# Pré-processamento de Traces para Auxiliar a Visualização de Programas Paralelos

*Denise Stringhini*[1]
*Phillippe O. A. Navaux*[2]

Universidade Federal do Rio Grande do Sul
Instituto de Informática - II
Curso de Pós-graduação em Ciência da Computação
Caixa Postal 15064. Porto Alegre, RS, 91501-970, BRASIL

**Resumo**

O trabalho descreve uma ferramenta de visualização lógica de programas paralelos baseada na análise de *traces* de execução. A idéia básica consiste em aproveitar as informações fornecidas pela monitoração e que em geral são utilizadas apenas para dirigir animação *post-mortem* dos programas, e utilizá-las também na montagem das janelas de visualização.

Este artigo descreve tanto o pré-processador, quanto as janelas de visualização idealizadas a partir das informações que podem ser obtidas através do pré-processador.

**Abstract**

This work describes a logical visualization tool to parallel programs based on execution trace analysis. The basic idea consists in to make use of information provided by the monitoring phase, which are generally used to drive the post-mortem animation of programs, and to utilize it to construct the animation views.

This paper describes as much the preprocessor as much as the animation views idealized from the information wich can be obtained by the preprocessor.

## 1. Introdução

Nos últimos anos, o processamento paralelo tem se tornado uma das principais alternativas em se tratando de computação de alto desempenho. Neste contexto, a necessidade de que se desenvolvam técnicas e ambientes de programação paralela cada vez mais apurados e próximos ao usuário é inegável. Deste ponto, depende em grande parte o avanço da computação paralela.

Assim como a computação convencional (seqüencial), a programação paralela requer ambientes que facilitem a programação, depuração e avaliação de desempenho de programas. Entretanto, ela se torna mais complexa, na medida em que cada programa é constituído de vários processos que devem interagir em busca de um resultado.

As ferramentas para programação paralela devem possuir uma preocupação extra que é a de não sobrecarregar o sistema com sua própria execução. Isto poderia acarretar um comportamento diferente ao programa paralelo, inclusive podendo mascarar ou ocasionar erros em sua execução que não fazem parte da execução original (realizada sem o auxílio da ferramenta).

---

[1] Aluna do Curso de Pós-Graduação em Ciência da Computação - CPGCC-UFRGS - Área de interesse: processamento paralelo. E-mail: string@inf.ufrgs.br

[2] Doutor em Informática (Grenoble, 1979). Professor Pesquisador - CPGCC-UFRGS - Área de interesse: processamento paralelo. E-mail: navaux@inf.ufrgs.br

A partir desta restrição surge um mecanismo muito utilizado com o objetivo de reduzir ao máximo esta interferência. Este mecanismo consiste na geração de um *trace* de execução do programa paralelo, que pode ser utilizado de forma *post-mortem* (após a execução da aplicação). O *trace* consiste na monitoração do programa durante sua execução, coletando informações que possibilitem sua reexecução completa, de uma forma determinística, ou forneçam uma espécie de "relatório", com informações importantes sobre o comportamento do programa. Este relatório tem o objetivo de fornecer parâmetros para que o programador tire suas próprias conclusões sobre o desenvolvimento da aplicação, e é o tipo de *trace* utilizado na construção do pré-processador.

O escopo deste trabalho abrange os programas paralelos baseados em troca de mensagens, o que faz com que as máquinas próprias a este tipo de paradigma de programação sejam as fracamente acopladas, incluindo redes de estações de trabalho.

Neste contexto, a visualização de processos e/ou processadores torna-se um mecanismo bastante eficaz quando se quer ter uma visão geral da aplicação. Vários trabalhos têm sido desenvolvidos nesta área, dos quais pode-se destacar o ParaGraph [HEA 91], escrito originalmente para programas que utilizam a biblioteca PICL [GEI 90], o PVaniM [STA 96] e o XPVM [GEI 95], para programas PVM [GEI 94].

Este trabalho tem como principal objetivo facilitar e tornar mais intuitiva a compreensão e a avaliação de desempenho de programas paralelos através da visualização lógica do programa. Para isto, fornece uma quantidade maior de informações em cada janela de visualização e as organiza de forma que, à primeira vista, o programador já tenha uma idéia de quais são os principais problemas da aplicação que está desenvolvendo. Tem, então, a possibilidade de executar uma animação da execução do programa em várias janelas, que fornecem diferentes parâmetros de avaliação.

O grande diferencial em relação às outras ferramentas, já citadas, é a utilização do *trace* de execução numa fase pré-animação. Através da análise do *trace* antes da montagem das janelas, é possível que se as organize de tal forma que facilite a visualização, uma vez que já serão conhecidas diversas informações relativas ao comportamento do programa durante sua execução.

Além disso, demonstra-se os aspectos lógicos da comunicação entre os processos, tornando transparente a rede de interconexão. A única preocupação com os aspectos físicos da execução do programa é o mapeamento dos processos nos diversos processadores da rede ou arquitetura.

Inicialmente, serão abordadas algumas características relativas às ferramentas de depuração e análise. Na terceira seção é descrito o sistema como um todo. Na quarta, são consideradas as características do pré-processador e na quinta as janelas de visualização são descritas. A sexta e última seção apresenta a conclusão e o estado atual do trabalho.

## 2. Características das ferramentas de depuração e análise

Através da observação de diversas características das ferramentas de análise de programas paralelos que estão em evidência no momento, apresentamos uma classificação, levando-se em conta seus diferentes aspectos. Entretanto, salienta-se que, apesar de termos dividido as ferramentas em grupos distintos, essa divisão nem sempre é tão clara, ou seja, as ferramentas podem ter características de dois tipos ao mesmo tempo.

Em primeiro lugar, verificou-se que as ferramentas de auxílio à programação dividem-se em dois grandes grupos em relação ao objetivo a que se propõem: estão voltadas à depuração ou à análise de desempenho dos programas paralelos.

Uma outra característica muito importante diz respeito ao tipo de execução da ferramenta, que é o segundo aspecto apresentado. Este, descreve o nível de compartilhamento de recursos entre a ferramenta e a aplicação, especificando se a ferramenta vai ser executada total ou parcialmente em tempo de execução da aplicação.

O terceiro aspecto classifica o tipo de interação entre usuário e ferramenta, descrevendo seu modo de utilização. Já o quarto e último aspecto analisado apresenta as duas formas básicas de apresentação dos resultados: textual ou gráfica.

Esta classificação é apresentada na tabela 1.1.

| Classificação | Tipos | Descrição |
|---|---|---|
| Quanto ao objetivo. | Correção | Ferramentas voltadas à depuração que utilizam técnicas semelhantes às da depuração de programas seqüenciais. |
| | Desempenho | Ferramentas com facilidades (geralmente gráficas) para fornecer os parâmetros de desempenho do programa. |
| Quanto à execução. | *On-line* | Monitoração e apresentação de resultados em tempo de execução da aplicação. |
| | *Post-mortem* | Apenas a monitoração é feita em tempo de execução da aplicação. |
| Quanto à utilização. | *Meta-tool* | Usuário desenvolve sua forma de visualizar o comportamento da aplicação. |
| | *Toolkit* | Usuário utiliza um conjunto de janelas prontas. |
| Quanto à apresentação dos resultados. | Textual | Resultados na forma de texto; principalmente para demonstração de código. |
| | Gráfica | Resultados em janelas gráficas; principalmente para parâmetros de desempenho. |

**Tabela 1.1 Características das ferramentas.**

A ferramenta descrita neste tabalho se insere nas seguintes características: voltada ao **desempenho**, execução ***post-mortem***, utilização do tipo ***toolkit*** e apresentação essencialmente **gráfica** dos resultados.

## 3. Descrição do sistema

O esquema de análise *post-mortem* normalmente é dividido em dois passos: monitoração e animação. A monitoração fornece dados, geralmente um histórico da execução do programa (*trace*), para que a ferramenta de animaÁão possa, de alguma maneira, apresentar estes resultados (visualização).

O sistema aqui apresentado inclui um passo a mais no esquema de análise *post-mortem*, interpondo o pré-processador entre a monitoração e a visualização. O objetivo é captar do arquivo de *trace* as informações que servirão para a montagem das janelas de visualização.

A figura 2.1 ilustra este esquema, onde a monitoração e a correspondente geração do arquivo de *trace* é realizada pela ferramenta TAPE/PVM [MAI 95] para aplicações PVM e o pré-processamento, a geração do arquivo Info e a ferramenta de visualização são partes integrantes do sistema aqui apresentado.

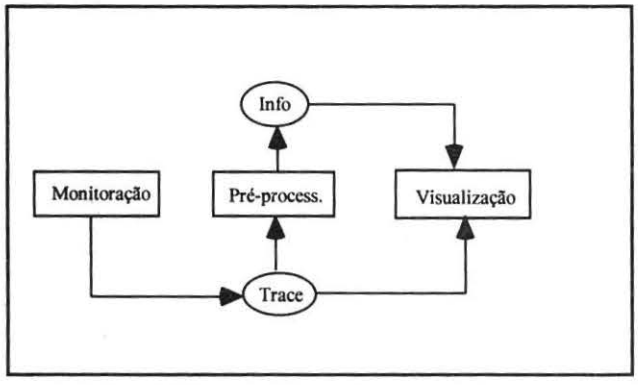


**Figura 2.1 *Post-mortem* com pré-processador**

O arquivo **Info** contém as informações coletadas do arquivo de *trace* e analisadas pelo pré-processador, que serão utilizadas pela ferramenta de visualização na construção e animação das janelas.

A ferramenta de visualização, por sua vez, é composta por quatro janelas que serão descritas adiante: Diagrama Espaço-tempo, Diagrama Kiviat, Diagrama Mapeamento de Processos e Grafo de Processos. Estas janelas possuem características de padronização e podem ser utilizadas em conjunto, onde cada uma demonstra um diferente aspecto da aplicação em análise.

## 4. Características do pré-processador

A monitoração é realizada pela ferramenta TAPE/PVM, que monitora programas PVM. Esta consiste num pacote que contém, além da ferramenta de geração de *traces*, um utilitário que os transforma do seu formato para o do PICL, utilizado pelo ParaGraph, e uma biblioteca de funções que facilitam a leitura dos *traces* gerados.

O pré-processador realiza uma leitura no arquivo de *trace*, de onde extrai as seguintes informações:

- identificadores numéricos de cada processo (*task id*);
- nomes dos *hosts* em que cada processo executou;
- nome do arquivo fonte de cada processo e sua ordem de criação (utilizados posteriormente para a formação da *string* de identificação de cada processo);
- processos para os quais cada processo enviou mensagens;
- número de mensagens enviadas por processo e por *host*.

Estas informações, depois de feita a leitura, são processadas. Este processamento consiste em arranjar as informações lidas de tal forma que possibilite que as janelas de visualização, descritas a seguir, possuam as informações necessárias ao seu funcionamento. Estas informações dizem respeito tanto ao posicionamento de cada elemento na janela, quanto a características específicas do comportamento do programa durante sua execução. Após este processamento, o resultado é gravado num arquivo de saída que contém informações para a criação das janelas de visualização.

## 5. Janelas de visualização

As janelas de visualização, que fazem parte do bloco de visualização (fig. 2.1) foram idealizadas considerando as vantagens trazidas pelo pré-processamento. Assim, algumas foram implementadas com base em outras ferramentas de análise, mas com alterações, e as outras foram construídas exclusivamente a partir de informações coletadas no arquivo de *trace*. A seguir são descritas as janelas implementadas: Diagrama Espaço-tempo, Diagrama Kiviat, Diagrama Mapeamento de Processos e Grafo de Processos.

### *5.1 Diagrama Espaço-tempo*

A primeira janela é bastante encontrada em ferramentas de avaliação de desempenho, mas que neste trabalho sofreu algumas alterações. É o Diagrama Espaço-tempo, que consiste numa representação bidimensional do sistema paralelo através do tempo, onde um eixo representa os processos e o outro, o tempo decorrido. Neste gráfico, cada processo é representado por uma linha horizontal, que representa seu estado, sendo a comunicação entre eles representada por uma linha que liga os dois processos comunicantes. Este tipo de diagrama está presente na maioria das ferramentas de análise, o que comprova sua grande utilidade no sentido de demonstrar o comportamento do programa paralelo, principalmente quanto à interação entre os processos.

As alterações implementadas a partir das informações obtidas com o pré-processador dizem respeito à ordem em que os processos estão dispostos no eixo x. Na maioria das ferramentas esta ordem não tem muita importância e, em geral, nem aparece especificada na documentação. Aqui, a ordem dos processos aparece como uma informação a mais, que antes da animação já pode ser interpretada.

Esta ordenação dos processos consiste em, num primeiro nível, agrupar os processos por *host* e apresentar estes grupos ordenadamente no eixo x, sendo o grupo relativo ao *host* que mais colocou mensagens na rede de comunicação, o primeiro de cima para baixo. Num segundo nível, os processos são ordenados internamente a cada

grupo, por quantidade de mensagens enviadas entre os próprios processos componentes da aplicação.

A figura 5.1 mostra um exemplo de diagrama espaço-tempo ordenado. Os processos estão distribuídos em quatro *hosts* que são ordenados por maior número de mensagens colocadas na rede. Dentro de cada grupo os processos estão ordenados por quantidade de mensagens enviadas.

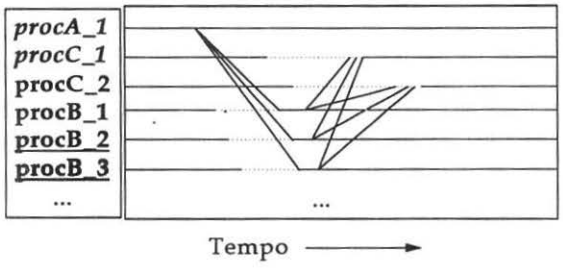


**Figura 5.1 Diagrama Espaço-tempo ordenado.**

A vantagem da ordenação dos processos no eixo x está, principalmente na inserção de mais uma informação ao programador sobre o comportamento da aplicação sem o acréscimo de poluição visual. Assim, o que ocorre é um reaproveitamento de um espaço da janela com informações que unem aspectos físicos e lógicos da comunicação global da aplicação. Físicos por que apresentam, no nível mais alto, informações sobre os *hosts* que formam a rede utilizada pela aplicação, e lógicos porque ao mesmo tempo demonstram a intercomunicação dos processos que a compõem. Note-se, também, que este tipo de organização permite identificar o mapeamento de processos na rede de *hosts*.

### *5.2 Diagrama Kiviat*

A segunda janela também foi idealizada através de um estudo de ferramentas que a utilizam, como o ParaGraph [HEA 91]. O Diagrama Kiviat possui diversas aplicações e tem sido constantemente utilizado para a avaliação de desempenho de sistemas paralelos. Consiste numa descrição geométrica do sistema, onde a unidade de informação é o processador (ou *host*). Nesta descrição, cada *host* é representado por um raio de um mesmo círculo que, usualmente, representa a utilização do *host* num determinado momento. A união destes pontos determinam os vértices do polígono que descreve o sistema, o que geralmente auxilia no balanceamento de carga.

Entretanto, o Kiviat aqui apresentado **não** demonstra o balanceamento de carga dos *hosts* que compõem a rede e sim o **percentual de mensagens colocadas na rede**

**por *host* em relação ao total da aplicação.** Assim, durante a animação, a figura geométrica apresenta os *hosts* que mais estão utilizando a rede de comunicação.

Além disso, a ordem em que as máquinas estão dispostas no círculo também é previamente processada. Ela corresponde à mesma ordem em que os grupos de processos estão dispostos no Diagrama Espaço-tempo, ou seja, a dos *hosts* que mais enviam mensagens. Assim, o sentido horário indica os *hosts* que mais utilizam a rede, o que acentua o desbalanceamento, se este estiver ocorrendo. A figura 5.2 mostra um exemplo de diagrama Kiviat de utilização da rede, que pode ser utilizado em conjunto com o Diagrama Espaço-tempo apresentado anteriormente.

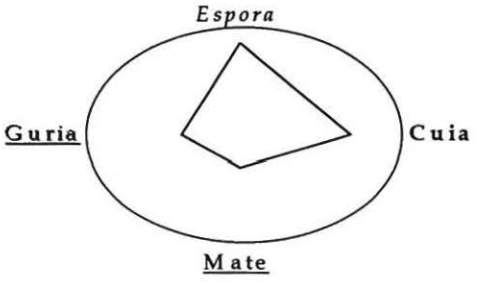


**Figura 5.2 Kiviat de utilização da rede.**

Esta abordagem para o diagrama Kiviat é vantajosa principalmente em se tratando de aplicações com um número razoável de processos, mas que serão executadas em redes ou arquiteturas com um número menor de processadores. Neste caso, o melhor desempenho é alcançado se os processos que se comunicam com maior freqüência forem mapeados para a mesma máquina, evitando sobrecarga na rede de comunicação. Neste sentido, o diagrama Kiviat aqui apresentado auxilia na detecção dos *hosts* que são os *gargalos* do sistema, ou seja, os responsáveis pela maior carga de comunicação da rede.

O funcionamento do Kiviat durante a animação é de análise do percentual de mensagens enviadas aos outros *hosts* a cada momento, em relação ao total de mensagens colocadas na rede pela aplicação. Assim, a figura tende a crescer e, ao final, mostra o percentual de cada *host* em relação a esse total. No caso de a figura ser homogênea indica uma utilização também homogênea da rede de comunicação pelos *hosts*.

### *5.3 Diagrama Mapeamento de Processos*

O terceiro tipo de janela não é encontrado nas ferramentas mais conhecidas de avaliação de desempenho de programas paralelos. Isto porque a maioria das ferramentas assume uma correspondência fiel entre a arquitetura paralela e o programa paralelo. Entretanto, a maioria das ferramentas de programação, como o PVM, são independentes da arquitetura e não obrigam um compromisso fiel entre os aspectos físicos e lógicos do programa.

Este diagrama leva em consideração esta observação e fornece o **mapeamento de processos** nos diversos *hosts* que compões a *máquina virtual* do PVM, considerando, inclusive, a hipótese de se ter mais de um processo por *host*.

O diagrama é composto por conjuntos de processos agrupados por *host*, que por sua vez, estão dispostos num grande círculo. Cada processo é um nodo cuja cor representa seu estado durante a animação (*enviando, recebendo, ocupado, inativo*). Esta representação dos estados do processo é bastante difundida em ferramentas deste tipo. A figura 5.3 apresenta um exemplo do Diagrama Mapeamento de Processos que pode ser utilizado em conjunto com os diagramas anteriormente apresentados.

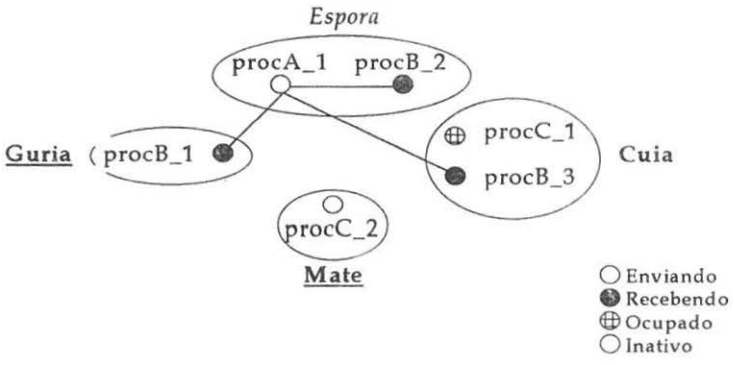


**Figura 5.3 Diagrama Mapeamento de Processos**

Este diagrama mostra de uma maneira direta e clara a máquina virtual montada com o PVM e a localização de cada processo durante a execução da aplicação. A animação demonstra a interação e o estado de cada processo a cada momento, o que é comum nas ferramentas de análise de desempenho. Entretanto, a maioria das ferramentas faz esta demonstração através dos *hosts* (fisicamente) e não dos processos (logicamente).

### *5.4 Grafo de Processos*

Por último, foi desenvolvida a janela Grafo de Processos que tem como objetivo principal levar ao extremo a representação lógica dos processos que compõem a aplicação paralela. Assim, o grafo é desprovido de qualquer informação que ligue os processos aos *hosts* em que foram executados.

Este grafo é obtido através de um algoritmo de desenho de grafos que aproxima os vizinhos: *Forced Directed Placement* [FRU 91]. As principais características deste algoritmo são:

- geração de um grafo não-orientado;
- distribuição uniforme de vértices na janela de visualização;
- tamanho uniforme de arcos e
- rapidez e simplicidade na implementação em relação a outros algoritmos com o mesmo propósito.

Neste grafo, da mesmo forma que no Diagrama Mapeamento de Processos, cada vértice corresponde a um processo e os arcos à sua comunicação. Além disso, o estado de cada processo, durante a animação, também é indicado por sua cor, obedecendo à mesma convenção do diagrama anterior.

Através desse grafo, pretende-se obter uma figura que represente logicamente o sistema, independentemente da configuração da rede ou arquitetura. Assim, cada aplicação terá uma figura correspondente, de acordo com a aproximação dos processos que mais se comunicam e da distribuição uniforme destes processos na janela de visualização. Salienta-se entretanto que, até o momento da finalização deste artigo, o algoritmo ainda não se encontra implementado e, portanto, não estão disponíveis dados sobre os resultados obtidos através desta implementação.

Apesar disso, apresentamos o que se pretende em termos visuais através desse grafo na figura 5.4. Esta, contém os mesmos processos apresentados nas anteriores, mantendo as características de padronização do conjunto de janelas.

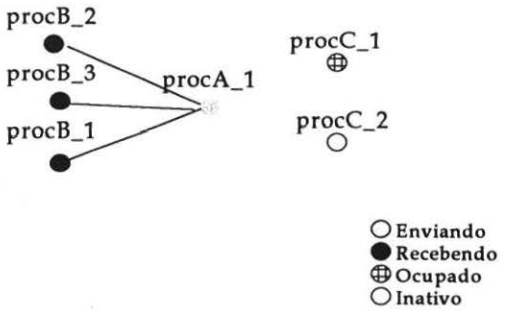


**Figura 5.4 Exemplo de Grafo de Processos**

## 6. Conclusão

Através de um estudo das ferramentas de avaliação de desempenho de programas paralelos mais importantes e utilizadas hoje em dia, observou-se algumas características que as tornam ainda alvo de críticas. A principal delas é a falta de janelas de visualização que demonstrem, de uma maneira clara e direta, a estrutura lógica da aplicação em análise. Isto se torna necessário principalmente para a análise de aplicações PVM, que são executadas a partir da máquina virtual criada pelo PVM e não têm qualquer compromisso com um tipo específico de arquitetura, podendo, inclusive ser composta de máquinas heterogêneas.

Além disso, faltam também mecanismos que, além de demonstrarem o mapeamento de processos na máquina virtual, possibilitem e facilitem a alteração deste mapeamento em função de um possível aumento de desempenho da aplicação. Isto se faz necessário à medida em que o PVM não obriga ao mesmo número de *hosts* e processos, permitindo mais de um por máquina. Ao mesmo tempo, não há uma política de balanceamento de carga, o que fica a cargo do programador da aplicação. Com isso, uma maneira de distribuição de processos através da máquina virtual poderia ser realizada a partir da colocação em um mesmo *host* de processos que possuem uma grande interação, evitando um alto tráfego de mensagens na rede de comunicação.

A partir destas observações, verificou-se que muitas das informações necessárias para se obter os tipos de visualizações requeridas teriam que ser obtidas **antes** da montagem das janelas e posterior animação das mesmas. Assim, chegou-se ao arquivo de *trace* que, por conter um histórico completo da execução do programa paralelo, contém

também todas as informações necessárias para suprir a montagem e animação das janelas de visualização.

O mecanismo de pré-processamento do arquivo de *trace* empregado no sistema apresentado possibilitou o aperfeiçoamento e criação de janelas de visualização que atendem aos requisitos e solucionam os problemas de análise de desempenho de programas PVM descritos acima.

O pré-processador se interpõem entre a fase de monitoração e a de animação, sendo um passo a mais na análise *post-mortem* de programas paralelos. Com isso, o pré-processador não influi nem no tempo de execução da aplicação, nem no tempo de execução da animação, ou seja, não há o indesejável compartilhamento de recursos que pode ser fatal em ferramentas de análise de desempenho de programas paralelos.

Foram aperfeiçoadas duas janelas já utilizadas em ferramentas de análise: Diagrama Espaço-tempo e Diagrama Kiviat, que sofreram alterações que possibilitam uma análise mais intuitiva e direta da aplicação paralela. Além disso, foram criadas duas novas janelas em função das informações obtidas através do pré-processamento: o Diagrama Mapeamento de Processos e o Grafo de Processos, que visam demonstrar os aspectos lógicos da aplicação e relacioná-los aos aspectos físicos.

O pré-processador foi implementado em linguagem C, com o auxílio da biblioteca de leitura de *traces* do TAPE/PVM, que é a ferramenta de monitoração que deve ser utilizada em conjunto com a aplicação PVM. A ferramenta de visualização está atualmente em fase final de implementação em ambiente de estações de trabalho Sun, linguagem C e interface baseada no sistema X Windows e desenvolvida com a biblioteca *Xlib*.

## 7. BIBLIOGRAFIA


[FRU 91] Fruchterman, T. M. J. e Reingold, E. M. *Graph Drawing by Forced-directed Placement* Soft. Pract. and Exp., vol 21(11), 1129-1164, Novembro 1991

[GEI 90] Geist, G. A.; Heath, M. T.; Peyton, B. W. e Worley, P.H. *A User's Guide to PICL - A Portable Instrumented Communication Library* Oak Ridge National Laboratory, 25 pp., Oak Ridge, Outubro 1990

[GEI 94] Geist, A.; Beguelin, A.; Dongarra, J.; Jiang, W.; Mancheck, R.; e Sunderam, V. *PVM* MIT Press, 1994

[GEI 95] Geist, G. A.; Kohl, James; Papadopoulos, Philip *Vizualization, Debugging, and Performance in PVM* Oak Ridge National Laboratory, 11 pp., Oak Ridge, Technical Report, 1995

[HEA 91] Heath, Michael T. e Etheridge, Jennifer A. *Visualizing the Performance of Parallel Programs* IEEE Software, vol. 8(5):29-39, Maio 1991

[MAI 95] Maillet, Eric *TAPE/PVM an Efficient Performance Monitor for PVM Applications - User Guide* LMC-IMAG, 19 pp., Grenoble, França, Technical Report, Março 1995

[STA 95] Stasko, John; Topol, Brad e Sunderam, Vaidy *PVaniM 2.0- - Online o Postmortem Visualization Support for PVM* disponível por ftp: ftp.cc.gatech.edu/pub/people/stasko