Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

Estudo de Escalabilidade em Sistemas de Processamento Paralelo Virtual

Autor: Fábio Soares Rocha
Orientador: Marco Aurélio Amaral Henriques

Dissertação de Mestrado apresentada à Faculdade de Engenharia Elétrica e de Computação da UNICAMP como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica. Área de concentração: Engenharia de Computação

Comissão Examinadora

Profa. Dra. Alice M. B. H. Tokarnia.....FEEC/UNICAMP
Prof. Dr. Célio Cardoso Guimarães.....IC/UNICAMP
Prof. Dr. Marco Aurélio Amaral Henriques.....FEEC/UNICAMP

Campinas
Março de 2010

FICHA CATALOGRÁFICA ELABORADA PELA
BIBLIOTECA DA ÁREA DE ENGENHARIA E ARQUITETURA - BAE - UNICAMP

R582e

Rocha, Fábio Soares
Estudo de escalabilidade em sistemas de processamento paralelo virtual / Fábio Soares Rocha. -- Campinas, SP: [s.n.], 2010.

Orientador: Marco Aurélio Amaral Henriques.
Dissertação de Mestrado - Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação.

1. Processamento paralelo (Computadores). I. Henriques, Marco Aurélio Amaral. II. Universidade Estadual de Campinas. Faculdade de Engenharia Elétrica e de Computação. III. Título.

Título em Inglês: Scalability study of virtual parallel processing systems
Palavras-chave em Inglês: Parallel processing (Computers)
Área de concentração: Engenharia de Computação
Titulação: Mestre em Engenharia Elétrica
Banca examinadora: Marco Aurélio Amaral Henriques, Célio Cardoso Guimarães, Alice Maria Bastos Hubinger Tokarnia
Data da defesa: 31/03/2010
Programa de Pós Graduação: Engenharia Elétrica

COMISSÃO JULGADORA – TESE DE MESTRADO

Candidato: Fábio Soares Rocha

Data da Defesa: 31 de março de 2010

Título da Tese: "Estudo de Escalabilidade em Sistemas de Processamento Paralelo Virtual"

Prof. Dr. Marco Aurélio Amaral Henriques (Presidente): _____

Prof. Dr. Célio Cardoso Guimarães: _____

Profa. Dra. Alice Maria Bastos Hubinger Tokarnia: _____

Agradecimentos

Aos meus pais, Wilson e Solange e meu irmão, Cris.

A minha esposa, Elen, pelo apoio e compreensão.

Ao meu orientador, Prof. Marco Aurélio, pela atenção e confiança.

Resumo

A utilização de sistemas de processamento paralelo virtual tem aumentado em várias áreas de aplicação, desde as estritamente matemáticas até as médicas e biológicas. Devido a esse crescimento, são cada vez mais necessários mecanismos para uma avaliação consistente de desempenho de sistemas desse tipo. É necessário também haver um entendimento mais preciso do conceito de escalabilidade, um dos principais indicadores de desempenho, tanto quanto ao seu significado como quanto à maneira de mensurá-la. Este trabalho traz um estudo comparativo sobre métricas de escalabilidade para sistemas de processamento paralelo homogêneos e heterogêneos, onde duas métricas homogêneas foram selecionadas para serem utilizadas na avaliação do limite superior de escalabilidade de duas plataformas de processamento paralelo virtual (JoiN a JPVM). A partir deste estudo foi proposta uma métrica de escalabilidade que utiliza como base para a sua análise o *speedup*, conceito muito familiar em processamento paralelo. Foram realizados testes de validação da métrica proposta, que destacam seu caráter prático e adequação para a aplicação em sistemas heterogêneos de processamento paralelo virtual.

Abstract

The utilization of Virtual Parallel Processing Systems has increased in several application areas, from strictly mathematical to medical and biological areas. Due to this increasing, mechanisms that offer a consistent performance evaluation of these systems, became more important and useful. It is also important to understand the concept of scalability, one of the main performance pointers, and the way of measuring it. This work shows a comparative study about scalability metrics of homogeneous and heterogeneous parallel processing systems, where two of the homogeneous metrics, were selected to evaluate the scalability upper limit on two virtual parallel processing platforms (JoiN and JPVM). As a result, We proposed a new scalability metric, which is based on the *speedup*, a well known parallel processing concept. Validation tests were performed using the proposed metric, which highlight its characteristics and suitability to heterogeneous virtual parallel processing systems.

Sumário

Lista de Figuras

Lista de Tabelas

Glossário

SPPV	Sistemas de processamento paralelo virtual
CAPP	Combinação algoritmo plataforma paralela
BoT	*Bag of Tasks*
EREW	*Exclusive Read Exclusive Write*
PRAM	*Parallel Random Access Machine*
flop	*Floating point Operations*
flops	*Floating point Operations Per Second*
JPVM	*Java Parallel Virtual Machine*
PASL	*Parallel Application Specification Language*
GSTR	*Generational Scheduler with Tasks Replication*
PVM	*Parallel Virtual Machine*
fdr	fator de desempenho relativo

Capítulo 1

Introdução

Não são poucos os problemas que exigem um grande poder computacional para sua resolução. São exemplos disso: previsão metereológica, processamento de imagens em tempo real, cálculo de volume/área de proteínas entre outros. Áreas como a Física, Matemática ou mesmo a Biologia, possuem diversos exemplos de desafios computacionais complexos que geralmente excedem a capacidade computacional que pode ser oferecida por uma única máquina monoprocessada. As máquinas multiprocessadas oferecem uma alternativa a esses casos, pois permitem a execução mais rápida de um grande problema que possa ser divido em tarefas menores, as quais são entregues e resolvidas paralelamente em cada processador presente na arquitetura. Apesar de propiciarem um melhor desempenho, máquinas multiprocessadas possuem arquiteturas que dificultam, ou mesmo, não permitem expansão, tendo assim sua capacidade computacional limitada.

Os sistemas de processamento paralelo virtual (SPPV) são sistemas computacionais que aproveitam uma rede já existente (internet, por exemplo) para interligar um número teoricamente ilimitado de computadores, que podem estar geograficamente espalhados e ter capacidades computacionais diferentes. O objetivo principal desses sistemas é agregar a maior quantidade possível de máquinas a seu serviço e utilizar todo o poder computacional resultante na resolução de problemas de grande porte. Para isso, devem possuir uma arquitetura que ofereça fácil expansão e possa ser gerenciada para distribuir paralelamente as tarefas que compõem o grande

problema. Cada máquina participante da plataforma processa as tarefas que recebe e retorna o resultado para uma unidade centralizadora que as coleta e compõe o resultado final do problema.

Dada à importância e crescente utilização de sistemas de processamento paralelo virtual, surge também a necessidade de mecanismos que propiciem uma análise detalhada de desempenho destes. Pelo fato de só serem realmente úteis quando utilizam eficientemente um número elevado de nós de processamento, é desejável saber como o desempenho da plataforma se comporta à medida que seu tamanho aumenta, ou mesmo, como se comporta à medida que recebe uma carga maior de trabalho.

O caminho para responder estas questões é a **avaliação da escalabilidade** da plataforma. A escalabilidade é um fator para a análise de desempenho de sistemas de processamento paralelo e costuma ser entendida como a capacidade que um sistema de processamento paralelo tem de oferecer um aumento no poder de processamento real proporcional ao aumento no número de nós processadores, ou seja, se um sistema de processamento paralelo sofre um aumento no número de máquinas a seu serviço, deseja-se saber quanto desse aumento é realmente refletido no poder de processamento. A avaliação da escalabilidade permite entender o comportamento do crescimento do sistema paralelo e prever o seu desempenho com uma quantidade maior de processadores. Isto é vantajoso quando, na prática for inviável realizar testes com um número realmente elevado de nós de processamento. Além disso, esta avaliação da escalabilidade torna-se mais importante à medida que surgem SPPV que se propõem a utilizar um número muito elevado de nós de processamento.

Entretanto, apesar de sua visível importância, não existe ainda uma definição nem um método de obtenção de escalabilidade que seja amplamente aceito e aplicado. Algumas tentativas foram feitas no sentido de buscar, além de uma definição suficientemente abrangente, uma maneira consistente de medir a escalabilidade de um sistema de processamento paralelo. As métricas de escalabilidade existentes na literatura são em sua maioria baseadas em sistemas homogêneos [1-6], onde todas as máquinas trabalhadoras podem ser consideradas idênticas sob o ponto de vista de processamento e comunicação. Poucas são as propostas feitas no sentido de definir uma métrica de escalabilidade para sistemas heterogêneos [15-17], caso onde os SPPV se enquadram, pois além da estrutura de comunicação poder variar ao longo do sistema, cada máquina participante do sistema pode possuir um poder computacional diferente, o que dificulta

significativamente a análise da escalabilidade. Como é mostrado nesse trabalho, em ambos os casos, as métricas propostas na literatura apresentam deficiências tanto em sua formulação e adequação a SPPV, quanto à sua aplicação prática.

Este trabalho faz um levantamento e analisa diversas métricas de escalabilidade para sistemas homogêneos propostas na literatura. Dentre elas duas das mais referenciadas foram selecionadas para uma aplicação prática visando avaliar o limite superior de escalabilidade de duas plataformas de processamento paralelo virtual (JoiN e JPVM). Os resultados dos testes somados à análise das propostas existentes de métricas de escalabilidade para sistemas heterogêneos ajudaram a compor uma nova métrica, que se destaca por utilizar o conhecido conceito de *speedup* como base para sua análise e possuir um caráter prático que a torna adequada para a utilização em SPPV, seja ele homogêneo ou heterogêneo. A nova métrica proposta foi testada demonstrando sua capacidade de caracterizar a escalabilidade em um ambiente heterogêneo utilizando a plataforma de processamento paralelo virtual JoiN.

1.1 Objetivos do trabalho e organização do texto

Os objetivos principais deste trabalho são: (i) estudar as métricas de escalabilidade existentes, tanto para sistemas homogêneos quanto para sistemas heterogêneos, a fim de identificar deficiências e qualidades; (ii) avaliar na prática duas das métricas homogêneas para análise do limite superior da escalabilidade de duas plataformas de processamento paralelo virtual, (iii) propor e validar uma métrica de escalabilidade que reúna os pontos fortes das existentes e seja aplicável a sistemas de processamento paralelo virtual heterogêneos e homogêneos.

Essa dissertação esta organizada da seguinte forma.

Cap. 2 Conceitos Básicos – apresenta alguns conceitos básicos para o estudo de sistemas de processamento paralelo. São apresentados alguns termos como *speedup*, carga de trabalho, aplicação paralela, Combinação Algoritmo-Plataforma Paralela, e como estes podem estar relacionados com a análise da escalabilidade de SPPV.

Cap. 3 Métricas de Escalabilidade – apresenta uma visão geral das diversas métricas de escalabilidade para sistemas paralelos homogêneos propostas na literatura, sendo que as duas mais referenciadas, Isovelocidade e Isoeficiência, são tratadas com mais detalhes. Ainda neste

capítulo, são apresentadas as propostas existentes na literatura de métricas de escalabilidade para sistemas heterogêneos.

Cap. 4 Testes com métricas de escalabilidade em sistemas homogêneos – apresenta e analisa os testes realizados com duas métricas de escalabilidade para sistemas homogêneos: isoeficiência e isovelocidade, quando aplicadas na avaliação do limite superior da escalabilidade de dois ambientes de processamento paralelo virtual.

Cap. 5 Proposta de métrica de escalabilidade para SPPV heterogêneos – propõe uma nova métrica de escalabilidade denominada métrica da isoeficiência de *speedup*, a qual é adequada a sistemas heterogêneos de processamento paralelo virtual. É proposta ainda uma adaptação do conceito de *speedup* para ambientes heterogêneos, o qual serve de base para a nova métrica de escalabilidade.

Cap. 6 Testes com a métrica da Isoeficiência de Speedup em sistemas heterogêneos – apresenta e analisa os testes realizados para a validação da métrica de escalabilidade da Isoeficiência de *Speedup* proposta neste trabalho.

Cap. 7 Conclusões e trabalhos futuros – apresenta os principais resultados deste trabalho e discute pontos que poderiam ser mais explorados no futuro.

Capítulo 2

Conceitos Básicos

Neste capítulo são apresentados alguns conceitos básicos para o estudo de sistemas de processamento paralelo. As definições apresentadas aqui servem de base para o entendimento das discussões e análises apresentadas nos demais capítulos desta dissertação.

2.1 Carga de trabalho

Um conceito discutido na literatura e que será utilizado largamente durante este trabalho é a definição de tamanho do problema ou carga de trabalho de uma aplicação. A carga de trabalho, denotada pelo símbolo *w*, deve ser um parâmetro que retrate diretamente a quantidade de computação realizada em uma determinada aplicação, sem permitir variações de interpretação [2-4].

Por exemplo, em uma aplicação envolvendo matrizes *n* x *n*, a escolha de *n* como parâmetro de carga de trabalho (*w*) pode parecer correta. Porém, seguindo a definição acima, ela não é adequada, pois dependendo do problema ao qual essa aplicação está relacionada, o parâmetro *n* reflete a carga do algoritmo de maneira diferente. Veja que ao dobrarmos o valor de *n*, teríamos a quantidade total de operações realizadas por um algoritmo de multiplicação de matrizes multiplicada por oito. No caso de uma aplicação de adição de matrizes a carga seria apenas quadruplicada quando *n* passasse para 2*n*.

Portanto para evitarmos esse tipo de situação, a definição da carga de trabalho *w* é dada como o número de operações básicas (instruções ou operações aritméticas) executadas em um único processador pelo melhor algoritmo sequencial que resolve o problema [2]. Na prática, nem sempre se conhece o melhor algoritmo, portanto utiliza-se o melhor algoritmo conhecido.

Uma vantagem da utilização dessa definição é a proporcionalidade entre *w* e o tempo de execução serial (T_s) em um único processador. Se t_c é o tempo de computação gasto em média pelo processador para executar uma única operação básica, podemos estimar o tempo médio de execução serial de uma aplicação com carga *w*, por:

$$T_s(w) = w.t_c \quad (1)$$

Uma possibilidade para a definição de *w* é restringir a contabilização a apenas operações de ponto flutuante realizadas no algoritmo [4], porém esta escolha ignora o efeito de outras operações que podem ser significativas no tempo de execução do algoritmo, como Entrada/Saída de dados por exemplo.

Uma abordagem diferente na definição de *w* é baseada na quantidade de memória requerida na execução da aplicação [14]. Esta considera que a maioria dos algoritmos necessita que todos os dados estejam alocados na memória principal para sua utilização. Portanto, o tamanho máximo do problema que pode ser resolvido, e consequentemente o máximo desempenho que um sistema de processamento poderia atingir, é limitado pelo tamanho da sua memória principal.

Concordamos que a definição de *w*, baseada na memória, seria mais adequada a alguns casos específicos de sistemas de processamento paralelo e algoritmos, o que requereria uma análise individualizada. Porém, por este trabalho ter como foco SPPV, onde se pressupõe a possibilidade de divisão de trabalho em muitas tarefas pequenas, os problemas de limitação de memória não são considerados. Portanto adotamos a definição de carga de trabalho *w*, da equação 1, que é baseada no número de operações básicas realizadas pela aplicação.

2.2 Aplicação paralela

Basicamente uma aplicação paralela é um algoritmo que resolve um determinado problema dividindo-o em partes (tarefas), que podem ser executadas paralelamente por vários

processadores ou trabalhadores, onde, trabalhador é uma máquina real ou virtual capaz de executar uma ou mais threads, sendo cada thread uma tarefa da aplicação paralela.

O objetivo principal da paralelização de um problema pode ser obter um tempo de resolução menor que o tempo que seria gasto para executar uma versão serial do algoritmo em uma única máquina, ou mesmo, conseguir processar um problema maior no mesmo tempo que sua versão serial de menor dimensão.

Considerando as características de SPPV, como número elevado de nós de processamento, fraca comunicação (internet, por exemplo) e grande volume de dados para processamento, o modelo de aplicação paralela BoT (*Bag of Tasks*) [13] foi escolhido para utilização neste trabalho. Nele a aplicação paralela é composta de tarefas independentes (que não se comunicam entre si). As tarefas são preparadas por uma máquina coordenadora e entregues às máquinas trabalhadoras participantes. Estas executam paralelamente o trabalho e devolvem seus resultados à coordenadora que os processa e obtém o resultado final. Em sua definição mais geral, o modelo BoT prevê aplicações que possuam mais de um lote de tarefas em uma mesma execução, isso significa que a partir de todos os resultados provenientes dos trabalhadores, a unidade coordenadora poderia ainda gerar um outro conjunto de tarefas a ser distribuído e reiniciar o ciclo descrito acima. Para tornar o modelo mais eficiente em SPPV, as tarefas criadas, pertencentes ao um mesmo lote, são preferencialmente iguais, ou seja, exigem a mesma quantidade de computação para serem finalizadas, o que facilita a distribuição e balanceamento da carga de trabalho entre os trabalhadores. Este modelo é utilizado por permitir que a capacidade de paralelização da aplicação seja altamente explorada, pois uma vez que não é permitida a troca de mensagens entre máquinas trabalhadoras os gastos extras com comunicação são reduzidos.

O gráfico da Figura 1 retrata, de maneira simplificada, como se comporta uma típica aplicação do tipo BoT, com apenas um lote de tarefas, em um sistema de processamento paralelo homogêneo com *N* trabalhadores. Cada seta horizontal na figura representa o eixo do tempo em cada componente da plataforma, onde o trecho em negrito corresponde ao intervalo de tempo em que o componente está efetivamente processando algum dado. As setas transversais representam a comunicação entre o componente coordenador e os componentes trabalhadores da plataforma, inicialmente enviando as tarefas e posteriormente os resultados. O componente chamado de

coordenador é aquele que prepara as tarefas, envia-as para os componentes trabalhadores, recebe os resultados e computa o valor final da aplicação.

Através de carimbos de tempo (t_1, t_2, t_3 e t_4), no componente coordenador da plataforma, é possível obter o tempo de cada etapa da aplicação paralela, assim como computar o tempo total de execução.

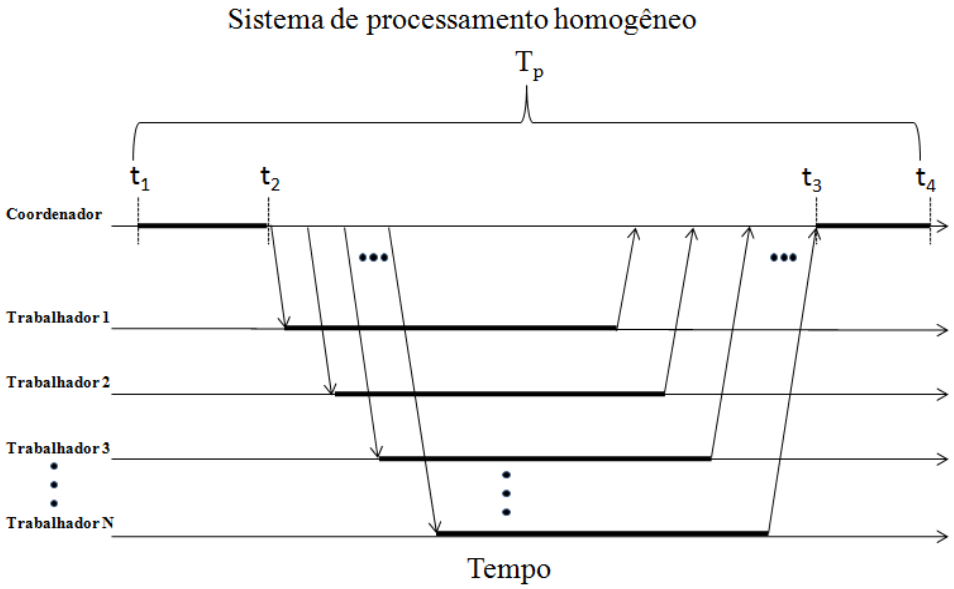

Figura 1- Tempos de execução de uma aplicação paralela do tipo Bag of Tasks em um sistema de processamento paralelo homogêneo.

Os seguintes intervalos de tempo são obtidos a partir das medições mostradas na Figura 1.

- (t_4 - t_1) - tempo total de execução da aplicação paralela (T_{par}): é computado a partir do momento que o componente coordenador inicia a aplicação paralela, até o momento que ele calcula o resultado final.

- (t_2 - t_1) - tempo de preparação das tarefas: trabalho serial que compreende o tempo gasto pelo coordenador para preparar as tarefas que serão entregues aos trabalhadores e eventualmente algum trabalho não paralelizável no algoritmo.

- (t_4 - t_3) - tempo de computação do resultado final: trabalho serial que compreende o tempo gasto pelo coordenador para computar o resultado final da aplicação após receber o resultado da última tarefa e eventualmente algum trabalho não paralelizável no algoritmo.

- (t_3 - t_2) - tempo de comunicação e processamento paralelo: o intervalo de tempo no coordenador entre, enviar a primeira tarefa e receber o resultado da última tarefa,

englobando o tempo gasto em comunicação e o tempo de processamento paralelo gasto nos trabalhadores.

Ainda na Figura 1, podemos observar que os tempos de processamento paralelo em cada um dos componentes trabalhadores, assim como os de comunicação, são iguais, o que permite que a ordem de envio seja mantida no recebimento das tarefas no coordenador. Isso se deve ao fato do sistema em questão ser homogêneo e das tarefas da aplicação terem cargas de trabalho iguais. Na Figura 2 temos um exemplo simplificado de como poderia ser o comportamento da mesma aplicação sendo executada em um sistema heterogêneo tanto do ponto de vista do processamento como de comunicação.

Em sistemas heterogêneos não se tem garantia de que a ordem de envio das tarefas será a mesma do recebimento nos trabalhadores, e muito menos, que será a ordem de término das tarefas. Repare que, como as tarefas são executadas em velocidades diferentes ocorre um aumento significativo do tempo em que os trabalhadores, principalmente os mais rápidos, ficam ociosos durante a execução. Isso no caso do critério de distribuição de tarefas do sistema homogêneo anterior ser mantido (uma tarefa por trabalhador).

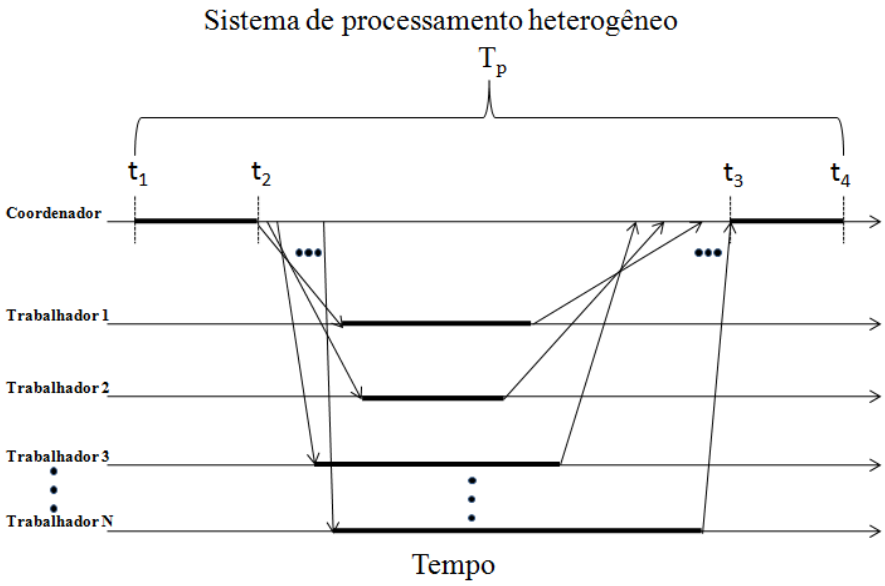

Figura 2 - Tempos de execução de uma aplicação paralela do tipo *Bag of Tasks* em um sistema de processamento paralelo heterogêneo.

Uma estratégia para minimizar esse tempo desperdiçado, é dividir o mesmo problema em um número maior de tarefas e enviá-las sob demanda, na medida em que os trabalhadores ficam ociosos. Um exemplo de uma nova execução em um sistema heterogêneo, agora com um número maior de tarefas (menores) é mostrado na Figura 3.

O exemplo da Figura 3 mostra que os trabalhadores permanecem mais tempo ocupados, o que significa dizer que temos um melhor aproveitamento do sistema, pois a carga de trabalho está mais balanceada entre os trabalhadores.

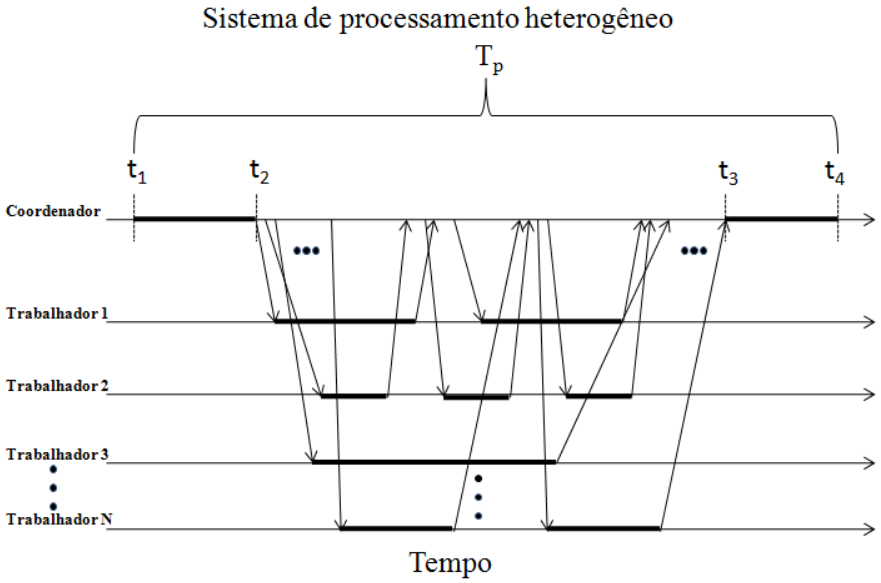

Figura 3 - Execução paralela em um sistema de processamento paralelo heterogêneo com um número grande de tarefas.

O modelo BoT utilizando tarefas com cargas de processamento iguais foi escolhido para ser adotado neste trabalho, pois se adéqua bem a SPPV por propiciar a exploração do paralelismo da aplicação restringindo a comunicação e facilitar o balanceamento de carga entre os trabalhadores através divisão da aplicação em tarefas iguais. Porém mesmo tratando-se de aplicações que seguem o modelo BoT, é possível perceber, pelo exemplo da Figura 3, que o balanceamento ideal de carga em sistemas heterogêneos não é um assunto simples de resolver e a capacidade computacional individual de cada trabalhador deve ser considerada na determinação do tamanho e da quantidade de tarefas a serem distribuídas.

2.2.1 Porção serial do algoritmo paralelo

Pode-se dizer que todo algoritmo paralelo possui uma porção serial que é também processamento útil inerente a ele, por menor que seja, sempre existe uma parcela do tempo que é empregada em operações seriais executadas em um único processador.

As operações seriais executadas em um algoritmo paralelo podem ser originadas a partir de: operações não paralelizáveis ou *overhead* de paralelização.

Operações não paralelizáveis: são as operações que já eram executadas na versão serial do algoritmo, e que apesar de estarem presentes na versão paralela, não permitem paralelização. Um exemplo disso seria um processamento adicional aplicado ao resultado proveniente do retorno das tarefas. As operações não paralelizáveis são uma característica exclusiva do próprio algoritmo e não dependem de fatores relacionados à plataforma.

***Overhead* de paralelização:** são todas as operações que não são necessárias na versão serial do algoritmo e foram adicionadas pelo processo de paralelização. Dentre essas operações encontram-se a preparação e envio das tarefas e dados e a posterior coleta e processamento dos resultados. Essas operações estão sempre presentes em um algoritmo paralelo e se originam do chamado *overhead* de paralelização. A dependência da relação entre arquitetura da plataforma e o algoritmo utilizado faz com que o *overhead* de paralelização varie em função do número de trabalhadores envolvidos [5] e por envolver as operações de envio e recebimento de dados, pode também sofrer influência da carga de trabalho *w*.

Portanto, o conjunto das operações seriais executadas pelo algoritmo paralelo é resultado da soma das operações não paralelizáveis e das originadas pelo *overhead* de paralelização.

Sendo $T_o(w,p)$ o tempo gasto com as operações seriais do algoritmo paralelo e $T_{par}(w,p)$ o tempo total de execução da aplicação paralela com carga *w* em *p* processadores, define-se a porção serial (q) do algoritmo paralelo como:

$$q(w,p) = \frac{T_o(w,p)}{T_{par}(w,p)} \tag{2}$$

Como pode ser visto na equação 2, devido ao fato de T_o (pelo *overhead* de paralelização) e T_{par} serem função de *w* e *p*, a porção serial do algoritmo paralelo pode ser influenciada pela

variação da quantidade de trabalhadores e/ou da carga de trabalho do algoritmo. Isto só não ocorre no caso de *w* e *p* produzirem variações exatamente iguais em T_o e T_{par}, algo pouco comum na prática.

2.3 Escalabilidade

No capítulo introdutório desse trabalho foi apresentada uma idéia geral do conceito de escalabilidade em sistemas de processamento paralelo.

"*Escalabilidade é a capacidade que um sistema de processamento paralelo tem de oferecer um aumento no poder de processamento real proporcional ao aumento no número de nós processadores a seu serviço*"

A definição acima é correta, porém um importante detalhe deve ser ressaltado. O termo "sistema de processamento paralelo" utilizado acima se refere não somente à plataforma de processamento paralelo, mas também à aplicação paralela que está sendo executada na plataforma.

O tipo de interconexão entre os nós de processamento e o tipo de escalonamento das tarefas para balanceamento e distribuição da carga de processamento são fatores que afetam a escalabilidade do sistema e estão diretamente ligados à arquitetura da plataforma. A característica de variação na porção serial do algoritmo devido ao *overhead* de paralelização é um fator de igual importância para a escalabilidade e está relacionado com a estrutura do algoritmo, ou seja, como a aplicação paralela foi implementada. Surge então o termo CAPP que significa uma **Combinação Algoritmo-Plataforma Paralela**. A partir disso não podemos discutir ou avaliar a escalabilidade de uma plataforma de processamento paralelo independentemente, sendo então a escalabilidade uma propriedade sempre associada a uma CAPP.

Uma CAPP que utiliza um ambiente de processamento paralelo homogêneo será referenciada neste trabalho como uma CAPP homogênea e analogamente aquela que utiliza um ambiente heterogêneo será tratada como CAPP heterogênea. Essa nomenclatura foi criada neste trabalho com a finalidade de simplificar a leitura.

2.4 *Speedup*

O *speedup* é certamente um dos conceitos mais importantes relacionados a processamento paralelo. Ele nos passa uma idéia do ganho (ou perda) associado à paralelização de um determinado algoritmo.

Seguindo sua definição clássica [26], o *speedup* é calculado pela razão entre o tempo de execução da melhor versão serial do algoritmo (T_s) e o tempo de execução da versão paralela (T_{par}) do mesmo algoritmo quando executado em p processadores, ambos utilizando a mesma carga de trabalho w.

$$S(w,p) = \frac{T_s(w)}{T_{par}(w,p)} \tag{3}$$

Existem duas abordagens principais que utilizam *speedup* para analisar a escalabilidade [8]. A primeira delas considera a **carga fixa**, onde o *speedup* é calculado à medida que o número de processadores aumenta e a carga de trabalho a ser executada é mantida fixa. Diferentemente da primeira, a segunda trabalha com a **carga variável** e nesta análise a carga do problema pode variar com o número de processadores envolvidos. Porém ambos os tempos (serial e paralelo) são medidos considerando a mesma carga de trabalho.

2.4.1 Carga fixa

Analisando a situação onde não temos variação na carga de trabalho (w) do algoritmo, se aumentarmos o número de processadores trabalhando paralelamente, podemos esperar que o tempo de execução total diminua proporcionalmente.

Em um sistema de processamento homogêneo, sendo $T_s(w)$ o tempo de execução serial de um algoritmo totalmente paralelizável com carga w em um único processador, o tempo de execução paralela em p processadores idênticos é dado por:

$$T_{par}(w,p) = \frac{T_s(w)}{p}, \tag{4}$$

desconsiderando qualquer *overhead* causado pela paralelização do algoritmo.

Portanto, como pode ser verificado a partir das equações (3) e (4), o comportamento ideal do *speedup* com carga fixa, em um sistema de processamento paralelo homogêneo, é uma reta do tipo *f(x) = x*, definida pela equação 5.

$$S(w,p) = \frac{T_s(w)}{T_s(w)/_p} = p \tag{5}$$

Na prática esse comportamento linear não ocorre. Segundo a chamada "Lei de Amdahl" [7], o *speedup* real na situação de carga fixa é limitado pela parcela serial original (intrínseca) do algoritmo. Sendo *r* esta porção serial original do algoritmo, o valor máximo obtido pelo *speedup* quando o número de processadores cresce indefinidamente ($p \to \infty$) e a carga é mantida fixa (w = cte), é dado por:

$$S_{max} = \frac{1}{r} \tag{6}$$

A Figura 4, mostra um gráfico do *speedup* versus o número de trabalhadores, em uma CAPP homogênea.

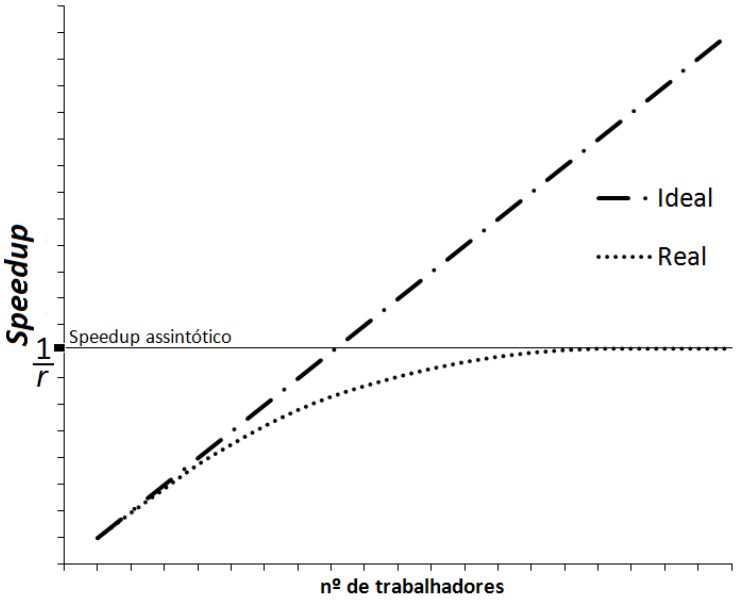

Figura 4 - Comportamento do *speedup* ideal e real em uma CAPP **homogênea com a carga fixa.**

As curvas no gráfico exemplificam o comportamento ideal e real do *speedup* se mantivermos a carga fixa e aumentarmos indefinidamente o número de trabalhadores. Repare que após certo número de trabalhadores, o *speedup* real atinge o valor máximo previsto pela de Lei de Amdahl.

A adição de processadores em uma execução paralela com carga fixa reduz progressivamente a parcela do tempo de execução que é gasta com as operações paralelizáveis do algoritmo. Porém o tempo gasto com a parcela serial original (operações não paralelizáveis) do algoritmo permanece fixo, tornado-se predominante no tempo de execução total que tende a atingir um valor mínimo. Este valor estabelece o *speedup* real máximo que é chamado por alguns autores de *speedup assintótico* [1].

2.4.2 Carga variável

A análise com carga fixa, objetivando atingir o menor tempo de execução possível, impõe um limite superior para os ganhos com a paralelização. Outra abordagem, proposta por Gustafson [9], prioriza trabalhar com uma CAPP, sempre extraindo o máximo que ela pode oferecer em termos de processamento. Nesse sentido a carga de trabalho é aumentada à medida que o número de processadores cresce.

Essa estratégia se justifica na prática, pois permite a resolução de problemas de maior complexidade ou mesmo com a obtenção de resultados mais precisos utilizando de maneira mais eficaz todo o recurso oferecido pela CAPP.

Ao fixarmos o número de processadores em uma CAPP e aumentarmos a carga da aplicação, observamos, em CAPP´s minimamente escaláveis, que o tempo gasto com as operações seriais no algoritmo ($T_o(w,p)$) cresce mais lentamente que o tempo de processamento paralelo ($T_{par}(w,p)$). À medida que $T_o(w,p)$ torna-se menos representativo em relação a $T_{par}(w,p)$, a execução paralela se aproxima da ideal, elevando o valor do *speedup*.

Quando o objetivo principal é a analisar o desempenho de uma CAPP à medida que o número de processadores aumenta, ou seja, analisar sua escalabilidade, o ponto de partida é garantir que a CAPP não esteja subutilizada. Por essa razão muitas das métricas de escalabilidade propostas na literatura após a formulação de Gustafson em 1988, passaram a adotar a variação de carga na avaliação da escalabilidade de CAPP´s.

2.5 Conclusões do capítulo

Neste capítulo foram definidos alguns conceitos básicos envolvidos no estudo de sistemas de processamento paralelo. O modelo de aplicação paralela BoT, adotado neste trabalho, foi apresentado, ressaltando as características que o tornam adequado para a utilização em SPPV. Foram discutidos os principais aspectos relacionados à paralelização de um algoritmo e como estes podem influenciar a escalabilidade de uma CAPP. O conceito de *speedup* foi apresentado, onde foram discutidas duas diferentes estratégias de análise: carga fixa e variável.

No próximo capítulo serão apresentadas as métricas de escalabilidade para sistemas homogêneos e heterogêneos propostas na literatura, muitas das quais utilizam os conceitos apresentados neste capítulo para sua formulação.

Capítulo 3

Métricas de Escalabilidade

Este capítulo apresenta uma visão geral das métricas de escalabilidade propostas na literatura, para sistemas homogêneos e heterogêneos. Algumas delas diferem bastante das demais, como a proposta por Srinivas e Janakiram [6], e foram mencionadas aqui com o propósito de demonstrar outras abordagens ao tema que, apesar de não terem obtido significativo reconhecimento ou citações, possuem sua importância pela busca de uma solução alternativa para a avaliação da escalabilidade.

Em sua maioria, as métricas seguem uma abordagem semelhante para analisar a escalabilidade de uma CAPP. Basicamente elas selecionam o que chamaremos de parâmetro de desempenho e analisam como esse parâmetro se comporta à medida que o tamanho da CAPP aumenta [19]. São exemplos de parâmetros de desempenho utilizados em algumas métricas: s*peedup*, eficiência paralela, velocidade unitária média e porção serial do algoritmo, entre outros.

Uma característica presente na maior parte das métricas é a variação na carga de trabalho do algoritmo de acordo com o número de trabalhadores, ou seja, quando o número de processadores no sistema aumenta, a carga de trabalho do algoritmo também aumenta. Baseado nesse comportamento, uma CAPP é considerada escalável se para um determinado aumento no número de nós de processamento (p), existe uma variação na carga de trabalho (w) que é capaz de manter o parâmetro de desempenho constante.

A variação necessária em *w* para manter o parâmetro de desempenho constante, oferece a noção do grau de escalabilidade da CAPP [18], ou seja, uma CAPP onde a carga de trabalho cresce linearmente com o número de processadores para manter o parâmetro de desempenho constante, é dita mais escalável do que outra onde a carga cresceria exponencialmente com o número de processadores para manter o desempenho.

Além disso, algumas métricas propõem ainda uma maneira de quantificar a escalabilidade de uma CAPP através de uma função *Ψ()*. Essa função considera a situação da CAPP antes e depois do aumento em *p*. Seus valores são variáveis entre 0 e 1, sendo 1 considerado a escalabilidade ideal.

3.1 Sistemas homogêneos

Como dito anteriormente, a maior parte das métricas de escalabilidade na literatura foram propostas para sistemas homogêneos, ou seja, onde a estrutura de comunicação, assim como todos os trabalhadores ou nós de processamento podem ser considerados idênticos. Esse tipo de métrica certamente é mais adequado para ambientes de processamento paralelo controlados, como máquinas multiprocessadas em laboratórios. Sua aplicação direta em ambientes heterogêneos, como no caso de SPPV reais, não é factível. Porém dado o maior número de propostas fundamentadas e experimentadas, o entendimento e a análise das métricas homogêneas é fundamental para o posterior estudo das métricas heterogêneas. Outro benefício desse estudo preliminar é a identificação de métricas que propiciam real aplicação prática, diferentemente de outras que acabam tornando-se específicas demais, dificultando seu uso em casos gerais.

3.1.1 Métrica do *speedup* assintótico

Esta métrica utiliza como parâmetro principal de análise o *speedup* assintótico [1]. Como definido na seção 2.4.1, o *speedup* assintótico (S_A) é o máximo valor de *speedup* atingido por uma CAPP, quando o número de processadores tende a infinito e a carga de trabalho (*w*) do algoritmo permanece fixa.

O cálculo do *speedup* assintótico é realizado a partir do tempo de execução serial (T_s) e o tempo de execução paralelo mínimo ($T_{par_{min}}$), que é o obtido aumentando-se o número de

trabalhadores na CAPP (com carga *w* fixa), até o momento em que o tempo total de execução paralelo (T_{par}) atinja um valor mínimo, ou seja, quando o tempo gasto com as operações não paralelizáveis do algoritmo torna-se predominante na execução. Sendo assim o *speedup* assintótico é calculado por:

$$S_A = \frac{T_s}{T_{par_{min}}} \tag{7}$$

Baseado nisso, a quantificação da escalabilidade é proposta pela equação 8 que envolve o *speedup* assintótico real ($S_{A_{real}}$) medido na CAPP analisada, e o *speedup* assintótico ideal ($S_{A_{ideal}}$) obtido pelo cálculo teórico da execução do mesmo algoritmo em uma *Exclusive Read Exclusive Write Parallel Random Access Machine* (EREW PRAM) uma máquina abstrata de memória compartilhada que idealiza aspectos como sincronização e comunicação e é utilizada para estimar o desempenho de algoritmos paralelos. Mais detalhes sobre esse tipo de máquina, adotada na definição de *speedup* assintótico ideal, podem ser encontrados na referência [20].

$$\Psi = \frac{S_{A_{real}}}{S_{A_{ideal}}} \tag{8}$$

O parâmetro *w* representa o valor fixo da carga de trabalho utilizada, que segundo os autores, deve ser grande suficiente para permitir o aparecimento do comportamento assintótico do *speedup*. Percebe-se então pela equação 8, que a escalabilidade da CAPP é avaliada pelo quanto seu *speedup* assintótico se aproxima do ideal, ou seja, se a CAPP atingir um *speedup* igual ao ideal sua escalabilidade seria unitária, que é a escalabilidade máxima esperada para uma CAPP.

Apesar de utilizar um conceito consagrado como o *speedup*, a análise de escalabilidade de uma CAPP nessa métrica é inteiramente baseada em carga fixa. Essa abordagem, quando comparada com as propostas feitas posteriormente na literatura, se mostra incompleta, pois não considera a influência da variação de carga, na análise da escalabilidade da CAPP. Também por não utilizar a idéia de variação de carga em sua formulação, não vemos nessa métrica a presença de um parâmetro de desempenho, como descrito no início desse capítulo.

Outro ponto a ser ressaltado é o próprio *speedup* assintótico. Enquanto o ideal requer para sua obtenção o conhecimento do algoritmo e seu comportamento em uma EREW PRAM, o real, medido na própria CAPP, deve ser tomado com um número suficientemente grande de

processadores, capaz de atingir a assíntota do *speedup*. Esse tipo de exigência dificulta, se não inviabiliza, a aplicação prática dessa métrica, já que pode ser difícil obter o número suficiente de máquinas para isso.

3.1.2 Métrica da Isoeficiência

A métrica da isoeficiência [2] é possivelmente a métrica de escalabilidade mais referenciada na literatura, servindo de base para outras [3,16,18]. Ela adota como parâmetro de desempenho a eficiência paralela (*E*), um conceito definido para sistemas de processamento paralelos homogêneos [22], que é calculada pela divisão do *speedup* pelo número de processadores.

$$E(w,p) = \frac{S(w,p)}{p} \tag{9}$$

A definição de *E* não estabelece que tipo de abordagem, carga fixa ou variável, deve ser utilizada na obtenção do *speedup*. Porém, cada uma delas age de maneira diferente no comportamento da eficiência de uma CAPP e quando analisadas juntamente com a variação no número de processadores facilitam o entendimento do conceito de escalabilidade utilizado na presente métrica.

Eficiência paralela com carga fixa e variação em *p*

Como discutido na seção 2.4.1, o *speedup* ideal em uma CAPP homogênea com carga de trabalho fixa é igual ao número de processadores utilizados (equação 5). Porém, na prática o *speedup* real cresce mais lentamente que *p*. Portanto, baseados na equação 9, podemos esperar que o comportamento da eficiência paralela seja decrescente com o aumento do número de processadores.

A Figura 5 mostra o comportamento real da eficiência paralela à medida que o número de trabalhadores aumenta em uma CAPP com carga fixa. Este gráfico foi obtido a partir de testes em ambientes homogêneos que serão mostrados e discutidos no Capítulo 4 (os detalhes como a carga aplicada e ambiente utilizado foram omitidos por não serem importantes neste momento, que visa apenas ilustrar o comportamento da eficiência paralela e aplicação da métrica).

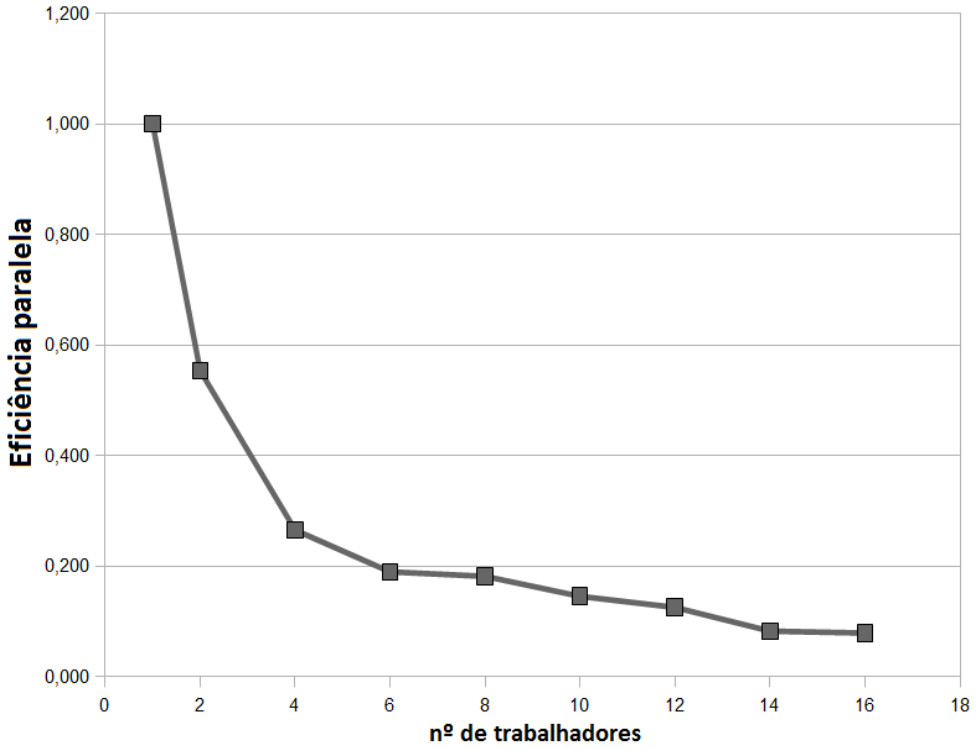

Figura 5 - Eficiência paralela em uma CAPP **homogênea com carga fixa.**

Eficiência paralela com carga variável e *p* fixo

Outro comportamento da eficiência paralela, que pode ser observado na prática, é o caso de fixarmos o número de processadores e variarmos a carga (*w*) do algoritmo. Neste caso, o aumento em *w*, provoca uma diminuição relativa na porção serial do algoritmo, pois faz com que mais tempo da execução seja empregado em trabalho paralelizável, o que provoca um aumento no *speedup* e, consequentemente, um aumento da eficiência paralela. Para exemplificar esse comportamento, temos a Figura 6 que traz um gráfico da eficiência paralela versus a carga *w*, (representada aqui pelo número de operações básicas realizadas em um algoritmo), para uma CAPP homogênea com um número fixo de processadores. Este gráfico também foi obtido nos testes do Capítulo 4, onde será mostrado com mais detalhes.

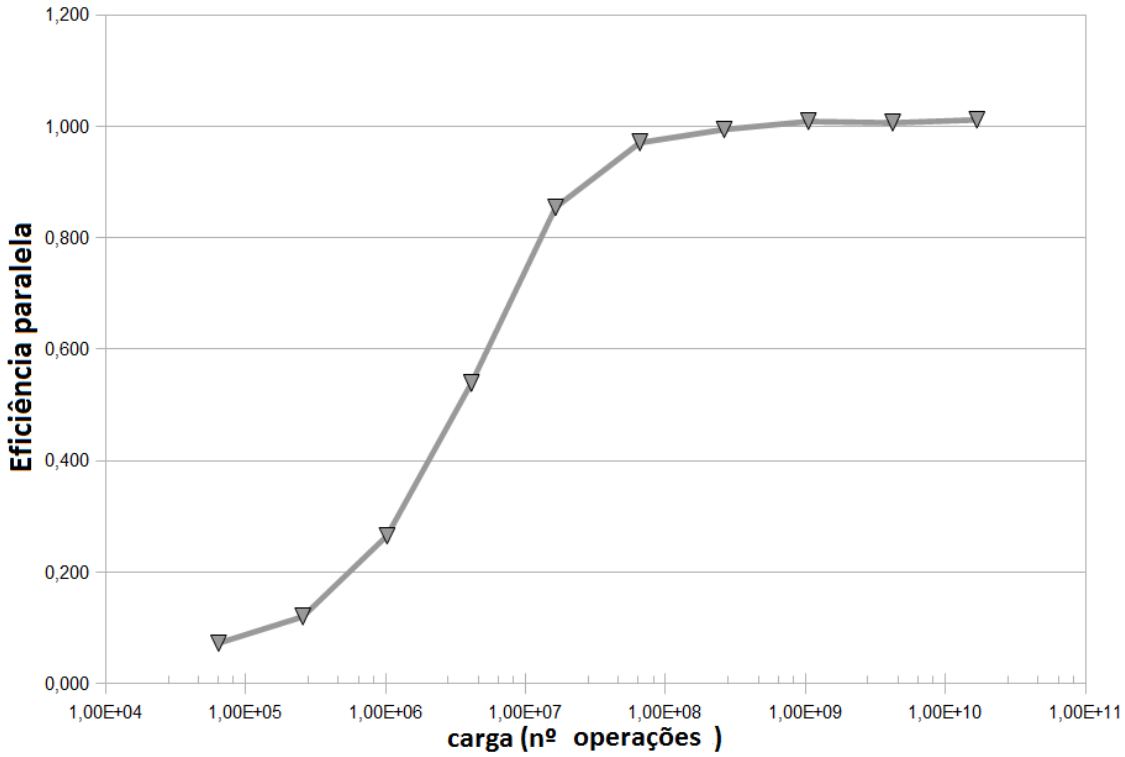

Figura 6 - Eficiência paralela em uma CAPP homogênea com o número fixo de processadores.

Como *w* e *p* têm efeitos inversos sobre a eficiência, deve ser possível manter a eficiência aproximadamente constante através de um aumento simultâneo em *w* e *p*.

Portanto uma CAPP é dita escalável se, para um determinado aumento em *p*, existe uma variação na carga *w* que mantenha a eficiência paralela constante. O grau de escalabilidade da CAPP é representado por uma *função isoeficiência* que determina a dependência entre *w* e *p* para manter a eficiência constante na CAPP. Essa função pode ser definida como:

$$w = K * f_o(w, p) \tag{10}$$

onde *K* é uma constante que depende apenas da eficiência paralela, e $f_o(w,p)$ é uma função que expressa matematicamente o comportamento do *overhead* de paralelização na CAPP.

Um ponto a ser ressaltado é que as métricas existentes não definem quais os valores mínimos de eficiência a serem mantidos constantes. Como consequência, uma CAPP que consiga manter constante uma baixa eficiência pode ser considerada escalável, sem distinção de outra que mantenha uma eficiência maior. Ou seja, é possível considerar qualquer CAPP escalável, desde

que sejam adotados níveis arbitrariamente baixos de eficiência, o que pode tornar o conceito inútil na prática.

3.1.3 Métrica da Latência

Esta métrica [3] foi baseada na teoria da isoeficiência, também utilizando como parâmetro de desempenho a eficiência paralela e seguindo os mesmos critérios para considerar uma CAPP escalável. A principal contribuição dos autores foi propor a quantificação da escalabilidade de uma CAPP utilizando o que eles chamaram de latência média, definida como o tempo médio gasto com *overhead* devido a cada processador do sistema.

A escalabilidade de uma CAPP, que aumenta de p para $p´$ processadores ($p<p´$) é obtida através das latências médias antes e depois da variação, como a seguir:

$$\Psi(p,p´) = \frac{L(w,p)}{L(w´,p´)} \tag{11}$$

A equação 11 é válida apenas na condição de isoeficiência, ou seja, o valor de $w´$ é tal que a eficiência paralela em p e $p´$ é a mesma.

A latência média, *L(w,p)*, é calculada pela equação 12.

$$L(w,p) = \frac{\sum_{i=1}^{p}(T_{par}(p) - T_i + L_i)}{p} \tag{12}$$

onde:

$T_{par}(p)$ – tempo de execução total da aplicação paralela com p processadores.

L_i – latência de *overhead* do processador i, inclui todo o tempo ocioso e o gasto com operações introduzidas por *overhead* na execução do processador i.

T_i – tempo de processamento gasto pelo processador i (inclui a latência L_i).

Apesar de sugerir uma forma de quantificar a escalabilidade, fato que não existia na métrica da isoeficiência, o cálculo da latência média não se mostra prático, visto que exige a medição da latência L_i de cada processador, o que em muitas plataformas significaria a introdução de pontos de medida no algoritmo que podem ser de difícil implementação e influenciar no próprio desempenho e latência.

3.1.4 Métrica da Isovelocidade

A seguinte métrica também obteve um reconhecimento diferenciado na literatura [4], sendo citada em vários outros trabalhos e se tornando base para uma extensão [17]. Foi batizada por seus autores como métrica da isovelocidade, pois utiliza como parâmetro de desempenho a chamada velocidade unitária média (V_{um}), que é dada pela equação 13.

$$V_{um}(w,p) = \frac{w/p}{T_{par}} \tag{13}$$

Uma CAPP é considerada escalável então, se a velocidade unitária média alcançada pelos nós de processamento permanece constante com o aumento do número de processadores e um aumento na carga de trabalho (w).

Essa métrica propõe o cálculo da escalabilidade $\Psi(p,p')$ de uma CAPP através da equação 14, que se baseia na variação de w que mantém V_{um} constante quando p muda para p'.

$$\Psi(p,p') = \frac{w/p}{w'/p'} \tag{14}$$

Baseados na idéia de desempenho assintótico [21], os autores da métrica sugerem que, durante a análise da escalabilidade, a V_{um} deva ser mantida constante em 50% do valor máximo atingido na CAPP, que representa um certo desperdício de capacidade em sistema.

Nesta métrica os autores consideram na contabilização da carga de trabalho w, apenas as operações de ponto flutuante (*flop*), o que pode ser um ponto negativo no caso do algoritmo utilizado executar outras operações que não envolvam pontos flutuantes e mesmo assim influenciem no desempenho da CAPP, como entrada e saída de dados por exemplo.

Por se basear em medidas de grandezas usuais em sistemas de processamento paralelo, como: carga do algoritmo, tempos de execução, número de processadores, essa métrica tem como vantagem o caráter prático. A seguir temos uma descrição passo a passo de como pode ser feita a aplicação prática dessa métrica em uma CAPP homogênea seguindo o sugerido pelos autores.

Para um determinado número de trabalhadores (p), varia-se a carga do algoritmo (w medida em *flop*) e mede-se o tempo de execução paralelo (T_{par}) correspondente. O cálculo da velocidade unitária média (V_{um}), dada em operações de ponto flutuantes por segundo (*flops*), é realizado

através da equação 13 e da origem ao gráfico da variação de V_{um} em função de w. O mesmo procedimento é repetido para várias quantidades de trabalhadores e todas as curvas podem ser colocadas no mesmo gráfico, como ilustra o gráfico da Figura 7.

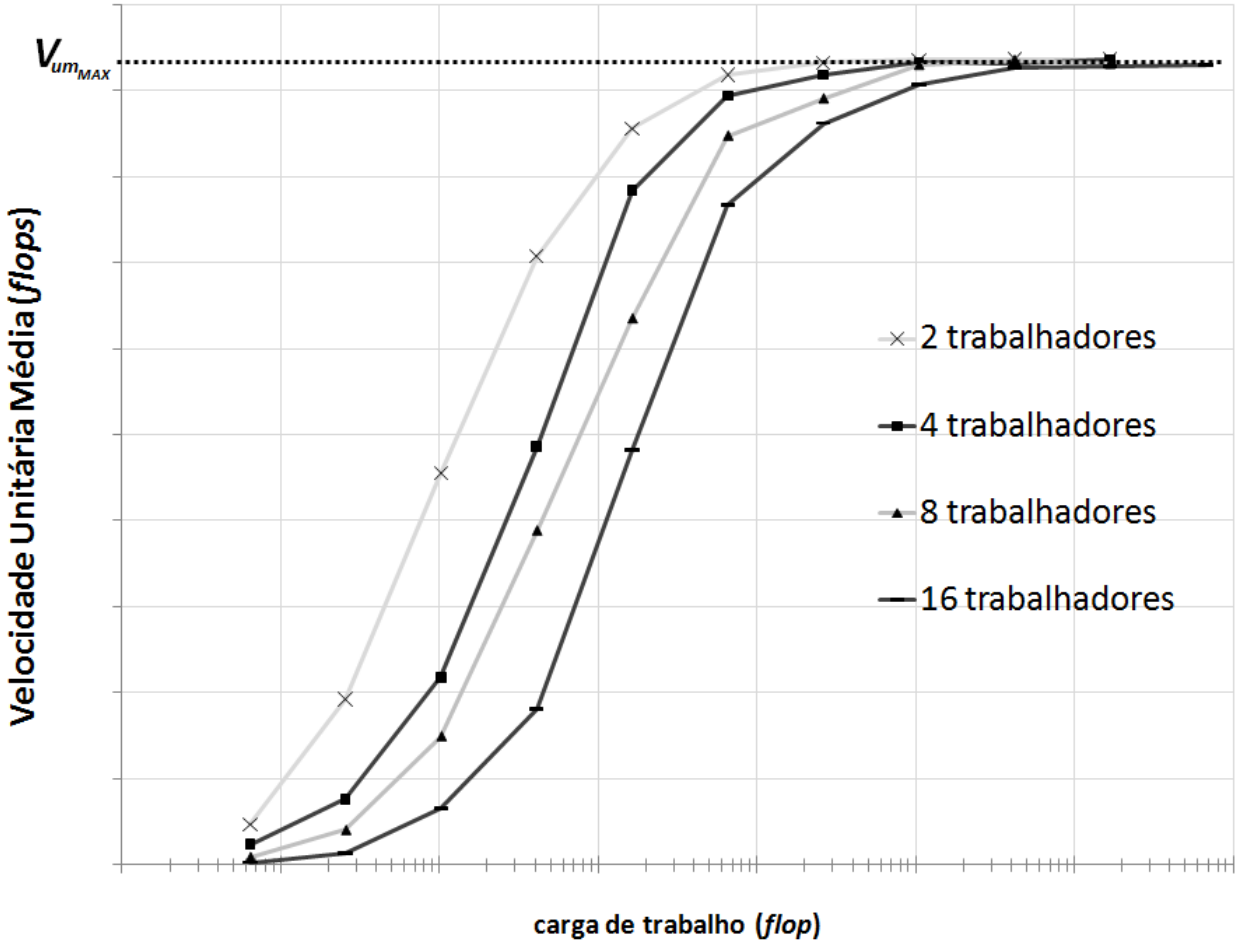

Figura 7 - Velocidade unitária média em função da variação da carga do algoritmo em uma CAPP homogênea.

Podemos perceber destacado na Figura 7 o valor da velocidade unitária média máxima atingida nesta CAPP ($V_{um_{MAX}}$). Seguindo as recomendações da métrica, estabelecemos como base a ser mantida constante na análise da escalabilidade o valor de 50% de $V_{um_{MAX}}$.

Por meio do gráfico da Figura 7 se obtém a variação de carga necessária para manter o parâmetro de desempenho constante, por exemplo, variando o número de trabalhadores de 2 para 16, como ilustra a Figura 8.

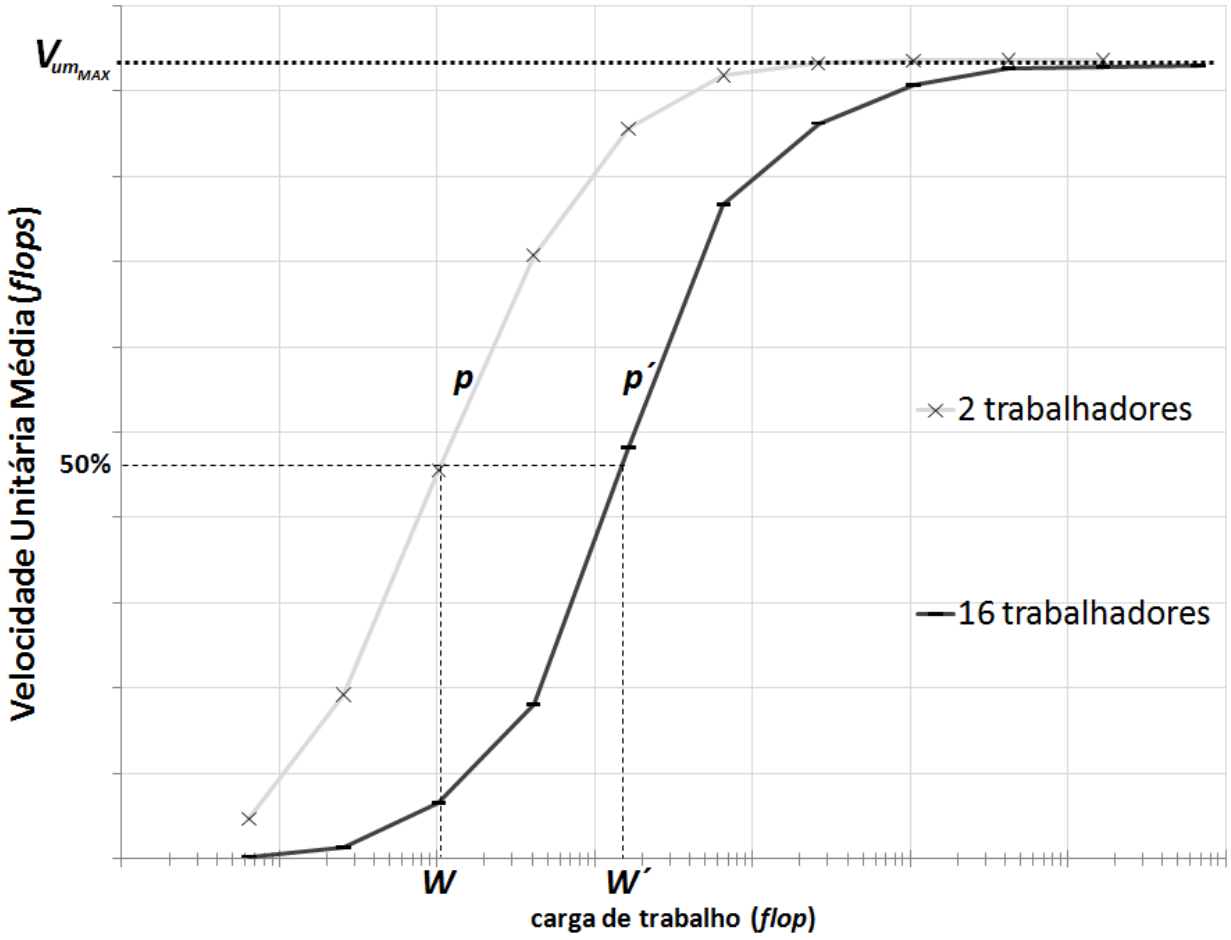

Figura 8 - Cálculo da variação de carga necessária, entre 2 e 16 trabalhadores, para manter a V_{um} constante em 50% da máxima.

Os valores de w e $w´$ obtidos do gráfico da Figura 8 são aplicados na equação 14, para determinar o valor da escalabilidade dessa CAPP quando varia de 2 para 16 trabalhadores. Considerando o exemplo usado sabemos que $p´ = 8p$. Se a variação necessária na carga, obtida no gráfico, resultasse em $w´ = 16w$. Então a escalabilidade seria:

$$\Psi(2,16) = \frac{w/p}{w´/p´} = \frac{w/p}{16w/8p} = 0{,}5$$

Diferentemente, se para a mesma variação no número de trabalhadores a variação na carga fosse igual a $w´ = 8w$, a escalabilidade da CAPP nessa condição de variação seria igual a 1, ou seja, a escalabilidade ideal. Percebe-se que o valor da escalabilidade está diretamente relacionado com a

proporcionalidade entre a variação de carga e a variação do número de processadores da CAPP em condição de isovelocidade.

O cálculo da escalabilidade pode ser realizado, como descrito na Figura 8, para todas as variações medidas de *p*.

3.1.5 Modelo alternativo de escalabilidade

Existe também uma visão bem diferente de escalabilidade [6], na qual é apresentada não propriamente uma métrica, mas sim, um modelo para caracterizar a escalabilidade de sistemas distribuídos e será mencionada aqui apenas com o propósito de demonstrar outras abordagens ao tema. Como argumentado pelos autores, este modelo permite comparar a escalabilidade de sistemas distribuídos similares. Segundo eles a escalabilidade deve ser uma função que depende dos seguintes fatores:

$$Escalabilidade = f(disp, cons, sinc, carga, falhas)$$

onde

disp: disponibilidade que pode ser quantificada pela razão do número de transações aceitas versus o número de submetidas ao sistema;

cons: consistência que é função do mecanismo de atualização e consistência entre as réplicas, número de vezes que é executado e granularidade da consistência que deve ser mantida entre as réplicas;

sinc: sincronização entre as réplicas, método utilizado e frequência com que é executado;

carga: carga de trabalho, o número de transações ou números de clientes;

falhas: a sequência e o número de falhas nas réplicas.

Observa-se pela descrição das características a serem consideradas na análise da escalabilidade que esse modelo se afasta das métricas anteriormente apresentadas, e por destinar--se a sistemas distribuídos de maneira geral não foca em características específicas de sistemas de processamento paralelo.

3.2 Sistemas Heterogêneos

A escalabilidade em sistemas heterogêneos, onde a velocidade de comunicação e o poder computacional de cada processador podem variar, precisa ser avaliada quando se trata de SPPV. Apesar de semelhantes aos desafios em métricas de escalabilidade para sistemas homogêneos, os desafios para caracterizar de maneira prática e consistente um sistema de processamento paralelo heterogêneo são maiores e, devido a isso, existe um número menor de propostas na literatura.

3.2.1 Métrica da Isoeficiência heterogênea

Essa métrica [15,18] é baseada na métrica homogênea da isoeficiência que foi estendida para se adequar a sistemas de processamento paralelo heterogêneos. A primeira definição feita pelos autores refere-se ao poder computacional do processador *i* (c_i),

$$c_i = \frac{w}{T_i}, \tag{15}$$

onde *w* é a quantidade de operações básicas executadas em uma tarefa específica realizada pelo processador *i* em um tempo T_i. Segundo os autores, na prática o valor de c_i pode ser obtido pela execução prévia do próprio algoritmo da CAPP em cada um dos processadores, com uma carga suficientemente grande para evitar erros provenientes do efeito de hierarquia de memória.

A partir disso é definido então o poder computacional total de um sistema heterogêneo como:

$$C(p) = \sum_{i=1}^{p} c_i \tag{16}$$

Essa métrica utiliza como parâmetro de desempenho uma redefinição da eficiência paralela convencional (equação 9), baseada nos tempos de execução paralelos, denominada eficiência heterogênea (E_H).

$$E_H(w,p) = \frac{T_{p_{ideal}}(w,p)}{T_{par}(w,p)} \tag{17}$$

onde $T_{par}(p)$ é o tempo de execução paralela real medido na CAPP e $T_{p_{ideal}}(p)$ é o tempo de execução paralela ideal, que é equivalente a execução do algoritmo em um único processador que possua o poder computacional igual ao poder computacional total do sistema. A equação 17 pode então ser rescrita utilizando essas definições da seguinte maneira.

$$E_H(w,p) = \frac{w}{T_{par}(w,p) * C(p)} \quad (18)$$

Sendo E_H o parâmetro de desempenho desta métrica, a CAPP é considerada escalável se, para um determinado aumento em *C(p)*, existe uma variação na carga *w* que mantenha a eficiência paralela heterogênea E_H constante. O aumento em *C(p)* pode ser conseguido adicionando novos processadores ou ainda substituindo algum processador por outro com maior poder computacional.

A obtenção de c_i, como sugerida pelos autores, é difícil do ponto de vista prático, pois exige que o próprio algoritmo, com uma carga específica, seja executado em cada um dos processadores, o que não é usual em SPPV.

Uma proposta feita por Kalinov [16] é essencialmente igual à isoeficiência heterogênea descrita aqui [15,18]. A contribuição adicional do autor concentra-se em deduzir teoricamente como seriam as funções isoeficiência heterogêneas para algumas situações de aplicações onde o balanceamento de carga entre os trabalhadores é imperfeito. Esses casos não serão tratados neste trabalho, pois não se aproximam do perfil das aplicações BoT que estamos focando aqui.

3.2.2 Métrica da Isovelo-eficiência

Esta métrica [17] é uma extensão da métrica de isovelocidade adaptada pelos próprios autores. Ela define inicialmente o conceito de velocidade de computação do processador *i* (v_i), que os autores obtêm através da execução de *benchmarks* tradicionais que expressem a capacidade do processador em *flops*. Assim, a velocidade ideal total do sistema ($V_{ideal}(p)$) pode ser obtida pela somatória de todos os v_i.

$$V_{ideal}(p) = \sum_{i=1}^{p} v_i \quad (19)$$

Para a determinação do parâmetro de desempenho a ser utilizado, os autores definem a velo-eficiência ($E_V(p)$), aqui traduzida do termo em inglês *speed-efficiency*, como:

$$E_V(w,p) = \frac{V_{real}(w,p)}{V_{ideal}(w,p)} \quad (20)$$

onde V_{real} é a velocidade real medida na CAPP, obtida dividindo-se a carga total do algoritmo, pelo tempo de execução paralelo total ($T_{par}(p)$). Assim podemos reescrever a velo-eficiência como:

$$E_V(w,p) = \frac{w/T_{par}(w,p)}{V_{ideal}(w,p)} = \frac{w}{T_{par}(w,p) * V_{ideal}(w,p)} \quad (21)$$

Se compararmos as equações (18) e (21), podemos perceber que suas versões finais são essencialmente as mesmas, diferindo apenas na maneira como os autores sugerem a obtenção de $C(p)$ e $V_{ideal}(p)$, respectivamente, onde ambos expressam a capacidade total do sistema.

Ainda na métrica da isovelo-eficiência é proposta uma equação para a quantificação da escalabilidade em uma CAPP heterogênea.

$$\Psi(V_{ideal}(p), V_{ideal}(p')) = \frac{w/V_{ideal}(p)}{w'/V_{ideal}(p')} \quad (22)$$

A equação 22 define a escalabilidade baseada na variação de carga necessária (w e w') para manter a velo-eficiência constante em uma CAPP que recebeu um aumento na velocidade ideal total de $V_{ideal}(p)$ para $V_{ideal}(p')$.

Para demonstrar a aplicação da métrica, os autores utilizam alguns exemplos clássicos de algoritmos sendo executados em um cluster heterogêneo. Contudo os resultados apresentados são pouco conclusivos e com algumas discrepâncias não explicadas. Por exemplo, a métrica é utilizada para comparar a escalabilidade do cluster executando dois algoritmos diferentes, porém em um algoritmo a velo-eficiência é mantida em 30% e no outro em 20% do valor máximo possível. Nenhuma explicação é dada para justificar por que as escalabilidades poderiam ser comparadas nesta situação e porque estes valores específicos foram escolhidos.

3.3 Conclusões do capítulo

Neste capítulo foram apresentadas e discutidas algumas das principais métricas de escalabilidade propostas na literatura para sistemas homogêneos, onde as métricas de Isoeficiência e Isovelocidade foram tratadas com maior detalhe. Foram descritas também as métricas de escalabilidade para sistemas heterogêneos existentes na literatura.

Apesar de muitas das características discutidas aqui terem servido de base teórica para a proposta da métrica de escalabilidade adequada para SPPV heterogêneos, que será apresentada no Capítulo 5, apenas o estudo das propostas na literatura não foi suficiente para prover as informações necessárias a essa formulação.

Com o objetivo de vivenciar a aplicação prática e identificar assim características desejáveis em uma métrica de escalabilidade, foram selecionadas duas das métricas mais consagradas na literatura, Isoeficiência e Isovelocidade, para serem avaliadas na caracterização da escalabilidade de CAPP´s homogêneas. O próximo capítulo apresenta em detalhes os resultados destes testes.

Capítulo 4

Testes com métricas de escalabilidade em sistemas homogêneos

Este capítulo descreve em detalhes os testes que foram realizados com duas métricas de escalabilidade para sistemas homogêneos apresentadas no Capítulo 3. Os testes tiveram como principais objetivos: identificar características importantes, do ponto de vista prático, na aplicação de uma métrica de escalabilidade; avaliar as métricas selecionadas na caracterização da escalabilidade de uma CAPP homogênea na situação de máximo desempenho; e investigar a influência da escolha do valor do parâmetro de desempenho (a ser mantido constante) no cálculo da escalabilidade, visando com isso reunir informações para auxiliar na proposição de uma nova métrica de escalabilidade para sistemas heterogêneos.

Foram escolhidas as métricas da isoeficiência [2] e da isovelocidade [4] que, apesar de serem destinadas apenas a sistemas homogêneos, quando comparadas às suas extensões em sistemas heterogêneos, apresentam um caráter mais prático que viabiliza sua real aplicação. Ainda por servirem de base para teorias mais recentes, as métricas escolhidas apresentam maior solidez na formulação, o que é refletido no considerável número de referências e citações existentes na literatura.

As métricas escolhidas foram aplicadas em dois ambientes distribuídos de processamento paralelo implementados em Java: JoiN [10,23] e JPVM (*Java Parallel Virtual Machine*) [11]. O

algoritmo utilizado em ambas as plataformas foi o cálculo do número Pi pelo método de Monte Carlo [12] que, como veremos a seguir, foi selecionado por possuir uma comunicação reduzida e assim tirar proveito do paralelismo que SPPV podem oferecer. A seguir temos uma descrição detalhada dos fatores envolvidos nos testes assim como a metodologia utilizada.

4.1 Cálculo do número Pi pelo método de Monte Carlo

A escolha de uma aplicação para compor as CAPP´s durante os testes buscou características que são altamente desejáveis em um algoritmo que visa utilizar SPPV extraindo o máximo de paralelismo que este pode oferecer. Além de possibilitar sua implementação seguindo o modelo BoT, existem outros aspectos que são buscados nesta escolha como: alto grau de paralelismo, *overhead* de paralelização reduzido e baixa comunicação, pois quando presentes, estes favorecem que a escalabilidade da CAPP seja minimamente afetada pelo algoritmo escolhido.

O cálculo do número Pi utilizando o método de Monte Carlo [12], que por simplicidade será referenciado a partir de agora neste trabalho como Pi-Monte Carlo, foi o algoritmo selecionado para compor as CAPP´s durante os testes e as seções a seguir discutem as características buscadas e como elas se apresentam no algoritmo escolhido.

4.1.1 Grau de paralelismo

Possuir um alto grau de paralelismo em um algoritmo significa dizer que a grande maioria do processamento utilizado nele pode ser dividido e processado paralelamente, ou seja, que o número de operações não paralelizáveis é pequeno em relação ao total de operações realizadas no algoritmo. A Figura 9 apresenta uma descrição básica dos passos realizados na versão serial do algoritmo Pi-Monte Carlo e permite avaliar a presença dessa característica.

```
1    - Inicializar o contador (counter = 0)
2    - Ler número de iterações (n);
3    - Executar n vezes
4         - Obter um número aleatório x entre 0 e 1;
5         - Obter um número aleatório y entre 0 e 1;
6         - Calcular z = x^2+y^2;
7         - Se (z<1) incrementar o valor de counter;
8    - Calcular pi = 4*counter / n;
```

Figura 9 - Versão serial do algoritmo Pi-Monte Carlo.

O *loop* de *n* iterações realizado entres as linhas 3 e 7 da Figura 9 gera coordenadas (*x*,*y*) aleatórias, onde (0,0) ≤ (*x*,*y*) ≤ (1,1), e incrementa um contador (*counter*) sempre que essa coordenada estiver contida dentro da área do quadrante de uma circunferência de raio 1 (linha 7).

Segundo o método de Monte Carlo, o valor de *n* deve ser suficientemente grande para permitir que a razão entre o número de pontos contidos no quadrante da circunferência unitária (*counter*) e o número total de pontos sorteados (*n*), seja proporcional à razão entre a área do quadrante de uma circunferência de raio unitário (π/4) e a área de um quadrado de lado unitário (1). Quanto maior o valor de *n*, maior a precisão do valor de Pi obtido ao final da aplicação. A equação 23 demonstra a proporcionalidade descrita acima:

$$\frac{counter}{n} = \frac{\pi/_4}{1} \tag{23}$$

que pode ser reescrita para obter o valor do número Pi.

$$\pi = \frac{4*counter}{n} \tag{24}$$

Pelo fato de *n* possuir um alto valor, o grande volume de cálculo realizado pela aplicação está concentrado no *loop* entre as linhas 3 e 7. Este é justamente o trecho da aplicação que é paralelizado conferindo ao algoritmo um alto grau de paralelismo e dando origem a tarefas que:

- recebem como dado de entrada o número de iterações do *loop*.
- retornam como resultado o número de coordenadas que estavam contidas no quadrante da circunferência de raio unitário.

A versão paralela do algoritmo Pi-Monte Carlo adiciona (*overhead* de paralelização) as operações de envio de dados (número de iterações a ser realizado em cada tarefa), recebimento e soma dos resultados contabilizando o valor de *counter*. Através do valor de *counter* e do número total de iterações executadas em todas as tarefas é possível calcular o valor de Pi através das equação 24 exatamente como é feito na versão serial.

A Figura 10 apresenta uma descrição das atividades realizadas na versão paralela do algoritmo Pi-Monte Carlo utilizada nos testes.

4.1.2 Comunicação

A baixa quantidade de dados enviados entre o coordenador e os trabalhadores também é uma característica favorável a SPPV. Por se tratarem de plataformas que muitas vezes utilizam meios de comunicação não garantidos, como a internet, aplicações que dependem do envio ou recebimento de grandes volumes de dados podem ter seu desempenho significativamente afetado por atrasos ou mesmo falhas na conexão. Para tirar proveito do máximo poder de processamento oferecido por um grande sistema de processamento paralelo, é desejável que o tempo gasto com comunicação seja desprezível em relação ao tempo gasto com processamento.

```
No coordenador
1    - Ler número total de iterações (n);
2    - Ler número de tarefas (t);
3    - Criar t tarefas contendo como entrada o valor (n/t);
4    - Enviar as t tarefas;
```

```
Nos trabalhadores
1    - Ler número de iterações (n/t);
2    - Inicializa o contador (counter = 0)
3    - Executar (n/t) vezes
4        - Gerar um número x aleatório entre 0 e 1;
5        - Gerar um número y aleatório entre 0 e 1;
6        - Calcular z = x^2+y^2;
7        - Se (z<1) incrementar o valor de counter;
8    - Enviar o valor final de counter para o coordenador;
```

```
No coordenador
1    - Receber os t resultados das tarefas;
2    - Iniciar o contador global (global_counter = 0);
3    - Somar os resultados das t tarefas e armazenar em global_counter;
4    - Calcular pi = 4*global_counter / n;
```

Figura 10 - Versão paralela do algoritmo Pi- Monte Carlo.

Assim a comunicação reduzida em uma aplicação paralela favorece níveis mais altos de escalabilidade, pois facilita o aproveitamento de cada novo processador introduzido por não sofrer grande impacto no aumento do fluxo de dados.

Como pode ser visto na versão paralela do algoritmo Pi-Monte Carlo, mostrada na Figura 10, o fato dos dados (referentes ao algoritmo) trocados entre a máquina coordenadora e os trabalhadores resumirem-se a um número apenas (nº iterações do *loop* e resultado da tarefa), é outro fator que potencializa a obtenção do máximo desempenho de SPPV. Outras mensagens de

controle que possam ser trocadas dizem respeito exclusivamente à administração da execução realizada pela plataforma paralela e independem da aplicação utilizada.

4.1.3 Carga de trabalho

O parâmetro escolhido para representar a carga de trabalho (*w*) do algoritmo de Pi-Monte Carlo foi o valor de *n* presente na Figura 9, que significa o número de iterações executadas no *loop*. Pelo fato de *n* ter um alto valor, o número total de operações realizadas em *loop* é aproximadamente igual ao total de operações realizadas na aplicação. Como o número total de operações do algoritmo é diretamente proporcional ao valor de *n*, este pode ser utilizado como parâmetro de carga de trabalho, segundo o que foi discutido no Capítulo 2.

4.1.4 *Overhead* de paralelização

Analisando a versão paralela do algoritmo mostrada na Figura 10, verifica-se outra característica importante na aplicação escolhida. As operações introduzidas pela paralelização não dependem da carga de trabalho do algoritmo. Como mencionado, o dado enviado para cada tarefa é apenas um número inteiro que indica quantas iterações devem ser executadas na tarefa. Para variarmos a carga da aplicação basta aumentar o valor deste número. Repare que a quantidade de dados, enviada e recebida para cada processador, continua a mesma, independente da carga. Portanto o *overhead* de paralelização neste caso não está ligado à carga de trabalho da aplicação, mas sim ao número de tarefas criadas, o que reflete diretamente no número de dados trocados entre os trabalhadores e o coordenador. Para um número fixo de tarefas, à medida que a carga da aplicação aumenta, o tempo total de processamento paralelo também aumenta, porém o tempo gasto com operações de *overhead* permanece fixo, o que eleva o desempenho da execução paralela.

4.1.5 Limite superior da escalabilidade da plataforma paralela

Por possuir um *overhead* de paralelização independente da variação de carga e pela baixa comunicação entre as tarefas, o que minimiza as influências do meio externo, a utilização do algoritmo Pi-Monte Carlo permite que o desempenho da CAPP seja avaliado trabalhando próximo ao limite superior de sua escalabilidade. Como esta aplicação tem características bem

próximas das ideais, podemos inferir que estamos trabalhando próximos ao limite superior de escalabilidade que qualquer CAPP possa ter com as plataformas paralelas escolhidas.

4.2 Plataformas paralelas

Duas plataformas que fornecem suporte a processamento paralelo distribuído foram selecionadas como instrumento dos testes. A seguir temos uma descrição de suas características principais assim como os passos básicos necessários para a utilização de cada uma delas.

4.2.1 A plataforma JoiN

A plataforma de software JoiN, ou simplesmente JoiN, vem sendo desenvolvida no Laboratório de Computação e Automação Industrial da Faculdade de Engenharia Elétrica e de Computação da UNICAMP. Essa plataforma se propõe a ser um sistema de processamento paralelo virtual aproveitando de maneira eficiente o grande poder computacional que pode ser proporcionado por muitos computadores pessoais conectados à internet.

Desenvolvido inteiramente em linguagem Java e dotado de uma estrutura baseada em componentes, JoiN tem como principal característica sua grande portabilidade, já que pode ser executado em qualquer computador que disponha de uma implementação da máquina virtual Java.

A plataforma JoiN possui quatro componentes distintos que definem sua estrutura lógica: servidor, coordenador, trabalhador e Jack. A seguir temos uma breve descrição da função que cada componente ocupa na plataforma JoiN.

Jack (*JoiN Administration and Configuration Kit*): módulo de administração que auxilia a configuração e operação da plataforma por parte dos administradores.

Servidor: gerencia as operações de mais alto nível e recebe novos integrantes para a plataforma.

Coordenador: gerencia as atividades dentro de cada grupo de trabalhadores da plataforma, prepara as tarefas, envia-as, recebe os resultados enviados pelos trabalhadores, processa o resultado final.

Trabalhador: responsável pela execução das tarefas da aplicação, o chamado processamento útil.

A Figura 11 ilustra a organização dos componentes na plataforma JoiN.

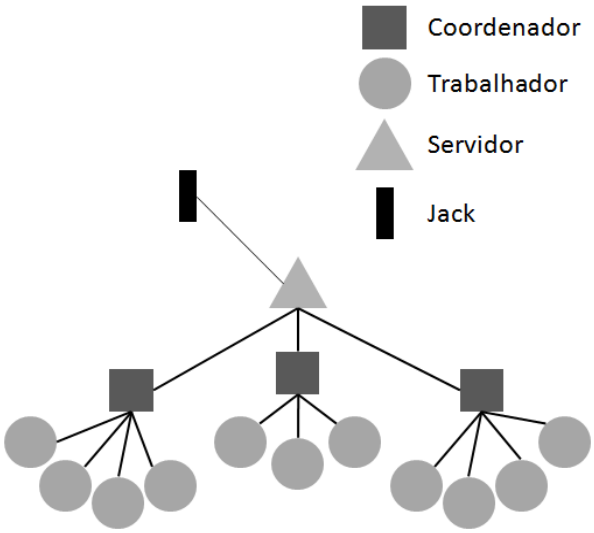

Figura 11 - Componentes da plataforma JoiN.

A plataforma JoiN foi originalmente projetada para ambientes heterogêneos, porém na fase de análise das métricas de escalabilidade homogêneas os testes realizados foram efetuados utilizando máquinas iguais formando um único grupo interligado por uma única rede.

Qualquer aplicação em JoiN precisa ter, além do algoritmo paralelo implementado em linguagem Java, uma especificação de como é o relacionamento entre as tarefas. Essa descrição é feita em um arquivo utilizando uma linguagem de especificação de aplicações paralelas (PASL – *Parallel Application Specification Language*), que determina, por exemplo, em quantas tarefas o problema deve ser dividido antes de ser distribuído.

A utilização do sistema JoiN segue basicamente o seguinte roteiro.

Um computador executando o componente servidor recebe a conexão de um computador preparado para a função de coordenador. A partir deste momento, todos os computadores que se conectam ao servidor, assumem a função de trabalhadores e são direcionados a algum dos coordenadores já existentes. Através do componente Jack são realizadas a instalação e inicialização da aplicação, autorizando o coordenador a iniciar a distribuição das tarefas.

Para minimizar o tempo de ociosidade em cada trabalhador (por espera de E/S), JoiN executa três *threads* simultâneas em cada um, ou seja, cada trabalhador pode receber e executar até três tarefas simultaneamente.

O usuário tem a opção de selecionar previamente, através de um arquivo de configuração, como deve ser a distribuição inicial das tarefas na execução. Ele pode optar por um número fixo de tarefas por trabalhador, ou ainda, pode utilizar integralmente o sistema de escalonamento GSTR (*Generational Scheduler with Task Replication*) [25]. Este segundo, basicamente, faz uma distribuição inicial de 50% do total de tarefas entre os trabalhadores, levando em consideração o poder computacional relativo entre eles, que é avaliado através de algoritmos de benchmark no momento que os trabalhadores entram na plataforma. Após a distribuição inicial, as tarefas remanescentes são enviadas aos trabalhadores à medida que estes retornam o resultado de suas tarefas e ficam ociosos.

4.2.2 JPVM

A biblioteca JPVM (*Java Parallel Virtual Machine*) [11] é um sistema de software composto por um conjunto de classes em Java que utilizam fundamentalmente a passagem de mensagens para prover suporte à programação paralela, seja em ambientes homogêneos ou heterogêneos. Por ser baseado em Java, também oferece as vantagens de portabilidade associadas a essa linguagem. A biblioteca suporta uma interface similar à provida em C pelo sistema PVM (*Parallel Virtual Machine*) [24], contando com algumas modificações na sintaxe e semântica que a adequam ao estilo de programação orientado a objeto presente na linguagem Java.

Por se tratar de apenas uma biblioteca, muitos dos recursos presentes em outras plataformas de processamento paralelo (como JoiN) não são oferecidos. Ficam a cargo do programador da aplicação funções como divisão, distribuição e agrupamento das tarefas na aplicação.

A utilização do sistema segue basicamente os seguintes passos.

Um processo trabalhador (*daemon*) do JPVM é inicializado em cada nó de processamento do sistema. Um dos nós de processamento é escolhido para coordenar os outros e nesse nó se executa uma classe chamada *Console*. Essa classe é utilizada para registrar, um a um, os nós participantes do sistema, entrando com informações de endereço na rede e porta de acesso. O

Console permite ainda algumas verificações básicas de monitoramento, como listar os nós presentes no sistema e o número de tarefas alocadas a cada um deles.

Após o registro pode-se executar a aplicação a partir do nó coordenador. A aplicação deve estar codificada em Java e basicamente pode fazer uso de métodos que geram réplicas de uma determinada classe (utilizada como tarefa para os trabalhadores), enviam e recebem dados. A biblioteca não impõe limites quanto à topologia de comunicação entre os trabalhadores, ficando também a cargo do programador da aplicação definir o fluxo correto dos dados durante a execução.

4.3 Ambiente de testes e metodologia

Os testes apresentados aqui foram realizados em um ambiente homogêneo na Faculdade de Engenharia Elétrica e de Computação da UNICAMP, onde as máquinas estavam dedicadas exclusivamente a este teste. Foi utilizado um conjunto de até 16 máquinas como trabalhadores, todas elas com a mesma configuração de *software* e *hardware*: *Pentium* 4, 1.8GHz, 1Gb RAM, Sistema Operacional Linux Ubuntu v9.04 e rede de 100 Mbps. Os carimbos de tempo apresentados na Figura 1 foram obtidos através de instruções oferecidas pela linguagem Java, conforme é mostrado nos códigos fontes listados no Apêndice A.

4.3.1 Configuração da aplicação

Como mencionado anteriormente a aplicação utilizada nos testes foi a de Pi-Monte Carlo. O mesmo algoritmo foi implementado em três versões, todas em Java para evitar as divergências de desempenho devido a linguagens de programação. A primeira delas foi uma versão serial do algoritmo executada diretamente em um dos computadores do grupo de teste. Essa primeira versão serve como base de comparação para todas as execuções paralelas, pois é a partir dela que obtemos o tempo serial T_s. As outras duas versões seguiram rigorosamente o mesmo padrão de paralelização, porém foram adequadas respectivamente aos ambientes JoiN e JPVM. Os códigos Java das classes principais das três versões da aplicação estão apresentados no Apêndice A.

Para cada versão da aplicação (serial, JoiN e JPVM) foram criadas 11 instâncias com valores crescentes de carga de trabalho. A Tabela 1 apresenta o número de iterações executadas em cada instância da aplicação.

Tabela 1 - Carga total de trabalho atribuída a cada instância da aplicação Pi-Monte Carlo nos testes com sistemas homogêneos.

Instância	Carga total de trabalho (nº iterações)
00	64.10^3
01	256.10^3
02	1024.10^3
03	4096.10^3
04	16384.10^3
05	65536.10^3
06	262.10^6
07	1049.10^6
08	4194.10^6
09	16777.10^6
10	67109.10^6

Para garantir o balanceamento de carga entre os trabalhadores e consequentemente obter o melhor desempenho nas execuções paralelas, a carga total de cada instância da aplicação foi dividida em tarefas iguais e o número de tarefas criadas foi determinado pelo número de trabalhadores envolvidos na execução. Seguindo a característica de *threads* simultâneas, oferecida na plataforma JoiN, e para manter a equivalência dos testes com JPVM, em ambas as plataformas testadas cada trabalhador recebeu exatamente 3 tarefas por execução. Por exemplo, para executar a instância 03 da aplicação em 10 trabalhadores, cada trabalhador recebeu 3 tarefas de um total de 30, onde cada tarefa realizou $136{,}5.10^3$ iterações ($4096.10^3/30$).

4.3.2 Execução dos testes

Todas as instâncias de carga foram aplicadas na versão serial do algoritmo, que foi executada em uma máquina participante do grupo de teste. É importante destacar que essa é uma etapa custosa das métricas e sabemos que nem sempre a execução serial é viável na prática, pois quando *w* atinge valores elevados o tempo de execução pode ser muito extenso.

Quatro grupos de trabalhadores foram formados para a utilização nos dois ambientes JoiN e JPVM contendo 2, 4, 8, 16 computadores trabalhadores. As instâncias da aplicação foram executadas em cada grupo de trabalhadores, sendo que para aumentar a confiabilidade dos resultados medidos, cada execução foi repetida cinco vezes, fornecendo assim informações para o cálculo de médias que foram utilizadas na composição dos gráficos de resultados. Apesar das

máquinas estarem dedicadas exclusivamente aos testes, foram calculadas médias a fim de minimizar o impacto de operações eventuais do sistema operacional ou algum tráfego extra na rede. Todos os dados obtidos nesta fase de testes estão contidos no Apêndice B.

4.4 Resultados

A partir dos tempos medidos foi possível analisar o desempenho das CAPP´s homogêneas com JoiN e JPVM sob a ótica das métricas de isovelocidade e isoeficiência, caracterizando assim escalabilidade de ambas as CAPP´s submetidas às mesmas condições de execução.

4.4.1 Análise da isovelocidade

Sob o ponto de vista da métrica da isovelocidade, estamos interessados em como a velocidade unitária média (V_{um}) definida na equação 13 (página 24), que é o parâmetro de desempenho dessa métrica, se comporta em cada CAPP à medida que ocorre variação na carga de trabalho *w* do algoritmo.

A partir dos tempos medidos na versão serial do algoritmo, mostrados no Apêndice B, é possível traçar o gráfico da velocidade unitária média em função da variação de carga do algoritmo Pi-Monte Carlo como mostrado na Figura 12.

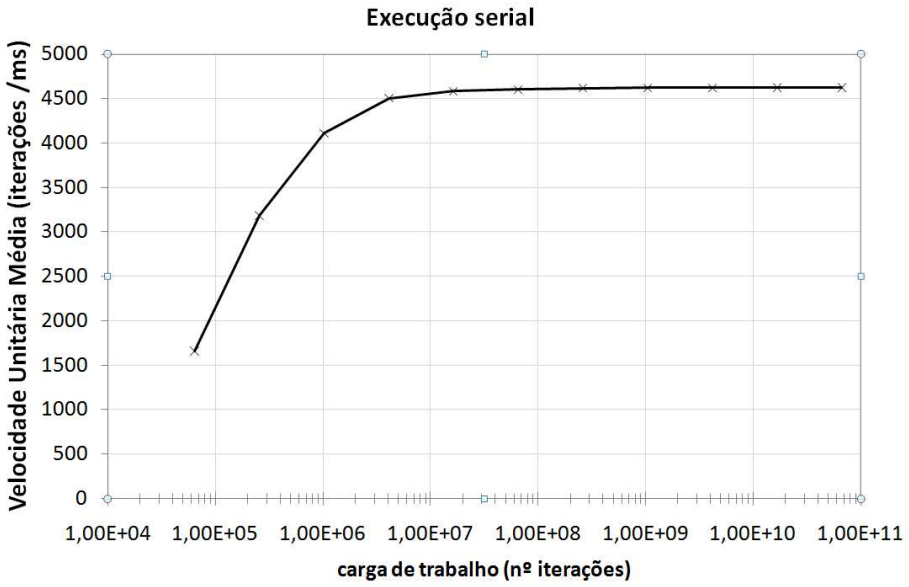

Figura 12 - Velocidade unitária média em função da carga do algoritmo Pi-Monte Carlo em sua versão serial.

A velocidade unitária máxima atingida na execução serial foi aproximadamente 4600 iterações/ms. Esse valor serve como referência para as execuções paralelas do algoritmo nesse ambiente homogêneo, pois define a máxima velocidade que se espera obter de uma máquina deste grupo na execução desta aplicação. Uma vez que as máquinas utilizadas possuem igual capacidade de processamento, não é esperado que o valor de saturação da velocidade unitária média na execução serial seja superado nas execuções paralelas.

Os gráficos da Figura 13 apresentam o comportamento das CAPP´s homogêneas com JoiN e JPVM, respectivamente, à medida que a carga de trabalho do algoritmo Pi-Monte Carlo é aumentada, onde cada curva nos gráficos representa o comportamento relativo a um grupo de trabalhadores.

Apesar de ambas as CAPP´s executarem o mesmo algoritmo, submetido à mesma variação de carga e utilizando o mesmo conjunto de máquinas, a métrica permite perceber diferenças entre as plataformas utilizadas. Isso propicia que a métrica, além de analisar o desempenho de uma CAPP, possa ser utilizada com instrumento de comparação de CAPP´s em situações equivalentes, como neste caso.

Os valores de V_{um} máximos atingidos nas duas plataformas foram aproximadamente 4600 iterações/ms, como era esperado através da execução da versão serial do algoritmo. Observa-se também que a CAPP com JoiN atingiu mais rapidamente valores elevados de velocidade, em relação à CAPP com JPVM, o que significa que JPVM necessitou de mais carga para mostrar todo o seu desempenho.

Como mostrado na seção 3.1.4, podemos utilizar os gráficos da Figura 13 para obter os valores de carga necessários ao cálculo das escalabilidades entre cada variação no número de trabalhadores, segundo a equação 14 (página 24).

Com a finalidade de avaliar como a região da curva pode afetar o cálculo da escalabilidade, foram considerados três valores base de V_{um}. Os valores escolhidos foram 1380, 2760 e 4140 iterações/ms, que representam respectivamente, 30%, 60% e 90% da V_{um} máxima atingida na CAPP (4600 iterações/ms). A Figura 14 ilustra o processo de obtenção dos valores de carga nas três regiões da curva de V_{um}.

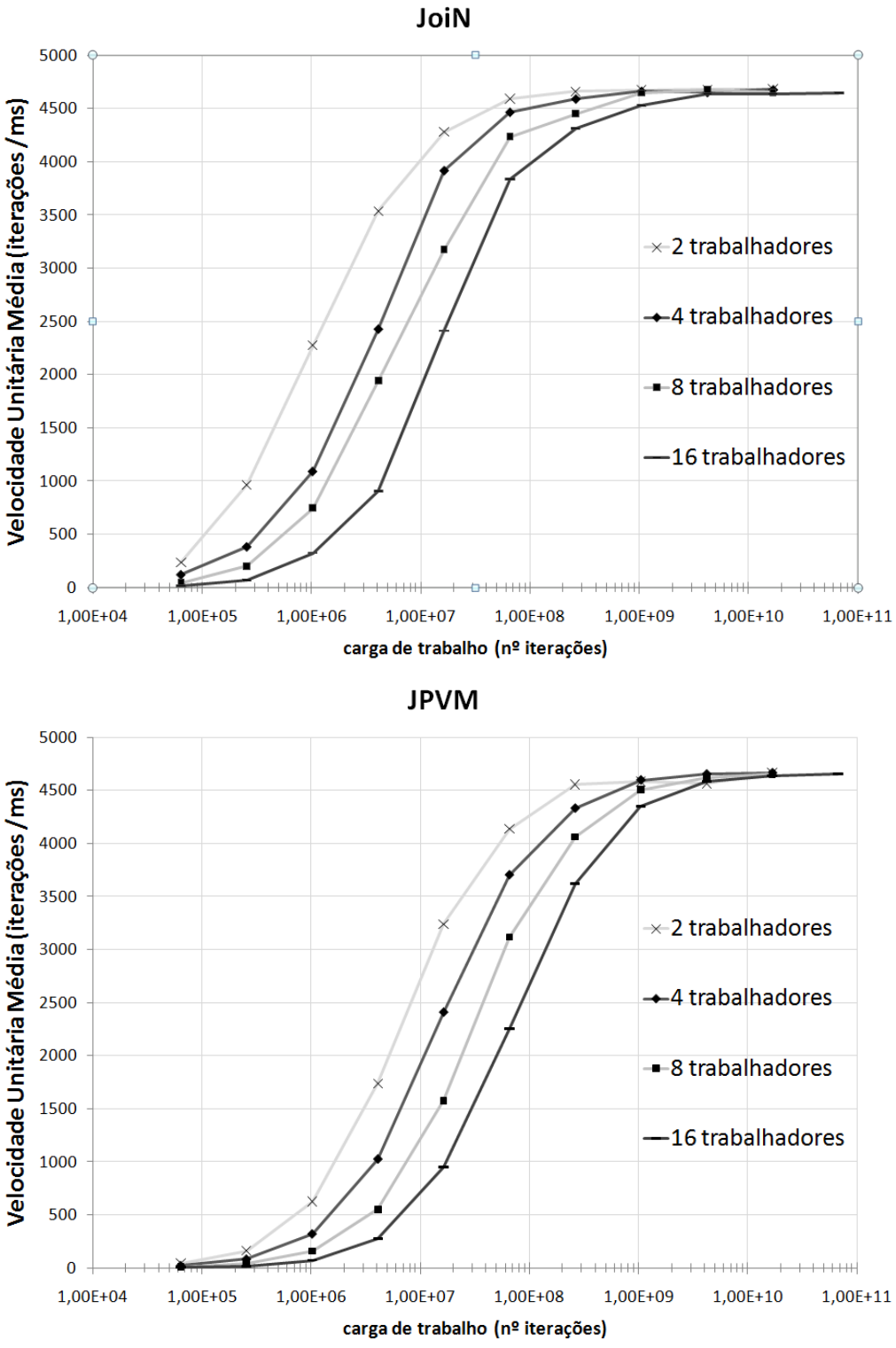

Figura 13 - Velocidade unitária média em função da carga do algoritmo Pi-Monte Carlo, em JoiN e JPVM.

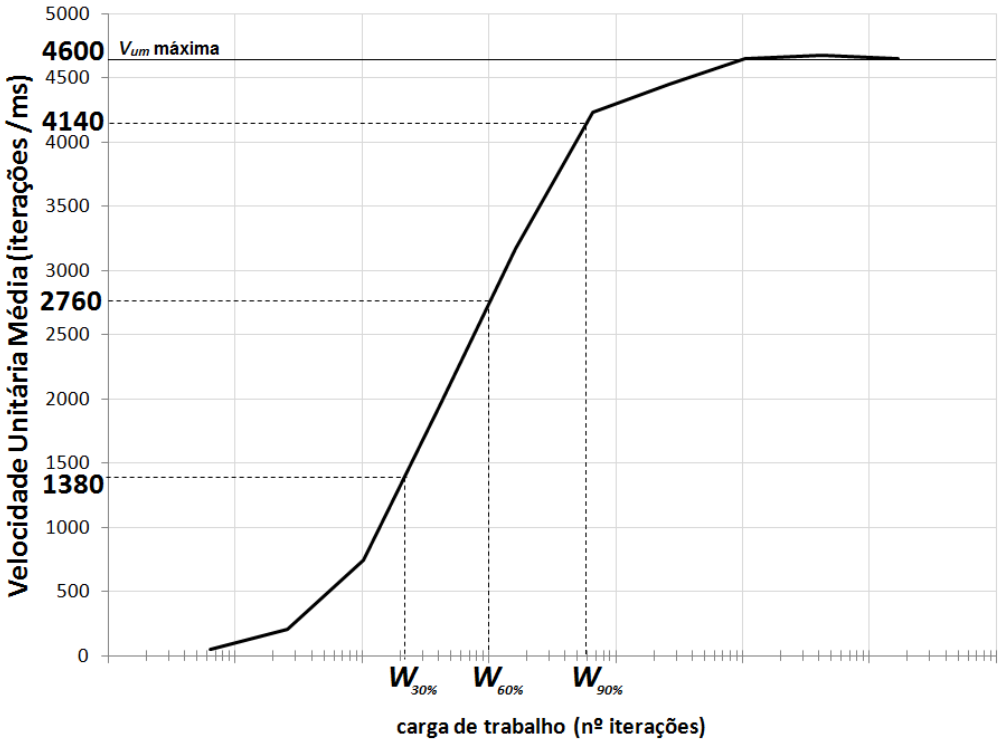

Figura 14 – Obtenção das cargas de trabalho necessárias para atingir 30%, 60% e 90% da V_{um} máxima.

Os valores aproximados das cargas $W_{30\%}$, $W_{60\%}$ e $W_{90\%}$ de cada grupo de trabalhadores foram obtidos através da interpolação logarítmica dos pontos de cada curva. No Apêndice C são mostrados os gráficos das curvas resultantes da interpolação mencionada.

A Tabela 2 apresenta, para as CAPP´s com JoiN e JPVM, os valores de cargas obtidos para cada grupo de trabalhadores, referentes aos três valores de V_{um} considerados, onde $W_{30\%}$, $W_{60\%}$ e $W_{90\%}$ equivalem às cargas de trabalho necessárias para atingir 1380, 2760 e 4140 iterações/ms, respectivamente.

Segundo o critério de escalabilidade adotado pela métrica da isovelocidade, neste ponto ambas as CAPP´s já podem ser consideradas escaláveis, pois observando os valores da Tabela 2, podemos perceber que existem valores de carga capazes de manter constante uma V_{um} dentro das variações de trabalhadores utilizadas. Para que a CAPP analisada fosse considerada não escalável teríamos como resultado dos testes que, por maior que possa ser, não existe valor de carga capaz de manter o parâmetro de desempenho constante, o que equivale a dizer que o valor de $w´$ tende a infinito e pela equação 14 (página 24) a escalabilidade tende a zero.

Tabela 2 - Cargas de trabalho para o cálculo da escalabilidade pela métrica de isovelocidade nas CAPP´s com JoiN e JPVM.

Isovelocidade			
JoiN			
Nº trabalhadores	Carga de trabalho (nº iterações)		
	$W_{30\%}$	$W_{60\%}$	$W_{90\%}$
2	369783	2038287	11235280
4	1064482	5512234	28544151
8	2163936	11137299	57321218
16	4630275	25693603	142574956
JPVM			
Nº trabalhadores	Carga de trabalho (nº iterações)		
	$W_{30\%}$	$W_{60\%}$	$W_{90\%}$
2	2427797	11898866	58317486
4	5205129	28718578	158450760
8	10360570	65296600	411526206
16	30064952	158922715	840062193

A partir da equação 14 (página 24) e dos valores de carga apresentados na Tabela 2, foram calculadas as escalabilidades das CAPP´s com JoiN e JPVM, referentes as variações no número de trabalhadores, para os três valores selecionados de V_{um}. Os valores obtidos estão na Tabela 3, onde as escalabilidades são mostradas nos cruzamentos linha e coluna, sendo o valor da linha o número de trabalhadores antes (p) e o valor da coluna o número de trabalhadores depois ($p´$).

As escalabilidades presentes na Tabela 3 refletem diretamente como deve ser a variação de carga, proporcionalmente ao aumento no número de processadores, necessária para manter a V_{um} constante. Quanto maior a equivalência entre a variação na carga e no número de processadores, mais próxima a escalabilidade está do seu valor máximo (unitário).

Apesar de ser a máxima, a escalabilidade unitária é a que intuitivamente se esperaria de uma CAPP. É natural imaginar que se dobrarmos o número de processadores, mantendo a carga fixa, o tempo de execução deveria cair à metade. Porém sabemos que isso raramente ocorre e o valor de escalabilidade calculado aqui representa o quão próxima a CAPP está de atingir o comportamento ideal esperado. Uma escalabilidade próxima de zero indica ser extremamente difícil mantermos constante o parâmetro de desempenho escolhido, mesmo se aumentarmos significativamente o volume de carga da aplicação.

Se analisarmos a tabela de cada CAPP comparando a mesma escalabilidade (p,$p´$) para valores de V_{um} diferentes, podemos observar que à medida que escolhemos um valor de V_{um} a ser mantido constante, as escalabilidades em ambas as CAPP´s variam significativamente. Esse fato demonstra que o valor de V_{um} escolhido como referência exerce influência real sobre as escalabilidades calculadas e, no caso da métrica ser utilizada com instrumento de comparação, os valores de V_{um} adotados como base devem então ser necessariamente os mesmos para ambas as CAPP´s comparadas, diferentemente das análises apresentadas por Chen et. al. [17], que comparam CAPP´s utilizando para cada um valor de referência diferente.

Tabela 3 - Valores de escalabilidade das CAPP´s com JoiN e JPVM, para diferentes valores de V_{um}

Isovelocidade							
Escalabilidade JoiN - $\Psi(p,p´)$				Escalabilidade JPVM - $\Psi(p,p´)$			
V_{um} = 1380 iterações/ms (30%)				V_{um} = 1380 iterações/ms (30%)			
p \ p´	**4**	**8**	**16**	**p \ p´**	**4**	**8**	**16**
2	0,69	0,68	0,64	**2**	0,93	0,94	0,65
4		0,98	0,92	**4**		1,00	0,69
8			0,93	**8**			0,69
V_{um} = 2760 iterações/ms (60%)				V_{um} = 2760 iterações/ms (60%)			
p \ p´	**4**	**8**	**16**	**p \ p´**	**4**	**8**	**16**
2	0,74	0,73	0,63	**2**	0,83	0,73	0,60
4		0,99	0,86	**4**		0,88	0,72
8			0,87	**8**			0,82
V_{um} = 4140 iterações/ms (90%)				V_{um} = 4140 iterações/ms (90%)			
p \ p´	**4**	**8**	**16**	**p \ p´**	**4**	**8**	**16**
2	0,79	0,78	0,63	**2**	0,74	0,57	0,56
4		1,00	0,80	**4**		0,77	0,75
8			0,80	**8**			0,98

Repare que para variações a CAPP utilizada apresenta

A métrica da isovelocidade permite então comparar a escalabilidade das CAPP´s com JoiN e JPVM, tomando sempre como base os mesmos valores de V_{um}. Neste caso, podemos perceber que a CAPP com JPVM mostrou-se mais escalável apenas na situação de menor valor de V_{um} (1380 iterações/ms). Para os valores de 2760 e 4140 iterações/ms, a CAPP com JoiN apresentou maiores escalabilidades na maioria das combinações (p,$p´$).

Podemos observar que ambas as CAPP´s, mesmo para uma V_{um} de referência em 90%, atingem altos valores de escalabilidade chegando, em alguns casos, à escalabilidade unitária ideal ou muito próxima dela. Essa informação sinaliza que as características presentes no algoritmo escolhido para a composição das CAPP´s nos permite trabalhar em regiões próximas ao limite superior de escalabilidade, extraindo assim o máximo de aproveitamento da plataforma paralela.

Um aspecto que deve ser ressaltado, e que pode ser observado na Tabela 3, é a influência do número inicial de processadores no valor da escalabilidade. Para qualquer situação na tabela, observa-se que, para um valor fixo de *p´*, os valores de escalabilidades crescem à medida que *p* se aproxima dele, ou seja, a escalabilidade da CAPP depende da situação inicial do sistema paralelo.

4.4.2 Análise da isoeficiência

Sob o aspecto da métrica de isoeficiência é possível visualizarmos a eficiência paralela homogênea, dada pela equação 9 (página 20), de duas maneiras. A primeira, mostrada na Figura 15, apresenta o comportamento da eficiência paralela nas CAPP´s com JoiN e JPVM quando o número de trabalhadores aumenta e a carga do algoritmo permanece fixa. Os gráficos apresentam curvas para quatro valores crescentes de carga de trabalho aplicadas ao algoritmo Pi-Monte Carlo.

Podemos observar que em ambas as CAPP´s a eficiência paralela, com carga fixa, decresce à medida que o número de trabalhadores aumenta. Quando a carga de trabalho do algoritmo permanece constante, o tempo total empregado em processamento útil (com operações paralelizáveis presentes também na versão serial) não aumenta, porém o tempo gasto com *overhead* de paralelização aumenta em função do número de trabalhadores. O aumento no tempo de *overhead* juntamente com o tempo de processamento útil fixo, diminui a eficiência da execução paralela. De maneira análoga observa-se, pela carga associada a cada curva, que à medida que o valor da carga é maior obtemos um aumento na eficiência inicial. Assim, para valores suficientemente grandes de carga é possível atingir a eficiência máxima (unitária), como ocorreu em ambas as CAPP´s no caso de $16{,}8.10^9$ iterações

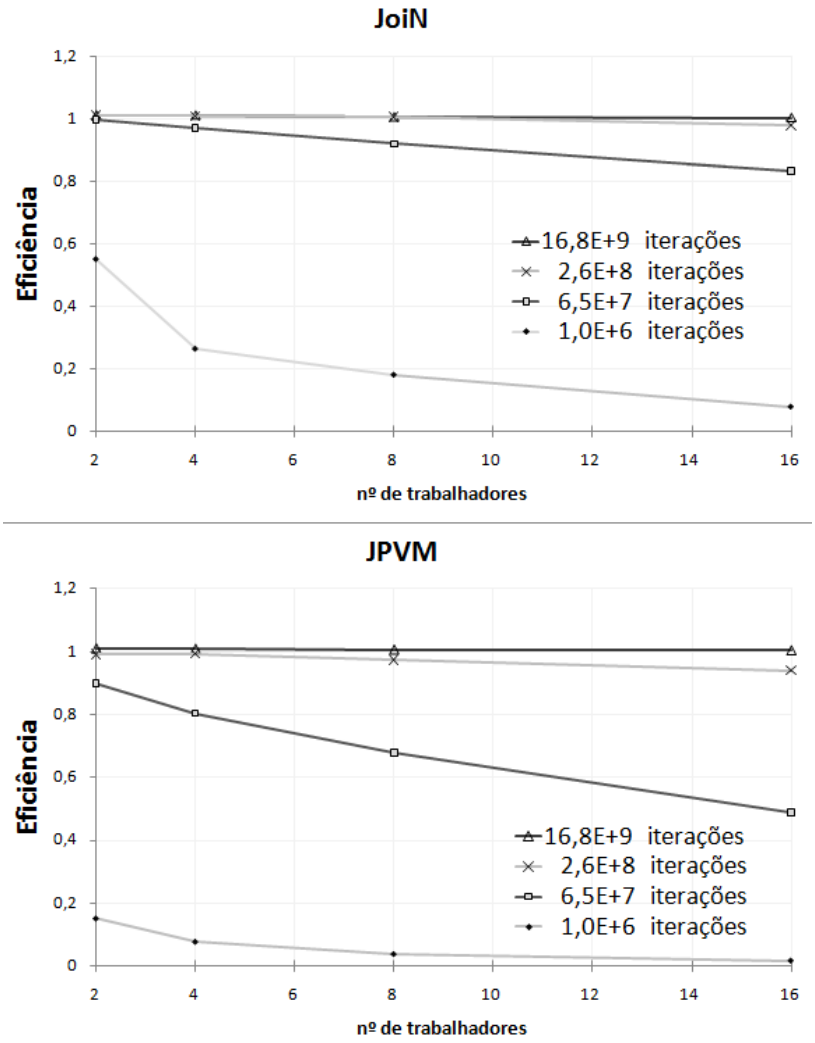

Figura 15 - Eficiência paralela com carga fixa, nas CAPP´s com JoiN e JPVM.

Utilizando a métrica como instrumento de comparação, podemos perceber pela Figura 15 que a CAPP com JoiN apresentou eficiências mais altas para valores menores de carga, em relação a CAPP com JPVM, fato que também foi verificado pela ótica da métrica da isovelocidade. Isso significa que a CAPP com JPVM necessitou de mais carga para compensar o *overhead* gerado pela entrada de trabalhadores na plataforma.

Outra visualização da eficiência paralela homogênea, prevista na métrica da isoeficiência, é seu comportamento à medida que fixamos o número de trabalhadores e variamos a carga do algoritmo. A Figura 16 e Figura 17 mostram esse comportamento para as plataformas JoiN e JPVM, respectivamente, executando o algoritmo Pi-Monte Carlo.

O que se observa aqui é um efeito contrário ao visto no gráfico da eficiência paralela com carga fixa. Com o aumento da carga de trabalho, a eficiência paralela aumenta até o momento em que satura próxima a 1. Isso ocorre porque as características das CAPP´s permitem que o tempo gasto com o *overhead* (que não depende da carga) permaneça constante e seja proporcionalmente cada vez menor em relação ao tempo de processamento total. Essa diminuição relativa no *overhead* do algoritmo aumenta seu *speedup* e consequentemente a eficiência paralela.

Comparando os gráficos da Figura 16 e Figura 17 é possível constatar novamente que a CAPP com JoiN se mostra mais eficiente que a com JPVM, pois obtém eficiências mais altas para valores menores de carga.

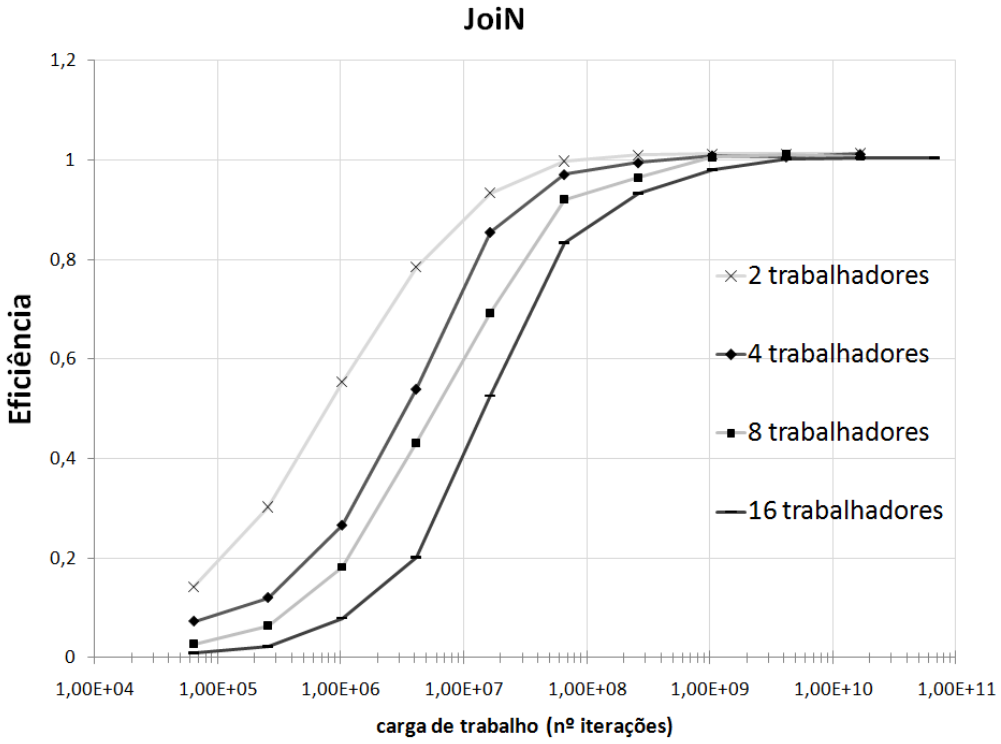

Figura 16 - Eficiência paralela em função da carga do algoritmo Pi-Monte Carlo em JoiN.

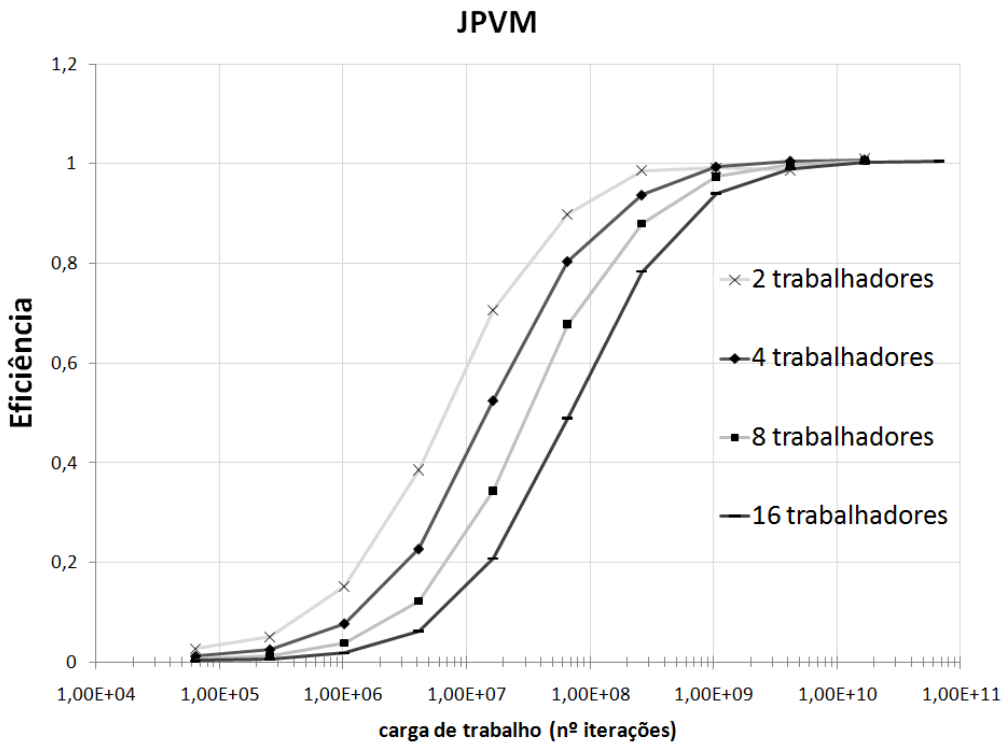

Figura 17 - Eficiência paralela em função da carga do algoritmo Pi-Monte Carlo em JPVM.

Pelo fato da métrica de isoeficiência relacionar a escalabilidade da CAPP com a variação de carga necessária para manter a eficiência paralela constante, podemos utilizar a mesma metodologia aplicada anteriormente na métrica da isovelocidade e determinar os valores de carga necessários para a manutenção da eficiência em valores selecionados.

Através da interpolação logarítmica dos pontos de cada curva dos gráficos da Figura 16 e Figura 17, foram obtidos os valores aproximados de carga necessários para atingir 30%, 60% e 90% da eficiência paralela. No Apêndice C são mostrados os gráficos das curvas resultantes dessa interpolação. A Tabela 4 mostra os valores de cargas obtidos.

Seguindo agora o critério de escalabilidade definido pela métrica da isoeficiência, novamente ambas as CAPP´s podem ser consideradas escaláveis, pois pelo exposto na Tabela 4, podemos perceber que existem valores de carga capazes de manter a eficiência paralela constante dentro das variações de trabalhadores utilizadas.

Tabela 4 - Cargas de trabalho necessárias para manter a eficiência paralela constante nas CAPP´s com JoiN e JPVM

Isoeficiência			
JoiN			
Nº trabalhadores	Carga de trabalho (nº iterações)		
	$W_{30\%}$	$W_{60\%}$	$W_{90\%}$
2	248189	1761102	12496431
4	1064476	5887661	32564905
8	2303052	12322549	65932173
16	4899123	28374659	164339884
JPVM			
Nº trabalhadores	Carga de trabalho (nº iterações)		
	$W_{30\%}$	$W_{60\%}$	$W_{90\%}$
2	2288038	11610968	58921482
4	5080856	28576805	160727613
8	10238932	65538189	419502150
16	28999515	156736834	847132631

Podemos utilizar a equação 14 (proposta originalmente para a métrica da isovelocidade na página 24), que considera a variação de carga para quantificar a escalabilidade de uma CAPP, aplicada aos valores da Tabela 4, porém agora mantendo a condição de isoeficiência. Os valores de escalabilidade obtidos estão mostrados na Tabela 5.

Primeiramente percebe-se aqui também a influência do valor de eficiência paralela utilizado como referência, nas escalabilidades calculadas. Comparando as CAPP´s, agora através dos valores obtidos pelo critério da isoeficiência, percebemos a CAPP com JPVM mostrando-se mais escalável para valores menores de eficiência (30%) enquanto a CAPP com JoiN se mostrou mais escalável para a maioria dos casos onde foi considerada a eficiência em 90%.

Percebe-se novamente, que as características do algoritmo escolhido para compor as CAPP´s permitiram explorar todo o potencial das mesmas, sendo possível atingir valores de escalabilidade muito próximos aos ideais, mesmo em condições de eficiência paralela alta (90%).

Um fator observado sob a ótica da isovelocidade, e que se repete aqui, é a relação de dependência entre a escalabilidade calculada e a situação inicial de sistema paralelo.

Comparando os valores de escalabilidades mostrados na Tabela 3 e Tabela 5, podemos constatar a semelhança entre as métricas de isovelocidade e isoeficiência, que se mostraram coerentes na caracterização de ambas as CAPP´s.

Tabela 5 - Valores de escalabilidade das CAPP´s com JoiN e JPVM, para diferentes valores de eficiência paralela.

Isoeficiência								
Escalabilidade JoiN - $\Psi(p,p')$					Escalabilidade JPVM - $\Psi(p,p')$			
E = 0,3 (30%)					E = 0,3 (30%)			
p / p´	**4**	**8**	**16**		**p / p´**	**4**	**8**	**16**
2	0,47	0,43	0,41		**2**	0,90	0,89	0,63
4		0,92	0,87		**4**		0,99	0,70
8			0,94		**8**			0,71
E = 0,6 (60%)					E = 0,6 (60%)			
p / p´	**4**	**8**	**16**	**p / p´**	**p / p´**	**4**	**8**	**16**
2	0,60	0,57	0,50	**2**	**2**	0,81	0,71	0,59
4		0,96	0,83	**4**	**4**		0,87	0,73
8			0,87	**8**	**8**			0,84
E = 0,9 (90%)					E = 0,9 (90%)			
p / p´	**4**	**8**	**16**		**p / p´**	**4**	**8**	**16**
2	0,77	0,76	0,61		**2**	0,73	0,56	0,56
4		0,99	0,79		**4**		0,77	0,76
8			0,80		**8**			0,99

4.5 Conclusões do capítulo

Neste capítulo foram descritos os testes realizados com sistemas homogêneos, utilizando as métricas da isoeficiência e isovelocidade quando aplicadas a duas plataformas de processamento paralelo virtual, JoiN e JPVM, onde ambas executaram o mesmo algoritmo Pi-Monte Carlo.

O algoritmo selecionado para testes foi descrito como uma aplicação propícia a ser utilizada na medição de escalabilidade de CAPP´s com SPPV. Por possuir alto grau de paralelismo, baixa comunicação e um *overhead* independente da variação de carga, permite extrair o máximo de desempenho e avaliar os limites superiores de escalabilidade das CAPP´s.

Foram apresentadas as principais características das plataformas utilizadas nos testes, JoiN e JPVM, assim como os passos básicos para sua utilização.

Os resultados dos testes permitiram identificar a influência da escolha do valor do parâmetro de desempenho (neste caso V_{um} e eficiência paralela) no valor final da escalabilidade calculada, o que determina que a comparação de escalabilidade entre CAPP´s deva considerar como base sempre um valor comum.

Pelo fato do comportamento ideal do tempo de execução em uma CAPP, com carga fixa, ser exatamente o inverso da variação no número de processadores, os valores calculados de escalabilidade podem ser entendidos como o quão próxima a CAPP está da situação ideal.

Observamos que, para ambas as CAPP´s e sob a ótica das duas métricas utilizadas, foi possível atingir altos valores de escalabilidade, chegando inclusive à escalabilidade unitária ideal (em algumas situações) ou muito próxima dela. Esses valores se mantiveram mesmo na condição onde os parâmetros de desempenho foram mantidos em 90% do máximo, indicando que as características do algoritmo escolhido para compor as CAPP´s permitiram explorar todo o potencial das mesmas.

Percebeu-se também que a escalabilidade de uma CAPP depende da situação inicial do sistema paralelo, pois mesmo se tratando de CAPP´s consideradas escaláveis, observa-se que, para um determinado número final de processadores, os valores de escalabilidades crescem à medida que o número inicial de processadores está mais próximo do final.

Utilizando as métricas para comparar as CAPP´s testadas, ambas as métricas identificaram a CAPP com JoiN como mais escalável nas condições de valores mais elevados de V_{um} e eficiência paralela enquanto a CAPP com JPVM mostrou-se mais escalável para baixos valores de parâmetro de desempenho.

Os resultados dos testes identificaram uma semelhança entre as métricas avaliadas, onde ambas consideraram as CAPP´s escaláveis e caracterizaram suas escalabilidades de maneira similar. Porém a métrica da isovelocidade se destacou por apresentar uma maneira direta de quantificar a escalabilidade e esta abordagem foi adaptada e utilizada também na métrica da isoeficiência.

Baseado nas características estudadas no Capítulo 3 e na experiência adquirida com os testes aqui descritos, o próximo capítulo apresenta a proposta de uma métrica de escalabilidade denominada Isoeficiência de *Speedup*, a qual é adequada a sistemas heterogêneos de processamento paralelo virtual.

Capítulo 5

Proposta de métrica de escalabilidade para SPPV heterogêneos

Este capítulo apresenta a proposta de uma métrica de escalabilidade adequada para a utilização em SPPV heterogêneos denominada Métrica da Isoeficiência de *Speedup*. Essa proposta teve como base o estudo das métricas de escalabilidade para sistemas homogêneos e heterogêneos presentes na literatura e foi influenciada também pela experiência prática proporcionada pelos testes realizados com as métricas de isoeficiência e isovelocidade aplicadas a sistemas homogêneos.

Um dos objetivos principais desta proposta foi oferecer uma alternativa confiável e ao mesmo tempo de fácil aplicação prática, para análise da escalabilidade de uma CAPP que utiliza SPPV heterogêneo. Para isso a métrica proposta utiliza o *speedup*, que é um parâmetro amplamente conhecido em sistemas paralelos, como base para sua formulação. O capítulo apresenta ainda uma nova abordagem para determinação e para visualização gráfica do *speedup*, a qual simplifica sua análise em sistemas heterogêneos.

5.1 Capacidade computacional em um sistema paralelo heterogêneo

Em sistemas homogêneos a definição de capacidade computacional total está diretamente ligada ao número de processadores (*p*) presentes na plataforma. Como todas as unidades de

processamento possuem as mesmas características, podemos esperar que um aumento em *p* reflita proporcionalmente em um aumento na capacidade total do sistema. Sendo assim o número de processadores é o parâmetro utilizado para representar o aumento ou diminuição na capacidade computacional total do sistema.

O mesmo não ocorre em sistemas heterogêneos; como a capacidade individual de cada processador pode variar, um aumento no número de processadores envolvidos não resulta necessariamente em um aumento proporcional na capacidade computacional total do sistema.

Sendo c_i a capacidade computacional do processador *i*, a capacidade computacional total (*C(p)*) de um sistema paralelo com *p* processadores é dada pela equação 25.

$$C(p) = \sum_{i=1}^{p} c_i \tag{25}$$

Podemos definir também a máxima capacidade individual (c_{max}) em um sistema paralelo heterogêneo com *p* processadores, como o maior valor de capacidade de processamento encontrado dentre os processadores existentes no sistema.

$$c_{max} = \max_{1 \le i \le p} c_i \tag{26}$$

Uma maneira prática e eficaz para obter a capacidade computacional individual de um processador (c_i) em SPPV é utilizando um conjunto padrão de *benchmarks*. Assim, sempre que um trabalhador é introduzido na plataforma, esse conjunto é executado no mesmo com a finalidade de mensurar sua capacidade computacional individual no momento em que ele se dispôs a fazer parte do SPPV. O conjunto de *benchmarks* deve ser suficientemente diversificado, contendo operações de várias naturezas como entrada/saída de dados, operações aritméticas com e sem ponto flutuante entre outras. Porém o tempo de execução desses testes deve ser reduzido, uma vez que não faz sentido que essa avaliação seja custosa em relação ao tempo de execução de uma aplicação paralela. Conhecendo o número de operações realizas no conjunto de *benchmarks* e o tempo gasto pelo processador para realizá-las, é possível determinar o valor de c_i dado em operações/segundo.

Essa abordagem, além de factível, mostra-se mais confiável em relação à utilização de dados teóricos tabelados pelos fabricantes que muitas vezes não representam as características atuais do processador, ou mesmo à execução de *benchmarks* tradicionais específicos que avaliam aspectos

isolados do processador e podem ter longa duração. Caso sejam conhecidas as características da aplicação, um *benchmark* específico é mais adequado, porém o custo de utilizar um *benchmark* para cada tipo aplicação pode ser desvantajoso na prática, frente à utilização de um genérico que seja satisfatório para várias.

5.2 Eficiência de *speedup* como parâmetro de desempenho

Um dos pontos principais de uma métrica de escalabilidade é seu parâmetro de desempenho. Portanto, a métrica proposta aqui define e adota como parâmetro de desempenho a eficiência de uma CAPP com relação ao seu *speedup*, denominada eficiência de *speedup* (E_s).

Considerando uma CAPP heterogênea onde c_{max} é a maior capacidade de processamento individual, $C(p)$ é a capacidade computacional total e w é a carga do algoritmo utilizado, a eficiência de *speedup* da CAPP é definida como a razão do *speedup* real (S_{real}) pelo *speedup* ideal (S_{ideal}).

$$E_s(w, C(p), c_{max}) = \frac{S_{real}(w,C(p),c_{max})}{S_{ideal}(w,C(p),c_{max})} \quad (27)$$

Veremos a seguir como obter os *speedups* necessários ao cálculo de E_s.

5.2.1 *Speedup* heterogêneo real

O *speedup* real em uma CAPP heterogênea que possui os parâmetros c_{max}, $C(p)$ e w, é obtido através da equação 28.

$$S_{real}(w, C(p), c_{max}) = \frac{T_{s_{min}}(w,c_{max})}{T_{par}(w,C(p))} \quad (28)$$

O termo T_{par} é tempo de execução paralela medido na CAPP com carga w e capacidade computacional total $C(p)$.

O termo $T_{s_{min}}$ é o tempo de execução serial mínimo, obtido com a melhor versão serial conhecida do algoritmo (com carga w) executada na máquina trabalhadora com maior capacidade de processamento da CAPP (c_{max}).

Com discutido anteriormente, na prática nem sempre a execução serial do algoritmo é viável, pois quando w atinge valores realmente elevados a execução serial pode ser muito demorada.

Uma opção neste caso é estimar o valor de $T_{s_{min}}$ através da equação 29, que considera a carga total *w* sendo executada a uma velocidade c_{max}.

$$T_{s_{min}}(w, c_{max}) = \frac{w}{c_{max}} \tag{29}$$

5.2.2 *Speedup* heterogêneo ideal

O *speedup* ideal de uma CAPP heterogênea que possui os parâmetros c_{max}, *C(p)* e *w*, pode ser calculado através da equação 30.

$$S_{ideal}(w, C(p), c_{max}) = \frac{T_{s_{min}}(w, c_{max})}{T_{par_{ideal}}(w, C(p))} \tag{30}$$

O termo $T_{par_{ideal}}$(tempo de execução paralela ideal) é tempo teórico obtido pela execução de uma versão paralela do algoritmo que não possui nenhum *overhead* de paralelização, em uma CAPP com a capacidade computacional total igual a *C(p)*. O tempo de execução paralela ideal pode ser calculado através da equação 31.

$$T_{par_{ideal}}(w, C(p)) = \frac{w}{C(p)} \tag{31}$$

Alternativa para a obtenção do *speedup* ideal

Uma alternativa simples para estimar o valor do *speedup* ideal heterogêneo é utilizar fatores de desempenho relativos entre os trabalhadores presentes na CAPP. Estes podem ser obtidos a partir das capacidades individuais medidas pelo conjunto de *benchmarks* aplicado aos trabalhadores. O fator de desempenho relativo (*fdr*) do trabalhador *i* é calculado por:

$$fdr_i = \frac{c_i}{c_{max}} \tag{32}$$

O cálculo dos fatores de desempenho pode ser efetuado logo após a execução dos *benchmarks* nos trabalhadores da plataforma.

O valor do *speedup* ideal heterogêneo de uma CAPP com *p* trabalhadores, é obtido então pela soma de todos os fatores de desempenho relativos.

$$S_{ideal}(p) = \sum_{i=1}^{p} fdr_i \ , \tag{33}$$

como demonstrado a seguir.

$$S_{ideal}(p) = \frac{w/c_{max}}{w/C(p)} = \frac{C(p)}{c_{max}} \quad (34)$$

$$S_{ideal}(p) = \frac{\sum_{i=1}^{p} c_i}{c_{max}} = \frac{c_1 + c_2 + \cdots + c_p}{c_{max}} = \sum_{i=1}^{p} fdr_i \quad (35)$$

5.2.3 Equivalência com eficiência de *speedup* em sistemas homogêneos

Podemos verificar a equivalência da eficiência de *speedup* proposta nesta métrica para sistemas heterogêneos com a eficiência paralela clássica (*E*) apresentada na equação 9 (página 20) e utilizada exclusivamente em sistemas homogêneos.

A partir do demonstrado na equação 35, o *speedup* ideal em uma CAPP homogênea com *p* processadores idênticos é dado por:

$$S_{ideal}(p) = \frac{c_1 + c_2 + \cdots + c_p}{c_{max}}$$

Como em sistemas homogêneos: $c_1 = c_2 = \cdots = c_p = c_{max}$

$$S_{ideal}(p) = \frac{c_{max} * p}{c_{max}} = p \quad (36)$$

Considerando o sistema homogêneo, quando aplicamos o valor do *speedup* ideal obtido na equação 36, na definição de eficiência de *speedup* E_s proposta na equação 27, temos como resultado a definição de eficiência paralela clássica E para sistemas homogêneos, apresentada na equação 9.

$$E_s(w, C(p), c_{max}) = \frac{S_{real}(w, C(p), c_{max})}{S_{ideal}(p)} = \frac{S_{real}(w,p)}{p} = E(w,p) \quad (37)$$

Portanto a eficiência paralela (*E(w,p)*) utilizada unicamente em sistemas homogêneos está contida na eficiência de *speedup* E_s proposta nesta métrica, como um caso particular. Esse fato propicia que E_s seja utilizada como parâmetro de desempenho na análise de escalabilidade tanto de sistemas heterogêneos como homogêneos.

5.2.4 Número de tarefas mínimo na execução paralela

O fator de desempenho relativo pode ser utilizado também para determinar o número total de tarefas mínimo a ser utilizado em uma execução paralela, em um ambiente heterogêneo, visando sempre aproveitar o potencial computacional individual de cada processador.

A idéia é atribuir para cada processador um número de tarefas proporcional ao seu poder computacional, de forma que todos os processadores permaneçam ocupados durante a maior parte da execução paralela e idealmente finalizem seus trabalhos simultaneamente.

O número mínimo de tarefas atribuídas ao processador *i* é dado pela equação 38.

$$Tasks_{i_{min}} = \frac{fdr_i}{fdr_{min}} \tag{38}$$

onde fdr_{min} é o menor fator de desempenho relativo encontrado na CAPP heterogênea.

O número total de tarefas mínimo na execução paralela é então a soma das tarefas atribuídas a cada processador na CAPP, como mostra a equação 39.

$$Tasks\ _{total_{min}} = \frac{1}{fdr_{min}} \sum_{i=1}^{p} fdr_i \tag{39}$$

Recomenda-se utilizar um número de tarefas total significativamente maior que o valor total mínimo calculado da equação 39 para reduzir o risco de trabalhadores ficarem ociosos. Entretanto, este número não pode ser muito elevado a ponto de tornar o tempo de computação de cada tarefa pequeno em relação ao tempo de comunicação da mesma.

5.2.5 Visualização gráfica do *speedup* em sistemas heterogêneos

A visualização gráfica do *speedup* em função do aumento no número de processadores é um recurso muito útil na análise do desempenho de sistemas paralelos, principalmente no caso em que a métrica utiliza o *speedup* para o cálculo do seu parâmetro de desempenho.

Quando tratamos de sistemas homogêneos, é fácil prever como seria o gráfico do *speedup* ideal. Como visto na equação 33, cada processador adicionado à CAPP contribui idealmente com o valor de seu *fdr*. Como em sistemas homogêneos todos os processadores possuem *fdr* unitários, o gráfico do *speedup* ideal é uma reta. Uma vez que estamos lidando com sistemas heterogêneos, a previsão do gráfico do *speedup* ideal não é algo tão simples. Além de não ser mais

necessariamente linear, a entrada aleatória de processadores com capacidades diferentes de processamento resulta em um gráfico irregular que pode dificultar a compreensão do comportamento do *speedup* heterogêneo. Cada processador introduzido na execução contribui, mesmo que idealmente, com incrementos diferentes no *speedup*.

Como exemplo, imagine uma CAPP heterogênea como 20 processadores cujos *fdr´s* estão descritos na Tabela 6.

Tabela 6 - Fatores de desempenho relativos em uma CAPP heterogênea com 20 processadores.

Processador *i*	***fdr*$_i$**
1	1,00
2	0,95
3	0,90
4	0,90
5	0,75
6	0,70
7	0,60
8	0,50
9	0,40
10	0,35
11	0,30
12	0,25
13	0,20
14	0,15
15	0,10
16	0,05
17	0,04
18	0,04
19	0,03
20	0,01

A Figura 18 apresenta como seriam os *speedups* ideais para a CAPP heterogênea descrita na Tabela 6, quando os 20 processadores são adicionados de maneira aleatória e quando os mesmos processadores são adicionados em ordem crescente e decrescente de *fdr*. O gráfico demonstra também como seria o *speedup* ideal para uma versão homogênea da CAPP com 20 processadores.

Podemos observar pela Figura 18 que ordenação dos processadores adicionados em uma CAPP heterogênea torna o gráfico do *speedup* ideal mais fácil de analisar, porém essa abordagem visa apenas facilitar visualmente a compreensão do comportamento do *speedup* frente ao idealmente esperado e não altera o desempenho da CAPP avaliada.

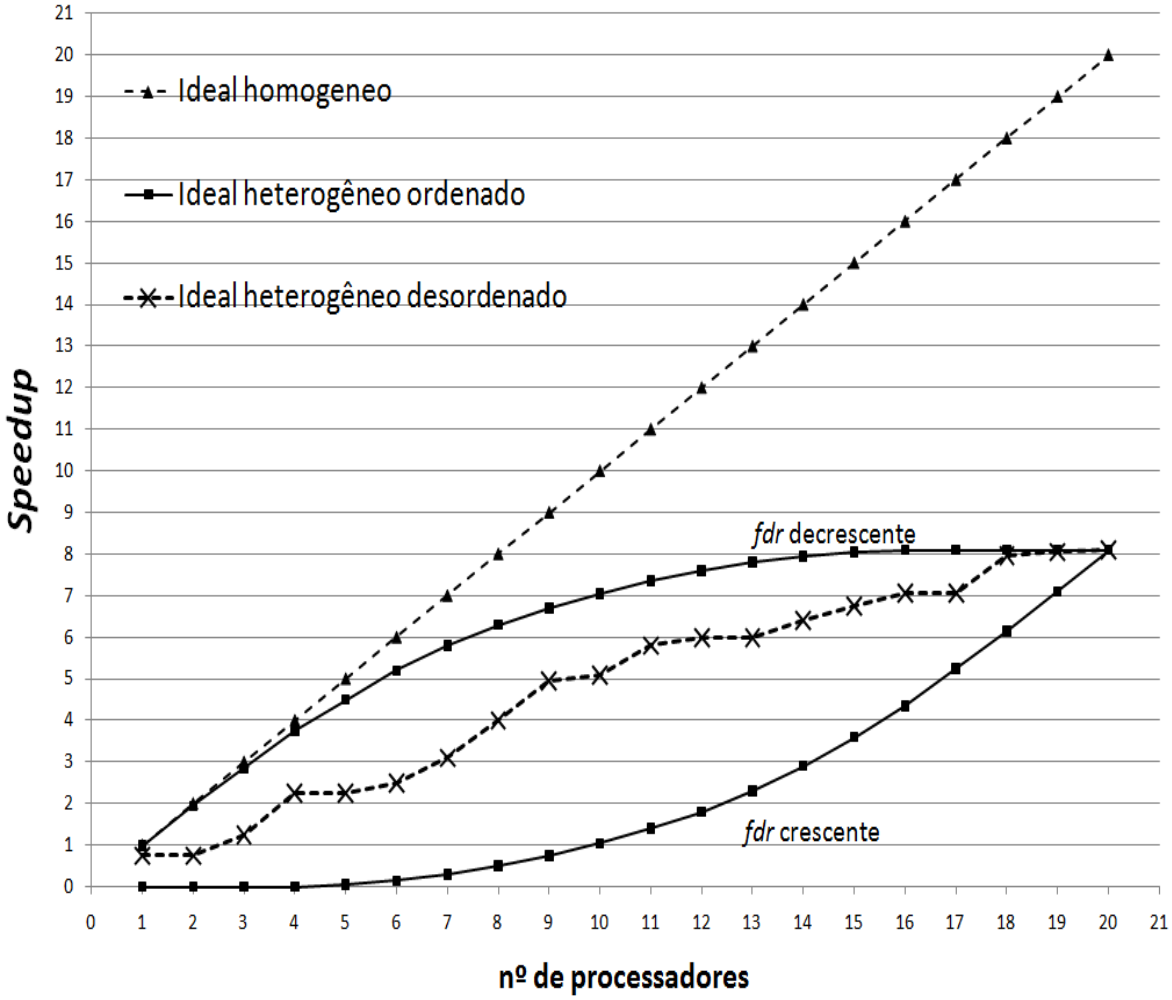

Figura 18 - *Speedup* ideal ordenado e desordenado em uma CAPP heterogênea.

A Figura 19 mostra um exemplo de como pode ser o gráfico dos *speedups* ideal e real ordenados (de maneira decrescente) em uma CAPP heterogênea, onde no momento da entrada do décimo trabalhador foram destacados graficamente os valores dos *speedups* que compõem o cálculo da eficiência de *speedup* (E_s).

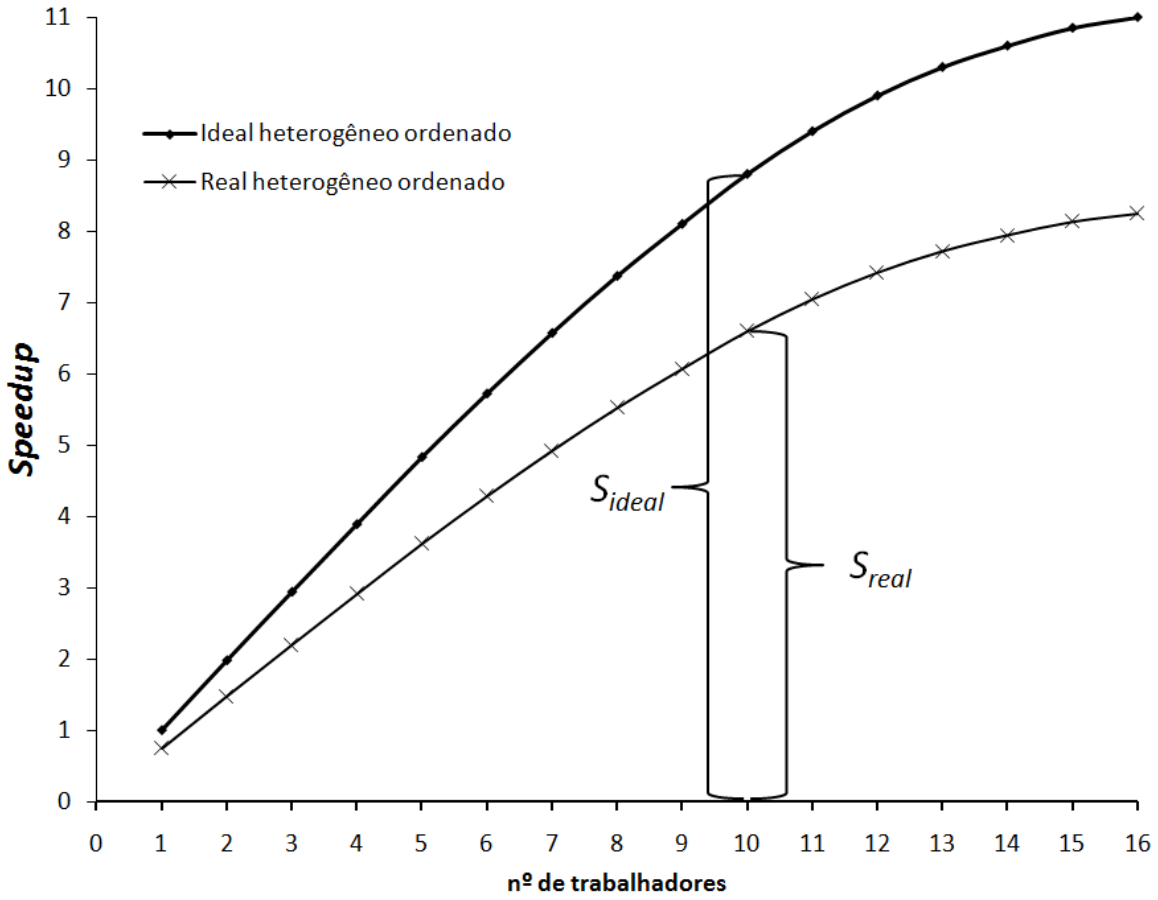

Figura 19 - *Speedups* ideal e real, ordenados (*fdr* decrescente) em uma CAPP heterogênea.

5.3 Escalabilidade de uma CAPP heterogênea em isoeficiência de *speedup*

Definimos a escalabilidade de uma CAPP heterogênea em condição de isoeficiência de *speedup* da seguinte forma.

Uma CAPP heterogênea que sofre um aumento em sua capacidade computacional total (C(p)) é considerada escalável, se a eficiência de speedup (E_s) depois do aumento pode ser mantida constante, mediante um aumento na carga de trabalho (w) da aplicação.

Baseado nesta definição, a quantificação da escalabilidade nesta métrica é proposta de maneira análoga à métrica de isovelo-eficiência apresentada no Capítulo 3, que também utiliza a variação de carga como medida do grau de escalabilidade. Porém, consideramos aqui a eficiência de *speedup* (E_s) como parâmetro de desempenho a ser mantido constante.

Consideramos uma CAPP que tenha sua capacidade computacional total aumentada de $C(p)$ para $C(p')$, e sua carga de trabalho também aumentada de w para w'. Se a eficiência de *speedup* (E_s) na CAPP($C(p),w$) é a mesma que na CAPP($C(p'),w'$), então a escalabilidade da CAPP nesta situação pode ser definida pela equação 40:

$$\Psi(C(p), C(p')) = \frac{w/C(p)}{w'/C(p')} \tag{40}$$

Uma visualização útil ao entendimento da variação de carga necessária em uma CAPP para manter a eficiência de *speedup* constante, pode ser vista no gráfico da Figura 20.

Como exemplo, imagine que é desejado manter a E_s da CAPP em 75%, isso equivale a termos uma curva de *speedup* real que se mantém sempre a 75% da ideal, como mostrado na Figura 20. O gráfico mostra também quatro perfis do *speedup* real da CAPP com carga fixa, onde $W_1 < W_2 < W_3 < W_4$.

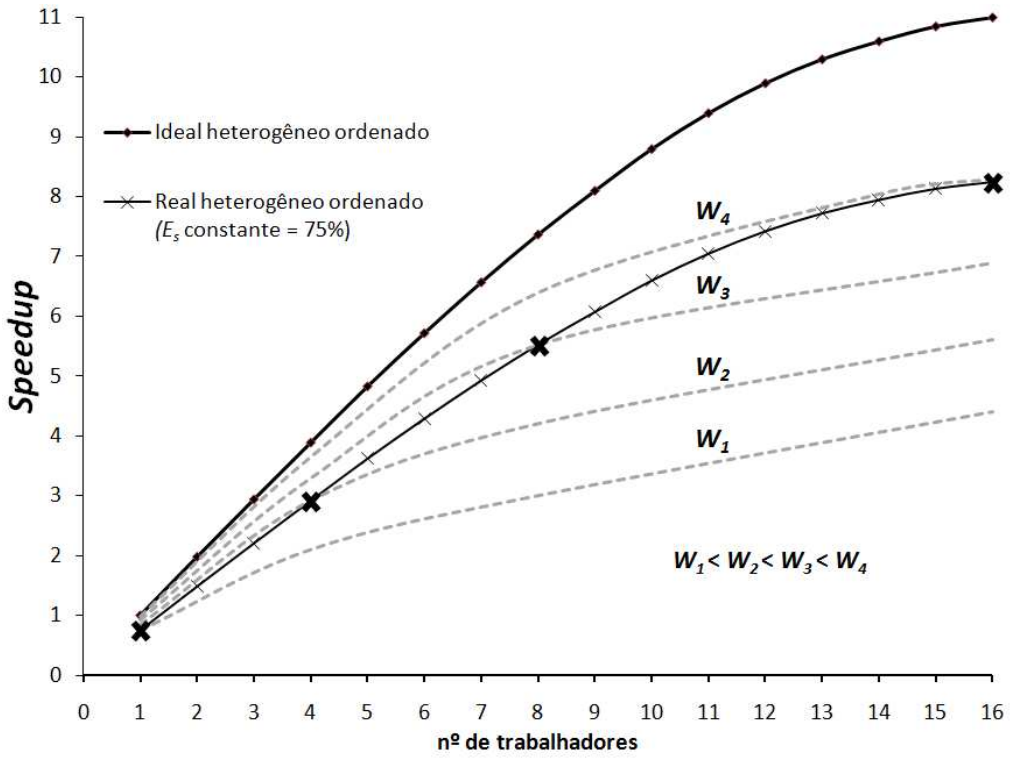

Figura 20 - Perfis de *speedup* de carga fixa em uma CAPP heterogênea.

Analisando os perfis de *speedup* de carga fixa deste exemplo, vemos que se mantivermos a carga fixa em um valor W_1, quando ocorrer a entrada do segundo trabalhador, a eficiência de

speedup da CAPP ficará abaixo do objetivo de 75%. Por outro lado, o valor W_2 atende o requisito de 75% no momento em que a CAPP recebe o quarto trabalhador Entretanto, a carga deve ser aumentada para W_3, se desejarmos manter 75% de E_s com a entrada do oitavo trabalhador na CAPP.

Ainda seguindo o exemplo, os valores de carga (w) necessários para atingir a eficiência pretendida, podem ser obtidos como exemplifica a Figura 21 que apresenta um gráfico de E_s em função da variação de carga em duas situações da CAPP ($C(p)$ e $C(p')$).

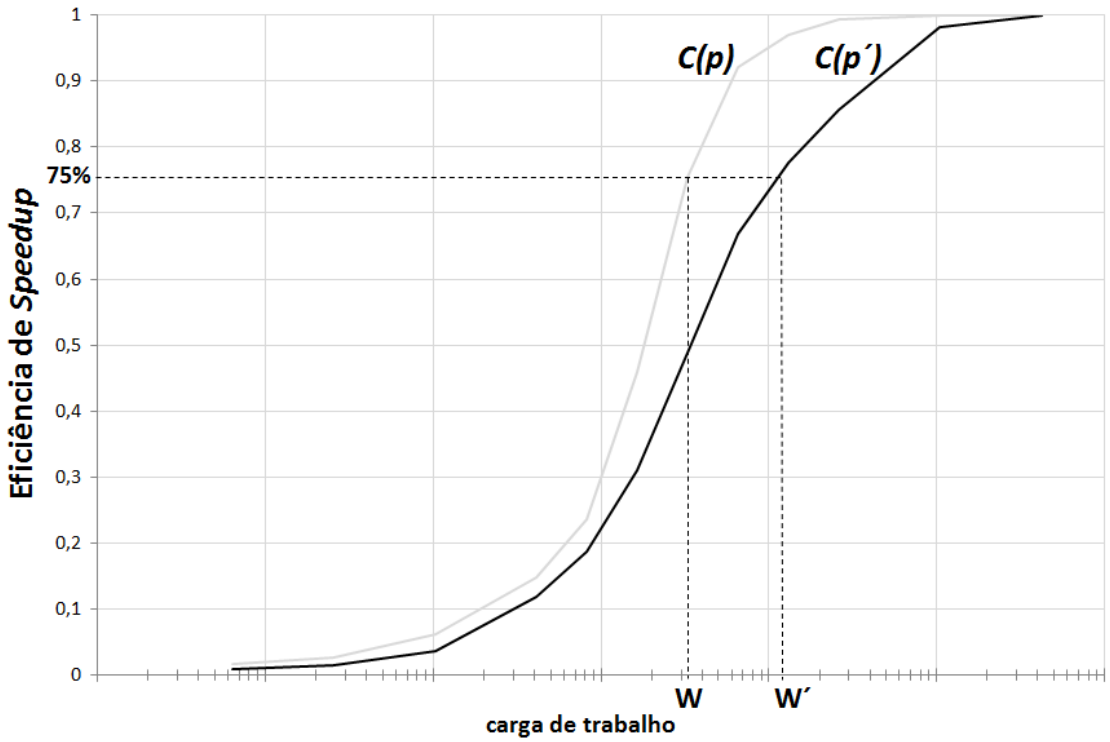

Figura 21 - Obtenção das cargas de trabalho necessárias para atingir 75% de E_s.

Os correspondentes valores $(C(p),w)$ e $(C(p'),w')$ fornecem as informações necessárias para o cálculo da escalabilidade proposta na equação 40.

5.4 Conclusões do capítulo

Neste capítulo foi apresentada uma nova métrica de escalabilidade destinada a SPPV heterogêneos, denominada métrica da isoeficiência de *speedup*.

A métrica define o conceito de eficiência de *speedup*, o qual é utilizado como parâmetro de desempenho do método.

Por utilizar o *speedup*, um conceito muito útil e familiar em processamento paralelo, a métrica proposta facilita a análise da escalabilidade de SPPV e é de fácil aplicação prática.

Além disso, sugerimos uma abordagem prática, baseada na aplicação de um conjunto padrão de *benchmarks*, para a medição da capacidade computacional individual dos trabalhadores de uma CAPP. Nesta abordagem propomos o uso de fatores de desempenho relativos, que auxiliam na obtenção do *speedup* ideal heterogêneo.

Como instrumento de análise, a proposta utiliza um gráfico de *speedup* ordenado, que facilita a compreensão do comportamento do *speedup* frente ao idealmente esperado.

Finalmente, propomos uma maneira de quantificar a escalabilidade baseada na variação de carga necessária para manter a eficiência de *speedup* constante à medida que a capacidade computacional da CAPP é aumentada.

O próximo capítulo apresenta testes realizados com a nova métrica da isoeficiência de *speedup* em uma CAPP composta pela plataforma JoiN e o algoritmo Pi-Monte Carlo em um ambiente heterogêneo.

Capítulo 6

Testes com a métrica da Isoeficiência de *Speedup* em sistemas heterogêneos

Este capítulo descreve os testes realizados para com a nova métrica da isoeficiência de *Speedup* proposta neste trabalho para sistemas heterogêneos.

6.1 CAPP utilizada

O algoritmo Pi-Monte Carlo utilizado para os testes em ambientes homogêneos, descritos no Capítulo 4, foi novamente utilizado nos testes com ambientes heterogêneos. Como mencionado, uma vez que o algoritmo permite aumento de carga e número de tarefas sem aumentar o *overhead*, a escalabilidade da CAPP torna-se predominantemente influenciada pelas características da plataforma, e pode ser avaliada em uma região próxima ao seu limite superior.

Para possuirmos uma referência, que pudesse servir como um dos parâmetros de validação da nova métrica proposta, escolhemos para compor a CAPP uma plataforma que já havia sido avaliada em ambientes homogêneos. A plataforma selecionada foi o sistema de processamento paralelo virtual JoiN, que também está descrito no Capítulo 4.

6.2 Ambiente de testes e metodologia

Os testes apresentados aqui foram realizados na Faculdade de Engenharia Elétrica e de Computação da UNICAMP com máquinas de uso exclusivo. Para a composição do ambiente heterogêneo, foram utilizados três tipos diferentes de máquinas num total de 12 máquinas trabalhadoras.

Como proposto na métrica de isoeficiência de *speedup*, todas as máquinas foram avaliadas segundo um conjunto de benchmarks no momento em que se conectaram na plataforma JoiN. Os *benchmarks* utilizados foram os desenvolvidos pelo grupo de pesquisas Java Grande Forum (http://www.epcc.ed.ac.uk/javagrande/), que realiza pesquisas em computação científica utilizando a tecnologia Java. A execução dos *benchmarks* em cada máquina permitiu o cálculo dos *fdr´s*, de acordo com a equação 32 e, consequentemente, a classificação das máquinas segundo suas capacidades computacionais. A Tabela 7, apresenta a classificação das máquinas utilizadas a partir de seus *fdr´s*, assim como a correspondente configuração de *hardware* e *software*.

Tabela 7- Nomes e configurações das máquinas utilizadas no ambiente heterogêneo.

Máquina	*fdr*	Configuração de HW e SW
le22-4	1,00	Core 2 Duo 2.5GHz, 4Gb RAM, S.O. Linux Ubuntu v9.04, rede 100 Mbps.
le22-2	0,99	
le22-3	0,97	
le22-5	0,96	
viviane	0,35	Pentium 4 1.8GHz, 1Gb RAM, S.O. Linux Ubuntu v9.04, rede 100 Mbps.
Bishop	0,35	
Tuck	0,35	
Vortigen	0,35	
iff	0,25	Pentium 4 1.48GHz, 512Mb RAM, S.O. Windows XP, rede 100 Mbps.
eps	0,24	
dxf	0,24	
pict	0,23	

6.2.1 Configuração da aplicação

Foram utilizadas 13 instâncias da aplicação Pi-Monte Carlo com diferentes cargas de trabalho. A Tabela 8 apresenta o número de iterações executadas em cada instância da aplicação.

Tabela 8 - Carga total de trabalho atribuída a cada instância da aplicação Pi-Monte Carlo nos testes com sistema heterogêneo.

Instância	Carga total de trabalho (nº iterações)
00	64.10^3
01	256.10^3
02	1024.10^3
03	4096.10^3
04	8192.10^3
05	16384.10^3
06	32768.10^3
07	65536.10^3
08	131072.10^3
09	262144.10^3
10	1049.10^6
11	4194.10^6
12	8389.10^6

Analogamente aos testes com sistemas homogêneos foi utilizada a mesma versão serial do algoritmo Pi-Monte Carlo implementada em Java. A versão serial foi executada na máquina com maior *fdr* (le22-4) da CAPP e cada instância da Tabela 8 foi executada cinco vezes para o cálculo de médias. Os dados obtidos na execução serial na máquina "le22-4" constam no Apêndice D.

Foi utilizada a mesma versão paralela do algoritmo Pi-Monte Carlo, implementada em Java para a plataforma JoiN na etapa de testes com sistemas homogêneos. Porém para obter o balanceamento de carga durante as execuções no ambiente heterogêneo, o número de tarefas a ser enviado a cada trabalhador durante a execução foi calculado proporcionalmente ao valor de seu *fdr*. Para não aumentar a complexidade da análise em ambiente heterogêneo, o *overhead* causado pela adição de tarefas foi mantido fixo, pela utilização de um número constante de 500 tarefas independentemente do número de trabalhadores envolvidos. Esse número atende o que foi discutido na seção 5.2.4. O total de trabalho de cada instância foi então dividido em 500 tarefas iguais que possuem um tempo de computação ainda significativo em relação ao tempo de comunicação. O número de tarefas atribuído a cada processador foi calculado segundo a equação 41.

$$Tasks_i = \left\lceil \frac{500 * fdr_i}{S_{ideal}(p)} \right\rceil \quad (41)$$

onde $Tasks_i$ é o número de tarefas a ser enviado ao trabalhador *i*, fdr_i é seu fator de desempenho relativo e $S_{ideal}(p)$ é a soma dos *fdr´s* dos trabalhadores envolvidos na referida execução. (Equação 33 – Página 59)

6.2.2 Condições de testes

Foram criados grupos de 2, 4, 8 e 12 máquinas, para a execução das instâncias da aplicação Pi-Monte Carlo mostradas na Tabela 8. A formação dos grupos de trabalhadores obedeceu a ordem decrescente de *fdr*, como determinado na métrica da isoeficiência de *speedup*. Assim os grupos foram formados como mostrado a Tabela 9.

Todos os tempos obtidos nas execuções estão mostrados no Apêndice D.

Tabela 9 - Grupos de máquinas utilizados nos teste heterogêneos.

Grupo	nº trabalhadores	Máquinas integrantes
1	2	le22-4, le22-2
2	4	le22-4, le22-2, le22-3, le22-5
3	8	le22-4, le22-2, le22-3, le22-5, viviane, bishop, tuck, vortigen
4	12	le22-4, le22-2, le22-3, le22-5, viviane, bishop, tuck, vortigen, iff, eps, dxf, pict

6.3 Resultados

Os dados coletados foram utilizados para verificar se a métrica da isoeficiência de *speedup*, proposta neste trabalho, é capaz de caracterizar adequadamente a escalabilidade de uma CAPP heterogênea.

O primeiro passo na utilização da métrica de isoeficiência de *speedup* é a determinação dos *speedup* ideais para cada grupo de execução, pois a partir deles poderemos calcular a eficiência de *speedup* (E_s), por meio da equação 27, e estabelecer o valor de referência a ser mantido constante durante a análise da escalabilidade. Como discutido no Capítulo 5, o *speedup* ideal pode ser estimado através da soma dos *fdr´s* envolvidos na execução. A Tabela 10 mostra os valores de *speedup* ideais obtidos para cada grupo de execução.

A partir dos valores da Tabela 10 é possível traçarmos o gráfico da eficiência de *speedup* (parâmetro de desempenho desta métrica) em função da quantidade de carga na aplicação Pi-Monte Carlo, para cada um dos grupos utilizados, como mostrado na Figura 22.

Pelo fato do conceito de escalabilidade nesta métrica estar diretamente ligado a variação na carga de trabalho necessária para a manutenção de E_s constante, o gráfico da Figura 22 é fundamental para a obtenção dos valores de carga necessários para atingir a eficiência desejada em cada grupo de execução.

Tabela 10 - *Speedups* ideais nos grupos heterogêneos testados.

Grupo	nº trabalhadores	*Speedup* heterogêneo ideal
1	2	1,99
2	4	3,92
3	8	5,32
4	12	6,28

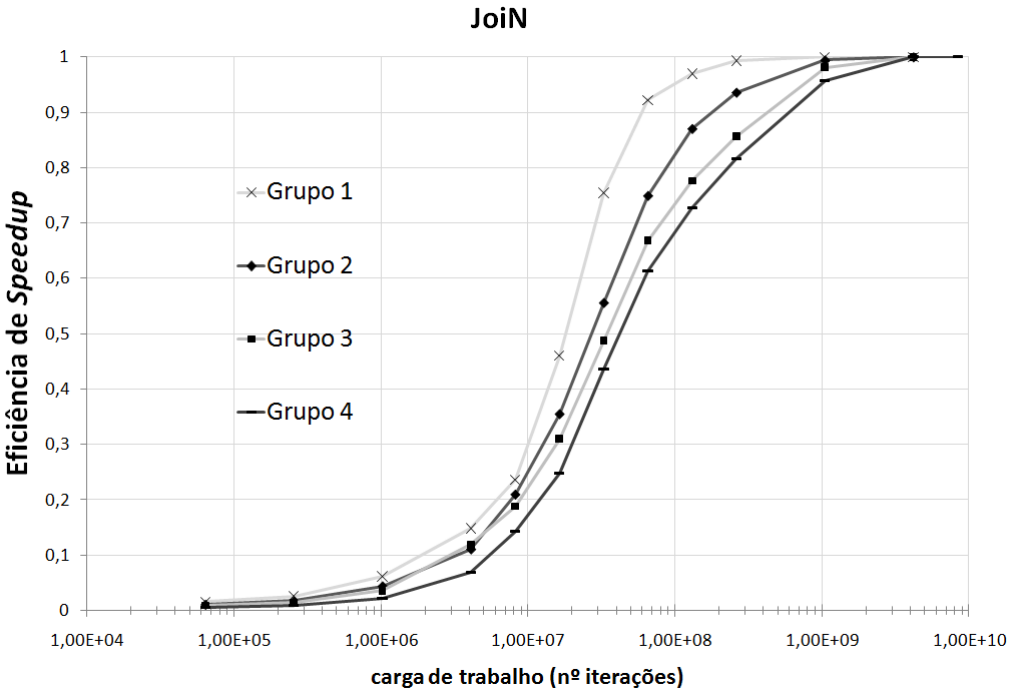

Figura 22 - Eficiência de *speedup* em função da carga do algoritmo Pi-Monte Carlo na plataforma JoiN em ambiente heterogêneo.

Podemos considerar que o valor ideal para a escolha da E_s de referência, seria exatamente o valor máximo atingido pelo parâmetro, uma vez que geralmente é desejado trabalhar com 100% do desempenho que as CAPP's podem nos oferecer. Porém, na região de saturação dos gráficos, estão envolvidas altas variações na carga *w* e a determinação do local exato onde o valor máximo é atingido requer uma granularidade de medidas elevada nesta região. Apesar de possível, do ponto de vista prático, esse procedimento é custoso, pois além de exigir inúmeras medidas para vasculhar uma extensa faixa de carga, as aplicações nesta região possuem tempos de execução elevados devido ao volume de carga envolvido. Portanto para preservar ao máximo o caráter prático da métrica, adotamos como referência o valor de 90% do *speedup* ideal heterogêneo, que continua consideravelmente próximo do máximo desempenho da CAPP.

Como ilustra a Figura 23, para um valor de E_s em 90% e utilizando interpolação logarítmica dos pontos de cada curva na região observada, obtemos os valores da carga de trabalho necessários para a manter E_s constante na CAPP com JoiN e Pi-Monte Carlo para cada um dos grupos heterogêneos. No Apêndice E são mostrados os gráficos das curvas resultantes da interpolação utilizada.

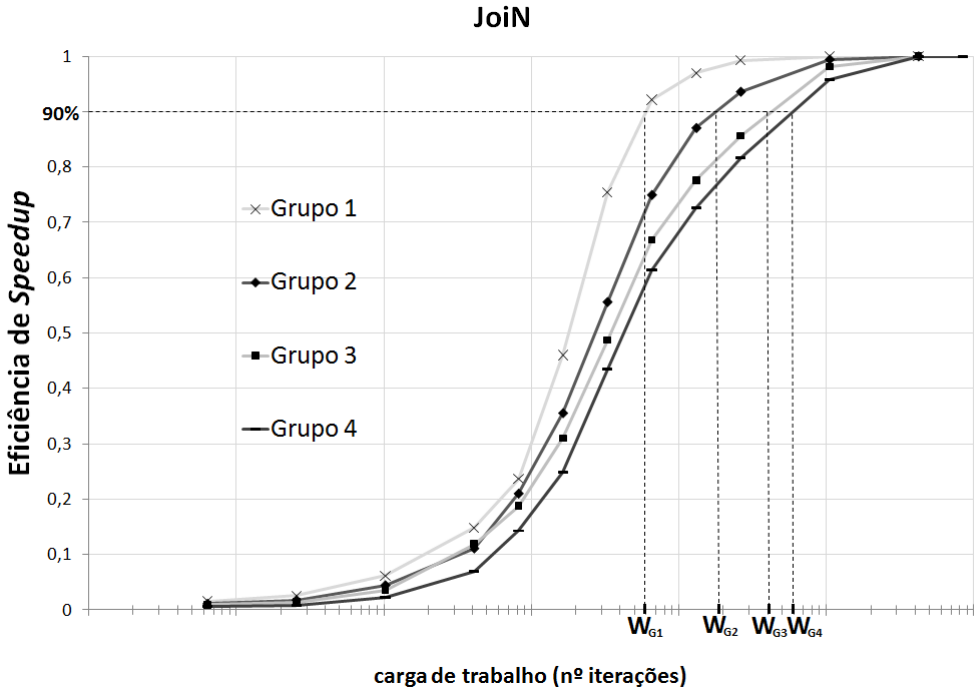

Figura 23 - Obtenção das cargas de trabalho necessárias para atingir 90% de E_s nos grupos heterogêneos de trabalhadores.

É importante ressaltar que apenas a escolha do valor de referência em 90% não garante que seja viável atingi-lo, pois é possível que a quantidade de carga exigida para esta situação fosse impraticável. Neste caso a análise de escalabilidade poderia ter sido feita para valores menores de eficiência, porém correndo o risco de avaliar um sistema paralelo subutilizado.

A Tabela 11 mostra os valores de cargas obtidos com E_s constante em 90% para todos os grupos de trabalhadores, assim como os valores da capacidade computacional total associada a cada grupo.

Tabela 11 - Valores de cargas com E_s constante em 90%, para os grupos heterogêneos de trabalhadores.

E_s = 90%		
Grupo	**Carga (nº iterações)**	**Capacidade computacional total ($C(p)$)**
1	57636677	1,99
2	187737942	3,92
3	442714750	5,32
4	606450077	6,28

Baseado no critério de escalabilidade proposto pela métrica de isoeficiência de *speedup* é possível considerar escalável a CAPP heterogênea com JoiN aqui analisada, uma vez que, pela Tabela 11, existe uma variação da carga da aplicação Pi-Monte Carlo capaz de manter constante a eficiência de *speedup*, à medida que a CAPP tem sua capacidade computacional total ($C(p)$) aumentada. Novamente ressaltamos que se a CAPP analisada não fosse escalável, teoricamente o valor de carga necessário para manter a eficiência de *speedup* constante tenderia a infinito e consequentemente a escalabilidade calculada pela equação 40 tenderia a zero.

A Figura 24 apresenta o gráfico do *speedup* ideal ordenado na CAPP analisada, juntamente com a curva do *speedup* na condição de E_s em 90%, também são apresentadas no gráfico (em tom mais claro) as curvas de *speedup* com carga fixa para algumas das instâncias da aplicação Pi-Monte Carlo. As curvas de carga fixa reforçam a afirmação de que existem valores de carga que satisfazem o critério de isoeficiência de *speedup*, uma vez que estas atingem (e até superam) o valor de 90% de E_s.

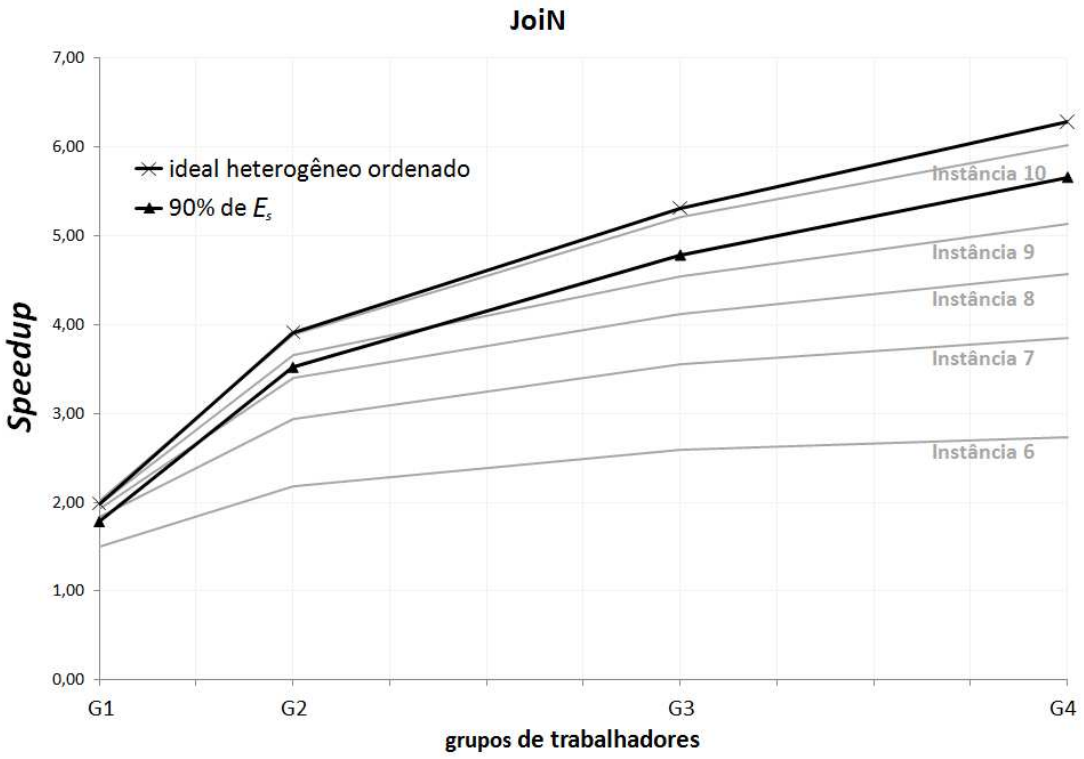

Figura 24 - Perfis de *speedup* de carga fixa para a CAPP heterogênea com JoiN.

A Tabela 12 apresenta os valores de escalabilidade calculados, segundo a equação 40, a partir dos valores presentes na Tabela 11.

Tabela 12 - Valores de escalabilidade da CAPP heterogênea com JoiN para uma eficiência de *speedup* de 90%.

		Escalabilidade JoiN - $\Psi(C(p),C(p'))$		
		E_s = 90%		
		Grupo 2	Grupo 3	Grupo 4
	C(p) \ C(p´)	3,91	5,31	6,28
Grupo 1	1,99	0,60	0,35	0,30
Grupo 2	3,91		0,58	0,50
Grupo 3	5,31			0,86

Na Tabela 12 observa-se que, para um valor fixo de *C(p´)*, os valores de escalabilidades crescem à medida que *C(p)* se aproxima de *C(p´)*, ou seja, a escalabilidade da CAPP heterogênea também depende da situação inicial do sistema paralelo. Este comportamento da escalabilidade e

sua similaridade com a condição homogênea podem ser percebidos mais facilmente devido à classificação ordenada dos trabalhadores.

Considerando que o algoritmo utilizado possui características que permitem analisar o limite superior de escalabilidade da CAPP, percebemos que mesmo nesta situação um ambiente heterogêneo oferece maior dificuldade para o aproveitamento total dos recursos, uma vez que nenhum dos valores calculados atingiu a escalabilidade ideal ou mesmo superou 0,9.

Ao observar os dados da Tabela 12 é importante ter em mente que a escalabilidade em ambientes heterogêneos é calculada a partir das variações na capacidade computacional total da CAPP e apesar do número de trabalhadores dobrar entre os grupos 1, 2, 3 e aumentar 50% no grupo 4, essas variações não são refletidas em *C(p)* que normalmente apresenta variações menores ou, no máximo, iguais as do número de trabalhadores.

Na situação de ambiente heterogêneo o número de trabalhadores (*p*) não pode ser usado diretamente, pois não oferece informação útil para a obtenção da escalabilidade. Portanto o parâmetro importante nestes casos é a capacidade computacional total da CAPP. Um dos diferenciais em métricas que se propõem a avaliar sistemas de processamento paralelo heterogêneos é a caracterização da *C(p)* da CAPP e como sua variação é percebida pela métrica. A isoeficiência de *speedup*, proposta neste trabalho, utiliza uma maneira prática e viável de avaliação da capacidade computacional dos trabalhadores e consequentemente da CAPP, por meio de um conjunto de *benchmarks*. Pelo exposto aqui pudemos verificar que essa abordagem conseguiu caracterizar as variações de poder computacional total e sua influência na escalabilidade da CAPP.

Utilizando os dados da Tabela 11, podemos observar ainda que para uma variação em *C(p)* de pouco mais de 3 vezes (Grupo 1 => Grupo 4), foi necessário um aumento de aproximadamente 10 vezes na carga para manter a eficiência de *speedup* em 90%, o que refletiu na escalabilidade de 0,3 mostrada na célula (Grupo 1 => Grupo 4) da Tabela 12.

6.4 Conclusões do capítulo

Neste capítulo foram descritos os testes realizados com um sistema heterogêneo, utilizando a métrica da isoeficiência de *speedup* aplicada a plataforma de processamento paralelo virtual JoiN, a qual executou o algoritmo Pi-Monte Carlo.

Para obter o balanceamento de carga durante a execução paralela em ambiente heterogêneo, foi utilizada uma abordagem onde a carga total de trabalho da aplicação foi dividida em 500 tarefas iguais e cada trabalhador recebeu uma quantidade de tarefas proporcional a seu desempenho relativo.

A métrica da isoeficiência de *speedup* foi aplicada a CAPP heterogênea com JoiN e verificou-se que a classificação dos trabalhadores em ordem decrescente de desempenho relativo ajudou na interpretação dos gráficos e dos resultados finais. Conforme pôde ser verificado, a utilização da eficiência de *speedup* como parâmetro de desempenho permitiu caracterizar a escalabilidade da CAPP, assim como quantificá-la para as variações de capacidade computacional consideradas. A utilização do *speedup* como parâmetro de desempenho da métrica também facilitou a compreensão dos dados, uma vez que este conceito é amplamente conhecido e está diretamente ligado ao desempenho em sistemas de processamento paralelo.

A utilização na CAPP heterogênea de um algoritmo que possui características próximas às ideais, facilitou a constatação de que em um ambiente heterogêneo o aproveitamento eficiente da capacidade computacional oferecida pelo sistema é mais difícil. Isso se refletiu no fato de nenhum dos valores calculados de escalabilidade ter atingido o valor unitário ideal ou mesmo superado 0,9, como ocorreu com a CAPP JoiN homogênea analisada no Capítulo 4 e que utilizou o mesmo algoritmo.

De maneira similar ao que foi constatado utilizando as métricas para sistemas homogêneos, verificou-se pela nova métrica, uma dependência entre a escalabilidade de uma CAPP heterogênea e a situação inicial da capacidade computacional do sistema paralelo adotado. Pois, para uma determinada capacidade computacional final, os valores de escalabilidades crescem à medida que a capacidade computacional inicial se aproxima da final. Este comportamento da escalabilidade e sua similaridade com a condição homogênea puderam ser mais facilmente percebidos devido à classificação ordenada dos trabalhadores.

Essa similaridade entre os resultados mostrados aqui e os obtidos com a aplicação das métricas homogêneas de isoeficiência e isovelocidade, confirmam a adequação da métrica da isoeficiência de speedup em caracterizar a escalabilidade de um SPPV heterogêneo.

Capítulo 7

Conclusões e trabalhos futuros

Neste trabalho foi abordado o desafio na área de sistemas de processamento paralelo relacionado à definição e avaliação da escalabilidade em sistemas homogêneos e heterogêneos de processamento paralelo virtual. A seguir é mostrado como os objetivos do trabalho, listados na seção 1.1, foram alcançados.

Primeiramente foram avaliadas métricas de escalabilidade para sistemas homogêneos e heterogêneos presentes na literatura, onde duas das métricas homogêneas, isoeficiência e isovelocidade, foram selecionadas para uma aplicação prática em duas plataformas de processamento paralelo virtual, JoiN e JPVM, ambas baseadas em Java. O algoritmo Pi-Monte Carlo, foi utilizado na aplicação dessas métricas, que possui características como baixa comunicação, alto grau de paralelismo e *overhead* por processador independente de carga. Os resultados dos testes em sistemas homogêneos permitiram:

- classificar ambas as combinações algoritmo-plataforma paralela (CAPP´s) como escaláveis segundo a ótica das duas métricas utilizadas, uma vez que foi possível encontrar valores de carga de trabalho capazes de manter constante o parâmetro de desempenho, frente a uma variação no número de processadores;

- entender que a escalabilidade calculada reflete o quão distante uma CAPP está da situação ideal (onde para uma carga fixa, o tempo de execução é exatamente o inverso da variação no número de trabalhadores);

- identificar que a escolha do valor de parâmetro de desempenho a ser mantido constante exerce real influência no valor de escalabilidade da CAPP;

- perceber a dependência entre a escalabilidade da CAPP homogênea e o número inicial de processadores da plataforma paralela utilizada;

- verificar que a utilização de um algoritmo com características próximas às ideais permite extrair o máximo desempenho de uma CAPP homogênea, atingindo assim valores de até 0,9 de escalabilidade;

Essas informações serviram para uma avaliação mais detalhada das métricas e permitiram identificar as características desejáveis em uma métrica de escalabilidade, tanto do ponto de vista da formulação, quanto da aplicação prática, o que auxiliou na proposta de uma nova métrica para sistemas heterogêneos de processamento paralelo virtual.

O foco principal da proposta da nova métrica foi garantir que ela fosse eficaz e de fácil aplicação prática em sistemas de processamento paralelo virtual. Para isso as abordagens consideradas levaram em conta características que podem estar presentes em ambiente reais desse tipo como: comunicação lenta, heterogeneidade das máquinas e alta carga de processamento. A métrica proposta, isoeficiência de *speedup*, utiliza como parâmetro principal o *speedup* que é certamente um dos conceitos mais difundidos em processamento paralelo. E para superar as dificuldades de obtenção e visualização do *speedup* em ambientes heterogêneos, foi proposta uma abordagem utilizando os desempenhos relativos (*fdr´*s) dos trabalhadores. Uma metodologia prática e viável foi proposta para cálculo dos *fdr´s* baseada na execução de um conjunto padrão de *benchmarks* que permite estimar a capacidade computacional no momento em que os trabalhadores entram na plataforma.

Foram apresentados testes para validar a adequação da proposta na avaliação da escalabilidade de uma combinação algoritmo-plataforma paralela heterogênea. Conforme pôde ser verificado, a utilização da eficiência de *speedup* como parâmetro de desempenho permitiu caracterizar a escalabilidade da CAPP, assim como quantificá-las para as variações de capacidade

computacional consideradas. O uso dos *fdr´s* para estimar a *speedup* ideal mostrou-se eficaz, assim como a estratégia de ordenação dos trabalhadores segundo seus *fdr´s*, fato que auxiliou na análise e visualização dos resultados. Os testes com a métrica da isoeficiência de *speedup* em um ambiente heterogêneo permitiram ainda:

- classificar a CAPP heterogênea com JoiN como escalável, uma vez que, frente a uma variação na capacidade computacional, foi possível encontrar valores de carga de trabalho capazes de manter constante a eficiência de *speedup*, utilizada nesta métrica como parâmetro de desempenho;

- identificar, de maneira similar ao que foi constatado utilizando as métricas para sistemas homogêneos, que existe uma dependência entre a escalabilidade de uma CAPP heterogênea e a situação inicial da capacidade computacional do sistema paralelo adotado;

- constatar que em ambientes heterogêneos, mesmo utilizando um algoritmo com características próximas às ideais, é mais difícil utilizar eficientemente os recursos do sistema e, portanto, atingir valores altos de escalabilidade;

Através dessas informações, e por apresentar resultados similares com os obtidos na aplicação das métricas homogêneas, a métrica da isoeficiência de *speedup* proposta neste trabalho foi considerada adequada para caracterizar a escalabilidade de SPPV heterogêneos de maneira prática e suficientemente confiável.

Contudo, este trabalho não tem a pretensão de apresentar uma solução definitiva para a avaliação da escalabilidade em sistemas heterogêneos de processamento paralelo. Existem algumas melhorias e análises que podem ser buscadas futuramente, tais como:

- proposta de uma metodologia para a obtenção da escalabilidade de uma CAPP que seja menos dependente da variação de carga na aplicação;

- criação de uma ferramenta automatizada capaz de utilizar os conceitos da métrica de isoeficiência na análise da escalabilidade e que possa ser utilizada em SPPV;

- avaliação do comportamento da métrica com outras situações de escalonamento de tarefas e outras classes de aplicação paralela que apresentem características de comunicação, paralelismo e *overhead* diferentes da de Pi-Monte Carlo;

Referências Bibliográficas

[1] D. Nussbaum, A. Agarwal. “Scalability of Parallel Machines**”**, In Communications of the ACM, 34(3):57-61, 1991.

[2] A.Y. Grama, A. Gupta, V. Kumar. “Isoefficiency: Measuring the scalability of parallel algorithms and architectures”, IEEE Parallel & Distributed Technology, 1(3):12-21, 1993.

[3] X. Zhang, Y. Yan, K. He. “Latency Metric: An experimental method for measuring and evaluating parallel program and architecture scalability”, Journal of Parallel Distributed Computing, 22:392-410, 1994.

[4] X. Sun, D.T. Rover. “Scalability of parallel algorithm-machine combinations”, IEEE Transaction on Parallel and Distributed Systems, 5(6):599-613, 1994.

[5] J.R. Zorbas, D.J. Reble, R.E. vanKooten. “Measuring the scalability of parallel computer systems”, ACM/IEEE Conference on Supercomputing, 832-841, 1989.

[6] A.V. Srinivas, D. Janakiram. “A model for characterizing the scalability of distributed systems”, ACM SIGOPS Operating Systems Review, 39(3):64-71, 2005

[7] G.M. Amdahl. "Validity of the single processor approach to achieving large scale computing capabilities" AFIPS Spring Joint Computer Conference, 30:483-485, 1967.

[8] X. Sun, M. Lionel. “Scalable Problems and Memory-Bounded Speedup”, Journal of Parallel and Distributed Computing, 19(1):27-37, 1993

[9] J.L. Gustafson. “Reevaluating Amdahl's law”, Communications of the ACM, 31(5):532-533, 1988.

[10] E.J.H. Yero, F.O. Lucchese, F.S.S., M. von Zuben, M.A.A. Henriques. “Join: The implementation of a java based massively parallel grid”, Future Generation Computer Systems: Parallel computing technologies, 21(5):791-810, 2005.

[11] A.J. Ferrari. “JPVM: Network Parallel Computing in Java”, Technical Report CS-97-29, Department of Computer Science, University of Virginia, Charlottesville, 1997.

[12] P. Beckmann. “A History of Pi”. Golem Press, 1977

[13] R. Ramanujam. "A foundation for parallel programming", disponível em http://www.imsc.res.in/~kabru/parapp/jam.pdf (último acesso em dezembro/2009)

[14] M. Quinn. “Parallel Programming in C with MPI and OpenMP”, McGraw-Hill, 2004

[15] J.L. Bosque, L.P. Perez. “Theoretical scalability analysis for heterogeneous clusters”, IEEE International Symposium on Cluster Computing and the Grid, 285-292, 2004

[16] A.Ya. Kalinov. ”Scalability of Heterogeneous Parallel Systems”, Programming and Computing Software, 32(1):1-7, 2006

[17] Y. Chen, X. Sun, M. Wu. “Algorithm-system scalability of heterogeneous computing”, Journal of Parallel and Distributed Computing, 68(11):1403-1412, 2008

[18] L.P. Perez, J.L. Bosque. “An efficiency and scalability model for heterogeneous clusters”, IEEE International Conference on Cluster Computing, 427, 2001

[19] P. Jogalekar, M. Woodside. “Evaluating the Scalability of Distributed Systems”, IEEE Transactions on Parallel and Distributed Systems, 11(6):589-603, 2000

[20] J. Keller, C. Kessler, J. Traeff. “Practical PRAM Programming”, Wiley, 2000

[21] R.W. Hockney. “Parametrization of computer performance”, Parallel Computing, 5, 1987

[22] K. Hwang. “Advanced Computer Architecture: Parallelism, Scalability, Programmability”, McGraw-Hill, 1993

[23] M.A.A. Henriques. "A proposal for java based massively parallel processing on the web", First Annual Workshop on Java for High-Performance Computing, ACM International Conference on Supercomputing, 59–66, 1999

[24] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V.S. Sunderam. “PVM: Parallel Virtual Machine”, MIT Press, 1994

[25] F. de O.Lucchese, “Um Mecanismo para Distribuição de Carga em Ambientes Virtuais de Computação Maciçamente Paralela”, dissertação de mestrado, Faculdade de Engenharia Elétrica e de Computação, Unicamp, 2002

[26] R.S. Barr, B.L. Hickman. “Reporting Computational Experiments with Parallel Algorithms: Issues, Measures, and Experts' Opinions”, ORSA Journal on Computing, 5(1):2-18,1993

Apêndice A

Implementações do algoritmo Pi-Monte Carlo em sistemas homogêneos

Para aumentar o grau de automatização dos testes, foi criada a classe *PiMonteCarlo_Constants* com o objetivo de manter os valores das cargas de trabalho aplicadas em cada instância da plicação. As posições do vetor APP_LOAD armazenam a carga de trabalho da instância da aplicação correspondente ao número do índice desse vetor, como mostrado na Figura 25.

```
1  public class PIMonteCarlo_Constants {
2
3      //Application load table, ApplicationID = ArrayIndex
4      public static final long[] APP_LOAD = {       64000L,
5                                                    256000L,
6                                                    1024000L,
7                                                    4096000L,
8                                                    16384000L,
9                                                    65536000L,
10                                                   262144000L,
11                                                   1048576000L,
12                                                   4194304000L,
13                                                   16777216000L,
14                                                   67108864000L,
15                                                 };
16 }
```

Figura 25 - Classe *PiMonteCarlo_Constants* utilizada por todas as versões de implementações do algoritmo.

Todas as versões implementadas do algoritmo Pi-Monte Carlo utilizaram essa classe para obter a carga de trabalho.

A.1 Versão Serial

A seguir temos a versão serial do algoritmo Pi-Monte Carlo implementada por meio da classe *PiSerial*, mostrada na Figura 26.

```
1  public class PiSerial {
2
3      public static void main(String[] args) {
4
5          int appID = Integer.parseInt(args[0]);
6          System.out.println("AppID = " + appID);
7          System.out.println("Starting task.... "+iterationSize);
8          long t1 = System.currentTimeMillis();
9
10         long iterationSize = PIMonteCarlo_Constants.APP_LOAD[appID];
11         long total = 0L;
12         double x,y;
13         for(long i=0; i<iterationSize; i++) {
14             x = Math.random();
15             y = Math.random();
16             if((x*x+y*y < 1))    total++;
17         }
18         double pi = (double) 4*total/iterationSize;
19
20         long t2 = System.currentTimeMillis();
21         long temp = (t2-t1);
22         System.out.println(temp+"@"+t1+"@SERIALTIME,T1\n");
23         System.out.println("Result receveid. PI= "+pi+".\n");
24     }
25 }
```

Figura 26 - Classe *PiSerial* utilizada na versão serial do algoritmo.

A.2 Versão paralela JoiN

A versão paralela do algoritmo Pi-Monte Carlo implementada para a plataforma JoiN, utiliza três classes Java.

A primeira delas, mostrada na Figura 27 (*DistributeData*), é executada pelo componente coordenador da plataforma e é responsável por montar um vetor (*limits[]*) contendo os dados de entrada para todas as tarefas da execução. Após a finalização desta classe, os dados armazenados em cada posição desse vetor são sequencialmente enviados pelo coordenador aos trabalhadores.

Ao receber o dado de entrada os trabalhadores executam a classe denominada *JoinTask*, mostrada na Figura 28, que realiza o *loop* de geração de coordenadas aleatórias e retorna como

resultado o número de coordenadas que atenderam ao requisito do método de Monte Carlo, discutido no Capítulo 4.

Após receber os resultados de todas as tarefas, a plataforma executa no coordenador a classe *GatherResults,* que coleta os resultados e calcula o valor de Pi, como mostra a Figura 29.

```
1  package br.unicamp.fee.dca.join.applications.PI;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileReader;
6  import java.io.Serializable;
7  import br.unicamp.fee.dca.join.framework.app.AppCode;
8
9  public class DistributeData implements AppCode{
10     public Serializable taskrun(Serializable par) {
11
12         try{
13             String fileName = "conf"+File.separator+
14                               "appManager"+File.separator+"pi_montecarlo.conf";
15             FileReader fr = new FileReader(fileName);
16             BufferedReader br = new BufferedReader(fr);
17             // Parsing pi_montecarlo.conf file
18             //First line: application ID(size)
19             int appID = Integer.parseInt(br.readLine());
20             //Second line: number of works, tasks per work
21             int numOfTasks = Integer.parseInt(br.readLine())*
22                              Integer.parseInt(br.readLine());
23             br.close();
24             fr.close();
25             System.out.println("\nApplication ID: "+appID+
26                                "  Number of tasks: "+numOfTasks);
27             long t1 = System.currentTimeMillis();
28
29             //Start actual distributing process
30             Object[] limits = new Object[numOfTasks];
31             long iterationSize = PIMonteCarlo_Constants.APP_LOAD[appID]/numOfTasks;
32             for(int i=0;i<numOfTasks;i++){
33                 Long[] parameters = new Long[2];
34                 parameters[0] = iterationSize;
35                 parameters[1] = new Long(appID);
36                 limits[i] = parameters;
37             }
38
39             long t2 = System.currentTimeMillis();
40             long temp = (t2-t1);
41             System.out.println("\n"+temp+"@"+t1+"@"+t2+
42                                "@TOTAL_INITIAL_SERIAL_TIME,T1,T2\n");
43             return limits;
44
45         }catch(Exception error){
46             error.printStackTrace();
47             return null;
48         }
49     }
50 }
```

Figura 27 - Classe *DistributeData* executada no coordenador na versão paralela em JoiN.

```
1  package br.unicamp.fee.dca.join.applications.PI;
2
3  import java.io.Serializable;
4  import br.unicamp.fee.dca.join.framework.app.AppCode;
5
6  public class JoinTask implements AppCode{
7      public Serializable taskrun(Serializable par) {
8          long t1 = System.currentTimeMillis();
9          Long[] parameters = (Long[])par;
10         long limit = parameters[0];
11         long total = 0L;
12         System.out.println("Starting task.... "+limit);
13
14         double x,y;
15         for(long i=0; i<limit; i++) {
16             x = Math.random();
17             y = Math.random();
18             if((x*x+y*y < 1))    total++;
19         }
20         Long[] res = new Long[2];
21         res[0] = new Long(total);
22         // Application ID to be used by GatherResults.java
23         res[1] = parameters[1];
24
25         long t2 = System.currentTimeMillis();
26         long temp = (t2-t1);
27         System.out.println("\n"+temp+"@"+t1+"@PARALLELWORK,T1\n");
28         System.out.println("Task processing finished. Found "+total+" numbers.");
29         return res;
30     }
31 }
```

Figura 28 - Classe *JoinTask* executada nos trabalhadores na versão paralela do algoritmo em JoiN.

```
1  package br.unicamp.fee.dca.join.applications.PI;
2
3  import java.io.Serializable;
4  import br.unicamp.fee.dca.join.framework.app.AppCode;
5
6  public class GatherResults implements AppCode{
7      public Serializable taskrun(Serializable par) {
8          long t1 = System.currentTimeMillis();
9          Object[] taskResults = (Object[]) par;
10         long total = 0;
11         int numOfTasks = taskResults.length;
12         int appID = 0;
13         for(int i=0; i<numOfTasks; i++) {
14             Long[] pos =  (Long[])taskResults[i];
15             long subTotal = pos[0];
16             total += subTotal;
17             appID = Integer.parseInt(pos[1].toString());
18         }
19         long iterationSize = PIMonteCarlo_Constants.APP_LOAD[appID]/numOfTasks ;
20         double aux = (double) total/iterationSize;
21         double pi = 4*aux/numOfTasks; // Calculating final result
22         long t2 = System.currentTimeMillis();
23         long temp = (t2-t1);
24         System.out.println("\n"+temp+"@"+t1+"@"+t2+"@TOTAL_FINAL_SERIAL_TIME,T3,T4\n");
25         System.out.println("\nResults receveid from workers. PI= "+pi+".\n");
26         long totalWork = iterationSize*numOfTasks;
27         System.out.println("\nTOTAL WORK = "+totalWork+"\n");
28         return par;
29     }
30 }
```

Figura 29 - Classe *GatherResults* executada no coordenador na versão paralela do algoritmo em JoiN.

A.3 Versão paralela JPVM

A versão paralela do algoritmo Pi-Monte Carlo implementada para a plataforma JPVM, utiliza duas classes. A classe denominada *JpvmTask* é executada nos trabalhadores e nela é realizado o *loop* que gera os pontos aleatórios e obtém o número de coordenadas que atingiram o quadrante do circulo unitário. Como no caso de JPVM a aplicação é responsável pelo envio de dados ao coordenador, a operação de envio do resultado da tarefa ao coordenado está incluída nesta classe, como mostrada na Figura 30. A outra classe é denominada *PiJpvm*, é executada na máquina coordenadora, sendo responsável por: preparar os dados de entradas das tarefas na execução, enviá-los aos trabalhadores, coletar os resultados de todas as tarefas e efetuar o cálculo do valor de Pi. O código dessa classe é mostrado na Figura 31.

```
1  import jpvm.*;
2
3  public class JpvmTask {
4
5      public static void main(String[] args) {
6          try {
7                  jpvmEnvironment jpvm = new jpvmEnvironment();
8                  long t1 = System.currentTimeMillis();
9                  // Get my parent's task id...
10                 jpvmTaskId parent = jpvm.pvm_parent();
11                 long iterationSize = 0L;
12                 jpvmMessage msg = jpvm.pvm_recv(appProtocol.TASK);
13                 if(msg!=null){
14                     iterationSize = msg.buffer.upklong();
15                 }
16                 System.out.println("Starting task.... "+iterationSize);
17
18                 long total = 0L;
19                 double x,y;
20                 for(long i=0; i<iterationSize; i++) {
21                     x = Math.random();
22                     y = Math.random();
23                     if((x*x+y*y < 1))    total++;
24                 }
25
26                 // Send a message to my parent...
27                 jpvmBuffer buf = new jpvmBuffer();
28                 buf.pack(total);
29                 jpvm.pvm_send(buf,parent,appProtocol.RESULT);
30
31                 // Exit from the parallel virtual machine
32                 jpvm.pvm_exit();
33                 long t2 = System.currentTimeMillis();
34                 long temp = (t2-t1);
35                 System.out.println("\n"+temp+"@"+t1+"@PARALLELWORK,T1\n");
36                 System.out.println("Task processing finished. Found "+total+
37                                    " numbers.");
38             }
39             catch (jpvmException jpe) {
40             System.out.println("Error - jpvm exception");
41             }
42     }
43 }
```

Figura 30 - Classe *JpvmTask* executada no trabalhador na versão paralela do algoritmo em JPVM.

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import jpvm.*;
4
5  public class PiJpvm {
6      public static void main(String[] args) {
7          try{
8              String fileName = "pi_montecarlo.conf";
9              FileReader fr = new FileReader(fileName);
10             BufferedReader br = new BufferedReader(fr);
11             // Parsing pi_montecarlo.conf file
12             //First line: application ID(size)
13             int appID = Integer.parseInt(br.readLine());
14             //Second line: number of works, tasks per work
15             int numOfTasks = Integer.parseInt(br.readLine())*
16                              Integer.parseInt(br.readLine());
17             br.close();
18             fr.close();
19             System.out.println("\nAppID:"+appID+" Workers:"+(numOfTasks/3)+
20                                " NumOfTasks:"+numOfTasks);
21             long t1 = System.currentTimeMillis();
22             // Distributing data
23             long workLoad = PIMonteCarlo_Constants.APP_LOAD[appID];
24             long iterationSize = workLoad/numOfTasks;
25             jpvmEnvironment jpvm = new jpvmEnvironment();
26             jpvmTaskId tids[] = new jpvmTaskId[numOfTasks];
27             jpvm.pvm_spawn("JpvmTask", numOfTasks, tids);
28             for(int i=0;i<numOfTasks;i++){
29                 jpvmBuffer buf = new jpvmBuffer();
30                 buf.pack(iterationSize);
31                 jpvm.pvm_send(buf, tids[i], appProtocol.TASK);
32             }
33             long t2 = System.currentTimeMillis();
34             // Gathering data
35             long total = 0L;
36             jpvmMessage firstMsg = jpvm.pvm_recv(appProtocol.RESULT);
37             long t3 = System.currentTimeMillis();
38             total += firstMsg.buffer.upklong();
39             for(int j=1;j<numOfTasks;j++){
40                 jpvmMessage message = jpvm.pvm_recv(appProtocol.RESULT);
41                 total += message.buffer.upklong();
42             }
43             double pi = 4*total/workLoad; // Calculating final result
44             long t4 = System.currentTimeMillis();
45             jpvm.pvm_exit();
46             long temp1 = (t2-t1);
47             long temp2 = (t3-t2);
48             long temp3 = (t4-t3);
49             long temp4 = (t4-t1);
50             long temp5 = temp1+temp3;
51             System.out.println(temp4+"@FULL_EXECUTION_TIME");
52             System.out.println(temp1+"@DISTRIBUTION_TIME");
53             System.out.println(temp2+"@COMMUNICATION+PROCESS_TIME");
54             System.out.println(temp3+"@GATHERING_TIME");
55             System.out.println(temp5+"@SERIAL_TIME");
56             System.out.println("Results receveid from workers. PI= "+pi);
57             long totalWork = iterationSize*numOfTasks;
58             System.out.println("TOTAL WORK = "+totalWork);
59         }catch(Exception error){
60             error.printStackTrace();
61         }
62     }
63 }
```

Figura 31 - Classe *PiJpvm* executada no coordenador na versão paralela do algoritmo em JPVM.

Apêndice B

Dados obtidos nos testes com sistemas homogêneos

A seguir temos os dados completos obtidos na execução dos testes em sistemas homogêneos utilizando a aplicação Pi-Monte Carlo nas plataformas JoiN e JPVM.

B.1 Execução serial

A Tabela 13 apresenta os tempos em *ms* medidos na execução da versão serial do algoritmo Pi-Monte Carlo em uma máquina com a seguinte configuração: *Pentium* 4, 1.8GHz, 1Gb RAM, Sistema Operacional Linux Ubuntu v9.04 e rede de 100 Mbps. O valor de T_{med} é obtido através da média aritmética dos tempos de cinco execuções realizadas para cada instância. Também é mostrado o desvio padrão relativo à T_{med}.

Tabela 13 - Tempos de execução serial para o algoritmo Pi-Monte Carlo nos testes com sistemas homogêneos.

Serial								
Instância da aplicação	Carga total de trabalho (nº iterações)	Tempos de execução serial (T_s) (*ms*)						Desvio padrão relativo(%)
		T1	T2	T3	T4	T5	T_{med}	
00	64.103	39	37	36	41	40	38,6	5,372
01	256.103	83	82	80	78	79	80,4	2,579
02	1024.103	248	249	250	248	251	249,2	0,523
03	4096.103	907	908	910	911	910	909,2	0,181
04	16384.103	3573	3572	3573	3574	3574	3573,2	0,023
05	65536.103	14235	14242	14238	14245	14241	14240,2	0,027
06	262.106	56765	56769	56768	56765	56764	56766,2	0,004
07	1049.106	226851	226843	226845	226825	226814	226835,6	0,007
08	4194.106	907107	907108	907029	907843	907031	907223,6	0,038
09	16777.106	3628037	3628152	3627943	3628182	3627897	3628042,2	0,003
10	67109.106	14509009	14505590	14505286	14506275	14505781	14506388,2	0,010

Obs: Desvio padrão relativo (%) = desvio padrão/T_{med}

B.2 Execução paralela - JoiN

A Tabela 14 apresenta os tempos medidos na execução da versão paralela do algoritmo Pi-Monte Carlo executada na plataforma JoiN. Esta tabela está dividida em quatro sub-tabelas onde cada uma corresponde a um grupo de trabalhadores utilizado na execução, iniciando em 2 e atingindo 16 trabalhadores. Cada sub-tabela está organizada da mesma forma que a Tabela 13.

Tabela 14 - Tempos de execução paralela para o algoritmo Pi-Monte Carlo na plataforma JoiN em ambiente homogêneo.

2 trabalhadores								
Instância da aplicação	Carga total de trabalho (nº iterações)	Tempos de execução paralela (T_p) (*ms*)						Desvio padrão relativo(%)
		T1	T2	T3	T4	T5	T_{med}	
00	64.103	185	132	124	128	109	135,60	21,35
01	256.103	139	104	159	129	133	132,80	14,91
02	1024.103	227	226	220	229	223	225,00	1,57
03	4096.103	575	623	563	580	555	579,20	4,56
04	16384.103	2036	1887	1881	1868	1899	1914,20	3,60
05	65536.103	7098	7144	7162	7159	7118	7136,20	0,39

06	262.106	28162	28141	28079	28119	28089	28118,00	0,12
07	1049.106	112127	112036	112138	112171	112079	112110,20	0,05
08	4194.106	447928	447923	448811	447920	447960	448108,40	0,09
09	16777.106	1791603	1791594	1791460	1792025	1792339	1791804,20	0,02
4 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	121	139	111	152	142	133,00	12,50
01	256.103	164	163	134	169	207	167,40	15,57
02	1024.103	205	293	222	231	222	234,60	14,48
03	4096.103	352	406	436	415	499	421,60	12,62
04	16384.103	1007	1056	1051	1097	1015	1045,20	3,45
05	65536.103	3640	3665	3647	3698	3686	3667,20	0,68
06	262.106	14185	14268	14466	14294	14145	14271,60	0,87
07	1049.106	56166	56132	56272	56243	56230	56208,60	0,10
08	4194.106	224116	224469	227167	224476	226423	225330,20	0,61
09	16777.106	896998	896070	896969	896500	896031	896513,60	0,05
8 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	219	151	171	166	177	176,80	14,41
01	256.103	146	184	169	152	140	158,20	11,40
02	1024.103	174	174	152	175	182	171,40	6,62
03	4096.103	266	254	258	253	287	263,60	5,33
04	16384.103	644	661	621	652	648	645,20	2,31
05	65536.103	1963	1939	1909	1932	1928	1934,20	1,01
06	262.106	7381	7447	7381	7383	7210	7360,40	1,21
07	1049.106	28261	28166	28225	28147	28233	28206,40	0,17
08	4194.106	112139	112138	112133	112194	112157	112152,20	0,02
09	16777.106	448198	460660	448109	448349	448452	450753,60	1,23
16 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	295	297	236	297	248	274,60	10,95
01	256.103	181	220	265	222	250	227,60	14,17
02	1024.103	180	242	195	196	171	196,80	13,90
03	4096.103	303	339	304	265	205	283,20	17,99

04	16384.103	398	449	387	495	394	424,60	10,93
05	65536.103	1004	1053	1058	1128	1096	1067,80	4,39
06	262.106	3672	3810	3967	3772	3787	3801,60	2,80
07	1049.106	14954	14336	14299	14355	14347	14458,20	1,92
08	4194.106	56775	56598	56525	56433	56520	56570,20	0,23
09	16777.106	226800	225538	225577	226254	225987	226031,20	0,23
10	67109.106	902324	903002	904380	903102	902587	903079,00	0,09

B.3 Execução paralela - JPVM

A Tabela 15 apresenta os tempos medidos na execução da versão paralela do algoritmo Pi-Monte Carlo executada na plataforma JPVM e está organizada da mesma maneira que a Tabela 14.

Tabela 15 - Tempos de execução paralela para o algoritmo Pi-Monte Carlo na plataforma JPVM em ambiente homogêneo.

2 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	750	688	722	718	695	714,60	3,44
01	256.103	723	869	739	896	727	790,80	10,68
02	1024.103	819	808	808	845	813	818,60	1,89
03	4096.103	1132	1179	1160	1184	1233	1177,60	3,15
04	16384.103	2501	2517	2514	2512	2594	2527,60	1,49
05	65536.103	7846	7876	7843	8173	7867	7921,00	1,79
06	262.106	28906	28892	28972	28473	28641	28776,80	0,73
07	1049.106	115944	115952	113856	113102	113128	114396,40	1,27
08	4194.106	458741	466241	466298	455711	449863	459370,80	1,54
09	16777.106	1796779	1799651	1796829	1796910	1797054	1797444,60	0,07
4 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	806	759	735	730	741	754,20	4,11
01	256.103	785	757	742	755	824	772,60	4,24
02	1024.103	796	794	818	797	787	798,40	1,46
03	4096.103	1008	1001	962	1041	977	997,80	3,05
04	16384.103	1711	1698	1694	1677	1723	1700,60	1,03
05	65536.103	4516	4433	4403	4379	4399	4426,00	1,22
06	262.106	15065	14895	15519	15203	15012	15138,80	1,58

07	1049.106	57142	57023	57043	57047	57023	57055,60	0,09
08	4194.106	225434	225390	225516	225584	225415	225467,80	0,04
09	16777.106	899027	899160	899000	901697	899044	899585,60	0,13
8 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	816	789	764	766	775	782,00	2,74
01	256.103	789	793	790	809	792	794,60	1,03
02	1024.103	847	807	790	794	787	805,00	3,07
03	4096.103	950	917	929	937	911	928,80	1,68
04	16384.103	1298	1275	1306	1337	1278	1298,80	1,93
05	65536.103	2628	2670	2602	2643	2591	2626,80	1,21
06	262.106	7944	8126	8142	8145	7971	8065,60	1,23
07	1049.106	29068	29065	29034	29229	29101	29099,40	0,26
08	4194.106	113399	113457	113477	113455	113446	113446,80	0,03
09	16777.106	450793	450781	450773	450764	450742	450770,60	0,00
16 trabalhadores								
Instância da aplicação	**Carga total de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.103	852	799	799	905	885	848,00	5,73
01	256.103	929	845	959	858	932	904,60	5,54
02	1024.103	851	864	905	892	847	871,80	2,93
03	4096.103	937	917	917	921	920	922,40	0,91
04	16384.103	1079	1083	1053	1072	1087	1074,80	1,25
05	65536.103	1843	1774	1819	1853	1792	1816,20	1,84
06	262.106	4524	4518	4518	4533	4538	4526,20	0,20
07	1049.106	15041	15118	15182	15028	14985	15070,80	0,52
08	4194.106	57149	57214	57256	57308	57194	57224,20	0,11
09	16777.106	225493	226358	226789	225508	225819	225993,40	0,25
10	67109.106	902721	901321	900547	901207	900874	901334,00	0,09

Podemos observar em nas tabelas mostradas que o desvio padrão relativo foi baixo maioria das situações de carga, o que realça a confiabilidade nas médias obtidas. Os maiores desvios ocorreram em situações onde a ordem de grandeza dos tempos medidos era pequena e portanto mais susceptíveis à variações introduzidas pelo ambiente de testes.

Apêndice C

Gráficos - Sistemas Homogêneos

A seguir temos os resultados da interpolação logarítmica, realizada por meio da função "*logarithmic Trendline"* do programa Microsoft Office Excel 2007, dos pontos obtidos nos testes efetuados com as CAPP´s homogêneas com JoiN e JPVM, para as métricas da isovelocidade e isoeficiência.

C.1 Métrica isovelocidade

A Figura 32 e Figura 33 mostram individualmente para cada grupo de trabalhadores nas plataformas JoiN e JPVM, respectivamente, as curvas de V_{um} resultantes da interpolação logarítmica utilizada para obtenção dos valores de cargas necessários ao cálculo da escalabilidade na métrica de isovelocidade.

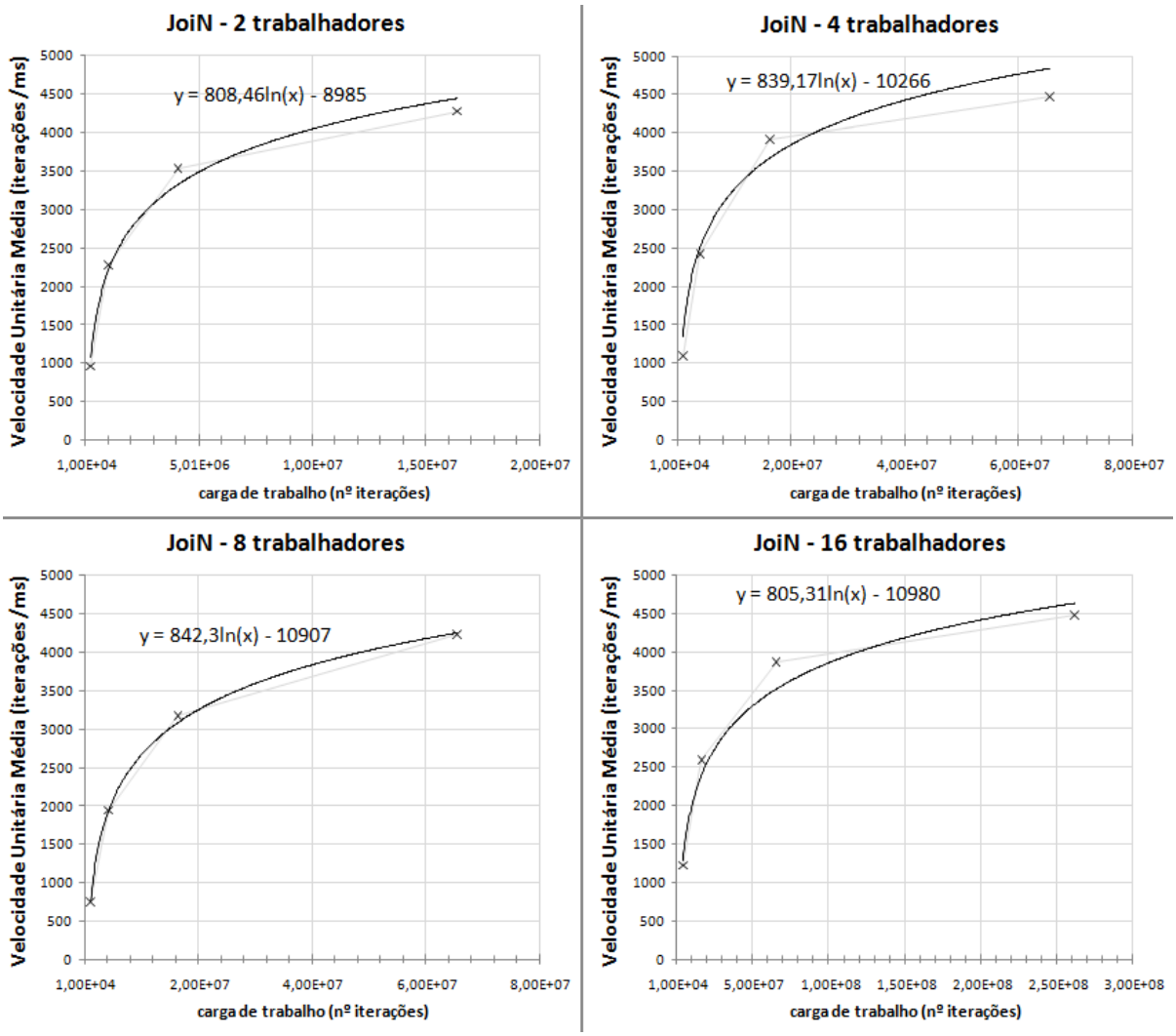

Figura 32 - Interpolação logarítmica das curvas de V_{um} na plataforma JoiN.

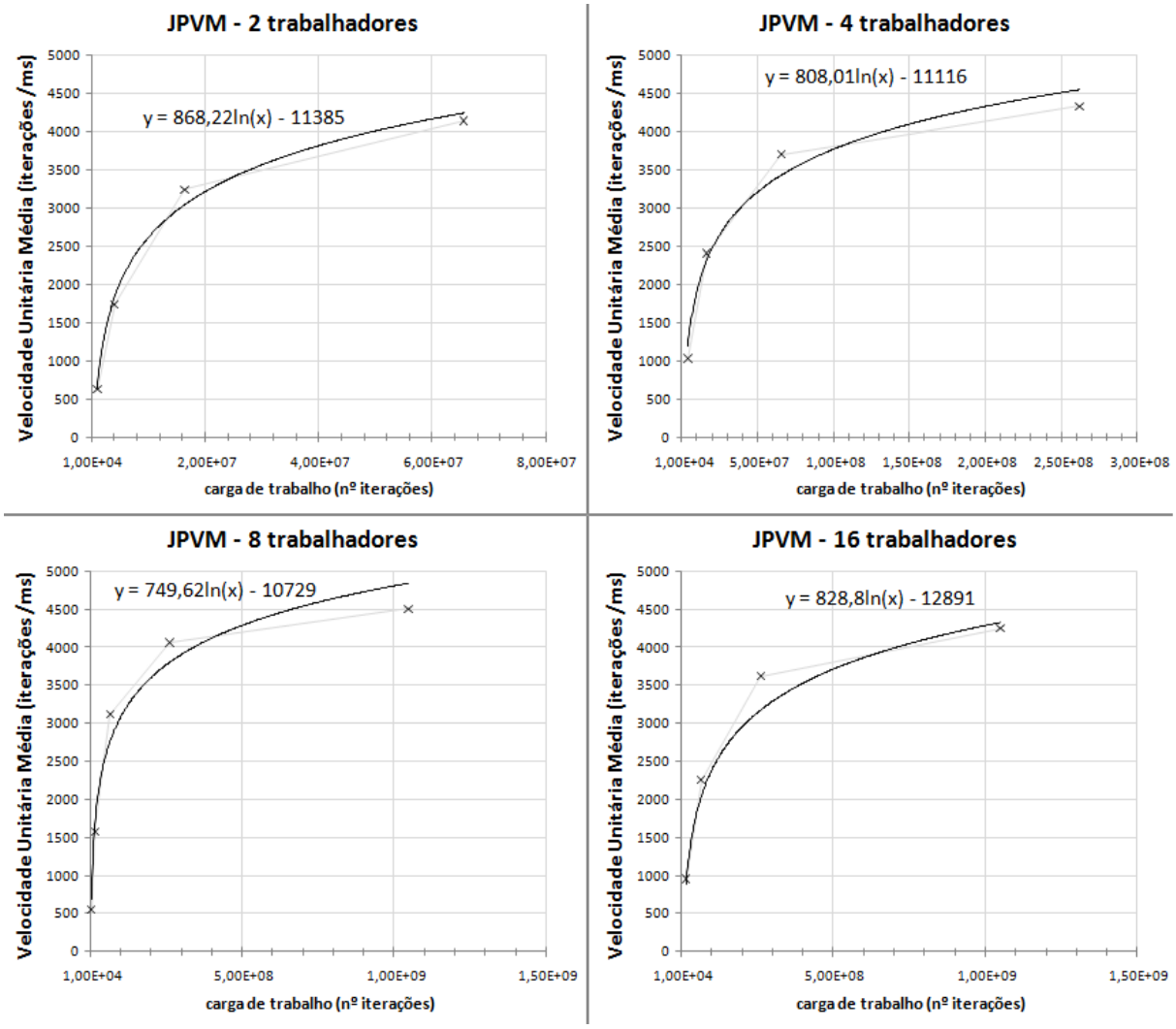

Figura 33 - Interpolação logarítmica das curvas de V_{um} na plataforma JPVM.

C.2 Métrica isoeficiência

A Figura 34 e Figura 35 mostram individualmente para cada grupo de trabalhadores nas plataformas JoiN e JPVM, respectivamente, as curvas de eficiência paralela resultantes da interpolação logarítmica utilizada para obtenção dos valores de cargas necessários para manter os valores selecionados de eficiência constantes.

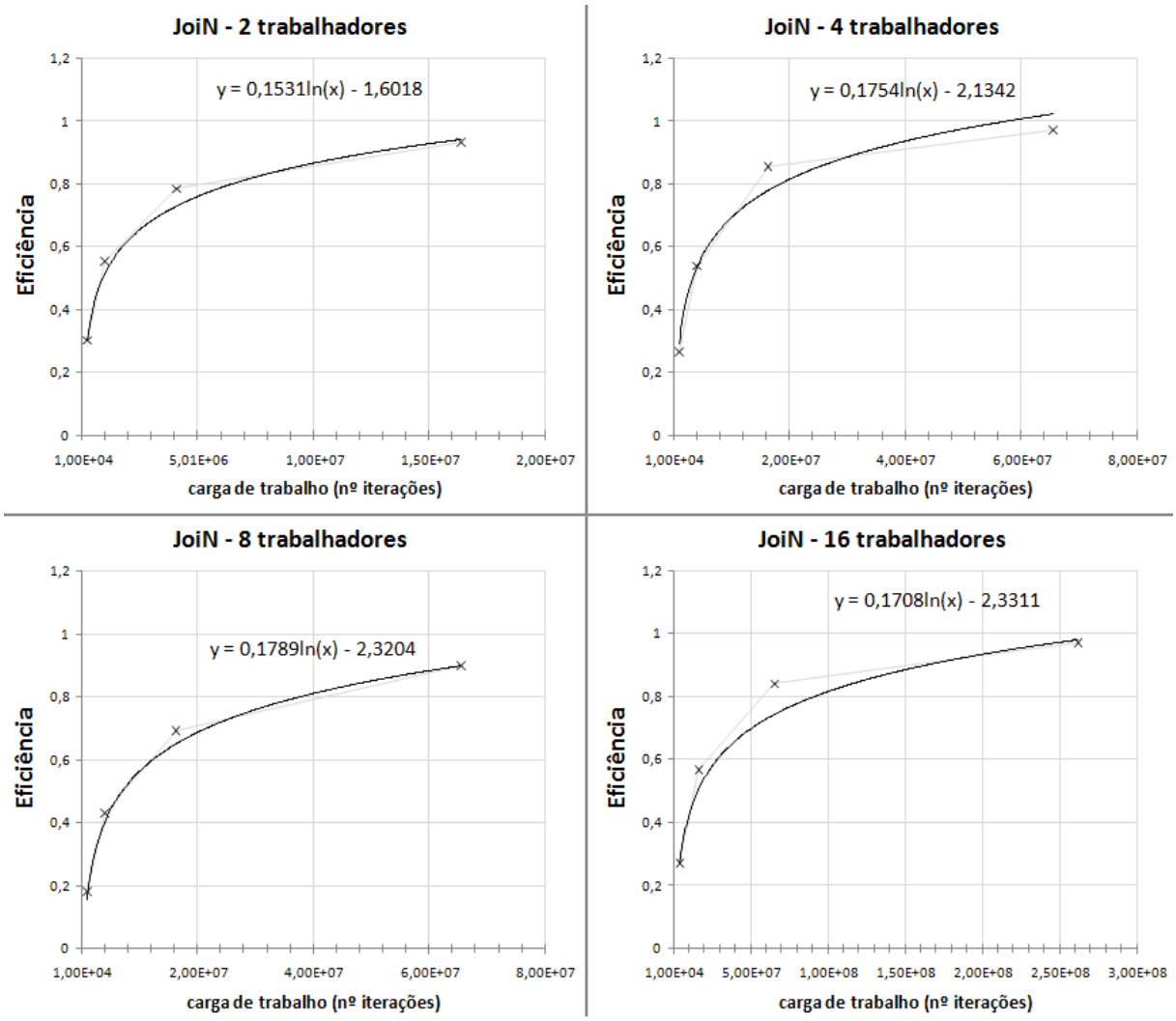

Figura 34 - Interpolação logarítmica das curvas de eficiência paralela na plataforma JoiN.

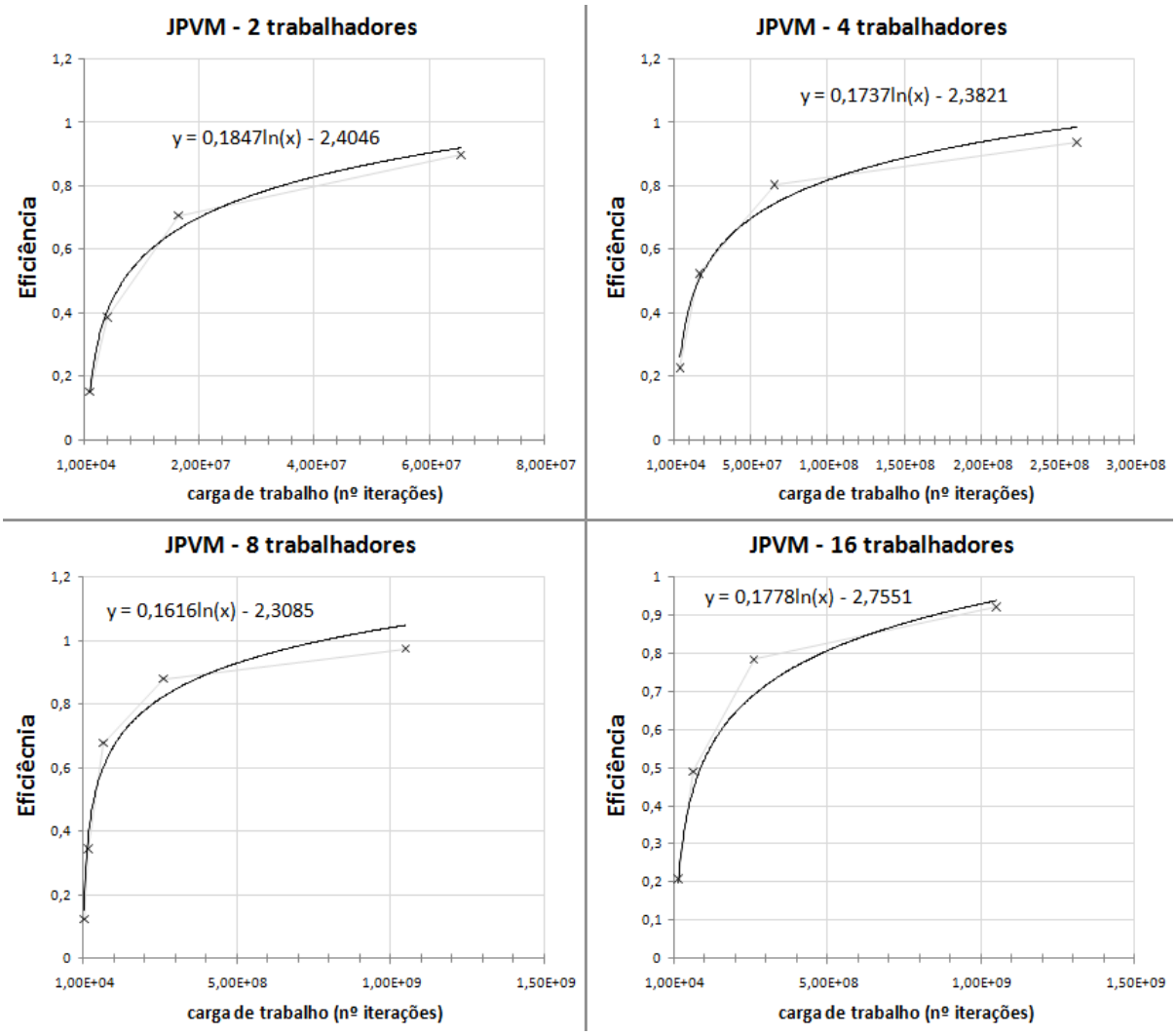

Figura 35 - Interpolação logarítmica das curvas de eficiência paralela na plataforma JPVM.

Apêndice D

Dados obtidos nos testes com sistemas heterogêneos

A seguir temos os dados obtidos na execução dos testes em sistemas heterogêneos utilizando a aplicação Pi-Monte Carlo na plataforma JoiN.

D.1 Execução serial

A Tabela 16 apresenta os tempos em *ms* medidos na execução da versão serial do algoritmo Pi-Monte Carlo na máquina “le22-4” que obteve o melhor *fdr* dentre as máquinas pertencentes ao grupo selecionado para participar dos testes com sistema heterogêneo. O valor de T_{med} é obtido através da média aritmética dos tempos de cinco execuções realizadas para cada instância. Também é mostrado o desvio padrão relativo à T_{med}.

Tabela 16 - Tempos de execução serial para o algoritmo Pi-Monte Carlo na máquina de maior *fdr* (le22-4).

Serial (le22-4)								
Instância da aplicação	Carga total de trabalho (nº iterações)	Tempos de execução serial (T_s) (*ms*)						Desvio padrão relativo(%)
		T1	T2	T3	T4	T5	T_{med}	
00	64.10^3	58	46	62	56	60	56,40	11,04
01	256.10^3	96	102	84	84	96	92,40	8,71
02	1024.10^3	252	238	238	238	248	242,80	2,77
03	4096.10^3	862	866	854	870	868	864,00	0,73
04	8192.10^3	1690	1674	1676	1692	1684	1683,20	0,48
05	16384.10^3	3326	3330	3340	3324	3346	3333,20	0,28
06	32768.10^3	6620	6628	6616	6700	6604	6633,60	0,57
07	65536.10^3	13174	13196	13180	13304	13220	13214,80	0,40
08	131072.10^3	26380	26374	26320	26376	26370	26364,00	0,09
09	262144.10^3	52694	52746	52796	52898	52608	52748,40	0,21
10	1049.10^6	210396	210582	211032	210406	210356	210554,40	0,13
11	4194.10^6	844898	844012	844078	843980	843142	844022,00	0,07
12	8389.10^6	1688068	1683226	1685408	1683842	1686118	1685332,40	0,11

Obs: Desvio padrão relativo (%) = desvio padrão/T_{med}

D.2 Execução paralela – JoiN ambiente heterogêneo

A Tabela 17 apresenta os tempos medidos na execução da versão paralela do algoritmo Pi-Monte Carlo executada na plataforma JoiN em ambiente heterogêneo. Esta tabela está dividida em quatro sub-tabelas onde cada uma corresponde a um grupo de trabalhadores utilizado na execução, iniciando em 2 e chegando a 12 trabalhadores. Os nomes das máquinas utilizadas em cada grupo estão referenciados no início de cada sub-tabela.

Tabela 17 - Tempos de execução paralela para o algoritmo Pi-Monte Carlo na plataforma JoiN em ambiente heterogêneo.

Grupo 1 (2 trabalhadores) (le22-4, le22-2)								
Instância da aplicação	Carga de trabalho (nº iterações)	Tempos de execução paralela (T_p) (*ms*)						Desvio padrão relativo(%)
		T1	T2	T3	T4	T5	T_{med}	
00	64.10^3	1717	1760	1818	1771	1794	1772,00	2,143
01	256.10^3	1750	1749	1778	1933	1828	1807,60	4,265
02	1024.10^3	2008	1968	2021	1969	1928	1978,80	1,861

03	4096.10^3	2936	2945	2969	2861	2913	2924,80	1,399
04	8192.10^3	3639	3497	3544	3597	3631	3581,60	1,684
05	16384.10^3	3613	3490	3811	3711	3608	3646,60	3,311
06	32768.10^3	4363	4430	4533	4457	4347	4426,00	1,700
07	65536.10^3	7219	7202	7237	7269	7157	7216,80	0,577
08	131072.10^3	13797	13611	13631	13586	13814	13687,80	0,795
09	262144.10^3	26774	26751	26755	26713	26723	26743,20	0,093
10	1049.10^6	104614	104669	104607	104579	104641	104622,00	0,033
11	4194.10^6	419700	419410	419519	419303	419207	419427,80	0,046
Grupo 2 (4 trabalhadores) (le22-4, le22-2, le22-3, le22-5)								
Instância da aplicação	**Carga de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.10^3	1295	1349	1316	1326	1356	1328,40	1,867
01	256.10^3	1288	1343	1330	1440	1342	1348,60	4,136
02	1024.10^3	1414	1364	1374	1459	1386	1399,40	2,731
03	4096.10^3	1989	1997	2017	1948	2002	1990,60	1,302
04	8192.10^3	2048	1963	2109	2073	2051	2048,80	2,626
05	16384.10^3	2398	2388	2349	2481	2374	2398,00	2,082
06	32768.10^3	3046	3045	3011	3065	3080	3049,40	0,850
07	65536.10^3	4465	4527	4597	4453	4494	4507,20	1,282
08	131072.10^3	7794	7834	7666	7729	7690	7742,60	0,909
09	262144.10^3	14373	14414	14435	14393	14420	14407,00	0,168
10	1049.10^6	54606	54267	53587	53708	54526	54138,80	0,864
11	4194.10^6	210520	210620	210398	210710	210702	210590,00	0,063
Grupo 3 (8 trabalhadores) (le22-4, le22-2, le22-3, le22-5, viviane, bishop, tuck, vortigen)								
Instância da aplicação	**Carga de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.10^3	1094	1093	1204	1192	1103	1137,20	4,907
01	256.10^3	1195	1292	1277	1248	1235	1249,40	3,033
02	1024.10^3	1228	1337	1380	1288	1204	1287,40	5,697
03	4096.10^3	1459	1287	1339	1461	1285	1366,20	6,465
04	8192.10^3	1668	1608	1790	1634	1728	1685,60	4,369
05	16384.10^3	2058	2013	2006	1916	2121	2022,80	3,721
06	32768.10^3	2474	2624	2561	2563	2585	2561,40	2,149
07	65536.10^3	3747	3827	3580	3738	3726	3723,60	2,405
08	131072.10^3	6437	6468	6325	6265	6470	6393,00	1,453
09	262144.10^3	11568	11609	11527	11665	11612	11596,20	0,447

10	1049.10^6	40327	40364	40186	40548	40613	40407,60	0,428
11	4194.10^6	156353	156664	156077	156158	156383	156327,00	0,146
Grupo 4 (12 trabalhadores) (le22-4, le22-2, le22-3, le22-5, viviane, bishop, tuck, vortigen, iff, eps, dxf, pict)								
Instância da aplicação	**Carga de trabalho (nº iterações)**	**Tempos de execução paralela (T_p) (*ms*)**						**Desvio padrão relativo(%)**
		T1	**T2**	**T3**	**T4**	**T5**	**T_{med}**	
00	64.10^3	1470	1441	1666	1553	1849	1595,80	10,423
01	256.10^3	1559	1789	1621	1492	1941	1680,40	10,870
02	1024.10^3	1462	1493	1839	1879	1951	1724,80	13,309
03	4096.10^3	1939	1993	2008	1987	2001	1985,60	1,372
04	8192.10^3	1862	1829	1848	2000	1840	1875,80	3,756
05	16384.10^3	1956	2153	2128	2157	2275	2133,80	5,369
06	32768.10^3	2216	2294	2734	2311	2566	2424,20	8,971
07	65536.10^3	3456	3401	3400	3409	3480	3429,20	1,067
08	131072.10^3	5794	5745	5718	5853	5773	5776,60	0,891
09	262144.10^3	10291	10435	10397	10148	10168	10287,80	1,263
10	1049.10^6	34747	35003	35127	34821	35316	35002,80	0,658
11	4194.10^6	132904	133648	133553	132711	133233	133209,80	0,303
12	8389.10^6	263842	263595	263763	262773	262898	263374,20	0,191

Podemos observar nas tabelas mostradas que o desvio padrão relativo foi baixo maioria das situações de carga, o que realça a confiabilidade das médias obtidas. Os maiores desvios ocorreram em situações onde a ordem de grandeza dos tempos medidos era pequena e portanto mais susceptíveis à variações introduzidas pelo ambiente de testes.

Apêndice E

Gráficos – Sistema Heterogêneo

A seguir temos os resultados da interpolação logarítmica, realizada por meio da função "*logarithmic Trendline"* do programa Microsoft Office Excel 2007, dos pontos obtidos nos testes efetuados na CAPP heterogênea com JoiN, para a métrica da isoeficiência de *speedup*.

A Figura 36 mostra individualmente para cada grupo de trabalhadores na plataforma JoiN, as curvas de E_s resultantes da interpolação logarítmica utilizada para obtenção dos valores de cargas necessários ao cálculo da escalabilidade.

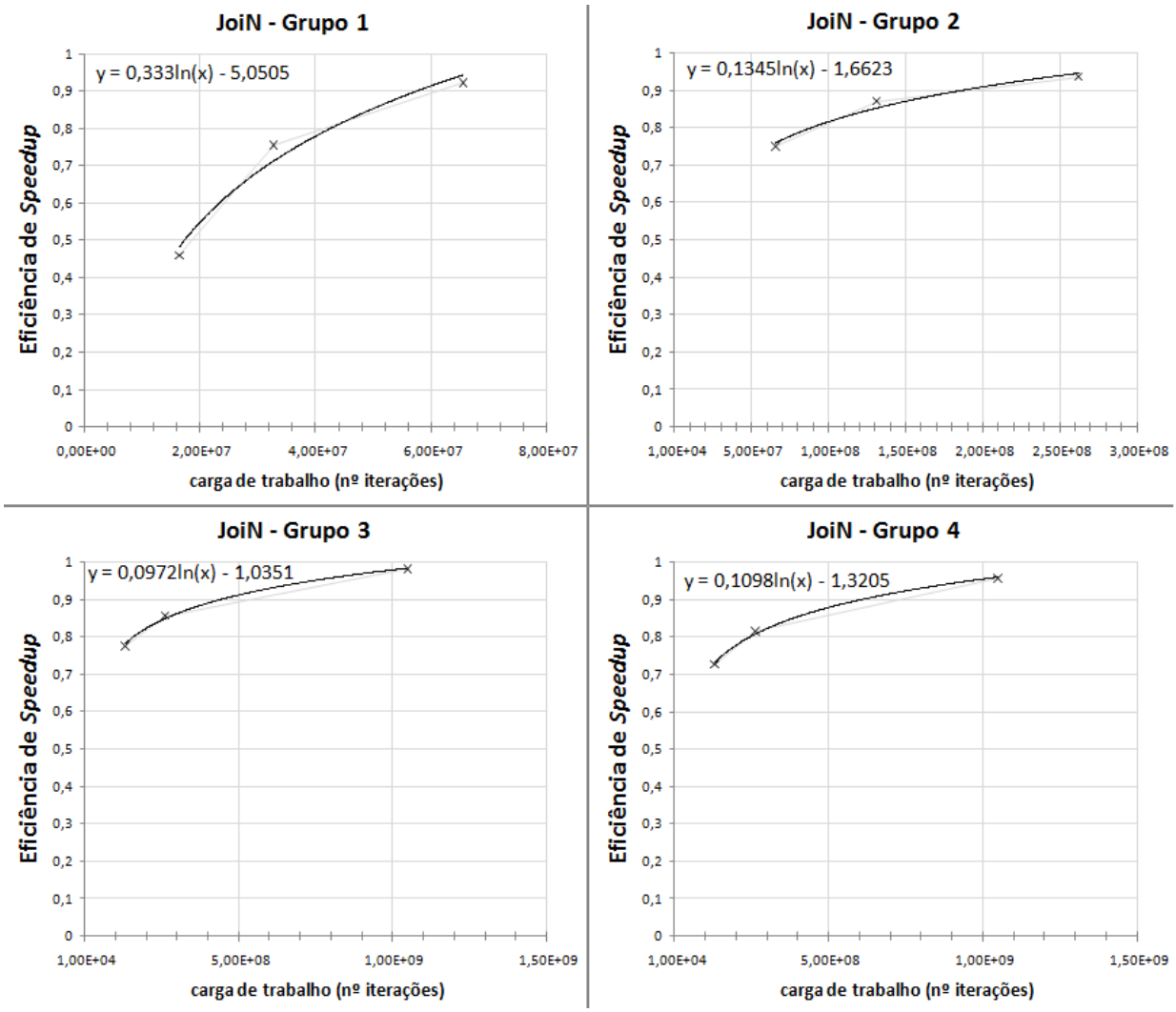

Figura 36 - Interpolação logarítmica das curvas de eficiência de *speedup* na plataforma JoiN.