturno	202,45	1,43	1,19

Tabela 5. 7 – Média aritmética, variância e desvio padrão das 30 execuções do *benchmark* Linpack no ambiente heterogêneo.

Benchmark Linpack	
Computador	Unidade – Tempo total em segundos (Média)
Colossus	1981,87
Tempestade	1982,72
Demolidor	1461,55
Venon	1462,84
Vampira	931,52
Gambit	932,62
Elektra	509,02
Anjo	509,10
Justiceiro	202,28
Noturno	202,45
GH	0,57

Tabela 5. 8 - Potência computacional das máquinas utilizando *benchmark* Linpack e a métrica GH no ambiente heterogêneo.

Benchmark Linpack – GH=0,57			
	Média	Variância	Desvio padrão
Aplicação Jacobi com Round-Robin	28,60	1,83	1,35
Aplicação Jacobi com DPWP	23,66	647,74	25,45

Tabela 5. 9 – Média aritmética, variância e desvio padrão das 30 execuções da aplicação Jacobi no ambiente heterogêneo.

Nesse cenário heterogêneo, como esperado, a política DPWP obteve um desempenho melhor em relação à *round-robin*. No entanto, a diferença entre as duas execuções foi menor, caracterizando uma perda de desempenho de 20,9% quando executando-se a *round-robin*. Acredita-se que a diminuição dessa diferença entre o desempenho das duas políticas deve-se ao fato dos computadores envolvidos (Elektra, Anjo, Justiceiro e Noturno) apresentarem cargas extras maiores do que no cenário anterior. Assim, a política DPWP demorou mais

tempo para descobrir quais máquinas e quantos processos cada uma dessas máquinas deveriam receber para realizar o balanceamento de carga no sistema.

5.8 CONSIDERAÇÕES FINAIS

Este capítulo detalhou o emprego da métrica GH no AMIGO, um ambiente de escalonamento real, voltado para a distribuição de processos paralelos em um *cluster* Beowulf com o PVM. Foram apresentados os detalhes de implementação pertinentes à inclusão da métrica GH no AMIGO e também da aplicação Jacobi, esta utilizada no estudo de caso realizado.

Os resultados obtidos demonstram que a métrica GH pode ser incorporada aos ambientes de escalonamento (ou diretamente nas políticas) de uma maneira simples e com ganhos de desempenho expressivos, neste caso de até 53,5%.

O estudo de caso realizado neste trabalho pode ser considerado simples e limitado às características do AMIGO, do PVM e suas políticas de escalonamento. No entanto, o mesmo indica que a GH consegue traduzir o grau de heterogeneidade de uma plataforma distribuída em uma única métrica capaz de aumentar a qualidade do escalonamento de processos.

Os resultados também confirmam, mais uma vez, que a métrica GH apresenta um comportamento estável ao se tratar os diferentes graus de heterogeneidade de plataformas distribuídas.

A elaboração deste estudo de caso também contribuiu para o desenvolvimento do AMIGO, permitindo que o mesmo conte a partir de agora com a informação sobre o grau de heterogeneidade da plataforma e também disponha de um ambiente de escalonamento adaptativo. Este recurso não estava disponível e pode fornecer ganhos expressivos de desempenho às aplicações também com outras políticas, diferentes daquelas utilizadas neste trabalho.

6 CONCLUSÕES

Neste capítulo são apresentadas as conclusões sobre o trabalho realizado, destacadas as contribuições desta dissertação e sugeridos alguns trabalhos futuros.

6.1 PONDERAÇÕES FINAIS DA DISSERTAÇÃO

Este trabalho apresentou um estudo elaborado sobre métricas para a obtenção do grau de heterogeneidade de um sistema computacional, tais como H_1, H_2, H_3, H_4 e GH, mostrando as diferenças e restrições. Dentre elas, destaca-se a métrica GH que tem como objetivo averiguar a dispersão das máquinas em torno de uma máquina virtual.

Além disso, foi realizada uma análise do comportamento dessas métricas e, conseqüentemente, examinados os efeitos causados pela escolha adequada de uma métrica para a obtenção do grau de heterogeneidade de um sistema computacional distribuído.

Para investigar a utilidade do uso da métrica GH, resultados empíricos foram apresentados em forma de estudos de caso.

A necessidade de um estudo como este foi motivada pela flexibilidade apresentada pela métrica GH, a qual pode assumir diversas métricas como entrada, estas relacionadas com processador, memória, rede ou outros recursos computacionais necessários à aplicação.

A pesquisa dos parâmetros de entrada foi guiada por dois requisitos: primeiro, a obtenção dos parâmetros no sistema operacional Linux em nível de usuário através do pseudo-sistema de arquivos */proc* e segundo, a obtenção dos parâmetros através de aplicações reais conhecidas como *benchmarks*.

O primeiro requisito apresentou como parâmetros: freqüência de velocidade da CPU, tamanho da *cache*, tamanho da memória RAM total disponível, tamanho total da área de *swap* e placa de rede. Os resultados obtidos com esse requisito exemplificaram a dificuldade de se determinar o grau de heterogeneidade de uma plataforma considerando apenas a configuração dos mesmos. Outro fator a ser reforçado é a necessidade de se verificar o quanto essas diferenças na configuração poderiam, de fato, ajudar a estimar as diferenças de desempenho esperadas quando as aplicações reais fossem executadas.

Em vista da necessidade de estimar a diferença efetiva de desempenho entre os computadores, o segundo requisito foi essencial para obter métricas de desempenho mais voltadas às aplicações finais dos usuários. Algumas dessas métricas foram: MIPS, MFLOPS, tempo de execução em segundos e *throughput*. Dentre essas métricas, as relacionadas com o

tempo são vistas por alguns pesquisadores da área como mais eficientes para representar o desempenho de um sistema computacional como um todo. A justificativa para tanto é que tais métricas tendem a encapsular as diferenças arquiteturais de *hardware* e *software* básico quando o objetivo é analisar o desempenho final de aplicações de usuários.

Os *benchmarks* encontrados na literatura e utilizados para analisar o desempenho da métrica GH através de diferentes perspectivas foram: processador (Whetstone, Dhrystone e Linpack), memória (Cachebench e Stream) e rede (Netperf). Os resultados obtidos com os mesmos, em um experimento realizado com 5 computadores instalados no Laboratório de Sistemas Distribuídos e Programação Concorrente – LaSDPC do ICMC/USP, mostraram que as mesmas máquinas podem apresentar grandes variações de desempenho e, conseqüentemente, no grau de heterogeneidade da plataforma distribuída que tais máquinas formaram.

Do ponto de vista de processador, memória e rede, as métricas de desempenho fornecidas pelos *benchmarks* Whetstone, Dhrystone, Linpack, Cachebench, Stream e Netperf, apresentaram valores equivalentes entre si e, conseqüentemente, a métrica GH obteve comportamento satisfatório frente às diferentes perspectivas de heterogeneidade.

Em resumo, a métrica GH de modo geral apresentou um comportamento satisfatório indo ao encontro do resultado do *benchmark* utilizado. A escolha correta do *benchmark* foi essencial para que o resultado final da métrica GH refletisse adequadamente o grau de heterogeneidade da plataforma, frente à demanda gerada pela aplicação, no caso os *benchmarks* executados.

Para verificar o impacto da métrica GH no escalonamento de processos, e conseqüentemente no desempenho final de uma aplicação paralela/distribuída, a métrica GH foi inserida no AMIGO, um ambiente de escalonamento real. Essa inserção permitiu a criação de um algoritmo de escalonamento adaptativo, onde foi possível adaptar o algoritmo de escalonamento à política *round-robin* ou à política DPWP, conforme o grau de heterogeneidade da plataforma computacional distribuída. Para as execuções foram definidos 3 cenários: homogêneo, parcialmente heterogêneo e heterogêneo. Cada cenário utilizou 2 políticas de escalonamento, *round-robin* e DPWP. No total, foram realizados 6 experimentos utilizando um *cluster*, constituído de 10 máquinas, o AMIGO, o PVM e a aplicação paralela para o método iterativo para a solução de sistemas lineares de Jacobi-Richardson (ou Gauss-Jacobi).

No cenário homogêneo, como era esperado, o escalonamento da aplicação Jacobi utilizando a política de escalonamento *round-robin* obteve melhor desempenho do que a DPWP. Isso ocorreu em virtude da sobrecarga de comunicação exercida pela política DPWP em um cenário homogêneo e sem demandas externas à aplicação. A política de escalonamento *round-robin* mostrou-se mais eficaz em virtude de apresentar uma sobrecarga de execução menor para realizar o balanceamento da carga no sistema.

Nos cenários parcialmente heterogêneo e heterogêneo, a política de escalonamento DPWP obteve um resultado mais eficiente do que a *round-robin* devido à natureza da política DPWP e levando-se em consideração a heterogeneidade do sistema.

Portanto, os resultados mostraram que a métrica GH pode ser empregada com sucesso em algoritmos de escalonamento reais, desde que adequada às características da plataforma, da aplicação e do próprio algoritmo de escalonamento. Os estudos de caso apresentaram um comportamento esperado e satisfatório no escalonamento de processos levando-se em consideração, a métrica tempo total em segundos, disponibilizada pelo *benchmark* Linpack frente à demanda da aplicação Jacobi.

6.2 CONTRIBUIÇÕES

As principais contribuições dos estudos apresentados nesta dissertação são:

- Realização de uma análise do comportamento da métrica GH frente às diferentes perspectivas reais de heterogeneidade. A eficiência dessa métrica foi comprovada pela utilização de *benchmarks*, fato que tornou a análise mais próxima à realidade. O emprego da GH com diferentes parâmetros de entrada permitiu ponderar com maior clareza a abrangência e confiabilidade da mesma;

- Implementação da métrica GH para obtenção do grau de heterogeneidade do sistema e a sua utilização no escalonamento de processos. A implementação realizada permitiu a realização de um escalonamento adaptativo e dinâmico frente à demanda da aplicação e da plataforma/ambiente utilizada;

- Verificação do impacto da métrica GH no escalonamento de processos frente à demanda de uma aplicação *cpu-bound* e do grau de heterogeneidade do ponto de vista de processador. Essa análise permitiu comprovar o comportamento da métrica GH utilizando 3 cenários distintos de graus de heterogeneidade;

- Levantamento e utilização dos principais *benchmarks* com código aberto e livre disponíveis na literatura. O uso dos *benchmarks* descritos nesse trabalho poderão ser utilizados em outros trabalhos futuros que vierem a ser desenvolvidos pelo grupo de pesquisa;

- Continuidade do desenvolvimento do ambiente de escalonamento AMIGO, desenvolvido pelo grupo de pesquisa. A partir deste trabalho o AMIGO pode contar com uma política de escalonamento adaptativa, fato antes não disponível.

6.3 SUGESTÕES PARA TRABALHOS FUTUROS

As sugestões para trabalhos futuros baseados nesta dissertação são:

- Realizar estudos específicos para encontrar os limites (*thresholds*) para a métrica GH frente às diferentes perspectivas de heterogeneidade (processador, memória e rede). Isso requer uma análise da execução de algumas aplicações reais, a fim de obter o valor ótimo para cada ponto de vista.

- Criar políticas de escalonamento direcionadas às aplicações de memória e rede e que englobam a heterogeneidade do sistema, aplicando a métrica GH para verificar o impacto desse escalonamento em ambientes reais ou simulados.

- Combinar as métricas de desempenho oferecidas pelos *benchmarks* de processador com as métricas de outros *benchmarks* e utilizá-las como parâmetro de entrada na métrica GH e verificar o efeito no escalonamento de outros tipos de aplicações que não são *cpu-bound*. A maioria das aplicações depende de quantidades consideráveis de memória para executar, mesmo não sendo consideradas *memory-bound*, portanto, essas métricas combinadas podem vir a ajudar no escalonamento dessas aplicações.

REFERÊNCIAS

Abdelzaher, T. F.; Shin, K. G. (2000). "**Period-Based Load Partitioning and Assignment for Large Real-Time Applications"**. IEEE Transactions on Computers, vol. 49, nº 1, p. 81-87, January.

Al-Jaroodi, J.; Mohamed, N.; Hong Jiang; Swanson, D. (2003). "**Modeling parallel applications performance on heterogeneous systems**". Proceedings of International Parallel and Distributed Processing Symposium, 2003, April.

Ambrosius, S. L.; Freund, R. F.; Scott, S. L.; Siegel, H. J. (1996). "**Work-Based Performance Measurement and Analysis of Virtual Heterogeneous Machines"**. In the 5th Heterogeneous Computing Workshop (HCW'96). P.669-685, April.

Amir, Y. et al. (2000). "**An Opportunity Cost Approach for Job Assignment in a Scalable Computing Cluster"**. IEEE Transactions on Parallel and Distributed Systems, v. 11, m. 7, p. 760-768, July.

Araújo, A.P.F. DPWP (1999). "**DPWP - Uma Nova Abordagem para o Escalonamento Dinâmico em Computação Paralela Virtual**". Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (ICMC/USP), maio, 1999.

Araújo, A.P.F.; Santana, M.J.; Santana, R.H.C.; Souza, P.S.L. (1999a)."**A New Dynamical Scheduling Algorithm**". In: International Conference on Parallel and Distributed Processing Techniques and Applications – PDPTA'99, Lãs Vegas, Nevada, U.S.A., junho, 1999.

Araújo, A.P.F.; Santana, M.J.; Santana, R.H.C.; Souza, P.S.L. (1999b). "**DPWP – A New Load Balancing Algorithm**". In: 5th International Conference on Information Systems Analysis and Synthesis – ISAS'99, Orlando, U.S.A., julho, 1999.

Bahi, J.M.; Contassot-Vivier, S.; Couturier, R. (2003). "**Coupling dynamic load balancing with asynchronism in iterative algorithms on the computational grid**". Proceedings of International Parallel and Distributed Processing Symposium, 2003, April.

Bai, G., Oladosu, K., Williamson, C. (2007). "**Performance benchmarking of wireless Web servers**". Ad Hoc Networks, v. 5, p. 392-412, April.

Bajaj R.; Agrawal, D.P. (2004). "**Improving scheduling of tasks in a heterogeneous environment**". IEEE Transactions on Parallel and Distributed Systems, v. 15, nº 2, p. 107-118, February.

Barker, K.; Chrisochoides, N. (2005). "**Practical Performance Model for Optimizing Dynamic Load Balancing of Adaptive Applications**". Proceedings of 19th IEEE Parallel and Distributed Processing Symposium, 2005, April.

Barton, C., Casçaval, C., Almási, G., Zheng, Y., Farreras, M., Chatterje, S., Amaral, J. (2006). "**Shared memory programming for large scale machines**". Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation, p.108-117.

Beaumont, O.; Legrand, A.; Robert, T. (2003).**"Optional algorithms for scheduling divible workloads on heterogeneous systems"**, In HCW'2003, the 12th Heterogeneous Computing Workshop, IEEE Computing Society Press.

Beitz, A.; Kent, S.; Roe, P. (2000). "**Optimizing Heterogeneous Task Migration in the Gardens Virtual Cluster Computer"**. 9th Heterogeneous Computing Wokshop, pp. 140-146, Cancun – México, May.

Benso, A., Di Carlo, S., Di Natale, G., Prinetto, P., Taghaferri, L. (2003). "**Data criticality estimation in software applications**". Proceedings of the International Test Conference - ITC 2003, p. 802-810, September.

Bhatia, S., Consel, C., Lawall, J. (2006). "**Memory-manager/scheduler co-design: optimizing event-driven servers to improve cache behavior**". Proceedings of the 2006 international symposium on Memory management, p. 104-114.

Binkert, N.L., Hsu, L.R., Saidi, A.G., Dreslinski, R.G., Schultz, A.L., Reinhardt, S.K. (2005). "**Performance analysis of system overheads in TCP/IP workloads**". 14th International Conference on Parallel Architectures and Compilation Techniques, 2005 - PACT 2005, p. 218-228, September.

Binkert, N. L., Saidi, A. G., Reinhardt, S. K. (2006). "**Integrated network interfaces for high-bandwidth TCP/IP**". ACM SIGOPS Operating Systems Review, v. 40, p. 315-324, December.

Branco, K. R. L. J. C. (2005). "**Índices de carga e desempenho em ambientes paralelos/distribuídos – modelagem e métricas"**. 2005 Tese (Doutorado), USP/ICMC, São Carlos, SP.

Branco, K. R. L. J. C.; Santana, M. J.; Santana, R. H. C. (2003a). "**A Novel Metric for Evaluation of Computer System Heterogeneity"**. Proceedings of The International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'2003). P.437-411 v. 1, Las Vegas – Nevada – USA, June.

Branco, K. R. L. J. C., Santana, M.J., Santana, R. H. C. (2003b). "**A Novel Performance Metric for Evaluation of Computer System Heterogeneity**". Proceedings of The International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS'2003). p. 292 – 301, V. 34, n. 04 SCS. Montreal – Canadá, July.

Branco, K. R. L. J. C.; Santana, M. J.; Santana, R. H. C. (2003c). "**A Novel Metric for Checking Levels of Heterogeneity in Distributed Computer Systems"**. Proceedings of The Fourth Congress of Logic Applied to Technology (LAPTEC'2003). p. 148-155, v. 101, IOS Press. Marília – São Paulo – Brazil, Novembro.

Boyer, F. W.; Hura, S. G. (2005). "**Non-evolutionary algorithm for scheduling dependent tasks in distributed heterogeneous computing environments**". Journal of Parallel and Distributed Computing, v. 65, nº 9, p. 1035-1046, September.

Braun, T. D.; Siegel, H. J.; Beck, N.; Boloni, L.; Maheswaran, M.; Reuther, A. I.; Rpbertson, J. P.; Thies, M. D.; Yao, B. (1998). "**A Taxonomy for Describing Macthing and Scheduling Heuristics for Mixed-Machine Heterogeneous Computing Systems"**. IEEE Workshop on Advances in Parallel and Distributed Systems. P.330-335, October.

Braun, T. D.; Siegel, H. J.; Beck, N.; Boloni, L.; Maheswaran, M.; Reuther, A. I.; Robertson, J. P.; Thies, M. D.; Yao, B.; Hensgen, D.; Freund, R. F. A. (1999). "**Comparison Study of Static Mapping Heuristics for a Class of Meta-Tasks on Heterogeneous Computing Systems"**. In The Proceedings of the 8th Heterogeneous Computing Workshop (HCW'99). San Juan, Puerto Rico, April.

Cameron, K.W., Ge, H., Feng, X. (2005). "**High-performance, power-aware distributed computing for scientific applications**". IEEE Computer, v.38, p. 40-47, November.

Cantonnet, F., El-Ghazawi, T.A., Lorenz, P., Gaber, J., (2005). "**Fast Address Translation Techniques for Distributed Shared Memory Compilers**". Proceedings of the 19th IEEE International on Parallel and Distributed Processing Symposium, 2005, p. 52b, April.

Cantonnet, F., Yao, Y., Annareddy, S., Mohamed, A.S., El-Ghazawi, T.A. (2003). "**Performance monitoring and evaluation of a UPC implementation on a NUMA architecture**". Proceedings of the International on Parallel and Distributed Processing Symposium, 2003, April.

Casavant, T. L., Kuhl, J.G. (1998). "**A Taxonomy of Scheduling in General-Purpose Distributed Computing Systems**". IEEE Transactions on Software Engineering,

p.141-154, fevereiro, 1988.

Chang, H., Li, K., Lin, Y., Yang, C., Wang, H., Lee, L. (2005). "**Performance issues of grid computing based on different architecture cluster computing platforms**". 19th International Conference on Advanced Information Networking and Applications, 2005 - AINA 2005, v. 2, p. 321-324, March.

Chen, H., Decker, J., Bierbaum, N. (2006). "**Future networking for scalable I/O**". Proceedings of the 24th IASTED international conference on Parallel and distributed computing and networks, p. 128-135.

Chen, S.; Eshaghian, M. M.; Khokhar, S.; Shaaban, M. E. (1993). "**A Selection Theory and methodology for Heterogeneous Supercomputing"**. 2nd Wokshop on Heterogeneous Processing (WHP'93), p. 15-22, April.

Clark, D. W., Levy, H. M. (1982). "**Measurement and analysis of instruction use in the VAX-11/780**", Proceedings of the 9th annual symposium on Computer Architecture, 1982.

Constantinescu, C. (2005). "**Dependability benchmarking using environmental test tools**". Annual Proceedings of the Reliability and Maintainability Symposium, 2005, January.

Conte, T.M., Hwu, W.-M.W. (1991a). "**Benchmark characterization**". IEEE Computer Society, v.24, p.48-56, January.

Conte, T.M., Hwu, W.-M.W. (1991b). "**A brief survey of benchmark usage in the architecture community**". ACM SIGARCH Computer Architecture News, v.19, p.37-44, June.

Dantas, M. A. R.; Zaluska, E. J. (1998). "**Efficient Scheduling of MPI Application on Network of Workstations"**. Future Generation Computer Systems – FGCS, v. 13, p. 489-499.

Ding, C., Kennedy, K. (2000). "**The memory of bandwidth bottleneck and its amelioration by acompiler**". Proceedings of the 14th International on Parallel and Distributed Processing Symposium, 2000 - IPDPS 2000, p. 181-189.

Dongarra, J. (2006). "**Trends in high performance computing: a historical overview and examination of future developments**". IEEE Circuits and Devices Magazine, v.22, p. 22-27.

Duncan, R. (1990). "**A Survey of Parallel Computer Architectures"**. IEEE Computer, p. 5-16.

Ekmecic, I.; Tartalja, I.; milutinovic, V. (1996). "**A Survey of Heterogeneous Computing: Concepts and Systems"**. Proceeedings of IEEE, n.84, p.1127-1144.

Ferrari, D.; Zhou, S. (1987). "**An Empirical Investigation of Load Indices for Load Balancing Applications"**. Im Proceedings of Performance'87, the 12th Int'l Symposium on Computer Performance Modeling, Measurement, and Evaluation, p.515-528.

Flynn, M. J. (1972). "**Some Computer Organizations and Their Effectiveness".** IEEE Transactions on Computers, v. c-21, n. 9, pp. 948-960, September.

Flynn, M. J.; Rudd, K. W. (1996). "**Parallel Architectures"**. ACM Computing Surveys. V. 28, nº 1, p.67-70, March.

Freund, R.; Conwell, D. (1990). "**Superconcurrency: A Form of Distributed Heterogeneous Supercomputing**". Supercomputing Review, p. 47-50, October.

Funk, A., Basili, V., Hochstein, L., Kepner, J. (2005). "**Application of a development time productivity metric to parallel software development**". Proceedings of the second international workshop on Software engineering for high performance computing system applications, p. 8-12.

Garg, R., Sabharwal, Y. (2006). "**Optimizing the HPCC randomaccess benchmark on blue Gene/L Supercomputer**". Proceedings of the joint international conference on Measurement and modeling of computer systems, p.369-370.

Gotsis, K.A., Goudos, S.K., Sahalos, J.N. (2005). "**A test lab for the performance analysis of TCP over ethernet LAN on windows operating system**". IEEE Transactions on Education, v.48, p. 318-328, May.

Grosu, D. (1996). "**Some Performance Metrics for Heterogeneous Distributed Systems"**. Proceedings of PDPTA'96. Las Vegas, June.

Grosu, D.; Chronopoulos, A.T.; Leung, M. (2002). "**Load balancing in distributed systems: an approach using cooperative games**". Proceedings of International Parallel and Distributed Processing Symposium – IPDPS, 2002, April.

Guermouche, A.; L'Excellent, J.-Y. (2005). "**A Study of Various Load Information Exchange Mechanisms for a Distributed Application using Dynamic Scheduling**". Proceedings of 19th IEEE Parallel and Distributed Processing Symposium, 2005, April.

Hagras T.; Janecek J. (2005). "**A high performance, low complexity algorithm for compile-time task scheduling in heterogeneous systems**". Parallel Computing, v. 31, nº 7, p. 653-670, July.

Harbaugh, S., Forakis, J. (1984). "**Timing studies using a synthetic Whetstone benchmark**", ACM SIGAda Ada Letters, v.4, p. 23-34, September-October.

Head, M. R., Govindaraju, M., Engelen, R., Zhang, W. (2006). "**Grid scheduling and protocols---Benchmarking XML processors for applications in grid web services**". Proceedings of the 2006 ACM/IEEE conference on Supercomputing.

Huang, B., Bauer, M., Katchabaw, M. (2005). "**Hpcbench - a Linux-based network benchmark for high performance networks**". 19th International Symposium on High Performance Computing Systems and Applications, 2005 - HPCS 2005, p. 65-71, May.

Hui, G., Jingli, Z., Yangkai, O., Shengsheng, Y. (2004). "**A design and evaluation of Ethernet links bundling systems**". 18th International Conference on Advanced Information Networking and Applications, 2004 - AINA 2004, v. 2, p. 313-316, March.

Hur, I., Lin, C. (2006). "**Adaptive History-Based Memory Schedulers for Modern Processors**". IEEE Computer Society, v. 26, p. 22-29, January-February.

Junwei, C.; Spooner, D.P.; Jarvis, S.A.; Saini, S.; Nudd, G.R. (2003)."**Agent-based grid load balancing using performance-driven task scheduling**". Proceedings of International Parallel and Distributed Processing Symposium, 2003, April.

Kamthe, A.; Lee S. (2005). "**A Stochastic Approach to Estimating Earliest Start Times of Nodes for Scheduling DAGs on Heterogeneous Distributed Computing Systems**". In HCW'2005, the 19th Heterogeneous Computing Workshop, IEEE Computing Society Press.

Kenny, E., Coghlan, B., Tsouloupas, G., Dikaiakos, M., Walsh, J., Childs, S., Callaghan, D. O., Quigley, G. (2005)."**Heterogeneous Grid Computing: Issues and Early Benchmarks**". Computational Science – ICCS 2005, v. 3516, p. 870-874, May.

Kerbyson, D. J., Lang, M., Patino, G., Amidi, H. (2004). "**An empirical performance analysis of commodity memories in commodity servers**". Proceedings of the 2004 workshop on Memory system performance, p. 42-50.

Khokhar, A. A.; Prasana, V. K.; Shaaban, M. E.; Wang, C. L. (1993). "**Heterogeneous Computing: Challenges and Opportunities"**. IEEE Computer, 26(6): 18-27, June.

Kishimoto, Y., Ichikawa, S. (2005). "**Optimizing the configuration of a heterogeneous cluster with multiprocessing and execution-time estimation**". Parallel Computing, v. 31, p. 691-710, July.

Kri, F., Feeley, M. (2004). "**Genetic instruction scheduling and register allocation**". 24th International Conference of the Chilean on Computer Science Society, SCCC - 2004, p. 76-83, November.

Kunz, T. (1991). "**The Influence of Different Workload Description on a Heuristic Load Balancing Scheme"**. IEEE Transaction on Software Engineering, v.17, n. 7, p. 725-730, July.

Lee, S., Huang J. (2002). "**A Theoretical Approach to Load Balancing of a Target Task in a Temporally and Spatially Heterogeneous Grid Computing Environment**". GRID, p. 70-81.

Liu, G. Q.; Poh, K. L.; Xie, M. (2005). "**Iterative list scheduling for heterogeneous computing**". Journal of Parallel and Distributed Computing, v.65, nº 5, p. 654-665, May.

Llcbench (2007). – "**http://icl.cs.utk.edu/projects/llcbench"**, acessado em 24/11/2007.

Mckee, S. A. (2004). "**Reflections on the memory wall**". Proceedings of the 1st conference on Computing frontiers, p.162.

Meeker, R. D. (2005). "**Comparative system performance for a Beowulf cluster**". Journal of Computing Sciences in Colleges, v. 21, p. 114-119, October.

Mehra, P.; Wah, B. W. (1993). "**Automated Learning of Load-Balancing Strategies for a Distributed Computer System"**. University of Illinois at Urbana-Champaign.

Minohara, T., Ishikawa, S., Amano, M. (2005). "**Centralized surveillance of unused address space by using virtual networks**". Proceedings of 11th Pacific Rim International Symposium on Dependable Computing, 2005, December.

Netlib (2007). – "**http://www.netlib.org/benchmark/"**, acessado em 24/11/2007.

Netperf (2007). – "**http://www.netperf.org/netperf/NetperfPage.html"**, acessado em 24/11/2007.

Parks, T. M., (2005). "**A comparison of MPI and process networks**". 19th IEEE International Proceedings of the Parallel and Distributed Processing Symposium, 2005, April.

Patterson, D. A., Hennessy, J. L. (2005). "***Computer organization and design: the hardware/software interface"***. Elsevier/Morgan Kaufmann. 3ª Edição.

Pheatt, C. (2007). "**An easy to use distributed computing framework**". ACM SIGCSE Bulletin, v. 39, p. 571-575, March.

Pisharath, J., Jiang, N., Choudhary, A. (2003). "**ERE: a framework for performance-energy tradeoffs in heterogeneous systems**". Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology - ISSPIT 2003, p. 778-781, December.

Pope, S., Riddoch, D. (2007). "**10Gb/s Ethernet performance and retrospective**". ACM SIGCOMM Computer Communication Review, v. 7, p. 89-92, April.

Potter, J. (1993). "**Heterogeneous Associative Computing. Proc Workshop on Heterogeneous Processing"**. IEEE CS Press, Los Alamitos, Calif., Order Nº 3532-02.

Radulescu, A.; van Gemund, A.J.C. (2000). "**Fast and effective task scheduling in heterogeneous systems**", In HCW'2000, the 9th Heterogeneous Computing Workshop, IEEE Computing Society Press.

Robinson, A., Garside, J. D. (2007). "**Sensitive registers: a technique for reducing the fetch bandwidth in low-power microprocessors**". Proceedings of the 17th great lakes symposium on Great lakes symposium on VLSI, p. 138-143.

Rodrigues, A., Murphy, R., Kogge, P., Underwood, K. (2004). "**Characterizing a new class of threads in scientific applications for high end supercomputers**". Proceedings of the 18th annual international conference on Supercomputing, p. 164-174.

Ruggiero, M. A. G., Lopes, V. L. R. (1998). "**Cálculo Numérico: Aspectos Teóricos e Computacionais"**. McGraw-Hill, 1988.

Rui, H., Zhang, L., Hu, W. (2007). "**Accelerating sequential programs on Chip Multiprocessors via Dynamic Prefetching Thread**". Microprocessors & Microsystems, v. 31, p. 200-211, May.

Rus, P., Stok, B., Mole, N. (2003). "**Parallel computing with load balancing on heterogeneous distributed systems**". ACM Advances in Engineering Software, v.34, p.185-201, February.

Saavedra, R. H., Smith, A. J. (1996). "**Analysis of benchmark characteristics and benchmark performance prediction**". ACM Transactions on Computer Systems, v.14, p.344-384, November.

Salminen, E., Kangas, T., Riihimaki, J., Hamalainen, T.D. (2005). "**Requirements for network-on-chip benchmarking**". 23rd Conference on NORCHIP, 2005, p. 82-85, November.

Schnor, B.; Petri, S.; Langendörfer, H. (1996). "**Load Management for Load Balancing on Heterogeneous Plataforms: A Comparison of Traditional and Neural Network Based Approaches"**. In Second International Euro-Par Conference – Euro-Par'96, Lecture Notes in Computer Science v. 1124, Lyon, France, p.611-614, August.

Shivam, P., Babu, S., Chase, J. (2006). "**Active and accelerated learning of cost models for optimizing scientific applications**". Proceedings of the 32nd international conference on Very large data bases, v.32, p.535-546.

Souza, P.S.L. (1996). "**Máquina Paralela Virtual em Ambiente Windows**". Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (ICMC/USP), maio, 1996.

Souza, P.S.L.; Santana, M.J.; Santana, R.H.C. (1999a). "**A New Scheduling Environment for Near-Optimal Performance**". In: International Conference on Parallel and Distributed Processing Techniques and Applications – PDPTA'99, Las Vegas, Nevada, U.S.A., junho, 1999.

Souza, P.S.L.; Santana, M.J.; Santana, R.H.C. (1999b). "**AMIGO – A Dynamical Flexible Scheduling Environment**". In: 5th International Conference on Information Systems Analysis and Synthesis – ISAS'99, Orlando, U.S.A., julho, 1999.

Souza, P.S.L.; Santana, M.J.; Santana, R.H.C.; Araújo, A.P.F. (1999c). "**PVM and a Viable and Flexible Scheduling**". In: Eleventh IASTED International Conference on Parallel and Distributed Computing and Systems - PDCS'99, Cambridge, MA, USA, novembro, 1999.

Souza, P.S.L.; Santana, M.J.; Santana, R.H.C. (2000). "**Escalonamento de Processos: uma contribuição para a Convergência da Área**". Notas do ICMC, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, maio, 2000.

Souza, P.S.L.; Santana, M.J.; Santana, R.H.C. (2001). "**AMIGO: Uma Contribuição para a Convergência na Área de Escalonamento de Processos**". Dissertação de Doutorado, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo (ICMC/USP), 2001.

Stream (2007). – "**http://www.cs.virginia.edu/stream"**, acessado em 24/11/2007.

Tanenbaum, A. S. (1995). "**Distributed Operation Systems"**. Prentice Hall. 2ª Edição.

Tanenbaum, A. S. (2002). "**Distributed Systems"**. Prentice Hall. 1ª Edição.

Teller, J., Silio, C.B. Jr., Jacob, B. (2005). "**Performance characteristics of MAUI: an intelligent memory system architecture**". Proceedings of the 2005 workshop on Memory system performance, p. 44-53.

Teresco, J.D., Fair, J., Flaherty, J.E. (2005). "**Resource-aware scientific computation on a heterogeneous cluster**". IEEE Computing in Science & Engineering, v.7, p.40-50, March-April.

Thid, R., Sander, I., Jantsch, A. (2006). "**Flexible Bus and NoC Performance Analysis with Configurable Synthetic Workloads**". 9th EUROMICRO Conference on Digital System Design (DSD'06), p. 681-688.

Thiruvathukal, G. K. (2007)."**Project Hosting: Expanding the Scientific Programmer's Toolbox**". IEEE Computing in Science & Engineering, v.9, p. 70-75, March-April.

Underwood, K.D., Brightwell, R. (2004). "**The impact of MPI queue usage on message latency**". International Conference on Parallel Processing, 2004 - ICPP 2004, v. 1, p. 152-160, August.

Vandierendonck, H., De Bosschere, K. (2004). "**Eccentric and fragile benchmarks**". IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS - 2004, p. 2-11.

Vetter, J.S., Yoo, A. (2002). "**An Empirical Performance Evaluation of Scalable Scientific Applications**". ACM/IEEE of Conference on Supercomputing, 2002, p. 16, November.

Xiao, L.; Zhang X.; Qu Y. (2000). "**Effective Load Sharing on Heterogeneous Network of Workstations**". Proceedings of IPDPS 2000, Mexico, May.

Xu, C.; Lau, F. C. M. (1997). **"Load Balancing in Parallel Computers: Theory and Practice"**. Kluwer Academic Publishers, Boston, USA, 1997.

Zhang, X.; Yan, Y. (1995). "**Modeling and Characterizing Parallel Computing Performance on Heterogeneous Networks of Workstations"**. Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Proceeding, p. 25-34, October.

Zhang, Z., Seidel, S. (2005). "**Benchmark measurements of current UPC platforms**". Proceedings of the 19th IEEE International on Parallel and Distributed Processing Symposium, 2005, April.

Weicker, R. P. (1984). "**Dhrystone: a synthetic systems programming benchmark**". ACM Computing Surveys, v.27, p.1013-1030, October.

Weiderman, N., Donohoe, P., Shapiro, R. (1990). "**Benchmarking for deadline-driven computing**". Proceedings of the conference on TRI-ADA '90, p. 254-264.

Whetstone (2007). – "**http://homepage.virgin.net/roy.longbottom/whetstone.pdf"**, acessado em 24/11/2007.

Woorsluys, W. "**Avaliação do Impacto de Índice de Carga Específico para Aplicações I/O-Bound no Escalonamento de Processos em Plataformas Distribuídas e Heterogêneas"**. 2005 Mini-Dissertação (Qualificação), USP/ICMC, São Carlos, SP.

APÊNDICE A

Este apêndice contém o código fonte da aplicação Jacobi utilizada nos experimentos feitos com o AMIGO e com a métrica GH. O código abaixo está escrito na Linguagem C e utiliza o PVM sobre o Linux para ser executado.

```
1   // para compilar: make all
2   // para rodar com 3 escravos: jacobi_pvm_n 6 3
3   // onde 6 e a ordem da matriz e 3 o nr de escravos
4   // para executar corretamente lembre-se que devera haver 1 Mestre e mais (nprocesses-1) Escravos.
5   // os dados da matriz sao gerados automaticamente durante a execucao
6
7   #include <stdlib.h>
8   #include <stdio.h>
9   #include <math.h>
10  #include <time.h>
11  #include <pvm3.h>
12  #define JACOBI "/home/hitoshi/jacobi/jacobi_pvm_n"
13  #define PRECISAO 0.000000001 // criterio de parada do algoritmo.
14                                  // determina o fim do processo iterativo
15  int nprocesses; // nr total de processos em execucao (contara tb o mestre)
16  int myrank; // nr do rank deste processo
17  int src, dest, msgtag; // origem, detino e tag da mensagem
18  int tid_src, tid_dest; // tid de origem e destino das mensagens
19  int * tid_tarefas, result;
20  int count, count_last; // qtde de linhas de A em cada faixa (para cd escravo)
21                          // se a divisao nao for exata da faixa, count_last determina a qtde da
22                          // ultima faixa, ou seja, do ultimo escravo
23  int stride=1; // usado para compactar o dado para o pvm_send
24  int converge; // indica se o sistema converge ou nao
25  int ordem; // usado para alocar as matrizes/vetores e parar os loops
26  int pai; // nr do processo pai. Usado para determinar mestre e escravos na main()
27  int meutid; // tid deste processo (mestre ou escravo)
28
29  //*****************************************************************************
30  void mestre(void){
31          double *A, *B;
32          double maxvetXnovo, maxdif, maxaux, mr;
33          double somalinha;
34          double *vetX;
35          int i, j, iteracao;
36          int sai;
37          int count_aux;
38          unsigned int seed;
39          struct timeval t1;
40          msgtag=0; // rotulo da primeira mensagem
41
42          pvm_initsend(PvmDataRaw); // create buffer
43          pvm_pkint(&ordem, 1, stride); // pack ordem da matriz
44          pvm_pkint(&nprocesses, 1, stride); // pack nprocesses
45          pvm_pkint(tid_tarefas, nprocesses, stride);
46          //*****************************************************************
47          // envia para os escravos (nprocesses-1) a ordem das matrizes
48          for(dest =1; dest < nprocesses; dest++){
49                  pvm_send(tid_tarefas[dest], msgtag); // pvm_send para todos os escravos
50          }
51          if(ordem<(nprocesses-1)) // deverao haver pelo menos uma linha de A para um escravo
52          {
53                  nprocesses=(ordem+1); // ordem escravos e mais 1 mestre
54          }
55          // determina qtde de valores a enviar para os escravos (com excecao do ultimo que recebera a dif)
56          // floor arredonda para baixo sempre
57          count=(int)floor(((double)ordem)/((double)(nprocesses-1)));
58          count_last=ordem-((nprocesses-2)*count);
59          // ****************************************
60          // aloca memoria para matrizes e vetores
61          A=(double *)malloc((ordem*ordem)*sizeof(double));
62          B=(double *)malloc(ordem*sizeof(double));
63          vetX=(double *)malloc(ordem*sizeof(double));
64          // ****************************************
```

```
65              // ************************************
66              // gera dados da matriz A e do vetor B
67              // valores de A[ ][ ] e B[ ] gerados de maneira pseudo-aleatoria. Garante convergencia.
68              // gera nova semente
69              (void)gettimeofday(&t1, 0);
70              seed=(unsigned int)t1.tv_sec+(double)(t1.tv_usec/1000000.0);
71              srand(seed); // gera nova seq de nrs pseudo-aleatorios com base no relogio
72              for(i=0; i<ordem; i++){
73                      somalinha=0;
74                      for(j=0; j<ordem; j++){
75                              if(i!=j){
76                                      while((A[i*ordem+j]=rand()%100)==0);
77                                      somalinha+=A[i*ordem+j];
78                              }
79                      }
80                      // aumenta o vlr da diagonal para garantir (e facilitar) a convergencia
81                      A[i*ordem+i]=(somalinha+200);
82                      while((B[i]=rand()%100)== 0);
83              }
84              // ************************************************************************
85              // envia dados para os (nprocesses-1) escravos
86              count_aux=count;
87              i=0;
88              for(dest=1; dest<nprocesses; dest++){
89                      if(dest==(nprocesses-1)){
90                              count_aux=count_last;
91                      }
92                      msgtag=1;
93                      tid_dest=tid_tarefas[dest];
94                      pvm_initsend(PvmDataRaw); // create buffer
95                      pvm_pkdouble(&A[ i*ordem ], (ordem*count_aux), stride);
96                      pvm_pkdouble(&B[i], count_aux, stride);
97                      pvm_send(tid_dest, msgtag);
98                      i=i+count_aux;
99              }
100             // processo mestre ira receber o resultado dos testes de convergencia feitos pelos escravos
101             msgtag=2; // tag para esta mensagem
102             sai=0; // flag que determina se o mestre deverah sair depois que receber todos os testes de convergencia
103             // mestre recebe o teste de convergencia vindos dos escravos.
104             for(i=0; i<(nprocesses-1); i++){
105                     pvm_recv(-1, msgtag);
106                     pvm_upkint(&converge, 1, stride);
107
108                     if(!converge){
109                             sai=1;
110                     }
111             }
112             msgtag=3;
113             // mestre envia mensagem para os escravos suspendendo ou autorizando o calculo
114             for(i=0; i<(nprocesses-1); i++){
115                     tid_dest=tid_tarefas[i+1];
116                     pvm_initsend(PvmDataRaw); // create buffer
117                     pvm_pkint(&sai, 1, stride);
118                     pvm_send(tid_dest, msgtag);
119             }
120             if(sai){
121                     return;
122             }
123             iteracao=0;
124             sai=0;
125             do{
126                     iteracao++;
127                     //*********************************************************************
128                     // recebe diferenca entre iteracoes para verificar criterio de parada
129                     // recebe tambem o vetor X com a solucao desta iteracao
130                     // mestre vai receber o maior valor de X calculado nessa iteracao pelo escravo
131                     // e tb a maior diferenca em relacao ao X anterior
132                     for(i=0; i<(nprocesses-1); i++){
133                         msgtag=5;
134                         pvm_recv(-1, msgtag);
135                         pvm_upkdouble(&maxaux, 1, stride);
136                         if((i==0) || (maxaux>maxvetXnovo))
137                             maxvetXnovo=maxaux;
138                         // o MPI manda duas mensagens. O PVM apenas uma contendo os dois valores (maxaux e maxdif)
139                         pvm_upkdouble(&maxaux, 1, stride);
140                         if((i==0) || (maxaux>maxdif))
```

```
141                                     maxdif=maxaux;
142                             }
143                             mr=maxdif/maxvetXnovo;
144                             if(mr<=PRECISAO){
145                                 sai=1;
146                             }
147                             msgtag=7;
148                             // mestre envia mensagem para os escravos suspendendo ou autorizando a continuacao do calculo
149                             for(i=0; i<(nprocesses-1); i++){
150                                     pvm_initsend(PvmDataRaw); // create buffer
151                                     pvm_pkint(&sai, 1, stride);
152                                     pvm_send(tid_tarefas[(i+1)], msgtag);
153                             }
154                     }while(!sai);
155                     msgtag=9;
156                     // mestre vai receber o vetX[ ]  final do escravo[1]. Sera o resultado do algoritmo
157                     pvm_recv(tid_tarefas[1], msgtag);
158                     pvm_upkdouble(vetX, ordem, stride);
159                     i=rand()%ordem; // escolhe uma linha do sistema para confrontar o vlr de x
160                     somalinha=0.;
161                     for(j=0; j<ordem; j++){
162                             somalinha+=(A[i*ordem+j]*vetX[j]);
163                     } // fim do for
164                     return;
165             } // fim da rotina mestre()
166             //*************************************************************************
167             //*************************************************************************
168             void escravo(void){
169                     int i, j;
170                     int mycount; // indica qtas linhas este escravo tera que processar
171                     int diagonal; // indica o elemento da diagonal principal de A
172                     int sai;
173                     double *A, *B;
174                     double *Aestrela, *Bestrela;
175                     double *vetX, *vetXnovo, *difvetXnovo;
176                     double maxvetXnovo, maxdif;
177                     double *somalinha;
178                     // recebe primeira mensagem do mestre com a ordem da matriz
179                     tid_src=pai;
180                     msgtag=0; // rotulo da primeira mensagem vinda do mestre
181
182                     pvm_recv(tid_src, msgtag);
183                     pvm_upkint(&ordem, 1, stride); // unpack ordem da matriz
184                     pvm_upkint(&nprocesses, 1, stride); // unpack nprocesses
185                     tid_tarefas=(int *)malloc(nprocesses*sizeof(int));
186                     pvm_upkint(tid_tarefas, nprocesses, stride); // unpack nprocesses
187                     // *********************************************************
188                     // determina myrank para ficar similar ao codigo MPI
189                     // *********************************************************
190                     for(i=1; i<nprocesses; i++){
191                             if(tid_tarefas[i]==meutid){
192                                     myrank=i;
193                                     break;
194                             } // fim do if
195                     } // fim do for
196                     // *********************************************************
197                     if(ordem<(nprocesses-1)) // deverão haver pelo menos uma linha de A para um escravo
198                     {
199                             nprocesses=(ordem+1); // ordem escravos e mais 1 mestre
200                     }
201                     if(myrank>=nprocesses){
202                             return;
203                     }
204                     // determina qtde de valores a enviar para os escravos (com excecao do ultimo que recebera a dif)
205                     // floor arredonda para baixo sempre
206                     count=(int)floor(((double)ordem)/((double)(nprocesses-1)));
207                     count_last=ordem-((nprocesses-2)*count);
208                     // se for o ultimo escravo, este pega a diferenca de elementos de A e B
209                     if(myrank==(nprocesses-1)) // ultimo escravo
210                             mycount=count_last;
211                     else
212                             mycount=count;
213                     // ****************************************
214                     // aloca memoria para matrizes e vetores
215                     A=(double *)malloc((mycount*ordem)*sizeof(double));
216                     B=(double *)malloc(mycount*sizeof(double));
```

```
217		Aestrela=(double *)malloc((mycount*ordem)*sizeof(double));
218		Bestrela=(double *)malloc(ordem*sizeof(double));
219		vetX=(double *)malloc(ordem*sizeof(double));
220		vetXnovo=(double *)malloc(mycount*sizeof(double));
221		difvetXnovo=(double *)malloc(mycount*sizeof(double));
222		somalinha=(double *)malloc(mycount*sizeof(double));
223		// ****************************************
224		//*********************************************************************
225		msgtag=1;
226		pvm_recv(tid_src, msgtag);
227		pvm_upkdouble(&A[ 0 ], (ordem*mycount), stride);
228		pvm_upkdouble(B, mycount, stride);
229		// o default eh que o sistema converge
230		converge=1;
231		//*********************************************************************
232		// calcula Aestrela para este processo escravo
233		for(i=0; i<mycount; i++){
234			// soma as linhas de Aestrela para ver se converge. Nao soma elem. da diagonal principal.
235			somalinha[i]=0.;
236			//encontra o indice para a diagonal principal desta linha. Deve usar count para determinar
237			//qtas linhas de A cada escravo anterior ja possui
238			diagonal=i+(count*(myrank-1));
239			for(j=0; j<ordem; j++){
240				if(j!=diagonal){
241					Aestrela[i*ordem+j]=A[i*ordem+j]/A[i*ordem+diagonal];
242					somalinha[i]+=(double)fabs(Aestrela[i*ordem+j]);
243				}
244				else{
245					Aestrela[i*ordem+j]=0.;
246				}
247			}
248			// calcula o Bestrela[ ] na posicao correta. Os outros vlrs de Bestrela serao recebido por mensagem depois
249			Bestrela[diagonal]=B[i]/A[i*ordem+diagonal];
250			if(somalinha[i]>=1){
251				converge=0;
252			}
253		}
254		tid_dest=tid_src; // processo mestre ira receber o resultado do teste de convergencia deste escravo
255		msgtag=2; // tag para esta mensagem
256		// enviando resultado teste de convergencia para o mestre.
257		pvm_initsend(PvmDataRaw); // create buffer
258		pvm_pkint(&converge, 1, stride);
259		pvm_send(tid_dest, msgtag);
260		src=0;
261		msgtag=3;
262		pvm_recv(tid_src, msgtag);
263		pvm_upkint(&sai, 1, stride);
264		if(sai){
265			return;
266		}
267		msgtag=4;
268		// este escravo enviara mensagens para os outros escravos, informando os elementos de Bestrela
269		// que estao aqui e os demais escravos nÃ£o tem.
270		for(i=1; i<nprocesses; i++) // envia para o escravo com rank 1,2,3,..., nprocesses-1
271		{
272			if(myrank!=i) // nao envia para ele mesmo
273			{
274				// i = 1 porque os ranks dos escravos comecam em 1, 2, 3, ...
275				// envia a porcao de Bestrela que este escravo calculou para os demais escravos
276				pvm_initsend(PvmDataRaw); // create buffer
277				pvm_pkdouble(&Bestrela[((myrank-1)*count)], mycount, stride);
278				pvm_send(tid_tarefas[i], msgtag);
279			}
280		}
281		// este escravo recebera mensagens dos outros escravos contendo os elementos de Bestrela
282		// que este escravo nao tem.
283		for(i=1; i<nprocesses; i++) // recebe do escravo com rank 1,2,3,..., nprocesses-1
284		{
285			if(myrank!=i) // nao recebe dele mesmo
286			{
287				if(i<(nprocesses-1)) {
288					pvm_recv(tid_tarefas[i], msgtag);
289					pvm_upkdouble(&Bestrela[((i-1)*count)], count, stride);
290				}
291				else // se for receber do ultimo escravo, entao recebe a sobra dos elementos (count_last)
292				{
```

```
293                                             pvm_recv(tid_tarefas[i], msgtag);
294                                             pvm_upkdouble(&Bestrela[((i-1)*count)], count_last, stride);
295                                     }
296                             }
297                     }
298                     // o primeiro vetor X Ã© igual ao vetor Bestrela
299                     for(i=0; i<ordem; i++){
300                             vetX[i]=Bestrela[i];
301                     }
302                     do{
303                             for(i=0; i<mycount; i++){
304                                     somalinha[i]=0.;
305                                     //encontra o indice para a diagonal principal desta linha. Deve usar count para determinar
306                                     //qtas linhas de A cada escravo anterior ja possui
307                                     diagonal=i+(count*(myrank-1));
308                                     for(j=0; j<ordem; j++){
309                                             if(j!=diagonal){
310                                                     somalinha[i]+=(Aestrela[i*ordem+j]*vetX[j]);
311                                             }
312                                     }
313                                     vetXnovo[i]=Bestrela[diagonal]-somalinha[i];
314                                     difvetXnovo[i]=(double)fabs((vetXnovo[i]-vetX[diagonal]) );
315                                     // se for a primeira iteracao ou se o vlr atual for maior que os anteriores, entao seleciona o maior
316                                     if((i==0) || ((double)fabs(vetXnovo[i])>maxvetXnovo))
317                                             maxvetXnovo=(double)fabs(vetXnovo[i]);
318                                     // se for a primeira iteracao ou se o vlr atual for maior que os anteriores, entao seleciona o maior
319                                     if((i==0) || (difvetXnovo[i]>maxdif))
320                                             maxdif=difvetXnovo[i];
321                             }
322                             dest=tid_tarefas[0]; // processo mestre ira receber o resultado do teste de convergencia
323                             msgtag=5; // tag para esta mensagem
324                             // enviando o maxvetXnovo (parcial) para o mestre.
325                             pvm_initsend(PvmDataRaw); // create buffer
326                             pvm_pkdouble(&maxvetXnovo, 1, stride);
327                             pvm_pkdouble(&maxdif, 1, stride);
328                             pvm_send(dest, msgtag);
329                             // mestre ja verificou o criterio de parada e vai informar se continua ou nao calculando
340                             src=tid_tarefas[0];
341                             msgtag=7;
342                             pvm_recv(src, msgtag);
343                             pvm_upkint(&sai, 1, stride);
344                             if(!sai){ // ainda nao atingiu o criterio de parada
345                                 msgtag=8;
346                                 // este escravo enviara mensagens para os outros escravos, informando os elementos do vetXnovo
347                                 // que estao aqui e os demais escravos nÃ£o tem.
348                                 for(i=0; i<(nprocesses-1); i++){
349                                         if((myrank-1)!=i) // nao envia para ele mesmo
350                                         {
351                                            // destino == i + 1 porque os ranks dos escravos nao comecam em zero, mas sim em 1, 2, 3, ...
352                                            // envia para os demais escravos a porcao de vetXnovo que este escravo calculou
353                                            pvm_initsend(PvmDataRaw); // create buffer
354                                            pvm_pkdouble(&vetXnovo[0], mycount, stride);
355                                            pvm_send(tid_tarefas[(i+1)], msgtag);
356                                         }
357                                         else{
358                                                 //acerta os seus proprios valores em vetX[ ]
359                                                 for(j=0; j<mycount; j++){
360                                                         vetX[((i*count)+j)]=vetXnovo[j];
361                                                 }
362                                         }
363                                 }
364                                 // este escravo recebera mensagens dos outros escravos contendo os elementos de de vetX
365                                 // que este escravo nao tem.
366                                 for(i=0; i<(nprocesses-1); i++){
367                                         if((myrank-1)!=i) // nao recebe dele mesmo
368                                         {
369                                                 if((i+1)<(nprocesses-1)) // nao eh o ultimo processo (que pega a sobra)
370                                                 {
371                                                         pvm_recv(tid_tarefas[(i+1)], msgtag);
372                                                         pvm_upkdouble(&vetX[(i*count)], count, stride);
373                                                 }
374                                                 else // se for do ultimo escravo, recebe apenas a sobra dos elementos (count_last)
375                                                 {
376                                                         pvm_recv(tid_tarefas[(i+1)], msgtag);
377                                                         pvm_upkdouble(&vetX[(i*count)], count_last, stride);
378                                                 }
```

```
                                        }
                                    }
                                }
                }while(!sai);
                // todos os escravos tem o vetX[ ] completo. Somente um precisa enviar esse vetor par o mestre no final das iteracoes.
                // escolhi o primeiro escravo...
                if(myrank==1){
                        msgtag=9;
                        dest=tid_tarefas[0];
                        // enviara para o mestre a sua porcao de vetX[ ]
                        // destino == i + 1 porque os ranks dos escravos nao comecam em zero, mas sim em 1, 2, 3, ...
                        // envia para os demais escravos a porcao de vetXnovo que este escravo calculou
                        pvm_initsend(PvmDataRaw); // create buffer
                        pvm_pkdouble(vetX, ordem, stride);
                        pvm_send(dest, msgtag);
                }
                return;
        } // fim de escravo( )
        //*************************************************************************
        //*************************************************************************
        int main(int argc, char *argv[]){
            pai=pvm_parent();
            meutid=pvm_mytid();
            if(pai==PvmNoParent) // e o processo mestre, pois nao ha pai
            {
                myrank=0;
                if(argc!=3){
                        exit(0);
                }
                ordem=atoi(argv[1]);
                nprocesses=atoi(argv[2]);
                if(ordem<1) {
                        exit(0);
                }
                if(nprocesses<1){
                        exit(0);
                }
                nprocesses++;
                tid_tarefas=(int *)malloc(nprocesses*sizeof(int));
                tid_tarefas[0]=meutid;
                result=pvm_spawn(JACOBI, (char **)NULL, PvmTaskDefault, (char*)NULL, (nprocesses-1), &tid_tarefas[1]);
                if(result<(nprocesses-1)){
                        pvm_exit();
                        exit(1);
                }
                mestre();
            }
            else{
                escravo();
            }
            pvm_exit();
            exit(0);
        } // main()
```