Exploração de Paralelismo de Dados em Ambiente Distribuído

Dinamérico Schwingel[1]
Dante Augusto Couto Barone[2]

Universidade Federal do Rio Grande do Sul
Pós-Graduação em Ciência da Computação
Caixa Postal 15064
CEP 91501-970 - Porto Alegre, RS, Brasil
Tel.: (051)336-8399 Fax: (051)336-5576

Resumo

Este artigo descreve o trabalho que vem sendo desenvolvido na UFRGS, tendo por objetivo pesquisar técnicas para aumentar o desempenho de algoritmos, através da exploração de seu paralelismo em ambiente de processamento paralelo fracamente acoplado ou distribuído, dando ênfase para algoritmos com paralelismo de dados.

Assuntos relacionados a balanceamento de carga, eficiência das técnicas de comunicação, processamento heterogêneo e tolerância a falhas em tal ambiente são discutidos, assim como são apresentadas comparações com um ambiente fortemente acoplado baseado em *transputers*.

Resultado práticos, usando um algoritmo de síntese de imagens, são apresentados e comparações de desempenho com um Cray Y-MP232/2E são estabelecidas.

Abstract

This paper describes the work being done at UFRGS seeking for techniques to enhance algorithms performance by exploiting its parallelism in a distributed environment, ephasizing algorithms with data parallelism.

Issues related to load balancing, efficient communication, heterogeneous processing and fault tolerance in such environment are discussed.

Practical results obtained with an image sinthesys algorithm are presented and performance comparisons with a Cray Y-MP232/2E are stated.

[1]Bacharel em Informática (UFRGS,1991); Mestrando do CPGCC/UFRGS; Processamento Paralelo e Distribuído; Computação Gráfica; e-mail: dino@inf.ufrgs.br

[2]Professor UFRGS/CPGCC; Dr. Eng. em Informática (Instituto Nacional Politécnico de Grenoble, França); Processamento Paralelo, Microeletrônica; Redes Neurais; e-mail: barone@inf.ufrgs.br

1 INTRODUÇÃO

Processamento paralelo tem sido largamente utilizado para a obtenção de alto desempenho a um custo relativamente baixo, mas para que esse objetivo seja atingido, deve-se explorar eficientemente a concorrência presente nos algoritmos. Há duas formas de se explorar essa concorrência dentro do domínio do problema.

Pode-se explorar o paralelismo dos processos, executando diversas operações independentes sobre um ou mais fluxos de dados. Tal enfoque é o mesmo utilizado por unidades *pipeline* a nível de instrução de máquina. No nível de processo, este esquema é somente aplicável a máquinas do tipo *MIMD* devido a necessidade de execução de diferentes instruções em um mesmo instante.

O outro enfoque é explorar o paralelismo presente nos dados a serem processados, o que permite a computação independente de diferentes conjuntos de dados. Dessa forma os processadores executam o mesmo algoritmo sobre diferentes dados, sendo esta uma técnica aplicável a ambas máquinas *MIMD* e *SIMD*.

Este artigo descreve o trabalho desenvolvido na UFRGS no sentido de utilizar um ambiente de processamento distribuído[3], composto de estações de trabalho heterogêneas, para a obtenção de alto desempenho.

Até o presente, tem-se concentrado esforços no estudo de técnicas para a exploração do paralelismo de dados, por melhor se aplicar ao ambiente, conforme será visto na seção 4.1.

Formas híbridas de processamento que exploram as duas técnicas podem, facilmente, ser construídas, da mesma forma que são projetadas máquinas *MIMD/SIMD*.

O projeto desenvolvido em um ambiente distribuído é uma continuação de trabalho realizado em *transputers* [SCH92] e, por utilizar-se do mesmo paradigma de programação, os resultados podem ser aproveitados para qualquer desenvolvimento sobre máquinas *MIMD* com memória distribuída.

2 PROCESSAMENTO PARALELO EM VISTA DA LINGUAGEM

Técnicas de processamento paralelo tem sido utilizadas para a construção de máquinas com desempenho superior às existentes já há alguns anos. Tais técnicas podem ser implementadas de duas formas bastante distintas se olhadas do ponto de vista do projetista de software.

Pode-se construir uma máquina com múltiplos processadores ou com apenas um processador e diversas unidades *pipeline*, na qual o compilador ficará encarregado de dividir as tarefas, tornando a programação idêntica a de uma máquina com apenas um processador. O aumento de desempenho é determinado pela eficiência do compilador em explorar os eventos concorrentes no código fonte. Máquinas típicas deste grupo são multiprocessadores com memória compartilhada como o Cray Y-MP.

Outro tipo de máquina são os multicomputadores com memória distribuída, bem exemplificados pelo pioneiro *Cosmic Cube* [SEI85] e seus sucessores. Para estas máquinas, o construtor do software supre o compilador com informações sobre a concorrência dos even-

[3] Ao longo do artigo, será utilizado o termo distribuído como sinônimo de fracamente acoplado por brevidade e onde for entendido que facilitará a compreensão do texto.

tos. O ganho de desempenho obtido, pela execução em uma máquina paralela deste tipo, dependerá de uma especificação do algoritmo que explore sua concorrência, o que não pode ser feito por um compilador restrito a transformações no código fonte que devem preservar a semântica das ações ali estabelecidas [SEI90]. Ou seja, enquanto o compilador trabalha sobre uma especificação imutável, o construtor de software trabalha com o problema e pode, muitas vezes, fazer uma especificação que explore mais concorrência do problema.

Atualmente, limitações físicas dificultam a diminuição do período de relógio de máquinas como o Cray Y-MP, impedindo-as de aumentar significativamente seu desempenho, sem o uso de multiprocessamento. Entretanto, ainda assim, máquinas *MIMD* com memória compartilhada possuem problemas de contenção no acesso à memória, limitando sua escalabilidade. Por entender que somente com o uso de multicomputadores será atingido o desempenho demandado por muitas áreas [HWA84], este trabalho se concentra sobre esse paradigma.

3 AMBIENTE DE PROCESSAMENTO DISTRIBUÍDO

O surgimento dos computadores conhecidos como estações de trabalho trouxe ao mercado máquinas com o poder de processamento de mainframes de alguns anos atrás com preço muito inferior.

Nesse novo conceito de máquina está incorporada a intercomunicação que permite compartilhamento de recursos e uso integrado através da instalação de redes.

Atualmente, a velocidade de comunicação entre os processadores é muito inferior a velocidade de acesso a memória local, entretanto com a evolução na velocidade da comunicação entre essas máquinas, em pouco tempo poder-se-á pensar em uma rede de estações de trabalho como um computador local com memória distribuída.

Este trabalho tem como meta determinar se tal ambiente oferece condições para ser usado em processamento de alto desempenho e quais são suas principais vantagens e limitações, de modo que o termo ambiente de processamento distribuído, neste artigo, se refere a redes de computadores do tipo estação de trabalho.

4 PONTOS CHAVE DE PROJETO PARALELO

Esta seção discute os principais pontos a serem considerados no projeto de um algoritmo paralelo, seja ele forte ou fracamente acoplado. Entretanto fogem do escopo deste trabalho detalhes não concernentes a paralelismo de dados em ambiente distribuído.

4.1 Eficiência de Comunicação

Em um ambiente distribuído, a comunicação entre processadores não é feita a mesma velocidade encontrada em máquinas fortemente acopladas. Este projeto foi desenvolvido sobre uma rede de comunicação Ethernet com velocidade de 10 Mbit/s, enquanto que máquinas paralelas como os *transputers* valem-se de links de comunicação dedicados a 20 Mbit/s.

Em vista disso, deve ser dispensada atenção especial a eficiência de comunicação neste ambiente e, desde já, fica claro que um algoritmo com muita dependência de dados, exigindo muita troca de informação, terá dificuldade para conseguir bom desempenho.

Devido a essa limitação, este trabalho se concentrou sobre paralelismo de dados e um exemplo simples elucida as vantagens dessa modelagem sobre outra que explora paralelismo de processo. Suponham-se os dados contidos em uma matriz bidimensional $A[N, M]$ e que sejam necessárias k unidades de tempo para processar o dado $A[i, j]$. A figura 1 exibe a modelagem que explora o paralelismo contido nos processos a serem executados sobre o dado, usando P processadores e um fluxo de dados. Idealmente, k/P unidades de tempo seriam utilizadas por cada processador para cada dado.

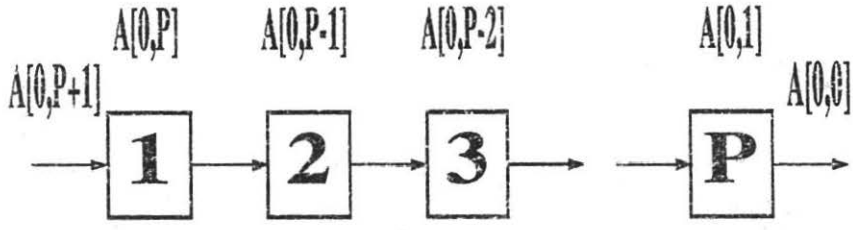

Figura 1: Modelagem explorando paralelismo de processo

No esquema apresentado, a cada k/P unidades de tempo, uma mensagem é enviada ao próximo processador. A eficiência das comunicações deve ser avaliada cuidadosamente em qualquer projeto paralelo e no caso distribuído, esse fator ganha crucial importância, uma vez que o meio físico para troca de mensagens é compartilhado por diversos processadores, senão todos.

Explorando-se o paralelismo de dados a comunicação diminui bastante, uma vez que há uma fase de inicialização, a partir da qual os processadores somente se comunicam ao final do trabalho (figura 2), gerando um fluxo de mensagens bastante inferior, na ordem de $2P$, enquanto que o anterior gera NMP. Nota-se que, no primeiro caso, o número de mensagens depende das dimensões do problema, enquanto que este último depende do número de processadores, o que é bem mais desejável.

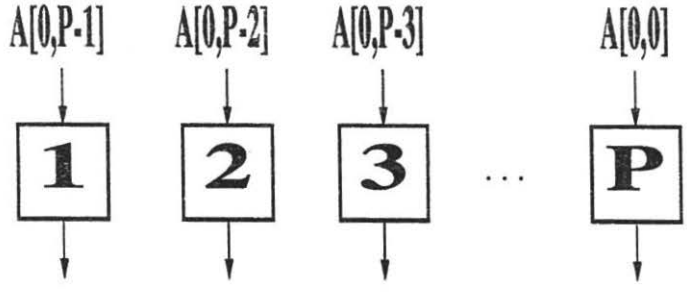

Figura 2: Modelagem explorando paralelismo de dados

Entretanto, nem sempre pode-se dividir igualmente o trabalho entre os processadores em apenas um passo de inicialização. Nesse caso, para atingir um bom balanceamento de carga, pode-se dividir os dados em T porções para que sejam distribuídas dinamicamente, gerando $2TP$ mensagens.

Para o mesmo número de processadores, então, a exploração de paralelismo de dados será mais eficiente somente se $T < MN/2$. Estabelece-se, assim, uma relação de compromisso: quanto maior o tamanho de cada porção, menor o número de mensagens, entretanto mais grosseiro será o balanceamento de carga.

4.2 Balanceamento de Carga

Depois das comunicações, este é o assunto mais importante em processamento paralelo, uma vez que o tempo de execução de um tarefa, dividida em T porções e levada a cabo por P processadores, é igual ao tempo de execução do processador mais lento. Assim, para que não haja processadores ociosos deve haver uma forma de se distribuir a mesma quantidade de trabalho para cada processador. Isso não significa o mesmo número de porções porque elas podem ter cargas diferentes.

Em um ambiente paralelo fortemente acoplado, por exemplo em uma rede de *transputers*, pode-se dispor de processadores dedicados o que facilita a monitoração da carga. Em ambiente distribuído é necessário considerar que não se está trabalhando com máquinas dedicadas, de modo que a carga do sistema pode variar por fatores externos, quer seja por processos de outros usuários ou por processos de sistema.

Uma estratégia eficaz e largamente utilizada para distribuição de carga, quando se explora paralelismo de dados, é a divisão dos dados em pacotes, valendo-se de uma distribuição dinâmica destes. Na maioria das aplicações, a carga de cada pacote não pode ser estimada previamente, ficando por conta de uma monitoração da carga do processador a decisão de enviar outro pacote para aquele processador. Essa monitoração pode ser feita pelo próprio processador que decide enviar parte de seu trabalho a outro ou, de uma forma menos complexa, pelo processo que distribui os pacotes.

4.3 Processamento Heterogêneo

O uso de protocolos e meio físico de interconexão comuns entre os diversos fabricantes de estações de trabalho permite que suas máquinas sejam conectadas em uma mesma rede e operem de forma cooperativa. Isso possibilita que o problema seja resolvido por máquinas construídas sobre processadores diferentes, o que é chamado de processamento heterogêneo.

Para que diferentes máquinas operem cooperativamente é necessário que disponham das mesmas estruturas de comunicação, o que pode ser provido pelo uso de um padrão ou então exige a implementação de um protocolo próprio de comunicação.

Com os dois lados da comunicação operando sobre as mesmas primitivas é facilitada a manutenção do programa, uma vez que existe apenas um código fonte para todas as máquinas e o interface dependente de máquina não é visível na codificação da solução do problema.

4.4 Tolerância a Falhas

Assim como a carga de cada processador pode ser alterada por fatores externos, muito facilmente pode-se perder contato com outro processador por esses mesmos fatores. Uma rede fracamente acoplada pode estar dispersa até mesmo em edifícios vizinhos e, muitas vezes, não há garantia de que as máquinas estarão operacionais nos mesmos horários. Ainda,

falhas adicionais podem ser introduzidas pelos usuários que estão operando localmente as máquinas.

Além de falhas que podem ocorrer nos processadores, também a rede de comunicação pode ser danificada obrigando o sistema a manter informações sobre o trabalho de cada processador.

5 MODELAGEM PROPOSTA

O resultado desses estudos foi aplicado a um algoritmo de síntese de imagens que utiliza o método *ray-tracing* [WHI80] e, a seguir, serão apresentados as bases para a modelagem e os resultados obtidos.

O algoritmo utilizado apresenta um alto grau de concorrência, permitindo que cada *pixel* da imagem seja computado de forma independente, possibilitando uma eficaz exploração de paralelismo de dados.

Como visto na seção 4.1, há um compromisso entre uma boa distribuição de carga e a minimização do número de mensagens. Considerando esse compromisso, foi utilizada uma modelagem do tipo *master/workers*, na qual um processo distribui dinamicamente trabalho para um conjunto de processos que executam o algoritmo, intensivo em computação, e retornam seus resultados ao processo distribuidor, conforme a figura 3.

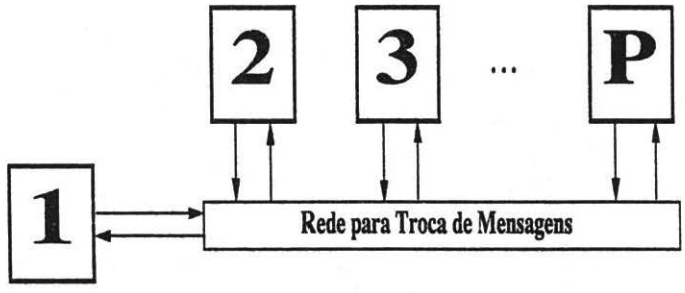

Figura 3: Processador *master* e *workers*

Parametrizando-se o tamanho de cada porção de trabalho a ser distribuída, pôde-se testar com quais valores se obtinham os melhores resultados. Nessa parametrização é importante salientar que o tamanho físico dos pacotes de trabalho é sempre o mesmo, o que muda é o tamanho do pacote de resultado, medido em número de *pixels*. Com a comunicação sendo feita sobre uma rede de pacotes, deve-se manter o tamanho do pacote de resultado como um múltiplo do pacote utilizado pela rede para otimizar seu uso.

Pitot et al [PIT90] dividem os algoritmos paralelos de *ray-tracing* em dois tipos: aqueles que copiam toda a estrutura de dados para todos os processadores e os que copiam apenas um determinado volume para cada processador. Este algoritmo encontra-se na primeira classificação e isso foi projetado assim, para que não houvesse um grande fluxo de mensagens, o que saturaria o meio físico compartilhado de comunicação. Ainda, a diferente carga apresentada pelos pacotes de trabalho diminui a probabilidade de colisões na rede.

O uso de máquinas de diferentes fabricantes é facilitado pelo adoção de sistemas operacionais e protocolos de comunicação compatíveis. No escopo deste trabalho, a comunicação

foi implementada através de *sockets* [SUN90a] que são facilidades disponíveis no sistema operacional UNIX, introduzidas na versão 4.2BSD. Os resultados apresentados aqui não foram obtidos em ambiente heterogêneo, uma vez que, até o presente, tal versão não se encontrava operacional.

Tolerância a falhas também será incluída na versão heterogênea e, a seguir, estão enunciados os problemas mais graves a serem tratados, juntamente com a forma como isso será feito:

- Impossibilidade de disparar um *worker* em processador remoto: um conjunto de máquinas é especificado para executar os processos *workers*, mas algumas delas podem estar desligadas ou com problemas na rede de comunicação. O processo *master* deve possuir formas de detectar e contornar esses problemas, não utilizando essas máquinas.

- Pacote de trabalho enviado e resultado não recebido: para determinar que o resultado não foi recebido, deve haver um tempo máximo de espera definido, pois pode tratar-se de um problema de excessiva carga naquele processador, carga esta causada por fatores externos. Outra causa pode ser falha de comunicação tanto no envio do pacote quanto na remessa do resultado e, nos dois casos, deve-se enviar o pacote de trabalho a outro processador.

5.1 Descrição da Implementação

A implementação consiste de dois processos: *master* e *worker*. O processo *master* é executado em uma estação e controla toda a geração da imagem. Processos *workers* são disparados em diversas outras máquinas que são responsáveis pela síntese da imagem, propriamente dita.

O algoritmo do processo *master* é o seguinte:

1. lê arquivo de descrição da cena;
2. dispara o número de *workers* especificado;
3. estabelece a comunicação com os *workers*;
4. transmite a estrutura de dados, lida do arquivo de descrição, para os *workers*;
5. os passos seguintes são executados concorrentemente, em um mesmo processador:
   (a) enquanto houver raios a processar, transmite pacotes de trabalho para os *workers* que estão livres;
   (b) recebe as porções de imagem que foram geradas pelos *workers* e grava em um arquivo;
6. após gravar toda a imagem, fecha os arquivos e finaliza as comunicações para que todos os *workers* encerrem sua execução.

Os pacotes de trabalho, enviados aos *workers*, consistem de informações bastante simples que informam ao processo a posição inicial, na imagem, e deslocamento nas duas dimensões, fazendo com que seja gerada aquela porção limitada da imagem. Esse pacote é composto de 17 bytes de informação útil sempre.

Por outro lado, os deslocamentos bidimensionais citados são parametrizados e quanto maiores forem os deslocamentos, maior será o pacote enviado ao final do trabalho. Com o uso de parametrização nas duas dimensões, é possível controlar, também, o formato do pacote, o que influencia a distribuição de carga.

Os processos *worker* executam o algoritmo de síntese de imagens, para uma dada região, conforme especificado pelo pacote de trabalho, até que recebam uma sinalização de final, quando encerram sua execução. Seu algoritmo é o seguinte:

1. estabelece comunicação com o *master*;
2. recebe estrutura de dados;
3. recebe pacote do *master*;
4. se for sinalização de fim, encerra execução;
5. processa pacote;
6. envia imagem gerada para o *master*;
7. volta ao item 3.

Não faz parte deste trabalho descrever o algoritmo de síntese de imagens utilizado, uma vez que ele é bem discutido na literatura especializada e o objetivo é generalizar e não especializar em determinadas aplicações.

A arquitetura global do sistema implementado pode ser vista na figura 4. Ao centro estão os processos *workers* que compartilham o mesmo código executável e, envolvendo-os, está representado o processo *master*. Dentro dele, são vistos círculos que representam diferentes fluxos de execução concorrentes, implementados através da biblioteca de processos leves disponível no ambiente SunOS [SUN90b].

6 AVALIAÇÃO DO MODELO

A avaliação da eficiência de um algoritmo paralelo é medida dividindo-se o ganho obtido, em relação ao seqüencial, pelo número de processadores utilizados, entretanto isso deve ser considerado somente para o caso em que todos os processadores têm o mesmo desempenho.

A tabela 1 apresenta os tempos obtidos em diferentes números de estações SPARC 1+, para uma das cenas testadas, composta de 97 objetos.

A figura 5 exibe um gráfico, no qual aparecem o ganho ideal, obtido pela razão entre o tempo seqüencial e o número de processadores, e o ganho obtido com o presente sistema. Observa-se que a linearidade do ganho foi muito boa, demonstrando um uso eficiente de troca de mensagens, aproximando a eficiência obtida em uma arquitetura fortemente acoplada para o mesmo problema, utilizando pequeno número de processadores [SCH92].

Para um número elevado de objetos, a implementação se mostrou bastante eficiente e cenas com grande quantidade de objetos são as mais comuns, quando é usado o algoritmo de *ray-tracing*. Apesar disso, para avaliar a eficiência em um caso limite, mediram-se os tempos de geração para uma cena com apenas 1 objeto. Com o algoritmo seqüencial obteve-se um

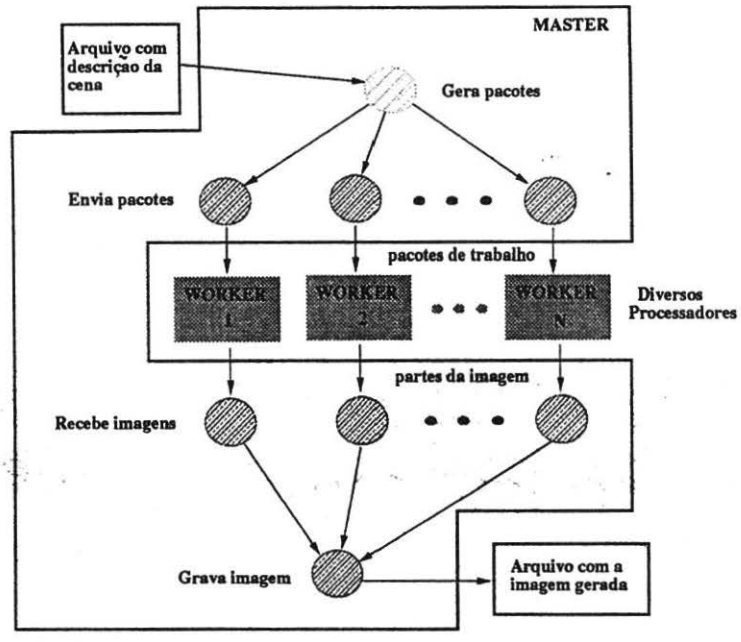

Figura 4: Estrutura do Sistema

tempo de 1m02s. Com 7 estações de trabalho, as mesmas utilizadas anteriormente, obteve-se um tempo de 15,7s.

Idealmente, com 7 processadores, poder-se-ia atingir um ganho de 7. No entanto, obteve-se apenas 3,94 de ganho, com uma eficiência, então, de apenas 56,4%. Disso conclui-se que a sobrecarga de troca de mensagens é muito grande para cenas não complexas, não justificando seu emprego nesse caso. Contudo, como salientado anteriormente, até mesmo uma cena com 97 objetos pode ser considerada simples se comparada com as que são normalmente usadas em animações experimentais e visualização científica. Além disso, o sistema pode detectar que uma cena é simples, baseado em algum limiar, e utilizar pacotes grandes, de forma a gerar um número bastante reduzido de mensagens, o que aumentaria a eficiência, com uma estratégia bastante simples. Para o teste com a cena mais simples, foi utilizado o mesmo tamanho de pacote da outra, a saber 512x2. Ambas as cenas são compostas de 512 colunas e 488 linhas.

Como os resultados foram obtidos sobre processadores de mesmo tipo, pode-se estimar o ganho do algoritmo para diferente números de máquinas mais velozes, desde que seja utilizada a mesma rede de comunicação (Ethernet) e a mesma modelagem de distribuição.

Os tempos apresentados são uma média de diversos testes realizados e foram medidos em redes de estações de trabalho SPARC, no II/UFRGS, em horários em que as máquinas estavam com carga muito baixa.

Número de Estações	Tempo	Ganho	Eficiência
1^{seq}	34m16s	-	-
2	17m23s	1,97	98,8%
3	11m36s	2,95	98,6%
4	08m42s	3,93	98,3%
5	07m00s	4,89	97,9%
6	05m52s	5,83	97,3%
7	05m04s	6,74	96,3%

Tabela 1: Tempos em SPARC 1+

7 CONCLUSÕES

Embora o algoritmo utilizado apresente um alto grau de concorrência entre seus dados, os resultados obtidos mostram que um ambiente de processamento distribuído pode ser usado com sucesso para a obtenção de alto desempenho. Não fizeram parte dos objetivos deste artigo discutir técnicas para diminuir o fluxo de mensagens entre sub-redes, entretanto elas podem prover um aumento de desempenho.

Resultados obtidos na implementação inicial do algoritmo de *ray-tracing* em um supercomputador Cray Y-MP232/2E [SAC92] apresentaram um ganho de apenas 16,85 vezes o desempenho obtido em uma SPARC 1+. Esse resultado foi obtido vetorizando-se a função mais intensiva em cálculos de ponto flutuante, responsável pela computação das intersecções dos raios de luz com os objetos.

O desempenho máximo teórico desse supercomputador, para cálculos em ponto flutuante, é de 330 MFLOPS, entretanto isso é atingido somente em condições muito especiais. Para os testes realizados com essa primeira implementação, o máximo obtido foi de 15,48 MFLOPS.

Acredita-se poder aumentar muito o desempenho do sistema no Cray, através da vetorização do restante do algoritmo, mas o que deve ser observado é que não se trata de um sistema muito estável, uma vez que pequenas alterações têm grande influência, positiva e negativa, no desempenho. O sistema implementado em ambiente distribuído se mostrou muito estável, apresentando um ganho muito próximo do linear.

Outro ponto a ser considerado é que o algoritmo intensivo em computação não teve que ser modificado para a implementação distribuída, enquanto que, para a implementação vetorizada, o código teve que ser reescrito. É claro que o processo *master* teve que ser desenvolvido no ambiente distribuído, mas sua estrutura pode ser aproveitada para outros algoritmos com as mesmas características.

Pretendeu-se, com este trabalho, mostrar as bases para o uso de um ambiente distribuído na busca por computação de alto desempenho a um custo mais baixo e evidenciar as vantagens e limitações de tal ambiente. Um exemplo prático foi apresentado com medidas de desempenho, mostrando a plausibilidade do uso de tal ambiente.

A melhor característica apresentada foi o ganho linear, mostrando uma boa estabilidade, em contraste com as medidas obtidas em um supercomputador, aliada ao custo muito inferior de *hardware*.

Os resultados obtidos com este trabalho pretendem ser aplicados imediatamente na ace-

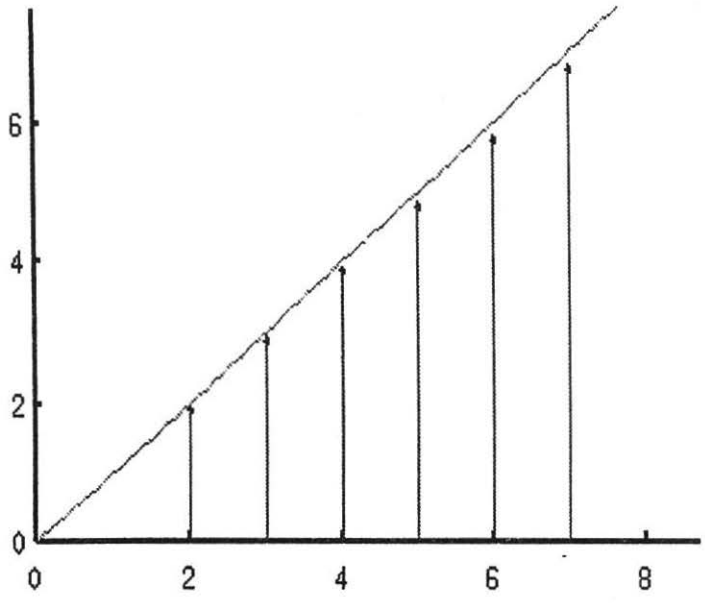

Figura 5: Ganhos: ideal e obtido

leração de algoritmos de aprendizado de redes neurais, o que deve fornecer mais dados para o estudo do comportamento do ambiente em questão. Estudos sobre a variação da linearidade com relação a granularidade também serão feitos.

8 AGRADECIMENTOS

Agradecimento especial a Daniel Francisco Sachet por ter tido contribuição decisiva na implementação das idéias aqui apresentadas.

Agradecimento ao CNPq pelo suporte parcial a este trabalho dentro do projeto CP^2.

Referências

[HWA84] HWANG, K.; BRIGGS, F. **Computer Architecture and Parallel Processing**. New York: McGraw Hill. 1984.

[PIT90] PITOT, P.; MOISAN B.; DUTHEN Y.; CAUBET R. A Transputer Based Implementation of the VOXAR Project. **Microprocessing and Microprogramming**, Amsterdam, v.30, n.1-5, p.347-354.

[SAC92] SACHET, D. F.; SCHWINGEL, D.; JOHANN, M. O. **Vetorização de um Algoritmo de Ray-Tracing**. Projeto em desenvolvimento junto ao Centro de Supercomputação da UFRGS.

[SCH92] SCHWINGEL, D. **APART²—Uma Arquitetura Paralela Assíncrona para Ray-Tracing em Transputers**. Porto Alegre: II da UFRGS, 1992. (Projeto de Diplomação).

[SEI85] SEITZ, C. L. The Cosmic Cube. **CACM**, New York, v.28, n.1, p.23-33, Jan. 1985.

[SEI90] SEITZ, C. L. **Concurrent Architectures**, in VLSI and Parallel Computation. San Mateo: Morgan Kaufmann. 474p. 1990.

[SUN90a] SUN Microsystems, Inc. **Network Programming Guide**. 1990. 353p.

[SUN90b] SUN Microsystems, Inc. **Programming Utilities and Libraries**. 1990. 430p.

[WHI80] WHITTED, T. An Improved Illumination Model for Shaded Display. **CACM**, New York, v.23, n.6, p.343-349, Jun. 1980.