Balanceamento de carga dinâmico. Estudo de caso: o programa MPIPOV*

André K. Menda, Márcia C. Cera, Nicolas Maillard,
Philippe A. O. Navaux

Universidade Federal do Rio Grande do Sul - Instituto de Informática - Campus do Vale
{akmenda, mccera, nmaillard, navaux}@inf.ufrgs.br

1. Introdução

Arquiteturas paralelas heterogêneas, como as grades de computadores [FOS 99], têm evoluído recentemente. Para a programação nessas arquiteturas são necessárias ferramentas que se adaptem dinamicamente a esses ambientes, proporcionando um melhor aproveitamento do hardware disponível, maior desempenho e facilidades ao programador.

Essa tendência de adaptação às evoluções das arquiteturas paralelas é percebida na biblioteca de comunicação MPI (*Message Passing Interface*) [MPI 93]. Ela é um padrão para programação com troca de mensagens em agregados. Recentemente foi acrescida a possibilidade de criação de processos dinamicamente, ou seja, em tempo de execução, através da norma MPI-2. Porém, seria importante para o programador um mecanismo que realizasse automaticamente o balanceamento de carga dinâmico, sem a necessidade de se estruturar o balanceamento a cada nova execução.

Isto é ilustrado ao longo do texto através do estudo da aplicação paralela do algoritmo Pov-Ray para renderização de imagens: MPIPOV. Neste exemplo, o balanceamento de carga é efetuado pelo próprio programa, possibilitando a obtenção de um bom desempenho. A seção 2 apresenta o contexto científico deste trabalho; a seguir será apresentado o programa MPIPOV. A seção 4 apresenta a análise de desempenho do programa. Por fim, se conclui este estudo na seção 5.

2. Contexto Científico: Programação Paralela

2.1. Arquiteturas Paralelas

Devido à impossibilidade de aumentarmos consideravelmente a freqüência de relógio dos circuitos eletrônicos internos aos computadores, buscam-se constantemente outras alternativas para o aumento do poder de processamento através de arquiteturas paralelas. Dentre as alternativas, tem-se os agregados de computadores (*clusters*) [KAU 90]. Estes são de fácil aquisição, instalação e manutenção e, em sua maioria, apresentam uma arquitetura homogênea. Um agregado é um sistema de computação paralela composto por um conjunto de computadores comuns e independentes trabalhando de forma integrada como se fosse um único recurso computacional [BUY 99]. Os agregados possuem a característica de escalabilidade, pois existe a possibilidade de aumentar seu tamanho quando for necessário mais poder computacional.

*Trabalho efetuado dentro do Projeto HP/UFRGS Clumssy

2.2. Programação Paralela com MPI

Como cada componente que integra uma arquitetura paralela possui sua memória local, faz-se necessário um modelo de programação com troca de mensagens. Para facilitar o desenvolvimento de aplicações com base nesse modelo, existem ferramentas que lhes oferecem suporte. Uma ferramenta muito utilizada é a biblioteca MPI-1.2.

A biblioteca MPI-1.2 possui cerca de 120 primitivas, possibilitando comunicações síncronas e assíncronas entre os processos, além de possibilitar comunicações em grupos. A característica estática de MPI-1.2 na criação de processos possibilita seu uso dentro da arquitetura de agregados, porém, em ambientes dinâmicos como as grades computacionais, onde o conjunto de computadores varia constantemente, ou em ambientes heterogêneos, o uso de MPI-1.2 é dificultado.

Para contornar essa deficiência, foi definida a norma MPI-2, possibilitando a criação dinâmica de processos, ou seja, a biblioteca permite criar processos em tempo de execução. Apesar disso, existe uma dificuldade na programação e controle de execução de aplicações paralelas em ambientes dinâmicos, pois a norma MPI não oferece especificação para o escalonamento dos processos. A seguir, é estudado um exemplo de aplicação que ilustra tal dificuldade.

3. Paralelização da Renderização

A utilização de algoritmos de geração de imagens realísticas através de *raytracing* é cada vez maior. Esta técnica calcula para cada pixel da imagem o valor de sua intensidade luminosa em relação a todos os outros objetos envolvidos na cena, demandando grande quantidade de processamento. Uma implementação de *raytracing* bastante conhecida é o Pov-Ray (*Persistence Of Vision Raytracer*) [LIN 94]. A entrada do Pov-Ray é um arquivo-texto externo que descreve a totalização do cenário, desde o conjunto de objetos até a posição da câmera e iluminação global. O resultado desses cálculos é a geração de imagens tridimensionais muito semelhantes à realidade. O algoritmo inicia seus cálculos pelo pixel superior esquerdo da imagem e faz uma varredura linha por linha, horizontalmente da esquerda para a direita e verticalmente de cima para baixo.

Para reduzir o tempo de renderização destas imagens, existe um código livre que executa paralelamente o algoritmo Pov-Ray através da biblioteca MPI, o MPIPOV [MIN 98]. A paralelização segue o modelo de programação mestre-escravo. Devido à diversidade nas arquiteturas paralelas foi implementada de duas formas: uma estática e outra dinâmica [FAV 99].

A paralelização estática do algoritmo de Pov-Ray é especial para arquiteturas de agregados homogêneos, pois ele divide o trabalho igualmente entre os processos que calcularão a intensidade dos *pixels* (inclusive para o mestre). Na paralelização dinâmica, cujo desempenho é maior em arquiteturas heterogêneas, o mestre não realiza cálculo e se encarrega de distribuir dinamicamente o trabalho para os escravos na medida em que estes vão encerrando a renderização das porções das imagens atribuídas a eles. Independente da forma de atribuição de trabalho, a imagem é dividida em porções de tamanhos regulares chamadas grades, cujas dimensões influem diretamente no tempo de processamento da imagem e podem ser definidas pelo usuário. O valor padrão é 32x32 *pixels*.

4. Análise de Desempenho

Esta seção apresenta os resultados obtidos na execução do MPIPOV com paralelização dinâmica. Optou-se por essa forma de paralelização por acreditar-se que esta ofereceria uma melhor distribuição das cargas de trabalho.

O arquivo de teste teve como resultado uma imagem de 500x500 *pixels*. A análise de paralelismo foi através de diferentes tamanhos de grade distribuídas em diferentes números de processadores, buscando obter a melhor relação de divisão de trabalho entre os nós do agregado. O desempenho de execução está fortemente ligado ao número de objetos, câmera e iluminação do arquivo inicial. Por esse motivo, dividir apropriadamente um tamanho de grade entre certo número de nós fornece um melhor desempenho.

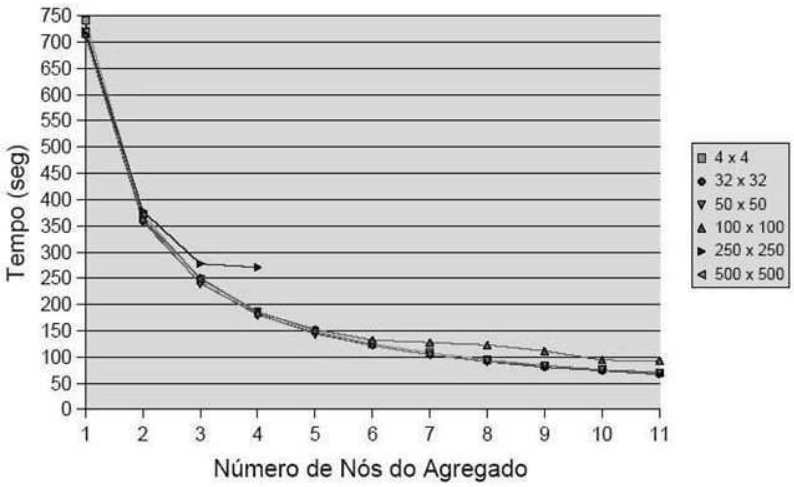

Figura 1: Desempenho de MPIPOV Dinâmico

O gráfico da Figura 1 apresenta os tempos de execução obtidos variando-se o número de nós do agregado e o tamanho da grade. Pode-se concluir que o tamanho de grade padrão (32x32) dividido por 11 nós obteve um desempenho maior (67 segundos). O pior desempenho (741 segundos) se deu com uma grade de menor tamanho (4x4) distribuída para apenas um escravo. Esse valor de tempo é muito próximo ao da execução seqüencial (711 segundos), porém o pior desempenho obtido é decorrente da comunicação entre o mestre e o escravo toda vez que se deseja atribuir uma nova grade para renderização.

O gráfico de aceleração (*speed-up*) é demonstrado na Figura 2. A aceleração é o tempo seqüencial dividido pelo tempo paralelo relacionado ao número de nós do agregado. Através desta figura pode-se observar a proximidade da curva ideal que é obtida pela execução com tamanho de grade 32x32 *pixels*.

5. Conclusão e Trabalhos Futuros

Este trabalho analisou uma aplicação paralela do algoritmo Pov-Ray para renderização de imagens através da técnica de *raytracing*, o MPIPOV. Verificou-se, com o exem-

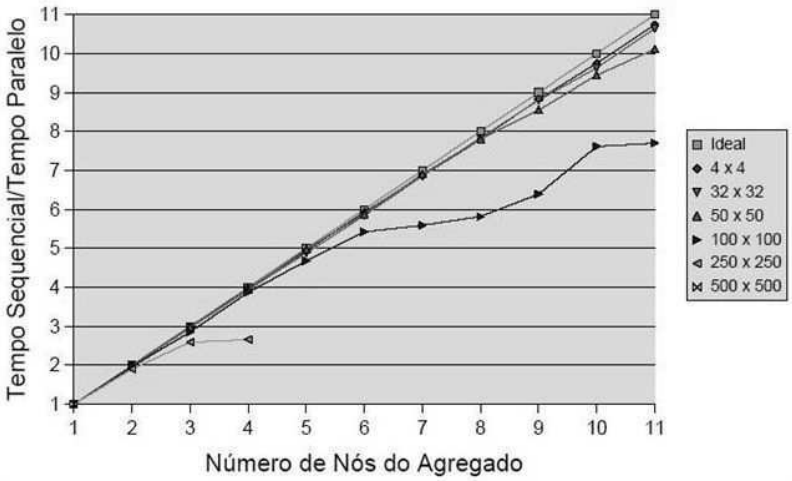

Figura 2: Aceleração de MPIPOV Dinâmico

plo, a importância de se ter um bom balanceamento dinâmico de cargas de trabalho. A experimentação validou o fato de se obter desempenho ideal com um certo mecanismo.

Pretende-se reimplementar essa aplicação usando MPI-2 para possibilitar a análise de seu comportamento quanto à criação dinâmica de processos. A biblioteca βMPI-2 [MAR 05], em desenvolvimento pelo grupo, visa promover o escalonamento dinâmico de processos. O objetivo dessa reimplementação será testar o escalonamento automático oferecido pela βMPI-2.

Referências

[BUY 99] BUYYA, R. (Ed.). **High performance cluster computing**: architectures and systems. [S.l.]: Prentice_Hall, 1999. 881p.

[FAV 99] FAVA, A.; FAVA, E.; BERTOZZI, M. MPIPOV: A parallel implementation of POV-ray based on MPI. In: PVM/MPI, 1999. **Anais...** [S.l.: s.n.], 1999. p.426–433.

[FOS 99] FOSTER, I.; KESSELMAN, C. (Eds.). **The grid**: blueprint for a new computing infrastructure. San Francisco, CA: Morgan Kaufmann, 1999.

[KAU 90] KAUFMAN, L. Finding groups in data: an introduction to cluster analysis. In: **Finding groups in data**: an introduction to cluster analysis. New York: Wiley, 1990.

[LIN 94] LINDLEY, C. A. Ray tracing and the POV-Ray toolkit. **Dr. Dobb's Journal of Software Tools**, v.19, n.7, p.68, 70, 72, 74, 76, 103, July 1994.

[MAR 05] MARCIA C. CERA NICOLAS MAILLARD, P. A. O. N. Betampi-2: uma biblioteca para o escalonamento dinâmico de processos mpi. In: III WORKSHOP PPD/UFRGS - WSPPD'2005, 05, Porto Alegre - RS. **Anais...** [S.l.: s.n.], 05.

[MIN 98] MINKE, T. D. **MPIPOV** :distributed ray tracing on a network of workstations. 1998. Dissertação (Mestrado em Ciência da Computação) — advisor: Steven Pruess, Dept. of Mathematical and Computer Sciences.

[MPI 93] MPI Forum. MPI: A message passing interface. In: SUPERCOMPUTING '93, 1993, Portland, OR. **Proceedings...** IEEE CS Press, 1993. p.878–883.