Metodologias para Análise de Desempenho de Sistemas de Computação Paralela

Leonardo Leiria Fernandes

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Metodologias para Análise de Desempenho de Sistemas de Computação Paralela

Leonardo Leiria Fernandes

Novembro de 2004

Banca Examinadora:

- Prof. Dr. Ricardo de Oliveira Anido
  Instituto de Computação - Unicamp (Orientador)
- Prof. Dr. Philippe Olivier Alexandre Navaux
  Instituto de Informática - UFRGS
- Prof. Dr. Célio Cardoso Guimarães
  Instituto de Computação - Unicamp
- Prof. Dr. Maria Beatriz Felgar de Toledo
  Instituto de Computação - Unicamp (Suplente)

Metodologias para Análise de Desempenho de Sistemas de Computação Paralela

Este exemplar corresponde à redação final da Dissertação devidamente corrigida e defendida por Leonardo Leiria Fernandes e aprovada pela Banca Examinadora.

Campinas, 4 de Novembro de 2004.

Prof. Dr. Ricardo de Oliveira Anido
Instituto de Computação - Unicamp
(Orientador)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Resumo

Com o crescimento do uso dos sistemas paralelos de computação em um grande número de aplicações, torna-se fundamental a análise de desempenho deste tipo de sistema. A complexidade dos sistemas distribuídos torna a análise de desempenho um grande problema para aqueles que desenvolvem aplicações paralelas. São apresentadas no presente trabalho algumas novas metodologias para resolver tal problema. Formas apropriadas de visualizar os programas paralelos em execução são importantes ferramentas para entender melhor o funcionamento dos sistemas e identificar gargalos e ineficiências. A identificação dos processos e ocasiões em que ocorrem tais problemas e ineficiências em um sistema simplifica muito a implementação de melhorias. Partindo do estudo de algumas técnicas de *profiling* e trabalhos relacionados, são desenvolvidas novas formas de visualização para os programas paralelos baseados em trocas de mensagens. É apresentado um conjunto de ferramentas de *profiling* para análise de desempenho de sistemas de computação paralela baseados na MPI. A partir de arquivos de *trace* são geradas algumas novas formas de visualização do funcionamento do sistema analisado. São propostas animações que representam o funcionamento do sistema, trocas de mensagens e distribuição de carga de processamento, bem como gráficos estatísticos referentes a tempos de espera de cada processo pelos demais. É discutido no texto o uso das ferramentas desenvolvidas no estudo de algumas aplicações paralelas reais. São apresentados ainda alguns aspectos da implementação do sistema de *profiling*.

Abstract

The continuous growth of parallel computing systems use on a large number of applications makes the performance analysis of such systems of fundamental importance. The distributed systems complexity turns performance analysis into a significant problem for parallel applications developers. This work presents some new methodologies for solving such problem. Appropriate visualizations of parallel programs executions are important tools for a better understanding of the systems and identifying bottlenecks and inefficiencies. Recognizing the processes and occasions in which such problems and inefficiencies take place in a distributed system makes the implementation of improvements significantly easier. After the study of some profiling techniques and related work, we present new forms of visualization for message passing parallel programs. We present a set of profiling tools for performance analysis of MPI-based parallel computing systems. From trace files we generate some new forms of visualization of system executions. We propose animations that represent system execution, message passing and load balancing. Statistical graphics representing wait times between each pair of processes are also developed. We discuss the use of our tools for analyzing some real parallel applications. We also discuss some details of the profiling system implementation.

À minha querida filha
Mariana.

Agradecimentos

Agradeço primeiramente aos meus pais por todo o apoio que me ofereceram durante a minha vida e à minha namorada Cláudia por estar a meu lado e me incentivar sempre a alcançar os meus objetivos.

Também sou grato à minha filha Mariana por seu sorriso e seu carinho que muito me motivam.

Ao meu orientador, Ricardo Anido, pela orientação, disponibilidade, amizade e compreensão.

Aos demais professores do IC-UNICAMP e aos meus professores da graduação na FURG pelos muitos ensinamentos fundamentais neste trabalho e nos próximos.

Ao Bruno Müller Jr e ao professor Edson Cáceres por permitirem a análise de suas aplicações (SORTTS e fecho transitivo de dígrafos, respectivamente) e por sanarem dúvidas a respeito das mesmas.

Ao Anthony Chan por responder todas as minhas dúvidas a respeito da MPE e dos formatos de arquivos CLOG e SLOG-2.

Aos colegas do IC pela amizade e pelas dicas, especialmente Cleo, Felipe Renon, Felipe Pereira, Fábio, Guilherme, Desirée, Ulisses, Patrick, Gregório e Evandro.

À minha tia, professora Mary Jane Fernandes Franco, pela revisão do português e do estilo do texto.

À Brenda, à minha avó Lina, à minha irmã Tatiane, aos meus primos Rafael e Gerson e aos meus amigos do Reduto e da época da graduação pela grande importância que têm na minha vida.

Aos demais familiares e amigos por serem pessoas especiais.

Ao CNPq pelo apoio financeiro.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Um sistema paralelo é constituído de dois ou mais processos que colaboram para a realização de uma determinada tarefa. Cada processo pode ser visto como um programa seqüencial em execução. Os processos cooperam através do uso de memória compartilhada ou de trocas de mensagens.

Nos últimos anos, tem crescido o interesse pela computação paralela. Com equipamentos cada vez melhores, e custos relativamente mais baixos, é crescente o uso de processamento paralelo nas empresas e instituições de ensino e pesquisa. As utilizações de programação paralela encontradas na bibliografia são inúmeras [10]. Algumas destas utilizações são apresentadas a seguir.

Muitas aplicações científicas e de engenharia com foco em simulações numéricas em que vastas quantidades de dados devem ser processados com o objetivo de criar ou testar um modelo são apropriadas para sistemas de processamento paralelo. Exemplos deste tipo de uso dos sistemas paralelos são:

- Simulações da circulação atmosférica;
- Circulação sangüínea no coração;
- A evolução das galáxias;
- Análises aerodinâmicas de componentes de aeronaves;
- Movimento de partículas atômicas;
- Otimização de componentes mecânicos.

Uma função cada vez mais importante da computação paralela está relacionada à bioinformática, no que diz respeito ao mapeamento de genes dos mais diversos seres vivos, possibilitando um entendimento da biologia nunca antes imaginado. Pesquisas na área

de bioinformática podem auxiliar a prevenir doenças, produzir alimentos mais saudáveis, combater pragas na agricultura, entre inúmeras outras possibilidades.

Outra aplicação significativa é a possibilidade de paralelizar sistemas de gerenciamento de bancos de dados, especialmente em situações com um alto número de transações por segundo ou quando casamentos de padrões complexos em grandes bases de dados estão envolvidos. Esse tipo de sistema tem se tornado bastante comum nas grandes empresas, que precisam manter bancos de dados cada vez maiores e mais eficientes.

Existem ainda aplicações da programação paralela em inteligência artificial, um vasto campo de pesquisa da ciência da computação. Algumas das utilizações da programação paralela em IA são:

- Busca através de regras de um sistema de produção;
- Implementação de algoritmos genéticos;
- Processamento de redes neurais;
- Processamento de visão computacional.

A produção de imagens realistas para a televisão e o cinema também é uma possibilidade de utilização do processamento paralelo. As aplicações da programação paralela continuam a se expandir para a resolução de diversos problemas nas mais diversas áreas tecnológicas e científicas.

Com a diminuição do custo dos equipamentos e a melhoria das tecnologias de comunicação, os *clusters* tornam-se uma opção interessante para a maioria das aplicações que demandam alto desempenho. Em comparação com computadores de grande porte, os *clusters* modernos apresentam custos de instalação e manutenção significativamente menores. Além disso, pode-se ampliar a capacidade de processamento de um *cluster* simplesmente acrescentando novos equipamentos ao sistema. Com a ampla utilização de *clusters*, o desenvolvimento de programas eficientes para sistemas multiprocessados é fundamental.

Os programas paralelos são naturalmente mais complexos do que os programas seqüenciais. De acordo com Andrews [2], em muitos aspectos, os programas paralelos estão para os seqüenciais assim como o jogo de xadrez está para o jogo de damas. Muitas vezes, centenas ou mesmo milhares de processos diferentes estão ativos no sistema. Os processos envolvidos precisam trocar mensagens entre si constantemente para realizar suas tarefas. Nos sistemas sem memória compartilhada a troca de mensagens é a única forma de comunicação e sincronização entre os processos. Tal complexidade dificulta significativamente o projeto e implementação das aplicações de processamento paralelo, bem como a identificação de eventuais ineficiências dos programas desenvolvidos. Weihl [41] ressalta que módulos em sistemas paralelos são mais complexos do que em sistemas seqüenciais. Um

procedimento, por exemplo, pode interagir com processos paralelos lendo e escrevendo em memória compartilhada ou trocando mensagens. Desta forma, uma simples relação de entradas e saídas não é suficiente para descrever o comportamento de um procedimento ou função em sistemas concorrentes. Outra característica apontada por Weihl [41] é o equilíbrio entre desempenho e funcionalidade, que deve ser uma preocupação para aqueles que desenvolvem sistemas paralelos.

Uma das questões centrais para análise do funcionamento deste tipo de sistema distribuído baseado em mensagens é a relação entre o tempo de comunicação [37] e o tempo de processamento.

A comunicação é um fator chave por tratar-se de uma operação que consome um tempo excessivamente grande se comparado com as altas freqüências de operação atingidas pelos processadores modernos. Quando não há um controle sobre essas operações e os momentos em que as mesmas devem ser executadas em um programa, pode-se comprometer demasiadamente a eficiência do programa, diminuindo e, em alguns casos extremos, até anulando as vantagens do processamento paralelo. Bertsekas e Tsitsiklis [4] dividem o atraso proveniente da comunicação em quatro partes:

- *Tempo de processamento da comunicação:* Tempo requerido para preparar a informação para a transmissão;
- *Tempo de fila:* Espera por uma série de eventos tais como: disponibilidade de canais de comunicação, envio de outras mensagens, disponibilidade de recursos como espaço em *buffer* no destino, retransmissões para correção de erros, etc;
- *Tempo de transmissão:* Tempo requerido para a transmissão de todos os bits da mensagem;
- *Tempo de propagação:* Tempo entre a transmissão do último bit da mensagem pelo emissor e o recebimento do mesmo bit no destino.

A sincronização entre processos é outro fator citado por Bertsekas e Tsitsiklis [4] como fundamental no desempenho de sistemas paralelos. Sistemas síncronos tendem a sofrer uma diminuição no seu desempenho. É necessário encontrar o equilíbrio entre sincronização e desempenho apropriado para cada aplicação específica.

Kumar et. al. [24] colocam entre as principais questões que envolvem computação paralela os métodos para a avaliação de algoritmos paralelos. Dado um determinado computador paralelo e um algoritmo paralelo, é necessário avaliar o desempenho do sistema resultante. Tal avaliação de desempenho deve buscar esclarecer questões sobre a velocidade e eficiência do sistema na solução do problema.

Percebe-se, atualmente, uma tendência para a pesquisa e desenvolvimento de sistemas e aplicações utilizando *grids* computacionais [18]. Aplicações paralelas neste tipo de plataforma apresentam um comportamento ainda mais complexo devido a heterogeneidade do sistema. Uma arquitetura de *grid* conta com diferentes tipos de equipamentos, sistemas operacionais e tecnologias de comunicação, ao contrário dos *clusters* que geralmente são mais homogêneos nestes aspectos.

Uma das principais interfaces para comunicação utilizada em sistemas paralelos é a *Message Passing Interface* (MPI) [28]. Trata-se de um conjunto de rotinas que realizam tarefas de comunicação entre os processos em um sistema paralelo, tais como: envio e recebimento de mensagens, *broadcast*, *multicast* e sincronização. A MPI oferece também uma interface para *profiling* que simplifica a obtenção de informações a respeito das rotinas de comunicação e sincronização em tempo de execução.

Para Lucas Jr. [26], a avaliação de desempenho é de vital importância na escolha de um determinado sistema entre duas ou mais opções, no desenvolvimento de aplicações e equipamentos, e na análise de equipamentos e *softwares* em uso. Desta forma, a comparação do desempenho de diferentes arquiteturas, a avaliação de protótipos em desenvolvimento e o monitoramento do comportamento real de um determinado sistema são os três propósitos gerais da análise de desempenho para Lucas Jr. [26].

Em face a complexidade dos sistemas paralelos, a importância da análise de desempenho e a crescente utilização de concorrência nas mais diversas aplicações, o presente trabalho apresenta os seguintes objetivos:

- Estudar o problema da análise de desempenho de sistemas distribuídos, com ênfase no processamento paralelo baseado em troca de mensagens;
- Apresentar algumas abordagens ao problema encontradas na literatura e em produtos comerciais;
- Desenvolver um conjunto de técnicas e metodologias, apoiadas por ferramentas de *software*, que possam ser utilizadas no *profiling* de programas paralelos que utilizam a MPI. Deseja-se atender da melhor forma possível à comunidade científica em suas necessidades de análise de estratégias e algoritmos de programação paralela em um número significativo de plataformas com diferentes configurações de *hardware*;
- Procurar novas formas de visualização da execução de sistemas que evidenciem características fundamentais na análise de desempenho, tais como: a eficiência das comunicações, identificação de gargalos, o funcionamento de algoritmos e o balanceamento da carga de processamento.

A elaboração de formas de visualização apropriadas para a análise de desempenho de sistemas concorrentes representa uma área ainda em desenvolvimento. Encontram-se

disponíveis ferramentas comerciais [15, 30] e, também, alguns trabalhos acadêmicos na área [7, 8, 21, 35].

Este trabalho apresenta metodologias e técnicas para o problema da análise de desempenho de sistemas paralelos através do uso de técnicas de *profiling* e do desenvolvimento de novas formas de visualização da execução de programas paralelos. No próximo capítulo é demonstrado o processo de *profiling*. A seguir, são descritos trabalhos relacionados que também buscam uma avaliação do desempenho de sistemas distribuídos. No capítulo 4 são apresentadas as ferramentas de *profiling* desenvolvidas neste trabalho. Após a apresentação das ferramentas, são mostrados resultados obtidos através do uso das ferramentas desenvolvidas para analisar o desempenho de algumas aplicações paralelas. São discutidos, ainda, alguns aspectos da implementação das ferramentas de *software*. Por fim, são expostas algumas conclusões do trabalho e suas referências bibliográficas.

Capítulo 2

O Processo de *Profiling*

Profiling é definido por Bernecky [3] como sendo a análise de um programa em execução a fim de determinar seu comportamento real em comparação ao esperado. As técnicas de *profiling* facilitam o acompanhamento de programas computacionais em execução. Tais técnicas têm por objetivo auxiliar aqueles que desenvolvem as aplicações a compreender melhor o comportamento dos programas. Como o próprio nome sugere, as ferramentas de *profiling* buscam traçar um "perfil" do funcionamento do programa, estabelecendo quais as rotinas e operações que consomem o maior tempo, a quantidade de tempo consumido, quais rotinas acionam quais outras, como se dá o fluxo de dados, entre outras características, buscando sempre facilitar a análise do sistema.

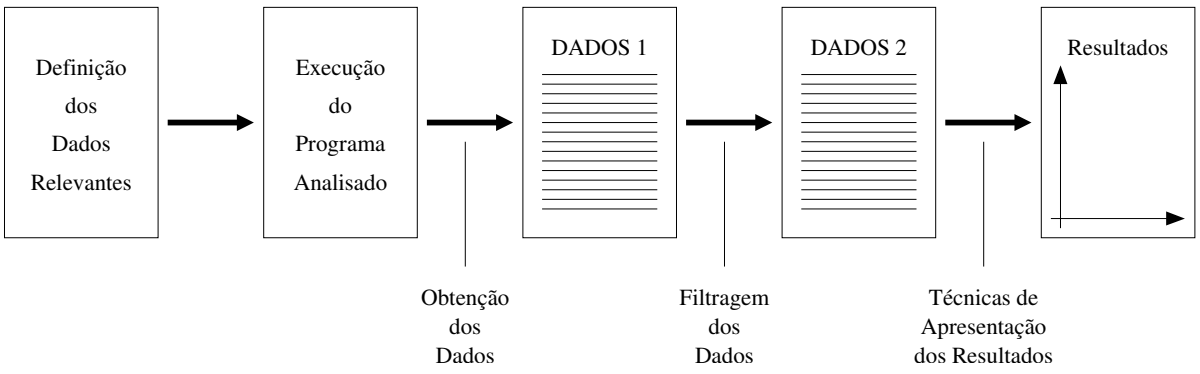

Figura 2.1: As etapas do processo de *profiling*.

Estas técnicas são especialmente importantes em sistemas com muitos processos, em que mesmo os analistas mais experientes encontram dificuldades para identificar eventuais ineficiências. Os sistemas distribuídos nos quais existe processamento paralelo costumam apresentar um nível de complexidade considerável, portanto representam uma importante

aplicação para sistemas de *profiling*.

O processo de *profiling*, em geral, envolve a coleta de um determinado conjunto de dados em tempo de execução e a exibição de tais dados de forma que facilitem a identificação de problemas relativos ao desempenho da aplicação em estudo. Especialmente em sistemas complexos com muitos processos envolvidos é fundamental uma apresentação dos dados que facilite a identificação de ineficiências.

Pode-se dividir, de maneira geral, um processo de *profiling* em quatro etapas distintas: definição dos dados a serem coletados, coleta de dados, filtragem ou tratamento dos dados e apresentação gráfica de resultados, conforme mostrado na figura 2.1. Nas próximas seções são descritas as etapas do processo de *profiling*.

2.1 Definição dos Dados Relevantes

A primeira etapa consiste na definição dos dados que devem ser coletados, tais como: tempo de processamento, tempo ocioso e tempo de comunicação de cada processo. É fundamental que sejam definidos os eventos do sistema que são importantes para a análise do desempenho do sistema, para que se possa obter os registros destes eventos nas etapas seguintes. Define-se um evento como sendo uma mudança de estado de um processo, como o início ou o fim de uma atividade de comunicação ou de uma operação de E/S, por exemplo.

2.2 Coleta de Dados

A segunda etapa se dá em tempo de execução dos processos estudados pelo sistema de *profiling*. Nesta etapa, deve-se obter os registros dos eventos na medida em que vão acontecendo. Deve ser registrado o tipo do evento, um rótulo relativo ao tempo em que o mesmo ocorreu (neste ponto, o tempo local do processador em que o evento acontece é suficiente), e alguns dados específicos como o tamanho de uma mensagem, sua origem e destino, por exemplo. Uma forma comum de se registrar estes dados é através da instrumentação de pontos apropriados dos programas.

A técnica básica utilizada por ferramentas de *profiling* de sistemas de processamento paralelo é a manutenção de arquivos de *trace*, que apresentam os tempos de início e final dos eventos a serem considerados em um sistema. A própria especificação da MPI [28] define uma interface para *profiling*. Esta interface pode ser utilizada para a criação de bibliotecas contendo rotinas com o mesmo nome das rotinas da MPI. Estas, por sua vez, acionam as rotinas originais da MPI, registrando a informação em um *buffer* antes e/ou depois da chamada a função MPI em tempo de execução. O *buffer* é, sempre que

necessário, gravado em disco.

Existe inclusive um banco de dados público [17] contendo uma série de arquivos de *trace* que são de extrema importância para a análise de diferentes problemas. Esses arquivos públicos podem servir como entradas para as ferramentas de *profiling*, ampliando assim as possibilidades de seus testes e de seu uso na análise das mais diversas situações, sem necessariamente ter de repetir implementações e experimentos já realizados para se obter uma análise dos respectivos resultados.

É também na etapa da geração dos arquivos de *trace* que devem estar concentrados os esforços para minimizar a intrusão, já que o registro dos eventos é a única etapa do *profiling* que deve, necessariamente, se dar em tempo de execução do sistema analisado. As demais etapas, em geral, são baseadas na análise posterior destes arquivos de *trace*, sem concorrer com o processo estudado. Uma das principais razões para que esta etapa consista apenas em registrar os eventos, minimizando qualquer carga adicional no sistema é justamente minimizar a intrusão, fazendo com que o programa se comporte de forma tão semelhante quanto possível a uma situação real.

O resultado desta etapa é um arquivo contendo dados brutos de cada evento considerado relevante. Este tipo de arquivo é chamado de arquivo de *trace*. Em geral, tais arquivos apresentam uma grande quantidade de dados, sem qualquer filtragem ou otimização que possibilite um estudo adequado do programa.

2.3 Tratamento dos Dados Coletados

Tel [36] apresenta três diferenças básicas entre sistemas distribuídos e sistemas centralizados: a falta de conhecimento do estado global do sistema, a falta de uma referência de tempo global e o não determinismo. Devido a estes aspectos, ordenar os eventos que acontecem em um sistema distribuído não é uma tarefa trivial. Uma diferença na freqüência dos *clocks* de dois processadores pode gerar registros de mensagens enviadas em um tempo posterior ao seu recebimento em outro processador, o que é absurdo. Na terceira etapa do processo de *profiling* mostrado na figura 2.1, tem-se a preocupação de ordenar os eventos de forma correta e buscar uma forma apropriada de estimar o tempo decorrido entre os eventos com o objetivo de analisar o desempenho do sistema. A ordenação dos eventos pode ser obtida através de técnicas conhecidas como relógios lógicos, mas o tempo transcorrido entre dois eventos em processadores diferentes requer um tempo de referência global para todo o sistema.

Nesta fase, é muito importante o estudo de formas de sincronização e ordenação de eventos do sistema.

Uma das principais metodologias para a ordenação de eventos é o uso de relógios lógicos [25, 32, 34]. Um relógio lógico se preocupa em determinar relações de causalidade

entre eventos que ocorrem em diferentes processos no sistema distribuído. É definida a relação de causalidade "acontece antes", denotada por $\rightarrow$. Assim, se o evento a "acontece antes" do evento b, escreve-se: $a \rightarrow b$. Esta relação é transitiva, ou seja, se $a \rightarrow b$ e $b \rightarrow c$, então $a \rightarrow c$.

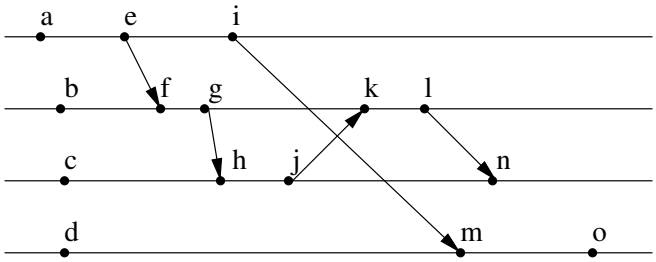

Figura 2.2: Diagrama de tempo de uma execução distribuída.

A figura 2.2 ilustra um diagrama de tempo onde cada linha representa um processo em um sistema distribuído. Pode-se perceber, por exemplo, que $e \rightarrow f$ e conseqüentemente, $e \rightarrow g$, $e \rightarrow k$ e $e \rightarrow l$. Por outro lado, nada podemos afirmar sobre a relação entre i e g sem uma referência de tempo global. Eventos que não apresentam relações de causalidade, como i e g são ditos eventos paralelos. Uma eventual troca na ordem de eventos paralelos não causa alterações em estados futuros do sistema. Os relógios lógicos ordenam os eventos do sistema, respeitando as relações de causalidade. Os eventos paralelos recebem valores ordenados de alguma forma arbitrária. Existem ainda relógios lógicos cujos valores são vetores ou matrizes, e não valores escalares. Neste tipo de abordagem, cada evento é registrado com o valor de seu tempo lógico local, mais as informações que o processo tinha a respeito dos tempos de outros processadores no momento do evento [34]. Com esse tipo de relógio lógico é possível, por exemplo, dados os valores de tempo lógico de dois eventos quaisquer, determinar se os mesmos são paralelos ou se obedecem a alguma relação de causalidade, o que não é possível com a utilização de valores de tempo lógico escalar.

Este tipo de algoritmo de sincronização é uma das bases de funcionamento dos sistemas distribuídos, evitando situações de conflito que seriam comuns se não houvesse este tipo de preocupação.

Mesmo com a utilização dos relógios lógicos, não é possível determinar o tempo decorrido entre dois eventos. Por exemplo, na figura 2.2, não é possível, sem uma referência de tempo global, determinar quanto tempo se passou entre os eventos e e f. Maillet e Tron [27] apresentam algoritmos para a implementação de uma referência de tempo global para avaliação de desempenho em sistemas multiprocessados. A idéia é colher amostras

dos tempos individuais de cada processador do sistema antes e/ou depois da execução dos processos em estudo. A partir destas amostras, identifica-se através de métodos estatísticos uma relação linear entre cada um destes tempos locais e o tempo global (em geral o tempo de um processador escolhido como referência). Quanto maior for a amostra, mais preciso será o tempo global obtido para os eventos. Coletar este tipo de amostra envolve um tempo considerável de trocas de mensagens entre os processadores do sistema, e pode tornar o processo de *profiling* um pouco lento, embora não haja nenhuma intrusão adicional na aplicação em si.

No caso do *profiling*, evidentemente, é fundamental uma preocupação com a ordenação e com o tempo de cada evento que ocorre no sistema, pois disponibilizar estas informações da maneira mais correta possível é a própria razão de ser das técnicas de *profiling* nos sistemas distribuídos.

Busca-se a extração de informações relevantes dos arquivos de *trace* obtidos anteriormente. Procura-se descobrir quais as informações mais importantes para o analista, o que é necessário para identificar eventuais ineficiências da implementação estudada. Tendo estipulado quais as informações desejadas, passa-se à implementação de ferramentas que automatizem a extração das informações dos arquivos de *trace*.

Abordagens conhecidas são a classificação dos eventos de acordo com um padrão determinado previamente como sendo o comportamento normal daquele evento [39]. Se o evento ocorrido no sistema e registrado no *trace* apresenta uma diferença significativa em comparação ao padrão, a execução deste evento pode ser classificada como ineficiente. É importante perceber que o padrão estabelecido deve ser específico para cada arquitetura de *hardware* e carga de *software*, para refletir a real característica da situação estudada. Este padrão pode ser estabelecido, por exemplo, em testes pouco antes da execução do programa que vai gerar o arquivo de *trace*, para refletir da melhor forma possível as características do sistema.

Também nesta fase são obtidas estatísticas de tempos dos eventos, de atividade de cada processador entre outras informações. Outra preocupação é tentar identificar no código fonte, as modificações que podem melhorar a performance do programa. Embora este tipo de abordagem possa requerer arquivos de *trace* diferenciados, e ferramentas mais sofisticadas.

2.4 Apresentação Gráfica dos Resultados

Tendo os eventos adequadamente ordenados e uma referência global, é necessário identificar as características do sistema. A última etapa consiste em desenvolver formas de apresentação destes resultados que facilitem a identificação, por parte do usuário, dos pontos cruciais do seu programa que podem melhorar o desempenho. Esta fase repre-

senta um dos maiores desafios do desenvolvimento de técnicas de *profiling*. É necessário buscar formas de sintetizar o grande volume de informações produzidos nas etapas anteriores simplificando a tarefa de identificar pontos fortes e fracos do sistema estudado. Isto é geralmente obtido através de gráficos.

A apresentação gráfica dos resultados facilita bastante a análise dos problemas, especialmente em sistemas grandes e complexos, onde a quantidade de dados é bastante extensa, dificultando a correção de problemas e ineficiências. A visualização do funcionamento dos programas paralelos através de gráficos e diagramas permite ao usuário um entendimento dos sistemas dificilmente obtidos de outra forma. Além disso, gráficos podem sintetizar grandes quantidades de dados em uma única imagem.

Para a geração dos gráficos de análise de desempenho existem opções que vão desde a programação com bibliotecas gráficas como a OpenGL [42] até a utilização de ferramentas de visualização científica existentes. As ferramentas de visualização científica, apesar de serem idealizadas para a visualização de fenômenos naturais podem ser utilizadas para representar o comportamento de sistemas de computação paralela.

Técnicas bastante utilizadas são gráficos de linha de tempo, ou diagramas de Gannt, que podem ser elaborados diretamente a partir dos arquivos de *trace* e representam uma visão excelente dos acontecimentos envolvidos no sistema. A figura 2.3 representa um exemplo simples deste tipo de gráfico, com apenas dois processos e as operações `MPI_Send` e `MPI_Recv` (envio e recebimento de mensagens, respectivamente). Observa-se facilmente que a segunda mensagem, enviada do módulo 2 para o módulo 1, foi enviada com atraso, pois o processo de destino está pronto para receber a mensagem muito antes do seu envio. Em situações onde existem inúmeras atividades e processos a serem considerados, gráficos deste tipo podem ser extremamente úteis.

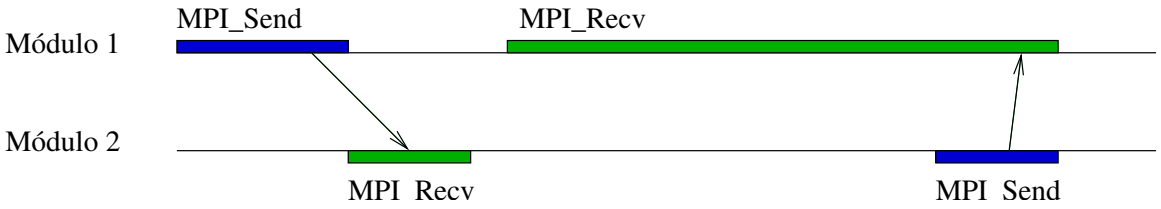

Figura 2.3: Exemplo simples de gráfico de linha de tempo com dois processos.

Os diagramas de Kiviat [23], por exemplo, foram uma das primeiras formas de visualizar a performance de sistemas multiprocessados. Neste tipo de gráfico, cada processador é representado por um eixo radial em uma circunferência. O grau de utilização do processador é um ponto no eixo correspondente. Sendo 0% de utilização representado por um ponto no centro do círculo e 100% de utilização por um ponto no seu perímetro. A

figura 2.4 foi gerada por Hackstadt e Malony [20], utilizando o *software* Visualization Data Explorer (DX) da IBM. Na representação utilizada na figura 2.4a, cada processador é mostrado em uma cor diferente. Este tipo de diagrama apresenta apenas o estado do sistema em um dado momento. Pode-se utilizar uma animação para exibir passo-a-passo os diagramas de Kiviat dos processos em execução, mas ainda assim não se tem uma idéia geral do funcionamento do sistema. Já na figura 2.4b, é gerado um tubo de Kiviat. Esta estrutura é desenhada a partir dos dados que geram diagramas de Kiviat, com um eixo perpendicular aos diagramas representando o tempo, possibilitando que se tenha uma visualização do comportamento de toda a execução. Já a figura 2.4c apresenta um quadro de uma animação que mostra passo-a-passo a execução, possibilitando que se tenha a visão do desempenho do sistema em cada etapa.

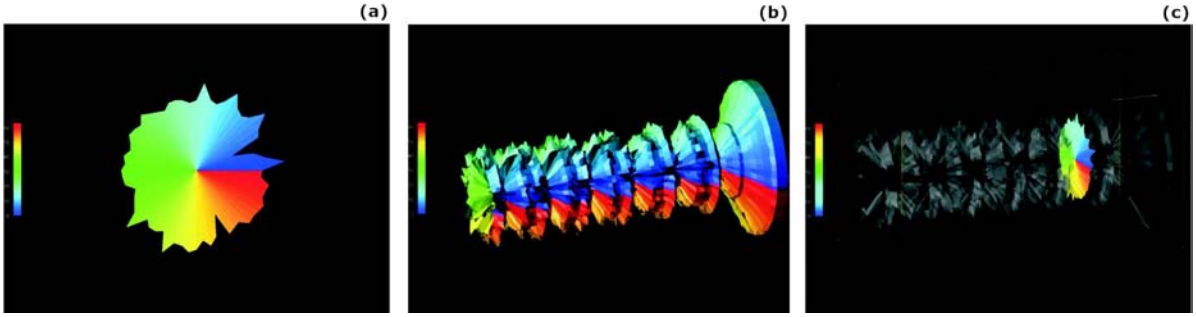

Figura 2.4: a) Diagrama de Kiviat tradicional, em 2D. b) Tubo de Kiviat formado pelas duas dimensões do diagrama de Kiviat tradicional e o tempo como terceira dimensão. c) Quadro de animação de um tubo de Kiviat.
Fonte: http://www.cs.uoregon.edu/research/paraducks/papers/ieeecga95.d/

Outro tipo de gráfico bastante útil são aqueles que mostram as estatísticas de cada processo. Através de gráficos circulares em setores, ou em barras, pode-se identificar facilmente características importantes como o tempo consumido pelos processos em cada atividade.

Pode-se ainda apresentar animações representando que atividade cada um dos processadores está executando a cada momento, possibilitando uma análise passo-a-passo do funcionamento do sistema como um todo.

A apresentação gráfica de dados obtidos em *profiling* e em simulações [9] da execução de sistemas distribuídos ainda é uma área em pleno desenvolvimento, tendo muitas possibilidades a serem exploradas pelos pesquisadores.

O presente trabalho procura encontrar formas adequadas de se conduzir cada uma das etapas descritas. Busca-se discutir as técnicas encontradas na literatura e em ferra-

mentas comerciais e identificar metodologias eficientes para a análise do funcionamento e do desempenho de sistemas multiprocessados que utilizam trocas de mensagens, mais especificamente o padrão MPI.

Capítulo 3

Trabalhos Relacionados

Neste capítulo são demonstradas algumas técnicas encontradas na literatura e ferramentas disponíveis para análise de desempenho de sistemas distribuídos. Em geral, as ferramentas disponíveis utilizam gráficos de linha de tempo como principal forma de visualização das atividades do sistema paralelo. Gráficos estatísticos também são bastante comuns nas abordagens estudadas.

3.1 Vampir/Vampirtrace

`Vampir` (*Visualization and Analysis of MPI Resources*) e `Vampirtrace` [30] formam um conjunto de ferramentas comerciais para análise de desempenho de programas paralelos que utilizam a MPI.

O `Vampirtrace` é o *software* responsável pela geração de arquivos de *trace*. O formato utilizado pelo Vampir é chamado de `STF` (*Structured Trace File Format*). Uma das principais vantagens deste formato, segundo o fabricante, é sua escalabilidade e uma representação dos dados compacta. O `Vampirtrace` inclui uma biblioteca de *profiling* para ser utilizada na geração dos dados do *trace*.

A partir do trace em formato `STF`, a ferramenta `Vampir` apresenta uma série de formas de visualização para programas MPI.

A figura 3.1 demonstra um diagrama de linha de tempo, conforme gerado pela ferramenta `Vampir`. A ferramenta permite a obtenção de informações adicionais sobre mensagens específicas tais como tamanho da mensagem, tempo de transmissão e comunicador MPI utilizado. Pode-se ainda identificar a mensagem no código-fonte. O gráfico em linha de tempo gerado pelo `Vampir` define diferentes atividades, tais como atividades da MPI e da aplicação. Cada atividade é representada por uma cor diferente. Na parte inferior da figura 3.1 são mostrados quantos processos estão realizando uma determinada atividade em cada instante. O diagrama de linha de tempo apresenta inicialmente todo o tempo de

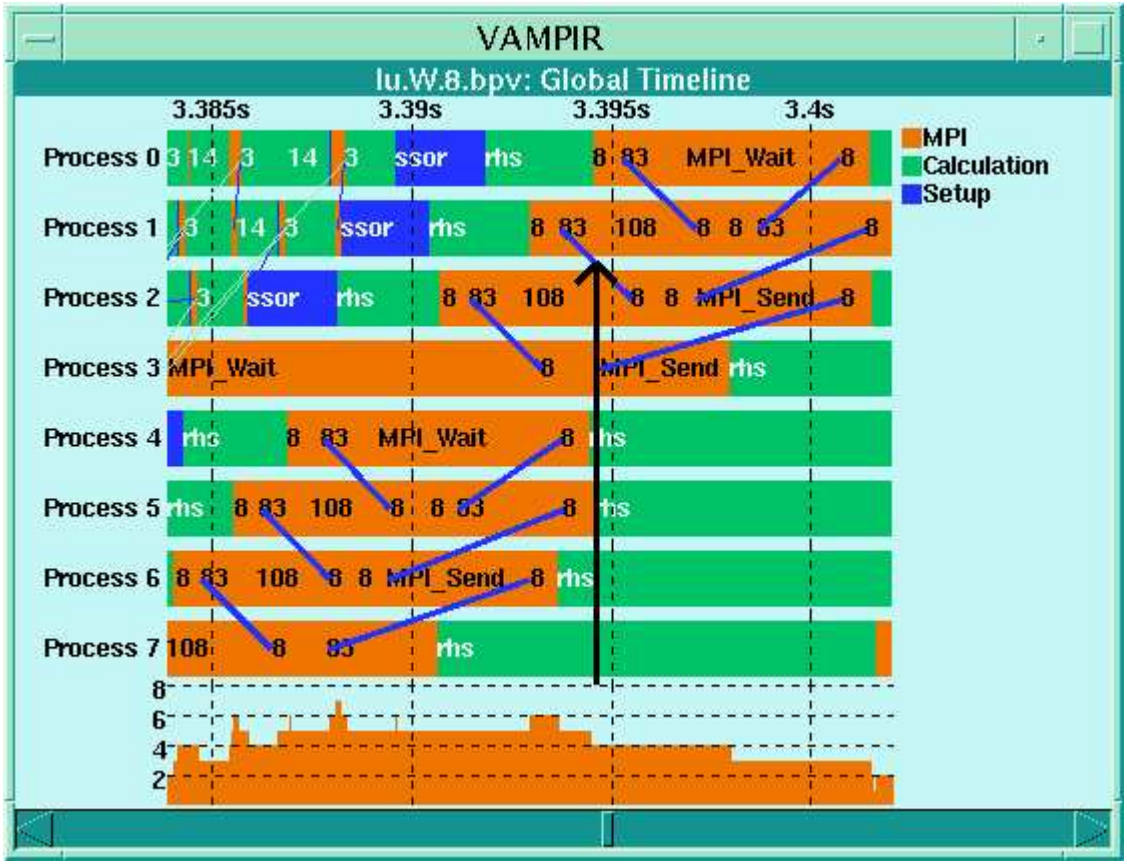

Figura 3.1: Gráfico de linha de tempo gerado pela ferramenta Vampir.
Fonte: http://pallas.com/e/products/vampir/index.htm

execução. A partir da representação inicial, pode-se realizar operações de *zoom* para que se possa visualizar mais claramente trechos específicos da execução.

Além do diagrama de linha de tempo, a ferramenta `Vampir` apresenta uma série de possibilidades de visualização diferentes. Os gráficos estatísticos são bastante úteis na análise de sistemas paralelos.

A figura 3.2 demonstra gráficos em setores, também gerados pelo *software* `Vampir`. Estes gráficos permitem que se observe o tempo total consumido pelas diferentes atividades (computação, comunicação, etc.) em cada um dos processos. Todas as operações da MPI são representadas como uma única atividade denominada MPI. Esta abordagem impossibilita que o usuário identifique o tempo consumido por cada uma das rotinas individualmente.

O *software* permite também determinar quais pares de processos trocam maiores quan-

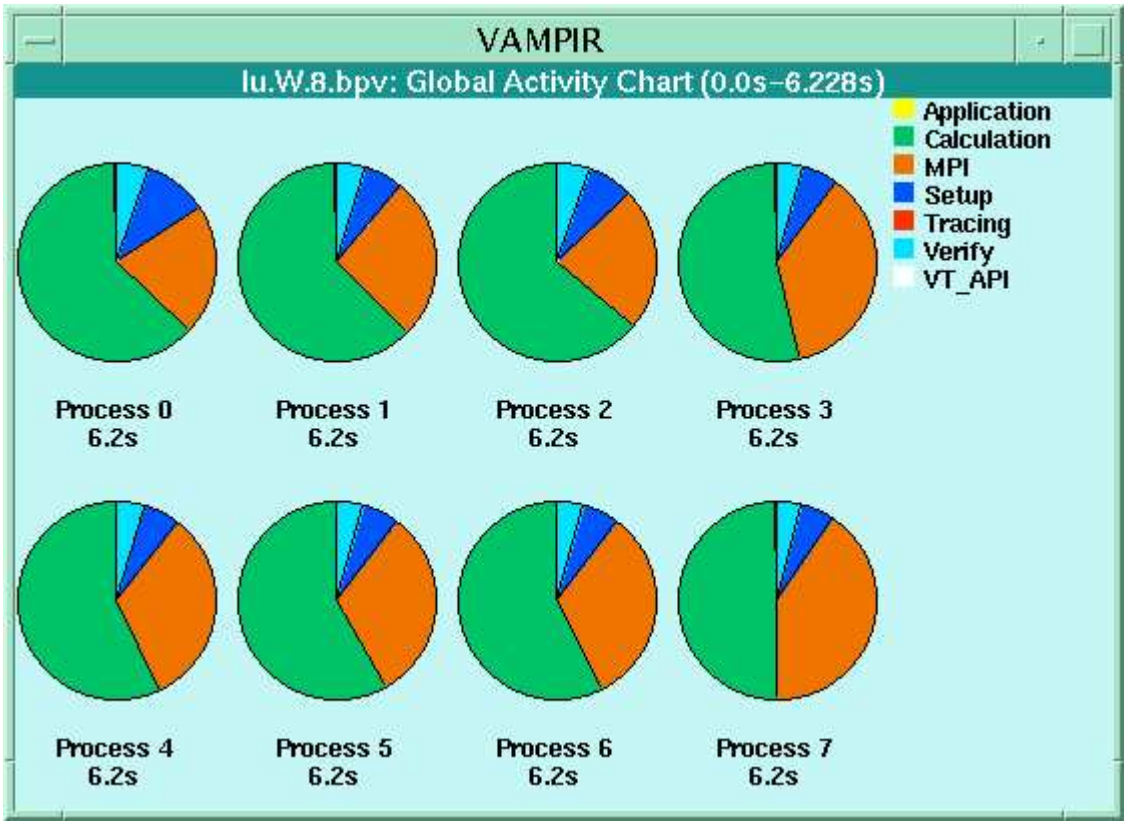

Figura 3.2: Gráficos em setores representando o tempo total de cada atividade.
Fonte: http://pallas.com/e/products/vampir/index.htm

tidades de dados. Um diagrama como o reproduzido na figura 3.3 revela este tipo de informação.

O uso de atividades em gráficos estatísticos e de linha de tempo pode, em muitos casos, ser uma generalização inconveniente. Colocar todas as rotinas MPI como uma única atividade, por exemplo, inclui operações como *broadcast*, *multicast*, mensagens ponto a ponto e operações de sincronização em uma mesma atividade. O fabricante justifica o uso das atividades como conjuntos de operações pelo fato de o número de operações eventualmente ser muito grande para que se possa representá-lo adequadamente por meio de cores.

O gráfico da figura 3.3 mostra claramente a quantidade de informação trocada entre os pares de processos, mas não apresenta nenhuma informação referente ao desempenho da comunicação realizada.

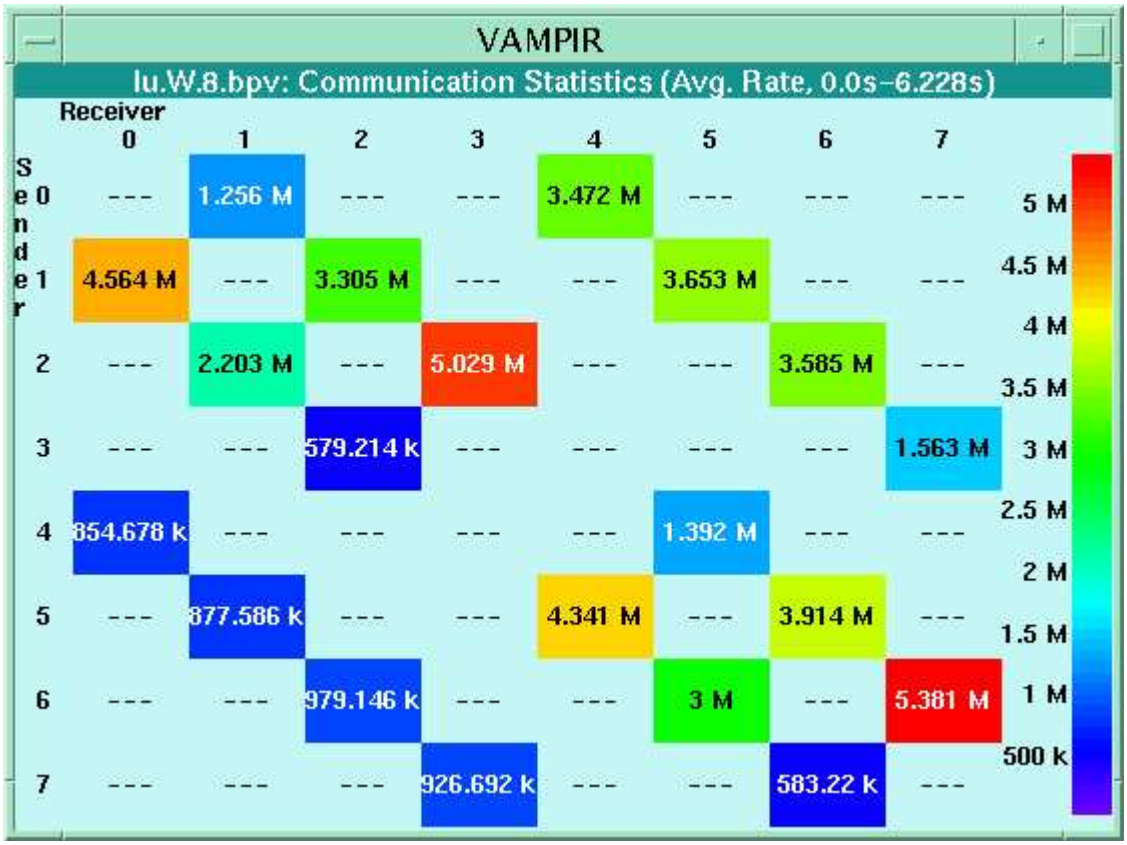

Figura 3.3: Troca de informações entre processos.
Fonte: http://pallas.com/e/products/vampir/index.htm

3.2 Jumpshot

O `Jumpshot` [43] é uma ferramenta de visualização baseada em gráficos de linha de tempo. A interface do *software* é mostrada na figura 3.4.

A aplicação utiliza arquivos de *trace* no formato `SLOG-2` (*Scalable Logfile*) [12], obtidos através da extensão a MPI chamada MPE (*MultiProcessing Environment*) [11]. Este tipo de arquivo também é utilizado pelas ferramentas propostas no capítulo 4.

De forma semelhante a ferramenta `Vampir`, discutida anteriormente, o `Jumpshot` permite operações de *zoom* para a visualização de detalhes da execução. São mostradas todas as rotinas MPI executadas pela aplicação alvo no diagrama de linha de tempo. Estados definidos pelo usuário no código-fonte através da MPE também são representados no gráfico. As mensagens são representadas por setas ligando processos emissores e receptores no diagrama.

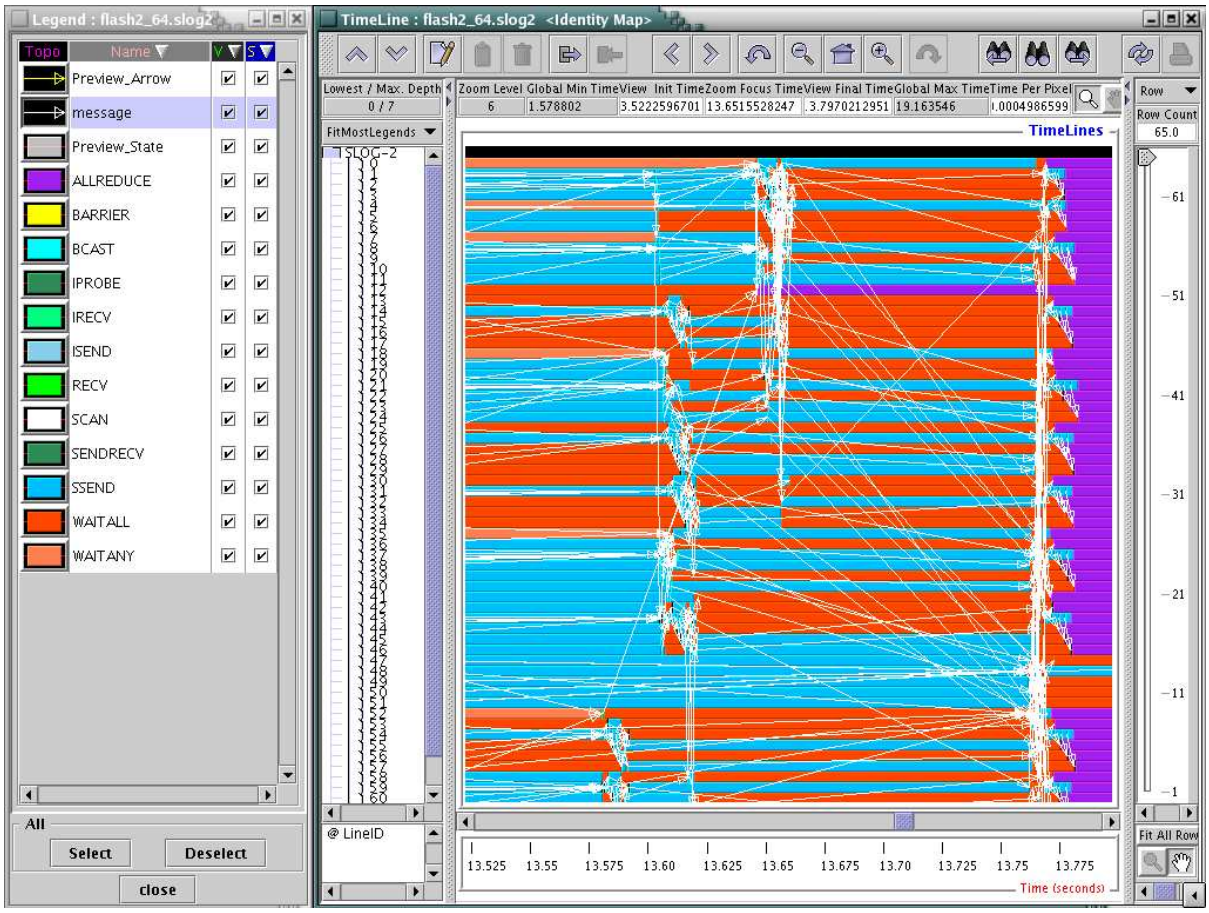

Figura 3.4: A ferramenta Jumpshot

Em visões mais globais de execução de programas, são utilizados estados e setas especiais (*preview states* e *preview arrows*) que representam conjuntos de estados e mensagens. Com um nível mais detalhado de *zoom* torna-se possível observar estados e mensagens individuais.

3.3 Paragraph

O `Paragraph` [21] é um conjunto de ferramentas de visualização para programas MPI. Para a geração dos arquivos de *trace* é utilizada uma biblioteca chamada `PICL` (*Portable Instrumented Communication Library*), posteriormente adaptada para a MPI e denominada `MPICL`.

A ferramenta apresenta uma série de formas de visualização que podem ser classificadas

em três tipos: visualizações de utilização, de comunicação e de tarefas.

As visualizações de utilização classificam os processadores do sistema em ociosos, ocupados ou sobrecarregados (utilizando comunicação). Tais visualizações incluem gráficos de Gannt, diagramas de Kiviat, contagem de processos em cada estado (ocioso, ocupado ou sobrecarga), entre outras.

As visualizações de comunicação incluem gráficos de linha de tempo, gráficos representando o tráfego de mensagens (curvas representando o total de mensagens ou bytes enviados mas ainda não recebidos pelo tempo), informações referentes a filas de mensagens e animações em grafos e hipercubos representando redes de comunicação.

Visualizações de tarefas demonstram estados definidos pelo usuário através da biblioteca `MPICL` em gráficos de Gannt, contagens do número de processos em cada tarefa pelo tempo, etc.

3.4 PVaniM

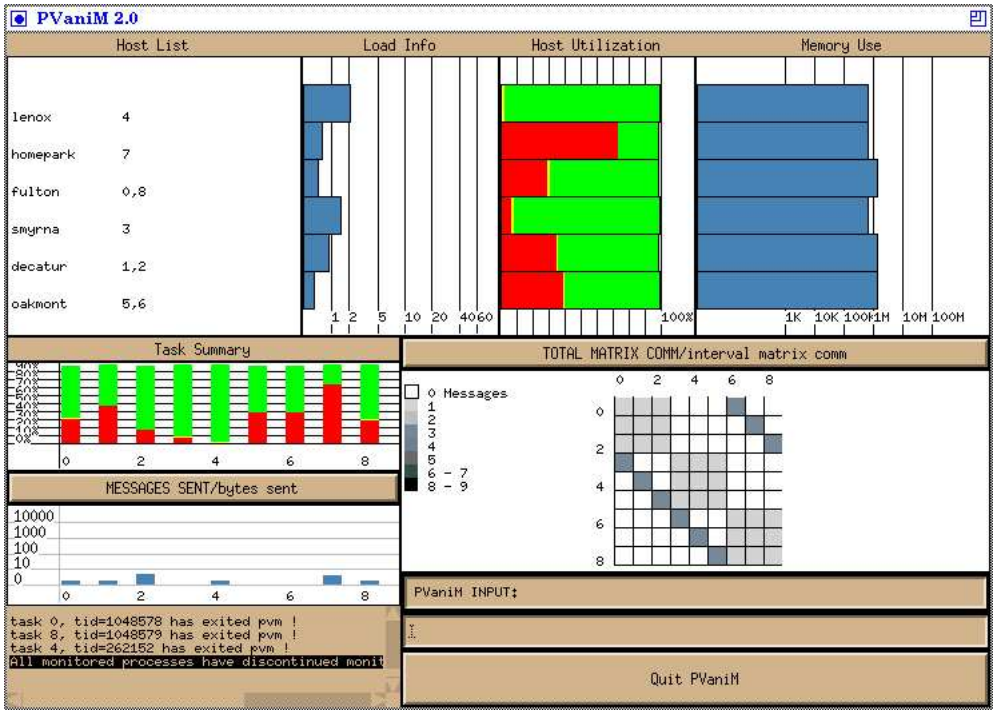

Figura 3.5: Visualizações da ferramenta PVaniM.
Fonte: http://www.cc.gatech.edu/gvu/softviz/parviz/pvanimOL/pvanimOL.html

`PVaniM` (*Online and Postmortem Visualization Support for PVM*) [38] é um conjunto

de ferramentas para *profiling* de aplicações PVM (*Parallel Virtual Machine*). A ferramenta apresenta algumas visualizações, mostradas na figura 3.5.

No canto superior esquerdo da figura 3.5 é apresentada uma lista dos computadores utilizados e a quantidade de tarefas alocadas em cada máquina. Ao lado, é mostrado um gráfico em barras indicando o número médio de tarefas em execução no nó. Ainda na figura 3.5, os gráficos com barras verdes, amarelas e vermelhas representa em verde o tempo consumido com computação, em amarelo o tempo consumido em envio de mensagens e em vermelho o tempo ocioso de espera por comunicações, sendo um gráfico com valores relativos a processadores e o outro com valores relativos a cada tarefa. O gráfico no canto superior direito da mesma figura representa a memória utilizada pelas tarefas PVM em cada processador. O gráfico na parte inferior esquerda da figura mostra o número de mensagens e de bytes enviados em um determinado intervalo. Por fim, a visualização da parte inferior direita da figura 3.5 mostra a quantidade de mensagens trocadas entre cada par de tarefas.

3.5 Outros Trabalhos

Existem na literatura ainda outras ferramentas de *profiling* [6, 8, 22, 35].

Uma abordagem proposta por Vetter [39] procura classificar as mensagens trocadas no sistema em normais, de recebimento tardio, de envio tardio, entre outras classificações. Desta forma, o analista pode alterar o programa, retardando ou antecipando as operações que estão sendo executadas antes ou depois de seu tempo ideal. Segundo o autor, alterações deste tipo podem melhorar significativamente o desempenho de aplicações distribuídas baseadas em troca de mensagens.

Rabenseifner [33] descreve uma solução mais voltada para os administradores de sistemas, apresentando estatísticas semanais e mensais do uso do equipamento pelos diferentes usuários e sistemas, indicando possíveis ineficiências dos programas. Existem ainda outras soluções que priorizam a carga de processamento de cada processador [5].

Oliveira e Midorikawa [13] propõem uma metodologia para predição da performance das rotinas MPI. Tal estudo é bastante relevante em teoria ou em situações próximas do ideal, como em *clusters* dedicados a uma aplicação específica. Em casos mais gerais como *clusters* não dedicados ou em *grids* torna-se inviável fazer uma previsão confiável do desempenho da comunicação devido a interferência de outras aplicações e usuários no tráfego da rede de comunicações e na carga dos processadores.

Capítulo 4

Apresentação das Ferramentas Desenvolvidas

Neste capítulo são discutidos brevemente alguns aspectos da coleta de dados utilizada. A seguir as ferramentas de *software* de visualização desenvolvidas são apresentadas.

Considera-se fundamental que seja possível através das ferramentas identificar as seguintes características básicas de um sistema paralelo:

- Desempenho das rotinas de comunicação e sincronização;
- Identificação da dependência (espera) entre os processos;
- Distribuição de carga entre os processadores;
- Comportamento de algoritmos.

Acredita-se que a identificação das características citadas permite a identificação de muitos gargalos e ineficiências de aplicações.

4.1 Coleta de Dados

Foram definidos como dados relevantes a serem coletados para que as ferramentas pudessem atingir seu objetivos:

- A ocorrência de todas as rotinas MPI na aplicação alvo com um rótulo de tempo de acordo com um relógio global, bem como a duração de cada chamada;
- Processos emissores e receptores de cada mensagem;
- As rotinas MPI associadas a cada mensagem nos processos emissor e receptor;

- Rótulos de tempo e duração de trechos do programa definidos pelo usuário;
- Informações referentes à utilização (carga) dos processadores.

Durante a pesquisa sobre *profiling*, percebeu-se que todos os requisitos citados, com exceção das informações sobre a carga dos processadores, eram atendidos pela extensão a MPI denominada MPE (*MultiProcess Environment*).

O sistema de armazenamento e acesso aos dados da MPE também mostrou-se adequado. Os formatos de arquivos `CLOG` e `SLOG-2` permitem o armazenamento e a pesquisa dos dados de *profiling* de maneira bastante escalar. Desta forma, grandes quantidades de dados podem ser armazenados de forma eficiente e dados específicos podem ser encontrados rapidamente mesmo em arquivos bastante grandes.

O grau de intrusão da MPE também mostrou-se bastante satisfatório. Os dados referentes aos eventos a serem registrados são armazenados em *buffers* de memória locais em tempo de execução. As informações da memória são gravadas em arquivos temporários locais conforme a necessidade. Os dados dos diferentes processos só são combinados em um único arquivo após o término da execução da aplicação alvo, de forma que não há operações de comunicação por parte da MPE durante a execução da aplicação alvo. Assim, a intrusão limita-se a operações de acesso a memória e eventuais operações locais de entrada e saída.

Outro fator favorável ao uso da MPE é sua popularidade. A biblioteca é distribuída individualmente ou em conjunto com a implementação da MPI chamada `MPICH`, sempre como *software* livre. Já existem versões da biblioteca para uma série de sistemas, incluindo várias versões de Unix, Linux, Windows (NT e 2000), plataformas como IBM SP, Intel i860, Delta, Paragon e Cray T3D.

O relógio global da MPE é extremamente simples. É utilizado apenas um ponto de sincronização entre os processos ao final da execução e o tempo desta sincronização é tido como referência para o cálculo dos tempos globais de todos os eventos. Esta abordagem mostrou-se apropriada, embora não seja totalmente confiável em períodos de execução muito longos. Dado que mecanismos de relógio global mais sofisticados tendem a aumentar a intrusão significativamente ou exigir longos períodos de sincronização antes e depois de cada execução da aplicação alvo [27], optou-se pelo uso da MPE apesar da limitação da sua implementação de relógio global.

Para obtenção dos dados referentes à carga dos processadores, foi desenvolvida uma aplicação MPI denominada `cpustatmpi` que, em conjunto com a MPE coleta dados da carga local em cada um dos processadores durante o tempo de execução da aplicação alvo. Mais detalhes sobre a coleta de dados e os formatos dos arquivos utilizados pela MPE e pelo `cpustatmpi` são discutidos na seção 6.1.

4.2 Animação de Processos

Esta ferramenta apresenta uma animação da execução do sistema. Cada processo é representado por um quadrado em uma matriz. A cor de cada processo muda de acordo com seu estado em cada momento da execução.

A ferramenta é mostrada na figura 4.1. A legenda na parte direita da figura apresenta todas as rotinas MPI presentes na aplicação, bem como estados definidos pelo usuário através da MPE. A ferramenta mostra também o tempo representado pelo quadro de animação mostrado, alguns botões de controle e menus.

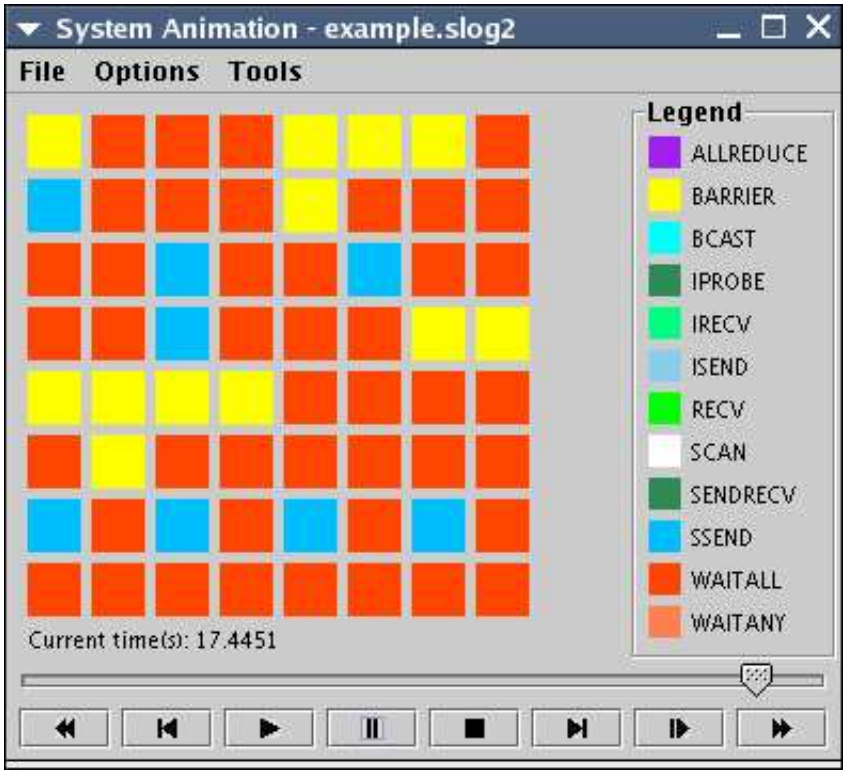

Figura 4.1: A ferramenta de animação de processos.

Os botões de controle na interface permitem a exibição da animação em velocidade normal, buscando aproximar-se do tempo real, ou em *slow motion*. O *slow motion* torna mais fácil a visualização do funcionamento de partes específicas da execução. Botões de avanço e retrocesso rápidos permitem uma navegação eficiente através da animação para que o usuário possa encontrar as partes mais relevantes da execução da aplicação rapidamente. Um controle deslizante também é disponibilizada para uma navegação ainda mais eficaz.

O número de quadros por segundo de tempo de execução da aplicação alvo pode ser alterado através do menu apropriado. Este recurso permite que se possa encontrar a melhor relação entre a velocidade da animação e o nível de detalhe com que se deseja trabalhar.

Muitas ferramentas de análise de desempenho para sistemas paralelos [21, 30, 43], discutidas no capítulo 3, usam gráficos de linha de tempo ou diagramas de Gannt para representar a execução de programas paralelos. Comparada a este tipo de gráfico, a animação de processos apresenta algumas vantagens, tais como a identificação de estruturas de repetição como *loops* freqüentemente encontradas em algoritmos distribuídos. A animação também torna simples a identificação de grupos de processos que se comportam de maneira semelhante. Também é possível identificar diferenças de comportamento indesejadas entre tais grupos de processos. A animação torna possível uma visão apropriada de detalhes de uma execução paralela, ao passo que gráficos de linha de tempo permitem uma melhor idéia geral de toda a execução. Neste sentido, as duas formas de visualização podem ser vistas como complementares.

A ferramenta permite ainda que se altere a topologia da animação. Através de uma opção em um menu pode-se escolher o número de linhas e colunas de processos representados. Esta opção é útil sempre que uma topologia física ou lógica do sistema é relevante para a aplicação ou para sua análise. É possível adaptar a visualização para representar adequadamente características da rede física ou características lógicas definidas pela aplicação.

4.3 Estatísticas de Tempo de Espera

Uma das maiores questões com relação ao desempenho de sistemas distribuídos é o tempo consumido por cada processo enquanto espera pelos demais. É proposta uma forma de visualização que torna possível identificar claramente o tempo de espera dos processos por cada um dos demais processos e também o tempo de espera por rotinas de comunicação coletiva e sincronização.

Através da definição de estados de espera como o recebimento de uma mensagem, pode-se calcular quanto tempo um processo espera por cada um dos demais. Este tempo de espera é comparado a seguir com o tempo total de execução do sistema e com o tempo total de espera do processo em questão.

A figura 4.2 demonstra a ferramenta. A representação é feita através de um quadro de tamanho $p \times p$, onde p é o número de processos no sistema. Cada linha i no gráfico representa um processo. Cada coluna j representa um processo pelo qual o processo i espera por um determinado tempo. A cor de cada célula $[i, j]$ no gráfico representa uma determinada percentagem de tempo consumida por i esperando por j. Esta percentagem

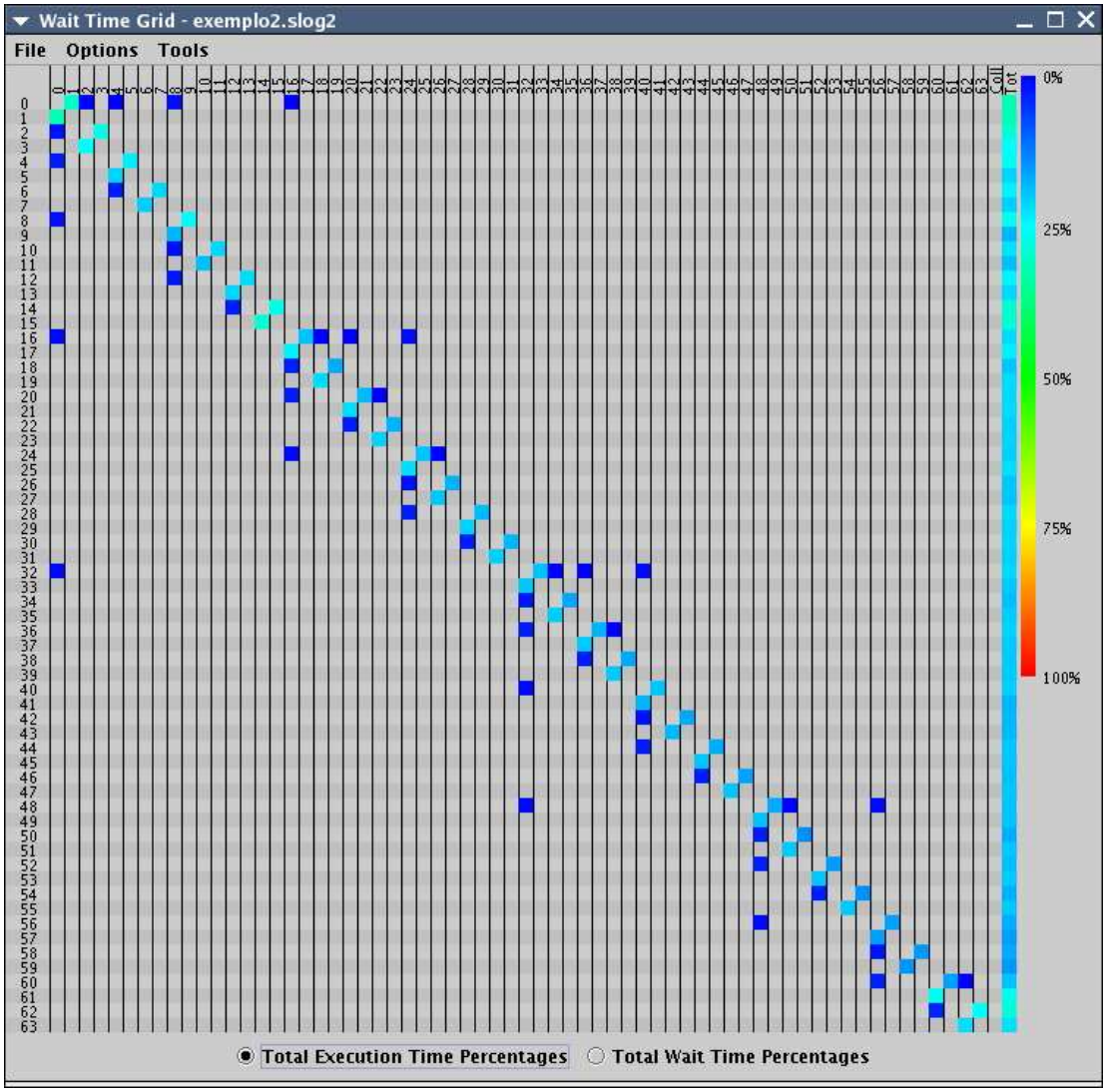

Figura 4.2: Tempo de espera relativo ao tempo total de execução.

pode ser em relação ao tempo de execução ou em relação ao tempo total de espera, de acordo com a opção escolhida pelo usuário através dos botões de rádio disponíveis na parte inferior da interface. O gradiente de cores a direita na figura 4.2 permite uma fácil identificação dos processos que esperam por maiores quantidades de tempo, e por quais processos esperam. Também é possível identificar quais processos fazem com que outros processos esperem mais tempo. Desta forma, pode-se identificar gargalos onde um grande número de processos esperam por alguns poucos. Nestes casos, uma possível solução é redistribuir a carga de processamento, passando algumas das tarefas dos processos ocu-

pados para os processos que esperam muito. As duas colunas mais a direita no gráfico da figura 4.2 tem significados especiais. A penúltima coluna da esquerda para a direita representa o tempo de espera por operações coletivas, como comunicação coletiva *broadcast*, *multicast*, etc.) e sincronização. A coluna mais a direita do gráfico representa o tempo de espera total de cada processo.

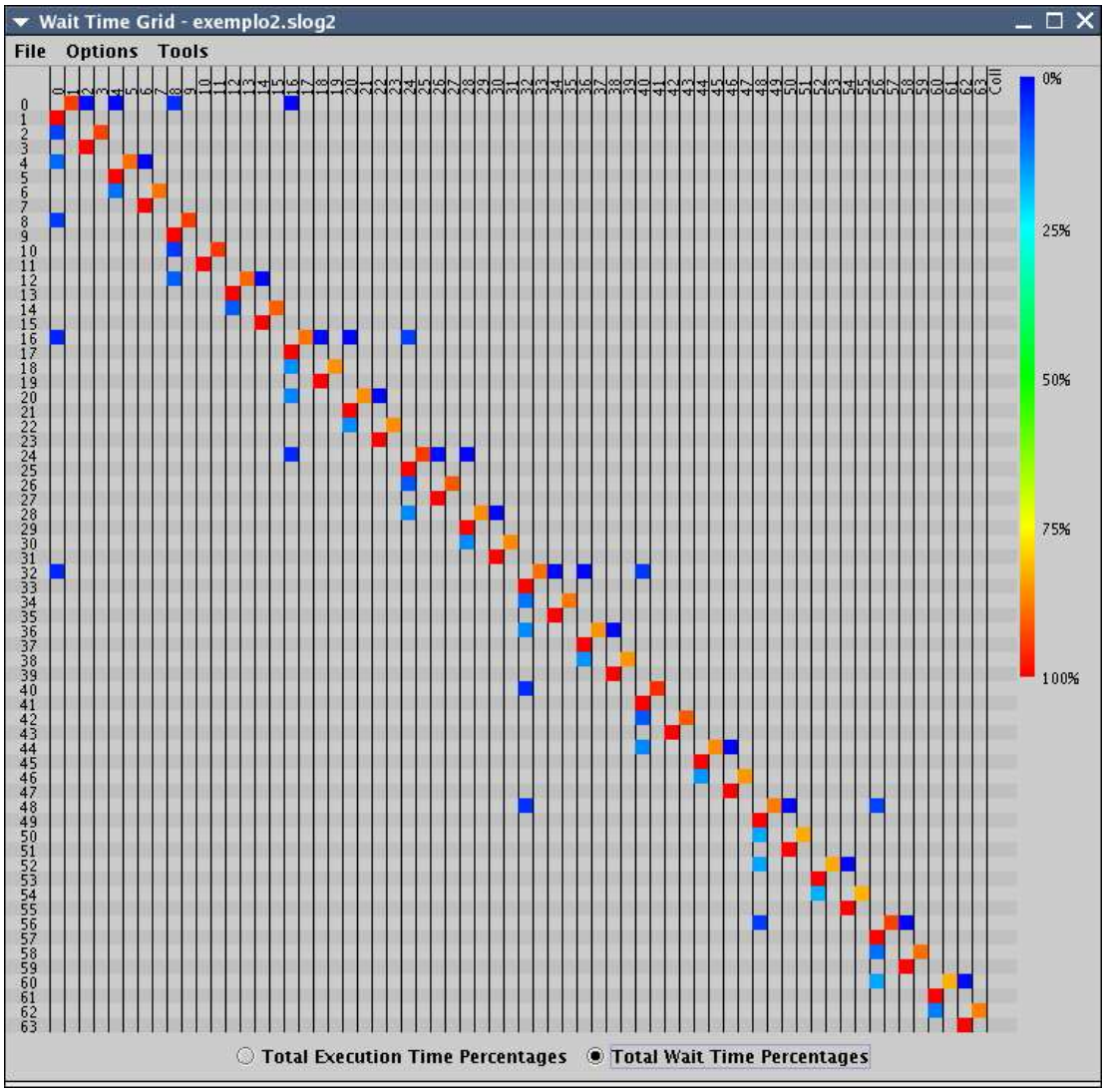

Figura 4.3: Tempo de espera relativo ao tempo total de espera.

A figura 4.3 demonstra a possibilidade de mostrar os valores como percentagens dos tempos totais de espera. Esta opção é importante, pois torna possível uma identificação mais precisa das diferenças entre os tempos de espera individuais de cada processo. Es-

pecialmente em casos onde o tempo de espera é pequeno em comparação com o tempo total de execução.

As formas de visualização são complementares. É possível identificar quais os processos que esperam mais tempo na representação mais geral da figura 4.2 e, a seguir, verificar a distribuição do tempo de espera de tais processos em mais detalhes conforme mostrado na figura 4.3.

Tempos de espera inferiores a 0,1 por cento do tempo de execução e do tempo de espera não são incluídos nas representações. Isto é feito porque tais tempos de espera geralmente representam trocas rápidas de mensagens que muitas vezes ocorrem em fases de sincronização que tendem a preencher a maior parte das células do gráfico com informação irrelevante. A exibição de tais dados dificultaria o reconhecimento por parte do usuário da informação realmente importante representada pelos casos em que ocorrem tempos de espera significativos.

Nas ferramentas de *profiling* pesquisadas não foram encontradas formas de visualização que permitissem este nível de detalhamento dos tempos de espera dos processos.

4.4 Animação da Distribuição de Carga

A ferramenta mostrada na figura 4.4 apresenta a distribuição de carga das aplicações. Como na ferramenta de animação de processos (seção 4.2), o sistema é representado por uma matriz de processos. A porcentagem de tempo de uso da CPU é mostrada para cada processo. Quanto maiores forem as porcentagens de tempo de uso, melhor será o aproveitamento dos recursos computacionais.

É importante notar que a ferramenta mede o uso das CPU's, independentemente das aplicações que estejam sendo executadas no momento da coleta de dados (tempo de execução da aplicação alvo). Isto significa que a ferramenta mede o uso da CPU por todos os processos ativos no sistema, e não somente pelos processos da aplicação alvo. Desta forma é possível avaliar a distribuição de carga do sistema em situações reais de utilização, com outras aplicações em execução. Também é possível estudar o comportamento da aplicação isoladamente, executando apenas a aplicação alvo e os processos básicos do sistema operacional no momento da coleta de dados (geração do arquivo de *trace*).

A ferramenta também exibe estatísticas totais da distribuição de carga. Isto é feito através da exibição dos valores médios obtidos durante toda a execução da aplicação alvo. As cargas totais são mostradas no quadro inicial da ferramenta e podem ser vistas a qualquer momento, sempre que a animação é parada.

Esta ferramenta, assim como a ferramenta de animação de processos, suporta visualizações com diferentes topologias através da definição do número de linhas e colunas em que os processos devem ser organizados na representação.

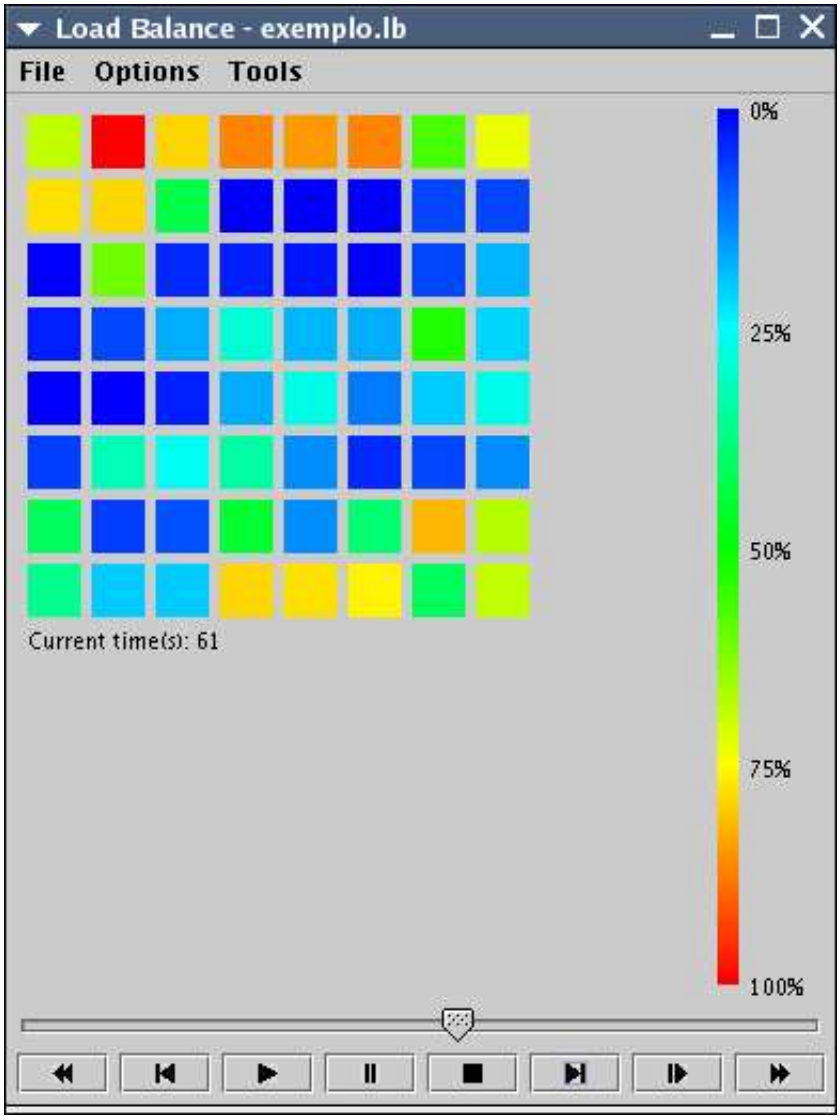

Figura 4.4: A ferramenta de distribuição de carga.

4.5 Animação do Tráfego de Mensagens

Esta animação, mostrada na figura 4.5, apresenta o tráfego de mensagens no sistema. O sistema é representado por uma matriz $p \times p$ de conexões entre processos, sendo p o número de processos. Uma área colorida representa a transmissão de uma mensagem no momento representado. As linhas representam processos emissores e as colunas, processos receptores de mensagens. As cores representam a taxa de transmissão efetiva da mensagem, conforme indicada pelo gradiente de cores a direita da figura 4.5.

Neste trabalho, considera-se como taxa de transmissão efetiva a quantidade de bits transmitidos dividida pelo tempo transcorrido entre o início do procedimento de envio da mensagem no processo emissor e o final do procedimento de recebimento no processo receptor. Desta forma, uma chamada tardia a um procedimento de recebimento de mensagem por um processo é uma causa provável para uma taxa de transmissão efetiva baixa. Através desta taxa de transmissão efetiva, pretende-se refletir tanto o tempo de resposta dos processos receptores quanto o tamanho da mensagem, uma vez que ambos são fundamentais no desempenho das comunicações.

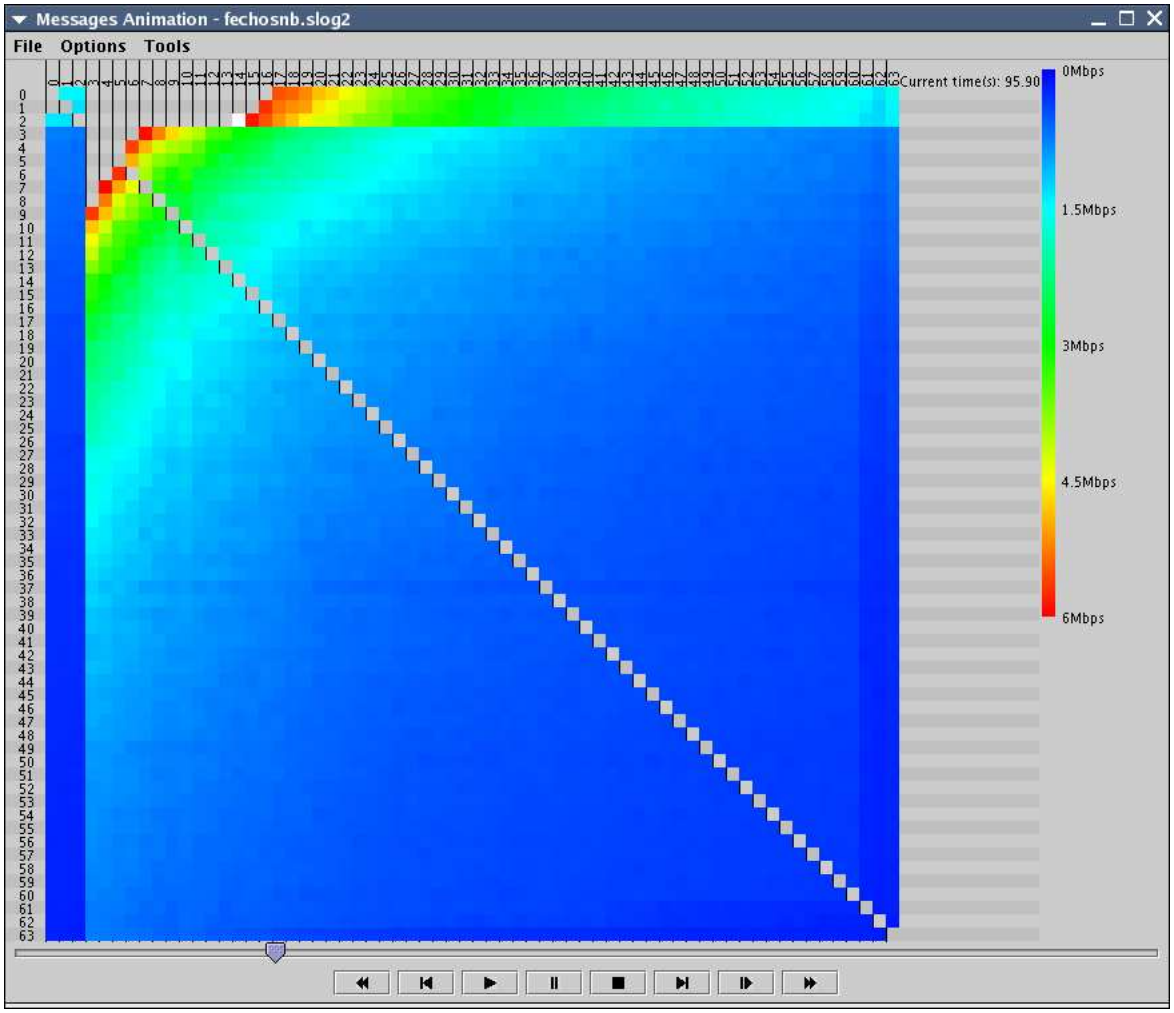

Figura 4.5: Tráfego de mensagens.

O gradiente de cores auxilia a visualização da eficiência da comunicação. Tanto um canal de comunicações ruim como recebimentos tardios tendem a reduzir a taxa de trans-

missão efetiva. Desta forma, ambos os problemas podem ser detectados através da ferramenta.

Também é possível visualizar, em um gráfico semelhante, a taxa de transmissão efetiva média durante toda a execução entre cada par de processos que se comunicam durante o tempo de execução. Tal gráfico facilita o reconhecimento de gargalos na comunicação entre processos evidenciando os pares de processos cuja comunicação foi, em média, mais ineficiente ao longo da execução.

Freqüentemente a taxa de transmissão efetiva dos dados mostra uma grande variação. Isto ocorre por muitas razões, tais como a ocorrência de mensagens de diferentes tamanhos e as diferenças na disponibilidade dos processos para receber mensagens. Manter um gradiente de cores grande o suficiente para incluir todas as velocidades de transmissão da aplicação é muitas vezes inadequado. Uma aplicação poderia, por exemplo, conter um subconjunto de mensagens bastante lentas e outro subconjunto contendo mensagens extremamente rápidas. Cada subconjunto, em um gradiente de cores grande, ficaria próximo a um dos extremos do gradiente, tornando difícil a identificação de diferenças de desempenho dentro de cada subconjunto. Para resolver este tipo de problema, é possível ajustar os limites do gradiente, definindo os valores máximos e mínimos a serem representados.

Quando as taxas de transmissão efetivas são maiores que o valor máximo do gradiente, a mensagem é representada pela cor branca. Mensagens com velocidades menores do que o valor mínimo representável aparecem na cor preta. Também é possível efetuar uma pausa na animação e otimizar o gradiente para o quadro corrente. Quando um gradiente é otimizado para um quadro de animação, seus valores mínimo e máximo correspondem às taxas de transmissão mínima e máxima representadas no quadro. De forma análoga, pode-se otimizar o gradiente para toda a execução.

4.6 Estatísticas de Processos

Pode-se acessar uma janela com mais detalhes a respeito de qualquer um dos processos da aplicação através de um clique com o mouse na área que representa tal processo em qualquer uma das ferramentas apresentadas anteriormente. A informação apresentada inclui a percentagem do tempo de execução consumido em cada estado e esperando por outros processos e por operações coletivas. Os dados nesta representação são referentes ao tempo total de execução.

A figura 4.6 mostra as estatísticas de um determinado processo. O gráfico de barras a esquerda na figura demonstra a percentagem de tempo consumido na espera por cada processo e por operações coletivas. O tempo total de espera também é apresentado no gráfico de barras. O processo representado na figura 4.6 consome um tempo muito grande

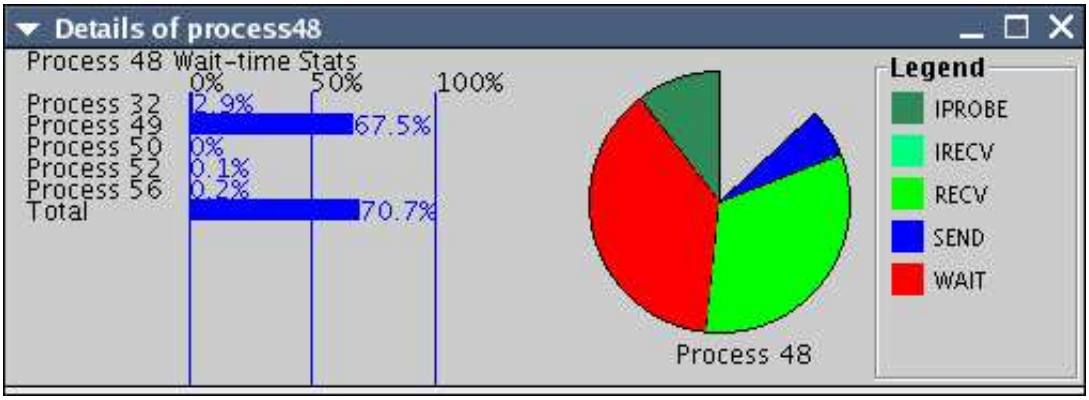

Figura 4.6: Detalhes de um processo.

esperando pelo processo 49, o qual pode ser um gargalo no sistema em questão. O gráfico em setores a direita na mesma figura representa a percentagem de tempo consumido por cada rotina MPI. O processo representado consome a maior parte do seu tempo de execução nas rotinas `MPI_Wait` e `MPI_Recv`.

Capítulo 5

Resultados

Neste capítulo são apresentados resultados obtidos através do uso das ferramentas desenvolvidas em algumas aplicações paralelas.

Todos os testes descritos foram realizados no *cluster* do Laboratório de Alto Desempenho (LAD) do Instituto de Computação (IC) da UNICAMP. O *cluster* é composto por dois computadores Pentium 4 com 512 megabytes de memória RAM cada e 64 computadores Pentium III de 450MHz, com 256 megabytes de memória RAM cada. A tecnologia de rede utilizada na comunicação entre os computadores do *cluster* é a Fast Ethernet de 100Mbps com *switches* ponto a ponto.

5.1 Soccer Real-Time Tracking System (SORTTS)

A aplicação estudada é um sistema de rastreamento de objetos em imagens obtidas através de câmeras de vídeo em tempo real. Particularmente, os objetos rastreados são aqueles envolvidos em uma partida de futebol.

O sistema utiliza processamento paralelo devido a grande demanda computacional necessária para rastrear os objetos em tempo real. A aplicação chama-se `SORTTS` (*Soccer Real-Time Tracking System*) [29] e foi desenvolvida no Instituto de Computação da UNICAMP.

A aplicação encontra-se dividida em três módulos: o módulo de leitura, o módulo de rastreamento e o módulo de iniciação e acompanhamento. O módulo de leitura é responsável pela leitura das imagens a partir das câmeras e por enviar estas imagens ou partes delas para outros processos. O sistema opera com imagens obtidas por seis câmeras, de modo que o módulo de leitura é composto por seis processos, cada um obtendo imagens de uma câmera diferente. O módulo de rastreamento envolve processos que realizam rastreamento de objetos em imagens completas e em imagens reduzidas. O rastreamento de imagens completas também é feito por seis processos, cada um recebendo

imagens de um dos processos de leitura. O módulo de inicialização e acompanhamento é o responsável por iniciar e interromper os rastreadores de imagens reduzidas de acordo com as coordenadas obtidas através dos rastreadores de imagens completas e é formado por um único processo.

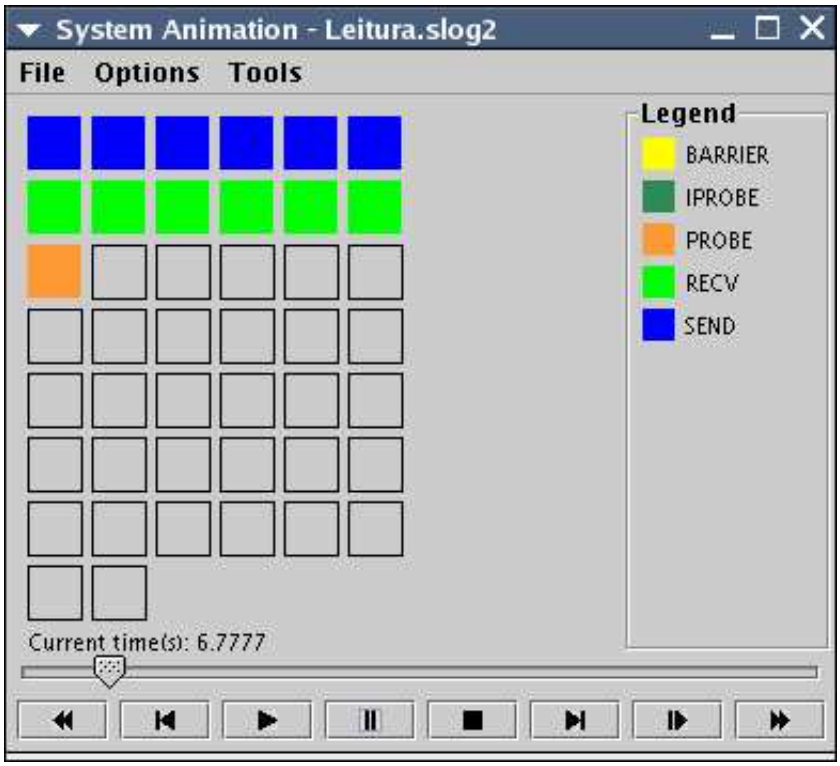

Figura 5.1: A ferramenta de animação de processos mostrando a execução do SORTTS.

Através da animação dos processos, foi possível identificar os diferentes grupos de processos devido ao seu comportamento diferenciado. O sistema funciona com os seis primeiros processos no módulo de leitura, os seis processos seguintes no módulo de rastreamento de imagens completas e com o décimo terceiro processo no módulo de iniciação e acompanhamento. Os demais processos atuam no rastreamento de imagens reduzidas. A figura 5.1 demonstra um quadro da animação. A ferramenta permite perceber a constante comunicação entre o módulo de leitura e os rastreadores de imagens reduzidas, bem como a constante atividade do processo que faz a iniciação e o acompanhamento. Periodicamente pode-se perceber o envio de imagens do módulo de leitura para o módulo de rastreamento de imagens completas. O envio de um conjunto de imagens completas é a situação retratada pela figura 5.1. Em outros momentos da animação é possível perceber a constante atividade de comunicação envolvendo os módulos de leitura, de acompanhamento e de

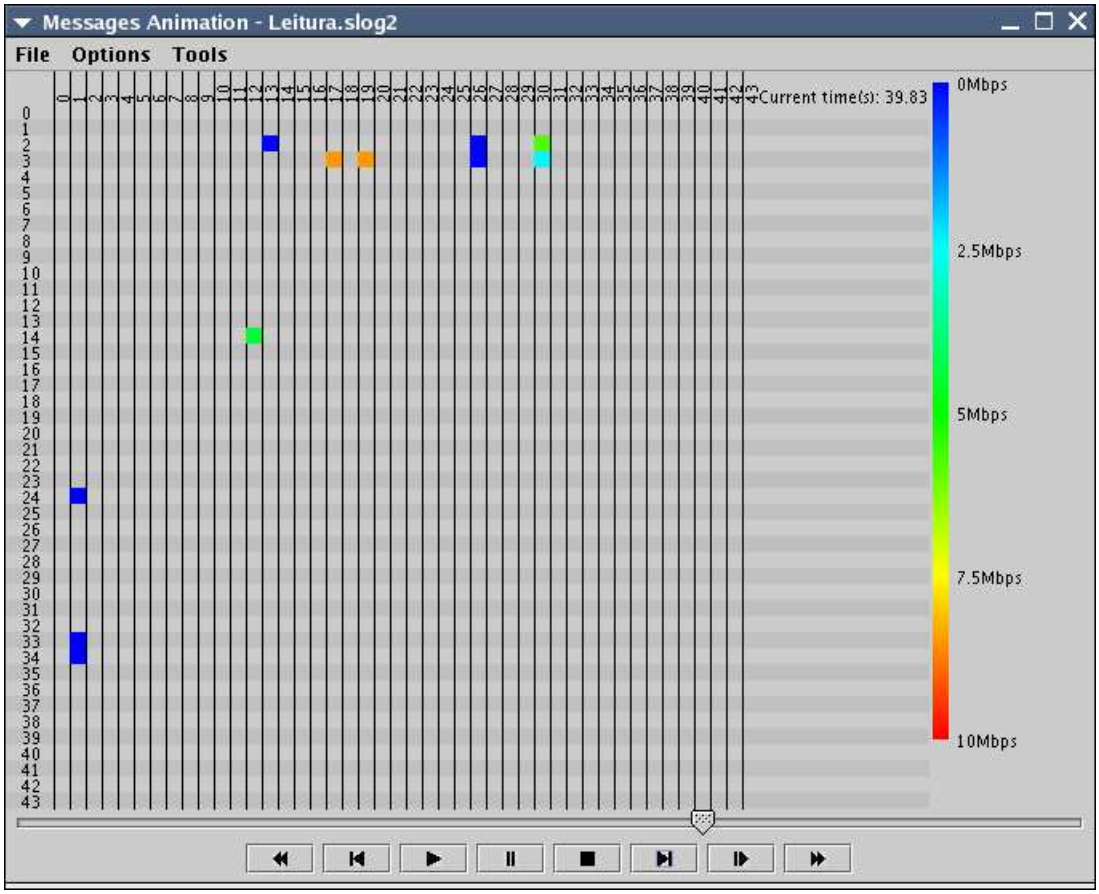

Figura 5.2: Tráfego de mensagens da aplicação.

rastreamento de imagens reduzidas.

O maior tráfego de mensagens, conforme mostrado na figura 5.2, ficou entre o módulo de leitura (seis primeiros processos) e os processos responsáveis pelo rastreamento de imagens reduzidas (processos 13 a 43). Este comportamento deve-se ao fato de o módulo de leitura precisar enviar constantemente imagens reduzidas para os processos 13 a 43. Salvo algumas exceções, o tráfego observado ficou abaixo de 10Mbps, em uma rede com capacidade nominal de 100Mbps. Esta observação se confirma no gráfico que representa o tráfego agregado de toda a execução, mostrado na figura 5.3. A comunicação mais eficiente foi dos processos zero a cinco (módulo de leitura) para os processos 6 a 11 (processos que fazem o rastreamento de imagens completas). A eficiência desta comunicação é justificada pelo fato de que são transmitidas imagens completas, formando mensagens grandes que efetivamente podem aproveitar o potencial da rede. A velocidade de comunicação efetiva nestes casos ficou em torno de 80Mbps.

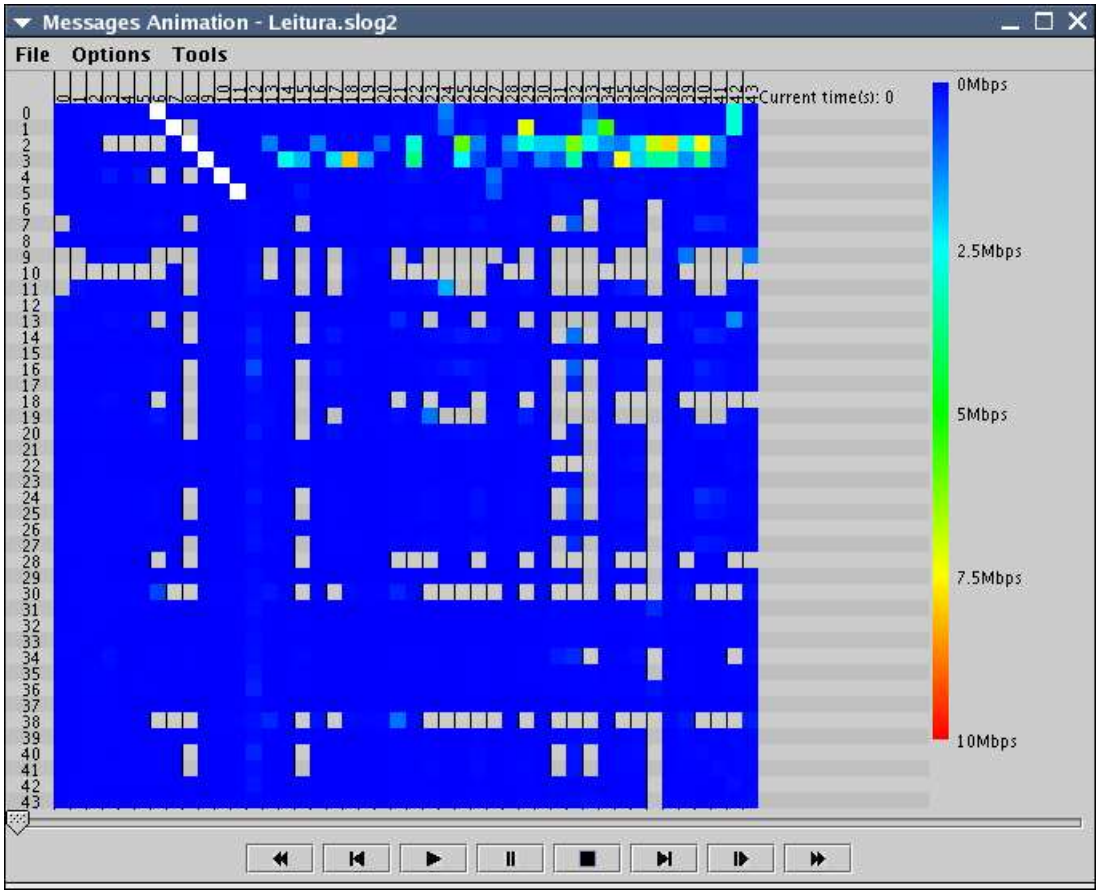

Figura 5.3: Tráfego de mensagens total da aplicação durante todo o tempo de execução.

As figuras 5.1 e 5.2 permitem apenas uma idéia aproximada das características do sistema apresentadas. Convém ressaltar que a observação das animações através das ferramentas possibilita uma percepção muito mais clara das atividades de comunicação do sistema e, conseqüentemente, a identificação das características da aplicação SORTTS discutidas anteriormente.

O tempo de espera mostrado nas figuras 5.4 e 5.5 evidencia a centralização das atividades no processador 12 (módulo de inicialização e acompanhamento), sugerindo que possivelmente este processador esteja sobrecarregado. Alguns processos esperam pelo processo 12 durante cerca de 50% do seu tempo de execução. Um grande número de processos consome mais de 75% do tempo de espera em função do processo 12. Novamente, os processos 6 a 11 se destacam por esperar bastante pelos seis primeiros. Mas, como foi visto que a comunicação está eficiente, o tempo de espera é grande apenas devido ao grande volume de dados transferido.

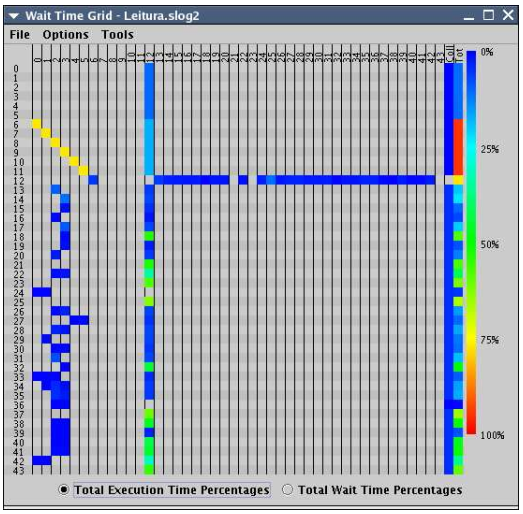

Figura 5.4: Estatísticas de tempo de espera mostram vários processos esperando pelo processo 12 por porcentagens significativas do tempo de execução do SORTTS.

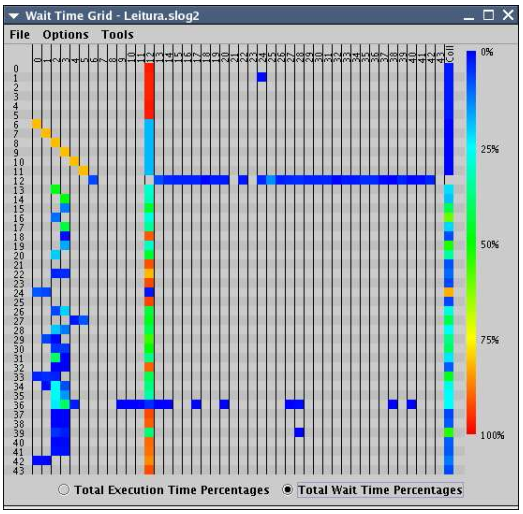

Figura 5.5: Tempo de espera de cada processo relativo ao seu próprio tempo total de espera no SORTTS.

Através do estudo da aplicação `SORTTS`, pode-se sugerir que o módulo de inicialização e acompanhamento inclua dois ou mais processos em vez de apenas um (o processo 12 na execução apresentada). Este tipo de modificação provavelmente melhoraria o desempenho do sistema, uma vez que é possível identificar o módulo de inicialização e acompanhamento

como sendo um gargalo do sistema.

5.2 HPL

A `HPL` (*High-Performance Linpack Benchmark for Distributed-Memory Computers*) [31] é uma implementação portável do *benchmark* Linpack [14] amplamente utilizada para sistemas paralelos. O *benchmark* resolve um sistema linear aleatório em precisão dupla em sistemas de memória distribuída.

Ao analisar a `HPL`, pode-se ver claramente a forma como o algoritmo da aplicação funciona. Na figura 5.6 é mostrado um quadro da animação do sistema. Através do ajuste da topologia da representação para uma topologia semelhante a definida na aplicação alvo, pode-se notar que a computação ocorre em uma linha de processadores por vez. A computação ocorre inicialmente na primeira linha de processos da animação. Após concluída a computação na primeira linha, pode-se visualizar o início do *broadcast* dos resultados para as demais linhas. A seguir é iniciado o processamento na segunda linha e assim sucessivamente.

A figura 5.6 demonstra a computação acontecendo na segunda linha de processos. No momento retratado pela figura, os resultados calculados pela primeira linha são enviados das linhas seis e sete para as linhas sete e oito, de acordo com o algoritmo de *broadcast* utilizado.

Na figura 5.7 é apresentada a animação da distribuição de carga da HPL. Pode-se perceber nesta animação um comportamento semelhante ao da animação de processos. Apenas algumas poucas linhas na matriz de processos apresentam atividade intensa a cada instante. O movimento das linhas em atividade da parte superior para a inferior da matriz lembra bastante o comportamento do algoritmo evidenciado pela animação de processos.

Tal possibilidade de visualizar tão claramente a execução de um programa é bastante útil para que os analistas possam avaliar o desempenho dos algoritmos implementados. Desta forma, torna-se possível aos usuários determinar se a execução de um determinado programa se comporta como o esperado ou não.

5.3 Implementação Paralela do fecho Transitivo de Dígrafos

Esta aplicação implementa um algoritmo para determinar o fecho transitivo de grafos dirigidos (dígrafos) [1]. A aplicação baseia-se no algoritmo de Warshall [40]. O dígrafo é representado por uma matriz de adjacências de tamanho $n \times n$, onde n é o número de

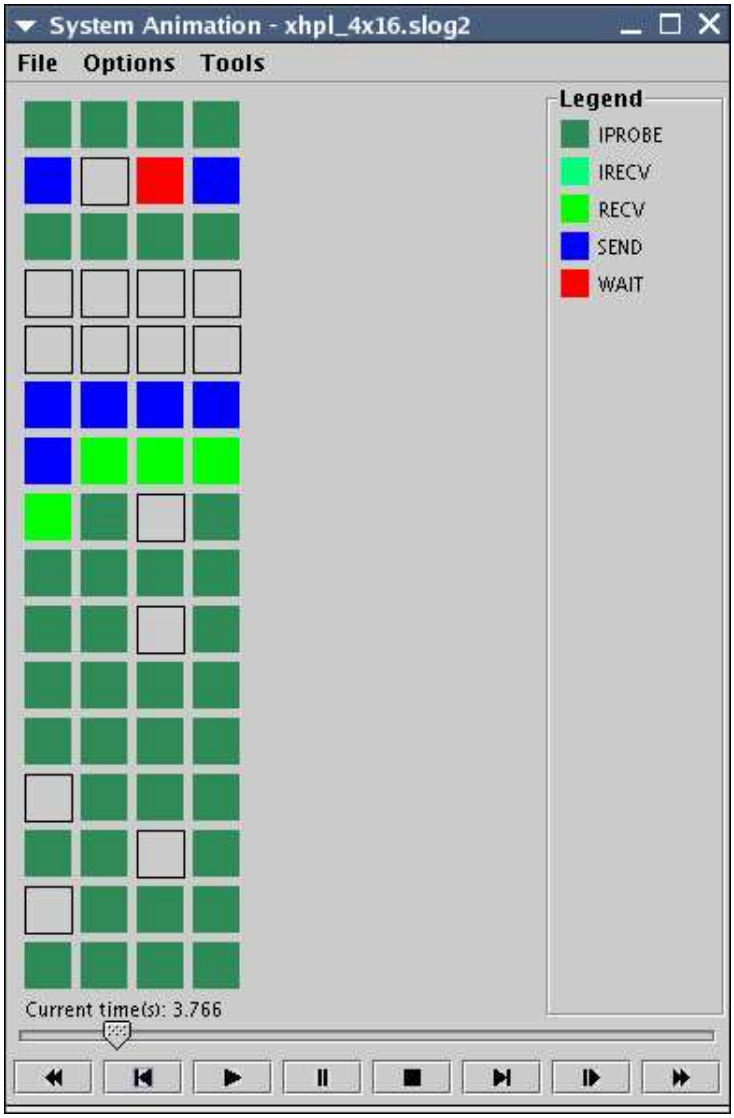

Figura 5.6: Animação da HPL com a computação ativa na segunda linha de processos e com *broadcast* sendo realizado nas linhas 6-8.

vértices do grafo. A matriz é dividida em p faixas horizontais e p faixas verticais, sendo p o número de processos. Os dados são distribuídos entre os processos de forma que cada processo receba uma faixa horizontal e uma faixa vertical.

Após a distribuição dos dados, cada processo calcula as novas arestas do dígrafo com base nas informações recebidas. O processo atualiza sua parte da matriz e armazena arestas a serem atualizadas pelos outros processos. Após a computação das arestas, os processos passam para uma fase de comunicações onde trocam as informações calculadas

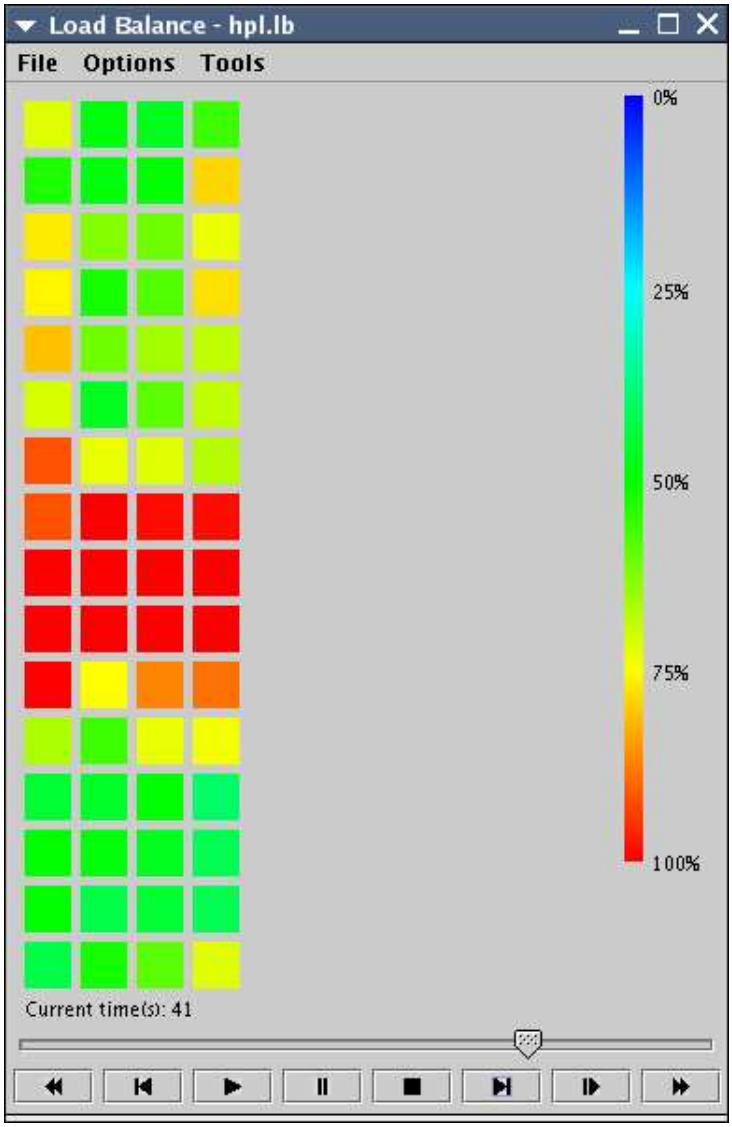

Figura 5.7: Animação da distribuição de carga da HPL com comportamento semelhante ao da animação de processos.

no passo anterior. A seguir, o sistema volta a fase de cálculo de novas arestas com base nas informações recebidas no passo de comunicação. O sistema continua alternando etapas de comunicação e computação até que nenhum processo realize atualizações.

O uso das ferramentas de *profiling* desenvolvidas permite notar o consumo de um tempo muito grande nas etapas de comunicação. A animação do sistema apresenta um tempo significativamente grande consumido pela rotina `MPI_Alltoallv` utilizada nas fases

de comunicação.

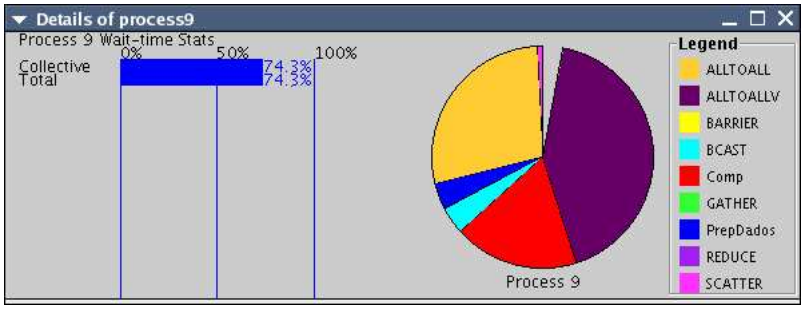

Figura 5.8: Detalhes de um processo em uma execução da aplicação original, com comunicação coletiva.

A figura 5.8 mostra claramente o tempo de `MPI_Alltoallv` em um processo típico da aplicação. O procedimento de comunicação coletiva consome quase a metade do tempo de execução da aplicação. Somando-se as duas rotinas de comunicação coletiva (`MPI_Alltoallv` e `MPI_Alltoall`) chega-se ao total de $74,3\%$ do tempo de execução do processo mostrado consumido em rotinas de comunicação.

Tendo reconhecido as etapas de comunicação como um gargalo do sistema, foram desenvolvidas algumas alternativas ao procedimento `MPI_Alltoallv`.

A aplicação foi modificada para utilizar cinco alternativas diferentes ao procedimento de comunicação coletiva original (`MPI_Alltoallv`). As cinco alternativas utilizadas foram: uso do procedimento `Sendrecv` com e sem algoritmo de *round-robin*; uso de envio de mensagens não bloqueante; uso de recebimento não bloqueante; e uso de ambos envio e recebimento não bloqueantes simultaneamente.

Com exceção da implementação utilizando *round-robin*, todas as implementações fazem o envio de mensagens em ordem, ou seja cada processo envia primeiro a mensagem para o processo zero, a seguir para o processo 1, e assim por diante. O recebimento, da mesma forma, é feito em ordem. Primeiro é recebida a mensagem do processo zero, e segue-se recebendo mensagens de cada um dos processos em ordem crescente. As quatro implementações deste tipo variam apenas quanto às rotinas de comunicação utilizadas, que podem ser envios e recebimentos bloqueantes, não bloqueantes ou o uso da rotina `Sendrecv`. Conforme mostrado a seguir, esta última abordagem foi a mais eficiente de todas.

A abordagem com *round-robin* também usa `Sendrecv`, mas faz os envios e recebimentos em uma ordem diferente. Utiliza-se uma topologia em anel onde cada processo n envia uma mensagem para o processo d a sua direita e recebe outra mensagem do processo e a

sua esquerda, a seguir, o processo n envia a mensagem para o nó a direita de d e recebe a mensagem do nó a esquerda de e, e assim por diante, até que o processo n tenha enviado e recebido mensagens de todos os outros processos.

Primeira Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	42.873.712	8,20	9,23	-12,56%	10,31	-25,64%	7,78	5,20%	6,40	22,02%	7,20	12,20%
16	86.114.460	13,50	10,88	19,39%	16,16	-19,67%	8,78	34,93%	7,02	48,03%	7,60	43.72%
32	80.225.532	8,80	7,09	19,45%	9,92	-12,67%	5,50	37,53%	4,26	51,65%	4,30	51,16%
64	27.548.820	1,22	1,10	9,36%	1,09	10,24%	1,06	13,30%	3,73	-205,88%	1,17	4,24%
Segunda Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	99.732	0,09	0,12	-41,40%	0,13	-45,08%	0,12	-44,33%	0,15	-77,14%	0,11	-32,43%
16	8.485.996	1,22	2,00	-64,57%	1,33	-8,94%	1,15	5,73%	0,75	38,56%	1,03	15,58%
32	115.791.334	20,37	53,69	-163,59%	14,25	30,04%	6,51	68,02%	6,24	69,38%	6,59	67,66%
64	375.953.316	97,19	140,97	-45,04%	31,42	67,67%	11,20	88,48%	9,42	90,31%	10,26	89,45%
Terceira Rodada												
Nós	Arestas	C. Col	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	0	0,08	0,12	-45,05%	0,12	-48,57%	0,12	-48,12%	0,14	-73,36%	0,10	-22,04%
16	0	0,08	0,09	-8,05%	0,12	-48,41%	0,10	-24,41%	0,07	22,15%	0,13	-56,14%
32	0	0,11	0,10	12,39%	0,09	20,16%	0,08	29,28%	0,10	11,71%	0,07	35,15%
64	0	0,32	0,27	15,50%	0,26	16,51%	0,22	30,42%	0,40	-27,15%	0,28	12,69%
Total												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	-	96,83	97,82	-1,02%	98,92	-2,16%	96,36	0,49%	93,86	3,07%	94,69	2,21%
16	-	56,76	54,93	3,23%	59,47	-4,77%	51,94	8,49%	49,00	13,66%	49,73	12,39%
32	-	68,90	100,56	-45,95%	63,89	7,28%	51,94	24,62%	50,77	26,32%	49,67	27,91%
64	-	123,93	167,85	-35,44%	58,06	53,15%	38,17	69,20%	37,09	70,07%	36,58	70,48%

Tabela 5.1: Grafo com 1280 vértices - Média de cinco execuções

Primeira Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	96.741.120	26,44	20,00	24,36%	21,19	19,85%	16,94	35,92%	13,98	47,13%	16,20	38,71%
16	214.207.202	27,57	22,60	18,04%	37,11	-34,59%	20,97	23,92%	17,54	36,37%	18,62	32,47%
32	446.695.252	48,43	83,63	-72,68%	70,45	-45,48%	25,57	47,20%	22,91	52,70%	22,13	54,30%
64	794.106.630	72,85	128,33	-76,15%	89,51	-22,86%	23,22	68,13%	19,18	73,68%	20,20	72,28%
Segunda Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	0	0,10	0,12	-19,64%	0,11	-10,60%	0,11	-2,46%	0,11	-4,02%	0,10	-0,16%
16	6.958	0,09	0,09	1,07%	0,10	-10,30%	0,11	-19,77%	0,09	-3,66%	0,11	-24,59%
32	3.058.578	0,29	0,33	-16,04%	0,37	-30,58%	0,30	-5,09%	0,32	-11,50%	0,34	-17,72%
64	125.301.880	18,43	36,79	-99,65%	7,47	59,47%	30,65	-66,35%	3,50	80,98%	4,44	75,89%
Terceira Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	-	-	-	-	-	-	-	-	-	-	-	-
16	0	0,09	0,09	0,07%	0,10	-11,44%	0,11	-20,41%	0,09	-3,14%	0,11	-23,72%
32	0	0,12	0,14	-10,34%	0,20	-58,82%	0,10	18,39%	0,15	-19,38%	0,14	-9,07%
64	0	0,12	0,14	-15,24%	0,13	-4,74%	0,14	-11,36%	0,14	-13,24%	0,13	-11,24%
Total												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	-	238,55	231,97	2,76%	233,18	2,25%	228,93	4,03%	224,48	5,90%	226,72	4,96%
16	-	181,07	176,01	2,79%	190,47	-5,19%	174,32	3,73%	169,31	6,50%	170,35	5,92%
32	-	198,55	230,48	-16,08%	216,26	-8,92%	171,34	13,70%	171,56	13,59%	166,16	16,31%
64	-	207,79	274,29	-32,01%	178,81	13,95%	136,63	34,25%	104,79	49,57%	105,48	49,24%

Tabela 5.2: Grafo com 1920 vértices - Média de cinco execuções

As tabelas 5.1 a 5.3 demonstram resultados de execuções da aplicação original em comparação às cinco alternativas. Cada tabela apresenta valores médios de cinco execuções das aplicações. As tabelas 5.1, 5.2 e 5.3 apresentam resultados obtidos em execuções de

Primeira Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	80.455.808	27,08	16,07	40,65%	18,58	31,39%	14,06	48,07%	10,80	60,11%	13,66	49,56%
16	7.265.378	1,44	1,20	16,75%	1,39	3,11%	1,20	16,81%	0,68	52,53%	1,15	20,24%
32	2.869.498	1,31	2,51	-90,78%	1,44	-9,32%	1,51	-14,83%	0,18	85,93%	2,67	-103,32%
64	2.268.370	0,55	0,55	-1,20%	0,58	-6,79%	0,58	-5,50%	3,09	-463,50%	0,21	61,61%
Segunda Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	79.485.494	24,35	15,87	34,82%	17,29	28,99%	15,28	37,27%	11,56	52,54%	13,63	44,05%
16	367.534.660	51,67	39,96	22,66%	63,19	-22,30%	36,69	28,99%	33,15	35,84%	32,74	36,64%
32	791.366.660	114,47	228,56	-99,67%	126,33	-10,36%	109,97	3,93%	109,42	4,41%	60,66	47,00%
64	1.490.196.114	282,49	348,56	-23,39%	217,66	22,95%	87,67	68,96%	128,23	54,61%	103,71	63,29%
Terceira Rodada												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	0	0,10	0,06	36,94%	0,05	48,14%	0,07	30,40%	0,08	22,77%	0,08	19,17%
16	0	0,06	0,06	10,73%	0,07	-3,56%	0,10	-50,10%	0,04	31,00%	0,09	-33,40%
32	2.817.774	0,32	0,29	7,43%	0,31	1,85%	0,30	7,27%	0,43	-36,36%	0,33	-3,57%
64	141.505.614	10,68	20,68	-93,62%	8,04	24,67%	15,29	-43,16%	3,92	63,25%	5,27	50,66%
Total												
Nós	Arestas	C. Col.	Não Bloqueante		Recv Não Bloq.		Send Não Bloq.		Sendrecv R.Robin		Sendrecv	
8	-	644,02	624,55	3,02%	628,47	2,41%	621,89	3,44%	601,38	6,62%	606,25	5,86%
16	-	333,39	321,50	3,57%	344,85	-3,44%	318,08	4,59%	311,11	6,68%	307,95	7,63%
32	-	513,40	504,02	1,83%	511,65	0,34%	745,53	-45,21%	656,73	-27,92%	436,89	14,90%
64	-	529,25	540,70	-2,16%	444,46	16,02%	363,74	31,27%	484,05	8,54%	313,57	40,75%

Tabela 5.3: Grafo com 2560 vértices - Média de Cinco Execuções

grafos de 1280, 1920 e 2560 vértices, respectivamente. Foram realizados testes com 8, 16, 32 e 64 processadores.

As tabelas apresentam o tempo das três primeiras rodadas de comunicação executadas pela aplicação para resolver o problema do fecho transitivo de grafos de diferentes tamanhos (1280, 1920 e 2560 vértices). Apenas as versões com 32 e 64 processadores da resolução do problema do grafo de 2560 vértices precisaram de mais de três rodadas para resolver o problema, ainda assim, em ambos os casos a quarta rodada representava o final do algoritmo, não sendo trocada mais nenhuma aresta. Além dos tempos das três primeiras rodadas em cada conjunto de execuções, também são mostrados os tempos totais de execução das aplicações, para que se avalie o impacto das alterações na aplicação como um todo e não apenas na comunicação em si. Os percentuais são calculados a partir da diferença entre o tempo da aplicação original e o tempo de cada alternativa em relação ao tempo da aplicação original, de forma que percentuais positivos representam melhorias e percentuais negativos representam pioras da implementação alternativa comparada à original.

Os gráficos mostrados nas figuras 5.9 e 5.10 apresentam um resumo dos dados apresentados nas tabelas 5.1 a 5.3 para as execuções com 32 e 64 processadores, respectivamente. O eixo vertical nos gráficos representa o tempo em segundos consumido por cada fase de comunicação e o eixo horizontal representa o número de arestas trocadas em cada fase. Deve-se notar que execuções relativas aos três grafos diferentes testados encontram-se representadas nos dois gráficos.

Ambos os gráficos mostram que três das cinco alternativas desenvolvidas são melhores

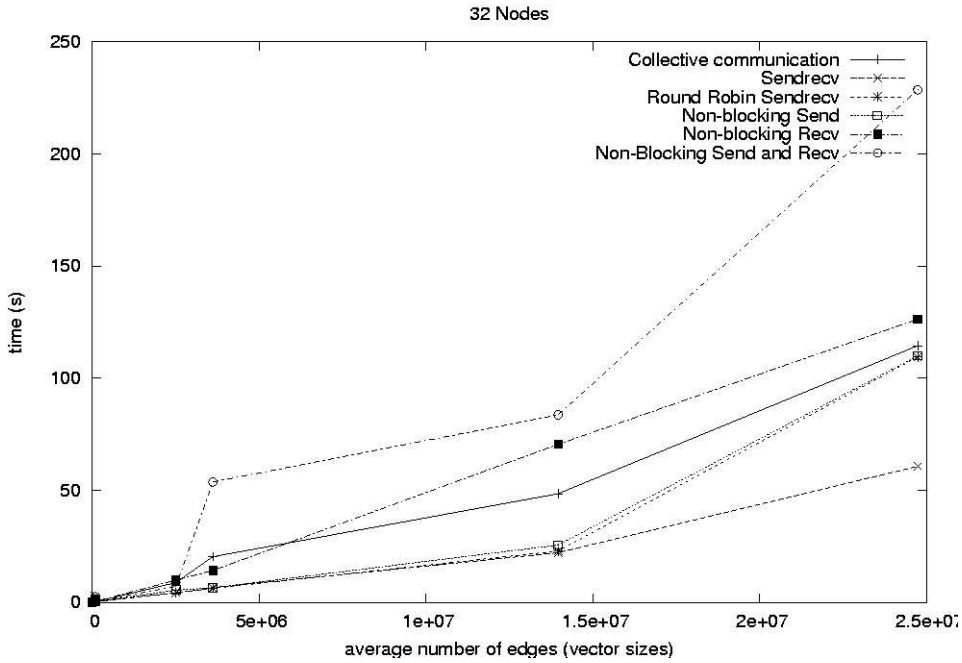

Figura 5.9: Tempos de diferentes rodadas de comunicação da aplicação utilizando 32 processos.

do que a implementação original com comunicação coletiva em todos as rodadas de comunicação significativamente grandes. A abordagem que utiliza recebimento não bloqueante é pior que a comunicação coletiva em alguns testes, mas aparenta ser mais estável. Apenas uma das alternativas mostrou um comportamento pior do que a aplicação original em todos os casos testados.

Pode-se comparar um processo da versão original com a versão que utiliza procedimentos `MPI_Sendrecv`. Os resultados são mostrados nas figuras 5.8 e 5.11. A figura 5.8 apresenta um total de $74,8\%$ do tempo de execução consumido por comunicação coletiva. Na figura 5.11, o tempo total de comunicação cai para $65,9\%$ do tempo de execução. Os gráficos em setores mostram que a porcentagem do tempo gasto em computação aumentou e que o tempo consumido pela rotina `MPI_Sendrecv` na figura 5.11 é muito menor que o tempo do `MPI_Alltoallv` correspondente na versão com comunicação coletiva.

Dada a significativa diferença entre o tempo consumido pela rotina `MPI_Alltoallv` e suas alternativas, uma outra possibilidade que provavelmente melhoraria ainda mais o desempenho da aplicação seria o uso de implementações alternativas também para a `MPI_Alltoall`.

A execução específica demonstrada na figura 5.11 foi $27,68\%$ mais rápida que a versão

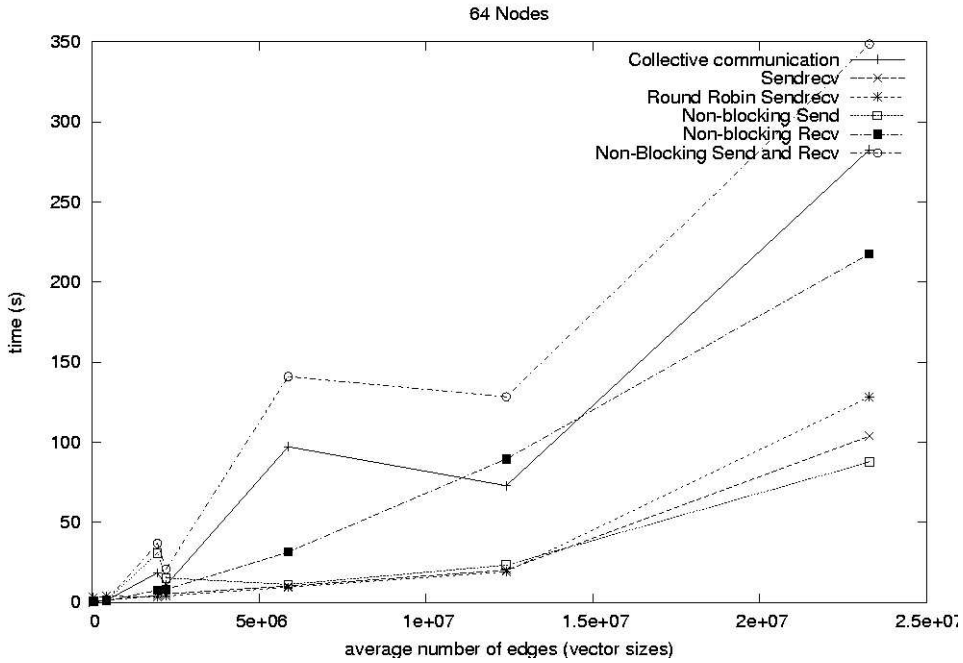

Figura 5.10: Tempos de diferentes rodadas de comunicação da aplicação utilizando 64 processos.

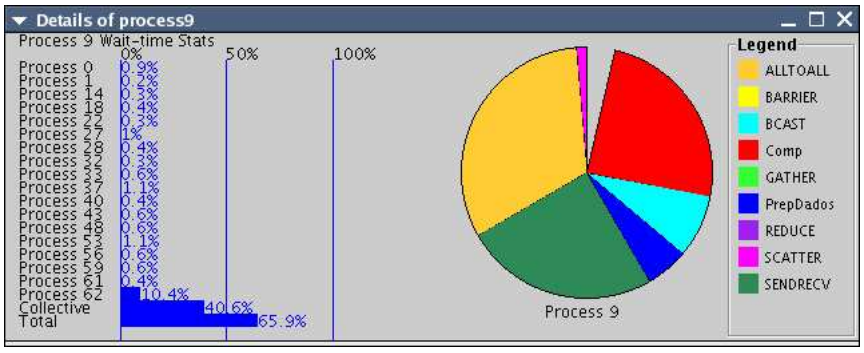

Figura 5.11: Um processo na versão com MPI_Sendrecv.

da figura 5.8. Para uma média de cinco execuções das duas aplicações, tal diferença de desempenho aumenta para $40,75\%$, conforme a tabela 5.3.

Pode-se afirmar, em conclusão ao estudo desta aplicação, que para os casos testados foi possível identificar um gargalo do sistema e desenvolver alternativas que melhoraram significativamente o seu desempenho.

Capítulo 6

Implementação

Este capítulo apresenta alguns detalhes da implementação das ferramentas apresentadas e avaliadas nos capítulos anteriores. São feitas algumas considerações sobre a geração dos arquivos de *log* e, posteriormente, é apresentada a implementação das ferramentas em Java.

6.1 Coleta de Dados

Para a maioria das ferramentas descritas neste documento, é utilizada a extensão a MPI chamada MPE [11]. Esta extensão inclui bibliotecas de *profiling* e facilidades para a geração de arquivos de *trace*.

A MPE permite a instrumentação de todas as rotinas MPI automaticamente, sem que se altere o código original. Também é possível a definição de outros estados e eventos através de chamadas a funções específicas no código da aplicação estudada. A intrusão da MPE é mínima, pois requer apenas que os eventos sejam gravados em um *buffer* local a cada processador. Toda a comunicação e a combinação dos dados obtidos em um único arquivo são feitos após a execução da aplicação alvo. A sincronização entre diferentes processadores para determinação de um relógio global também é feita apenas ao final da execução, não ocasionando intrusão. Para aplicações executadas por períodos muito longos, o modelo de relógio global da MPE pode ser inadequado devido a diferenças entre relógios locais dos processadores.

Com o auxílio da MPE, é gerado um arquivo de *trace* no formato `CLOG`. O formato `CLOG` é baseado em eventos, ou seja, trata-se de um conjunto de eventos com os respectivos rótulos de tempo.

O arquivo `CLOG` pode, posteriormente, ser convertido para o formato `SLOG-2` com o conjunto de aplicações `slog2sdk` [12]. O formato `SLOG-2` é baseado em objetos Java denominados *drawables*, e não em eventos como o formato `CLOG`. Tais objetos podem

representar um ou mais estados de processos ou mensagens do sistema. Um *drawable* indica, entre outras características, em que processo acontece e quais os tempos de início e fim no caso de tratar-se de um estado ou os processos de origem e destino e os tempos de envio e entrega no caso de mensagens. O formato foi desenvolvido para ser visualizado por ferramentas de visualização. Uma característica fundamental do formato `SLOG-2` é sua escalabilidade. É utilizada uma estrutura baseada em árvores binárias que classifica os objetos no arquivo de acordo com sua posição no tempo. Tal estrutura possibilita uma navegação eficiente mesmo em arquivos de alguns gigabytes de tamanho. Os arquivos `SLOG-2` são processados pelas ferramentas propostas.

Para a ferramenta de distribuição de carga, foi desenvolvida uma aplicação MPI em linguagem C, bastante simples, denominada `cpustatmpi`, que se comunica diretamente com o *kernel* do sistema operacional Linux para gerar arquivos de *log*, conforme especificado a seguir. Para o uso da ferramenta de distribuição de carga em outras plataformas deve-se desenvolver programas semelhantes que obtenham as informações do sistema operacional específico e gerem os arquivos de *trace* apropriados.

Para a geração dos arquivos de *trace* da distribuição de carga são necessárias pequenas modificações na biblioteca de *profiling* da MPE. Mais especificamente, é preciso alterar as funções `MPI_Init` e `MPI_Finalize` desta biblioteca. É conveniente ressaltar que toda aplicação MPI executa, obrigatoriamente, estas funções. A função `MPI_Init` deve criar um arquivo temporário que será utilizado para sincronização. Na implementação deste trabalho, convencionou-se o nome deste arquivo como `cpuloglock`, o qual deve ser criado no diretório `/tmp` do Linux. A função `MPI_Finalize` deve simplesmente apagar o arquivo criado. O programa `cpustatmpi` coleta os dados da distribuição de carga em cada processador, gerando um arquivo de *log* para cada processador. Os dados são coletados uma vez a cada período de tempo Δt. Quanto menor for o valor de Δt, mais fina será a granularidade e maior será o nível de intrusão. Valores de Δt em torno de um segundo permitem uma boa análise da distribuição de carga sem excesso de intrusão. O programa `cpustatmpi` coleta os dados durante o tempo de existência do arquivo `/tmp/cpuloglock`, ou seja, apenas durante a execução da aplicação alvo. Os dados são mantidos em um *buffer* na memória e copiados para o arquivo de saída de acordo com a necessidade.

Em alguns casos, o tempo de início da execução pode variar significativamente entre processadores diferentes. Tal variação também pode ser obtida através do `cpustatmpi`. O valor do tempo de início do *profiling* t_p é medido em cada processador p com relação a um tempo inicial de referência t_r obtido através da rotina de sincronização `MPI_Barrier`. O menor t_p entre todos os processadores será considerado posteriormente, nas etapas de visualização, como o início da execução t_0. Os tempos de início de todos os processadores podem então ser determinados com relação a t_0.

O formato dos arquivos a serem gerados em cada um dos processadores é extremamente

simples. Na primeira linha do arquivo, deve constar o valor de Δt em segundos. A seguir, na segunda linha, o valor do t_p específico do processador. As linhas subseqüentes contêm os valores percentuais de tempo ocioso da CPU para cada período Δt a partir de t_p, conforme informado pelo *kernel* através do sistema de arquivos `/proc`.

Foi desenvolvida ainda uma aplicação em Java que, a partir do conjunto de arquivos de *log* gerados pelo `cpustatmpi`, gera um objeto denominado `BalanceData` e grava este objeto em um arquivo, o qual serve de entrada para a ferramenta de distribuição de carga. Convencionou-se a extensão deste tipo de arquivo como `lb`. O objeto `BalanceData` e outros objetos Java das ferramentas desenvolvidas são discutidos na próxima seção.

6.2 Implementação das Ferramentas de *Profiling*

Para a implementação das ferramentas de visualização desenvolvidas foi escolhida a linguagem de programação Java. A implementação de todas as ferramentas totalizou aproximadamente 3500 linhas de código. Foi adotado para compilação e testes o J2SE (*Java 2 Platform, Standard Edition*) em sua versão 1.4.1.

Os principais motivos que levaram a opção pela tecnologia Java foram:

- Portabilidade, possibilitando que as ferramentas possam ser executadas em qualquer plataforma que possua uma máquina virtual Java;
- Compatibilidade com o formato `SLOG-2`, de forma que as ferramentas possam se integrar de maneira simples e eficiente com os arquivos de *log* que devem ler;
- Adequação da estrutura modular da programação orientada a objetos ao sistema desenvolvido;
- Suporte ao desenvolvimento de interfaces gráficas.

Foram desenvolvidos dois pacotes de classes Java para implementar as funcionalidades de todas as ferramentas. Foram utilizados ainda vários pacotes da biblioteca padrão Java e também alguns pacotes incluídos no `slog2sdk` para a leitura e utilização de objetos dos arquivos `SLOG-2`. A seguir são detalhados os dois pacotes desenvolvidos.

6.2.1 O Pacote *profiling*

A figura 6.1 mostra o diagrama de classes em UML (*Unified Modeling Language*) [19] das ferramentas desenvolvidas.

A classe abstrata `ProfilingAnimation` é a classe base dos três tipos de animação desenvolvidos. Os tipos de animação são representados pelas classes `SysAnim`, `MsgAnim`

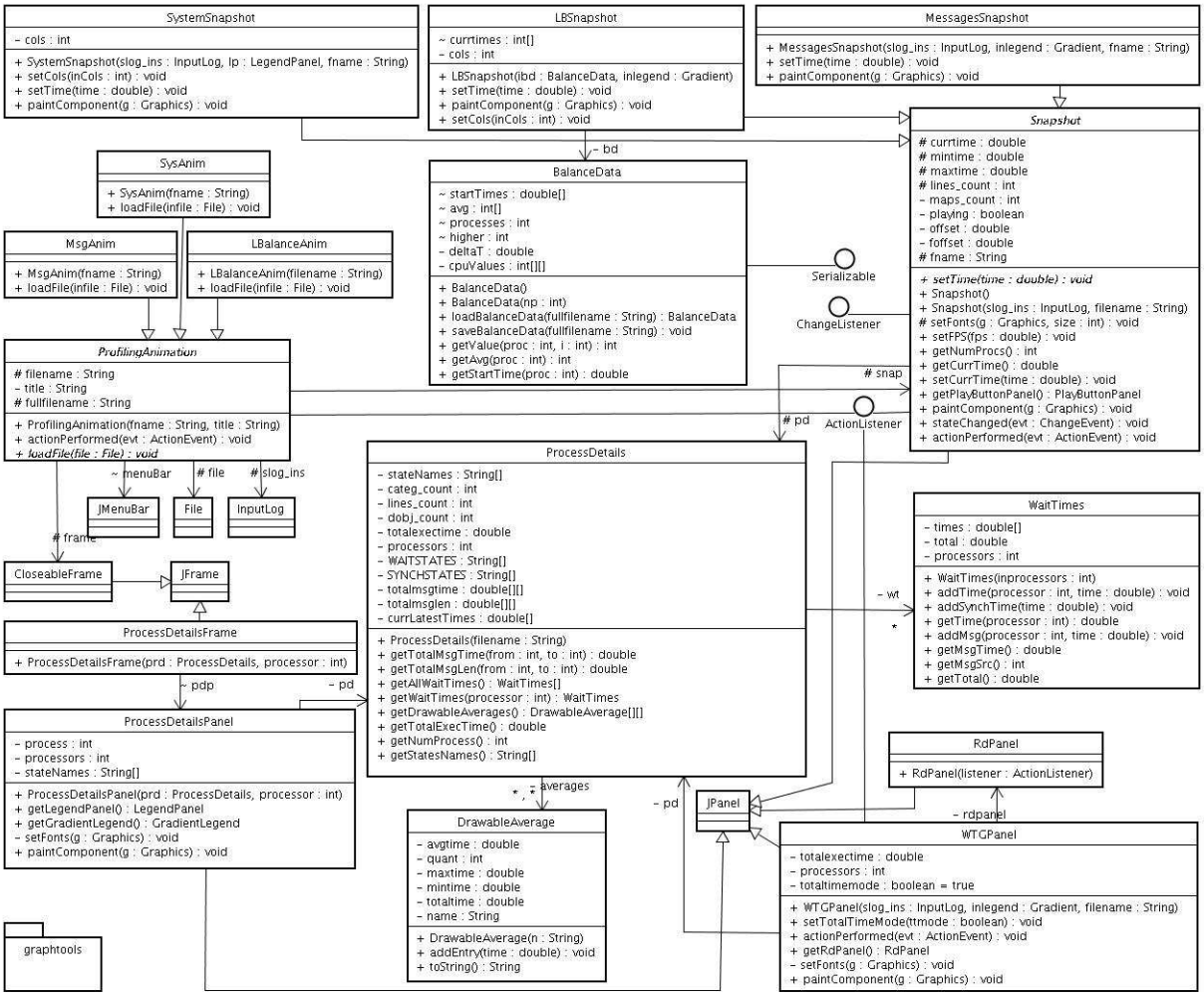

Figura 6.1: Diagrama de classes das ferramentas de *profiling*.

e `LBalanceAnim` utilizadas nas ferramentas de animação de processos, de mensagens e de distribuição de carga, respectivamente. Cada uma das três classes derivadas implementa um método `LoadFile` diferente para carregar um arquivo. As classes `SysAnim` e `MsgAnim` carregam arquivos `SLOG-2` obtendo os dados que necessitam para cada animação específica. O mesmo método `LoadFile` na classe `LBalanceAnim` carrega um arquivo `lb`.

Um dos principais atributos da classe `ProfilingAnimation` é um objeto `Snapshot`. A classe `Snapshot` também é abstrata e é progenitora das classes `SystemSnapshot`, `MessagesSnapshot` e `LBSnapshot`, uma para cada tipo de animação. A classe `Snapshot` e suas derivações contém uma série de atributos e métodos que descrevem características

fundamentais de uma animação em exibição. Os principais atributos são: tempos corrente, inicial e final, número de processos, tempo de simulação transcorrido entre quadros consecutivos em velocidade normal e rápida (avanço e retrocesso rápidos) e nome do arquivo utilizado. O principal método de um `Snapshot` é o método abstrato `setTime`. O método `SetTime` recebe um valor de tempo como parâmetro e gerencia as alterações em atributos e dispara métodos de atualização de componentes (`paintComponent`) apropriados para exibir o quadro de animação correspondente ao tempo recebido como parâmetro. A classe `Snapshot` implementa ainda as interfaces `ActionListener` e `ChangeListener` para responder a ações do usuário como cliques com o mouse na animação, em botões e no controle deslizante. As classes `SystemSnapshot` e `LBSnapshot` possuem um atributo e um método para armazenar e alterar o número de colunas de processos a serem mostradas. Desta forma, pode-se alterar a topologia apresentada nestas animações.

A classe `BalanceData` armazena alguns dados específicos da ferramenta de distribuição de carga. O atributo `cpuValues` armazena todos os valores percentuais de ociosidade de todos os processadores em cada período medido. A classe deve implementar a interface `Serializable`, pois seus objetos devem ser salvos em arquivos.

Uma classe muito importante utilizada por, praticamente, todas as ferramentas é chamada `ProcessDetails`. Esta classe armazena informações importantes como tempos e tamanhos de mensagens trocadas, número de operações (estados) diferentes dos processos e seus nomes, as rotinas definidas como de espera e de sincronização além de objetos `WaitTimes`, que armazenam informações referentes a tempos de espera, e `DrawableAverage` que guardam informações específicas para cada tipo de operação em cada processo.

A classe `ProcessDetailsPanel`, utiliza um objeto `ProcessDetails` para exibir os detalhes de processos individuais.

Por fim, a ferramenta de estatísticas de tempo de espera é implementada na classe `WTGPanel`.

6.2.2 O Pacote *graphtools*

O pacote `graphtools` foi desenvolvido como um conjunto de classes auxiliares para o pacote `profiling`. Este pacote tem por objetivo prover acessórios gráficos utilizados pelo pacote principal. O diagrama de classes do pacote `graphtools` é mostrado na figura 6.2.

A classe `PlayButtonPanel` representa um painel que contém os botões de controle da animação que possibilitam exibir a animação, efetuar uma pausa, a exibição em *slow motion*, avanço e retrocesso rápidos, avanço e retrocesso quadro a quadro, bem como parar a animação. Além dos botões, a classe disponibiliza também o controle deslizante para navegação e visualização do progresso da animação.

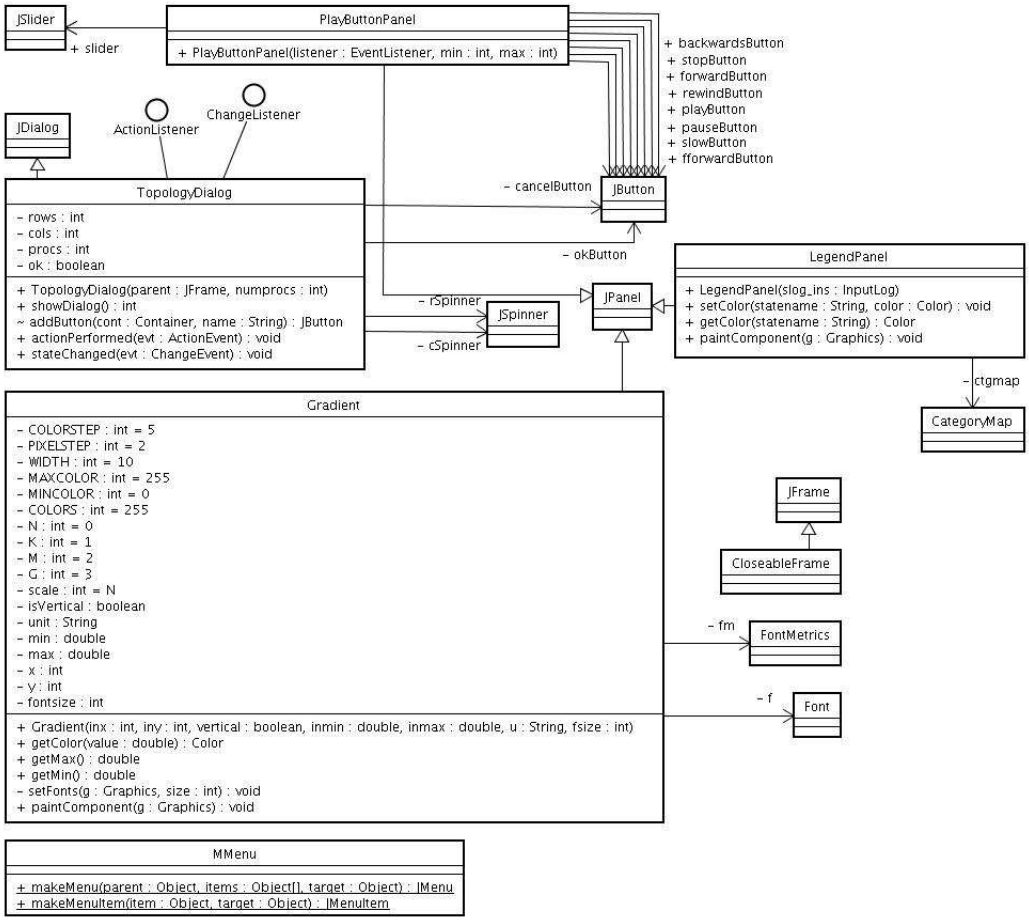

Figura 6.2: Diagrama de classes do pacote *graphtools*.

`TopologyDialog` é uma caixa de diálogo que permite que o usuário altere a topologia através da seleção do número de linhas e colunas de processos a serem apresentadas. A classe é utilizada sempre que a opção correspondente a mudança de topologia é selecionada na barra de menus de qualquer animação que suporte este recurso. A classe implementa as interfaces `ActionListener` e `ChangeListener` para interagir com o usuário.

As legendas utilizadas em várias ferramentas são implementadas na classe `LegendPanel`. A partir de um objeto `InputLog` disponível em arquivos `SLOG-2` obtém-se os nomes das operações que devem ser representadas. São disponibilizados métodos para obter e para alterar cores que representam os diferentes estados. O objeto `CategoryMap` armazena os estados e as respectivas cores.

A classe `Gradient` armazena as características e métodos referentes aos gradientes

de cores utilizados. Na criação de um `Gradient`, são informados os valores máximo e mínimo a serem apresentados e a unidade de medida. Com base nos valores máximo e mínimo, o gradiente utiliza uma escala normal, de quilos, megas ou gigas de acordo com a necessidade na apresentação dos valores.

A classe `MMenu` possui métodos estáticos que simplificam a criação de menus e é utilizada em todas as ferramentas.

Capítulo 7

Conclusão

O presente trabalho apresentou o problema da análise de desempenho de sistemas paralelos de computação e desenvolveu algumas metodologias para auxiliar tal análise. Foram implementadas ferramentas úteis na análise dos algoritmos paralelos. As metodologias permitem um estudo apropriado dos sistemas distribuídos.

Foram apresentados conceitos básicos de sistemas distribuídos e de análise de desempenho. Ressaltou-se a importância e as aplicações dos sistemas distribuídos na atualidade. Algumas abordagens da literatura e de produtos comerciais ao problema proposto foram discutidas.

As ferramentas de *software* desenvolvidas foram apresentadas detalhadamente. Tais ferramentas tornam possível um grande detalhamento das comunicações envolvidas nos sistemas. Pode-se identificar facilmente, através das novas formas de visualização desenvolvidas, as comunicações eficientes e ineficientes em um programa. As ferramentas permitem a identificação de gargalos e o estudo do comportamento dos processos envolvidos com mais detalhes.

A visualização de gargalos e da eficiência das comunicações, bem como o funcionamento de algoritmos paralelos e a distribuição de carga dos sistemas através das ferramentas foram objetivos alcançados com sucesso. Algumas das contribuições do trabalho já encontram-se publicadas [16].

Foram apresentados estudos referentes a aplicações paralelas reais que comprovam a eficiência das ferramentas em atingir seus objetivos. Especificamente, a discussão da aplicação SORTTS permitiu a identificação de um gargalo naquele sistema, no sistema HPL foi possível visualizar claramente o funcionamento do algoritmo e na aplicação de fechos transitivos de grafos foi possível identificar e melhorar uma ineficiência da comunicação utilizada.

Foram apresentados ainda aspectos da implementação do trabalho. Tal discussão incluiu aspectos dos arquivos de *trace* e também a implementação orientada a objetos das

ferramentas e suas interfaces.

Pode-se sugerir, como trabalho futuro, a implementação de melhorias no relógio global da MPE. Isto pode ser feito através de etapas de sincronização periódicas durante a execução ou através de análises estatísticas da variação entre os relógios locais em períodos de teste antes e/ou depois da execução da aplicação alvo. Este tipo de melhoria permitiria que as ferramentas fossem utilizadas também em execuções de longa duração.

Outra possibilidade é procurar alternativas para que seja possível acompanhar a execução das aplicações em tempo real, e não apenas em análises *post-mortem* como foi apresentado neste trabalho. Análises em tempo de execução permitem identificar problemas, como *deadlocks* por exemplo, em que o programa não termina. Além disso, a possibilidade de monitorar sistemas em tempo de execução permite que aplicações críticas possam ser monitoradas constantemente de forma que eventuais medidas corretivas possam ser tomadas o mais rapidamente possível.

Evidentemente, o presente trabalho não esgota o assunto de análise de desempenho ou as maneiras de se visualizar a execução de sistemas paralelos. Formas de visualização são bastante subjetivas no sentido de que cada pessoa pode se sentir mais confortável com um determinado tipo de visualização de um mesmo fenômeno. Além disso, cada diferente forma de visualização tende a ressaltar determinadas características específicas dos sistemas. Assim, o presente trabalho atinge seus objetivos através da ampliação da gama de possibilidades de visualização de execuções de sistemas paralelos existentes atualmente.

Referências Bibliográficas

[1] C. E. R. Alves, E. N. Caceres, A.A. Castro Jr., S. W. Song, and J.L. Szwarcfiter. Efficient parallel implementation of transitive closure of digraphs. *Lecture Notes in Computer Science*, 2840:126–133, 2003.

[2] Gregory R. Andrews. *Concurrent Programming*. Addison-Wesley Publishing Co., 1991.

[3] Robert Bernecky. Profiling, performance and perfection. In *Proceedings of the ACM/SIGAPL conference on APL as a tool of thought*, pages 31–52. ACM Press, 1989.

[4] Dimitri P. Bertsekas and John N. Tsitsiklis. *Parallel and Distributed Computation Numerical Methods*. Prentice Hall, 1989.

[5] Azer Bestavros. Load profiling, a methodology for scheduling real-time tasks in a distributed system. In *Proceedings of ICDS'97: The IEEE International Conference on Distributed Computing Systems*, Baltimore, Maryland, USA, 1997. IEEE Computer Society.

[6] Shirley Browne, Jack Dongarra, and Kevin London. Review of performance analysis tools for mpi programs. Technical Report 98–02, The U.S. Army Engineer Research and Development Center Major Shared Resource Center (ERDC MSRC), 1998.

[7] M. Calzarossa. Techniques and tools for performance analysis. In Allan Kent, James Williams, and Carolyn Hall, editors, *Encyclopedia of Computer Science and Technology*, volume 40, pages 309–335. Marcel Dekker, 1999.

[8] M. Calzarossa, L. Massari, A. Merlo, D. Tessera, and M. Vidal. A tool for performance analysis of parallel programs. *Rivista di Informatica*, 28(2):103–111, 1998.

[9] C. Carothers, B. Topol, R. M. Fujimoto, J. Stasko, and V. Sunderman. Visualizing parallel simulations in network computing environments: A case study. In *Proceedings of the 1997 Winter Simulation Conference (WSC '97)*, Atlanta, GA, USA, 1997.

[10] A. Chamlers and J. Tidmus. *Practical Parallel Processing*, chapter 1. Thompson Computer Press, 1996.

[11] Anthony Chan, William Gropp, and Ewing Lusk. *User's Guide for MPE: Extensions for MPI Programs*, 1998.

[12] Anthony Chan, William Gropp, and Ewing Lusk. Scalable log files for parallel program trace data. Technical report, Argonne National Laboratory, 2000.

[13] Helio M. de Oliveira, Jean M. Laine, and Edson T. Midorikawa. Algumas contribuições para modelagem de programas paralelos mpi. In *Anais do II Workshop de Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*, Campinas, SP, Brazil, 2003. SBC - Sociedade Brasileira de Computação.

[14] Jack J. Dongarra, Piotr Luszczek, and Antoine Petitet. The linpack benchmark: Past, present, and future. *Concurrency and Computation: Practice and Experience*, 15:1–18, 2003.

[15] Etnus totalview. http://www.etnus.com/Products/TotalView, 2004.

[16] Leonardo L. Fernandes and Ricardo Anido. Ferramentas de visualização para análise de desempenho de sistemas de computação distribuída. In *Anais do III Workshop de Desempenho de Sistemas Computacionais e de Comunicação (WPerformance)*, Salvador, BA, Brazil, 2004. SBC - Sociedade Brasileira de Computação.

[17] K. Ferschweiler, M. Calzarossa, C. Pancake, D. Tessera, and D. Keon. A community databamk for performance tracefiles. In Y. Cotronis and J. Dongarra, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface: 8th European PVM/MPI Users' Group Meeting, Proceedings*, volume 2131 of *Lecture Notes in Computer Science*, pages 233–240. Springer Verlag Heidelberg, 2001.

[18] Ian Foster and Carl Kesselman. Computational grids. In Ian Foster and Carl Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure*, pages 15–52. Morgan Kaufmann Publishers, 1998.

[19] Gilleanes T. A. Guedes. *UML Uma Abordagem Prática*. Novatec, 2004.

[20] Steven T. Hackstadt and Allen D. Malony. Visualizing parallel programs and performance. *IEEE Computer Graphics and Applications*, 15(4):12–14, July 1995.

[21] M. Heath and J. Etheridge. Visualizing the performance of parallel programs. *IEEE Software*, 8(5):29–39, September 1991.

[22] T. Kerola and H. Schwetman. Monit: a performance monitoring tool for parallel and pseudo-parallel programs. In *Proceedings of the ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 163–172. ACM Press, 1987.

[23] K. Kolence and P. Kiviat. Software unit profiles and kiviat figures. *ACM SIGMETRICS Performance Evaluation Review*, 2(3):2–12, September 1973.

[24] Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Comkputing Design and Analysis of Algorithms*. The Benjamin/Cummings Publishing Company, Inc., 1994.

[25] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978.

[26] Henry Lucas, Jr. Performance evaluation and monitoring. *ACM Computing Surveys*, 3(3):79–91, 1971.

[27] Eric Maillet and Cécile Tron. On efficiently implementing global time for performance evaluation on multiprocessor systems. *Journal of Parallel and Distributed Computing*, 28:84–93, July 1995.

[28] Mpi: a message passing interface standard. *International Journal of Supercomputer Applications*, 8(special issue on MPI), 1994.

[29] B. Muller Junior and R. Anido. Object detection with multiple cameras. In *Proceedings of the IEEE Workshop on Motion and Video Computing*, Orlando, FL, USA, 2002. IEEE Computer Society.

[30] W. E. Nagel, A. Arnold, M. Weber, H. Hoppe, and K. Solchembach. Vampir: Visualization and analysis of mpi resources. *Supercomputer*, 12(1):69–80, 1996.

[31] Antoine Petitet, R. C. Whaley, Jack J. Dongarra, and A. Cleary. Hpl - a portable implementation of the high-performance linpack benchmark for distributed-memory computers. http://www.netlib.org/benchmark/hpl/index.html, 2004.

[32] Rolf Rabenseifner. The controlled logical clock - a global time for trace based software monitoring of parallel applications in workstation clusters. In *Proceedings of the 5th EUROMICRO Workshop on Parallel and Distributed Processing (PDP'97)*, pages 477–484, London, UK, January 1997. IEEE Computer Society.

[33] Rolf Rabenseifner. Effective performance problem detection of mpi programs on mpp systems: From the global view to the details. In *Proceedings of the Parallel Computing'99*, Delft, Netherlands, 1999.

[34] Michael Raynal and Mukesh Singhal. Logical time: Capturing causality in distributed systems. *IEEE Computer*, 29(2):49–57, 1996.

[35] D. A. Reed, R. A. Aydt, R. J. Noe, P. C. Roth, K. A. Shields, B. Schwartz, and L. F. Tavera. Scalable performance analysis: The pablo performance analysis environment. In *Proceedings of the Scalable Parallel Libraries Conference*, pages 104–113. IEEE Computer Society, 1993.

[36] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

[37] D. Tessera and A. Dubey. Communication policies performance: A case study. In *Proceedings of the 9th EUROMICRO Workshop on Parallel and Distributed Processing (PDP'01)*, pages 491–497, Mantova, Italy, February 2001. IEEE Computer Society.

[38] B. Topol, , J. Stasko, and V. Sunderman. Pvanim: A tool for visualization in network computing environments. *Concurrency: Practice & Experience*, 10(14):1197–1222, 1998.

[39] Jeffrey Vetter. Performance analysis of distributed applications using automatic classification of communication inefficiencies. In *Proceedings of the 14th international conference on Supercomputing*, pages 245–254. ACM Press, 2000.

[40] Stephen Warshall. A theorem on boolean matrices. *Journal of the ACM*, 9(1):11–12, 1962.

[41] William E. Weihl. Specifications of concurrent and distributed systems. In Sape Mullender, editor, *Distributed Systems*, pages 27–53. ACM Press/Addison Wesley Publishing Co., 1993.

[42] Mason Woo, Jackie Neider, Tom Davis, and Dave Shreiner. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.2*. Addison-Wesley Professional, 3rd edition, 1999.

[43] Omer Zaki, Ewing Lusk, William Gropp, and Deborah Swider. Toward scalable performance visualization with Jumpshot. *High Performance Computing Applications*, 13(2):277–288, Fall 1999.

Apêndice A

CD-ROM

Para possibilitar uma melhor visualização das potencialidades das ferramentas propostas neste trabalho foi incluído um CD-ROM contendo algumas animações. A utilização do CD-ROM requer um *software* navegador de Internet capaz de abrir arquivos HTML e exibir figuras no formato GIF animado (i. e. Netscape, Internet Explorer, Mozilla, etc). O CD-ROM inclui também o texto completo desta dissertação em formato PDF. Para visualizar arquivos PDF é necessário um *software* apropriado.

Todos as animações e o texto da dissertação podem ser acessados a partir do arquivo `index.html` disponível no diretório raiz do CD-ROM. O arquivo citado funciona como um índice com *links* de hipertexto para todo o conteúdo do CD-ROM.

Nas próximas seções são descritas brevemente as animações do CD-ROM. Estas descrições também são disponibilizadas no próprio CD-ROM, nos arquivos HTML correspondentes a cada animação.

A.1 Apresentação das Ferramentas Desenvolvidas

Estas animações correspondem às figuras apresentadas no capítulo 4 da dissertação.

A.1.1 Animação de Processos

Esta animação apresenta a ferramenta de animação de processos (figura 4.1). Cada quadrado representa um processo e a legenda a direita mostra o significado de cada uma das cores. A cada instante, os diferentes processos mudam de cor de acordo com a tarefa que executam a cada momento. A aplicação alvo apresentada nesta animação é uma implementação da funcionalidade da rotina `MPI_Alltoallv`, onde cada processo recebe/envia mensagens de diferentes tamanhos de/para todos os demais processos. A implementação utiliza envio de mensagens não-bloqueante (`MPI_ISend`) e recebimento

bloqueante (`MPI_Recv`). A comunicação se dá em ordem, ou seja, os processos enviam mensagens primeiramente para o processo 0, depois para o processo 1 e assim por diante. A seguir, cada um dos processos passa a receber suas mensagens na mesma ordem crescente dos identificadores dos processos. Pode-se notar claramente, na animação, que os três primeiros processos encontram-se atrasados em relação aos demais. A animação que demonstra o funcionamento da ferramenta de tráfego de mensagens apresenta esta mesma situação.

A.1.2 Distribuição de Carga de Processamento

A ferramenta de distribuição de carga (figura 4.4) é apresentada nesta animação. A animação apresenta o sistema inicialmente em um período de relativamente pouca atividade, culminando com um período de utilização plena da capacidade de processamento.

A.1.3 Tráfego de Mensagens

A animação ilustra o funcionamento da ferramenta referente ao tráfego de mensagens (figura 4.5). O sistema é representado por uma matriz $p \times p$, sendo p o número de processos. Uma área colorida em uma posição qualquer (i,j) representa uma mensagem sendo transmitida do processo i para o processo j. As cores representam a taxa de transmissão efetiva da mensagem, sendo mensagens com taxas de transmissão maiores que 6Mbps representadas em branco (A ferramenta permite que se alterem os limites superior e inferior do gradiente de cores). A aplicação alvo apresentada nesta animação, bem como na animação de processos anterior é uma implementação da funcionalidade da rotina `MPI_Alltoallv`, onde cada processo recebe/envia mensagens de diferentes tamanhos de/para todos os demais processos. A implementação utiliza envio de mensagens não-bloqueante e recebimento bloqueante. A comunicação se dá em ordem, ou seja, os processos enviam mensagens primeiramente para o processo 0, depois para o processo 1 e assim por diante. A seguir, cada um dos processos passa a receber suas mensagens na mesma ordem crescente dos identificadores dos processos. Pode-se notar claramente, na animação, que os processos 0-2 começam a enviar e receber suas mensagens tardiamente, ocasionando uma comunicação ineficiente. Os demais processos se comportam conforme esperado, com as primeiras mensagens recebidas (mensagens dos processos de valores mais baixos) sendo transmitidas mais rapidamente do que mensagens de processos com valores mais altos.

A.2 Resultados

Esta seção discute animações referentes às aplicações apresentadas no capítulo 5, utilizadas para validação das ferramentas.

A.2.1 Aplicação SORTTS

Animação de Processos

A animação permite identificar cada um dos módulos da aplicação devido a seus comportamentos diferenciados. Os processos 0-5 representam o módulo de leitura, os processos 6-11 rastreiam imagens completas, o processo 12 faz a inicialização e acompanhamento e os demais processos funcionam rastreando imagens reduzidas enviadas pelo módulo de leitura. Periodicamente, é possível notar na animação o envio de imagens completas do módulo de leitura (processos 0-5) para o módulo de rastreamento de imagens completas (processos 6-11).

Tráfego de Mensagens

Esta animação permite verificar que a maior parte do tráfego de mensagens da aplicação se concentra entre os processos 0-6 que representam o módulo de leitura e os processos 13-43, módulo de rastreamento de imagens reduzidas. Este comportamento se deve ao freqüente envio de imagens reduzidas do módulo de leitura para o de rastreamento de imagens reduzidas. Pode-se notar que periodicamente ocorre o envio de imagens completas do módulo de leitura (processos 0-5) para o módulo de rastreamento de imagens completas (processos 6-11). Estas transmissões de imagens completas são as mais eficientes devido ao tamanho das mensagens. Pode-se notar também a comunicação entre o módulo de inicialização e acompanhamento (processo 12) e os demais módulos.

A.2.2 Aplicação HPL

Animação de Processos

A animação de processos da aplicação HPL permite identificar claramente o comportamento do algoritmo da aplicação. Selecionando na ferramenta a topologia apropriada, o processamento se dá em uma linha por vez, com os resultados sendo repassados as linhas inferiores através de um algoritmo de *broadcast*. Pode-se identificar o processamento na linha em que há maior atividade, com processos constantemente alternando entre envio de mensagens (cor azul), espera (cor vermelha) e recebimento não bloqueante (verde claro). Os demais processos encontram-se efetuando o *broadcast* de resultados anteriores (azul

para envio e verde para recebimento de mensagens) ou aguardando por novas mensagens (verde escuro).

Distribuição de Carga

Nesta animação, apesar da granularidade de um segundo, é possível identificar comportamento semelhante ao da animação anterior, com algumas poucas linhas em atividade mais intensa a cada momento.

A.2.3 Implementação Paralela do Fecho Transitivo de Dígrafos

Animação de Processos - Aplicação Original

Esta animação apresenta o comportamento dos processos na aplicação original. É possível notar a grande quantidade de tempo consumido em comunicação, especialmente na rotina `MPI_AlltoAllv`. Este comportamento encorajou o desenvolvimento de formas alternativas de se implementar tal comunicação. A animação mostra uma única rodada de comunicação representada por `MPI_AlltoAllv` com duração de cerca de quatro minutos e meio. Uma rodada de comunicação semelhante, com uma implementação alternativa apresentada na próxima animação consome apenas cerca de um minuto e vinte segundos.

Animação de Processos - Aplicação Alternativa

A implementação alternativa da rotina `MPI_AlltoAllv` com o uso de rotinas de envio não bloqueante (`MPI_ISend`) e recebimento bloqueante (`MPI_Recv`) mostra-se muito mais eficiente que a versão original. A animação do tráfego de mensagens desta mesma situação é discutida na seção A.1.3.

Apêndice B

Glossário de Funções MPI

Este glossário tem por objetivo apresentar brevemente o funcionamento de cada função MPI apresentada no texto e/ou nas figuras do presente trabalho.

MPI_Allreduce combina valores de todos os processos e distribui o resultado

MPI_Alltoall envia dados de todos para todos os processos

MPI_Alltoallv envia vetores de tamanho variável de todos para todos os processos

MPI_Barrier bloqueia o processamento até que todos os processos tenham chamado a rotina

MPI_Bcast envia mensagem de um processo para todos os demais

MPI_Finalize finaliza o ambiente de execução MPI

MPI_Gather coleta valores de um grupo de processos

MPI_Init inicia o ambiente de execução MPI

MPI_Iprobe testa se uma mensagem chegou de maneira não bloqueante

MPI_Irecv recebimento não bloqueante de mensagem

MPI_Isend envio não bloqueante de mensagem

MPI_Probe testa se uma mensagem chegou, bloqueando o processamento até que a mensagem chegue

MPI_Recv recebe uma mensagem

MPI_Reduce reduz valores em todos os processos para um só valor

MPI_Scan	calcula reduções parciais de dados em um grupo de processos
MPI_Scatter	envia dados de um processo para todos os demais em um grupo
MPI_Send	envia uma mensagem
MPI_Sendrecv	troca mensagens entre dois processos
MPI_Ssend	envia uma mensagem síncrona
MPI_Wait	espera a conclusão de um envio ou recebimento
MPI_Waitall	espera que todas as comunicações sejam concluídas
MPI_Waitany	espera que qualquer envio ou recebimento seja concluído