"Prober: Uma Ferramenta de Análise Funcional e de Desempenho de Programas Paralelos e Configuração de Cluster"

Luiz Eduardo S. Ramos[1], Luís Fabrício W. Góes[2], Carlos Augusto P. S. Martins[3]
Laboratório de Sistemas Digitais e Computacionais (LSDC) - Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais - PUC-MG
Av. Dom José Gaspar 500, Belo Horizonte, MG, Brasil
[1]{luizedu@pucmg.br} [2]{lfwg@pucmg.br} [3]{capsm@pucminas.br}

Resumo—
Este trabalho descreve uma ferramenta de análise funcional e de desempenho de programas paralelos que utiliza monitoramento para gerenciar a execução do programa, coletar métricas de desempenho e mostrar resultados através de gráficos e valores estatísticos. Além disso, o Prober é uma ferramenta que facilita toda a configuração de um ambiente paralelo em um cluster.

Palavras-chave— **Programas Paralelos, Análise Funcional e de Desempenho, Monitoramento, Cluster**

Abstract—
This work describes a performance and functional analyzer tool of parallel programs which uses monitoring to manage the program's execution, collects performance metrics and displays results through graphics and statistic values. Moreover the Prober is a tool that facilitates all the parallel environment's configuration in a cluster.

Keywords—**Parallel Programs, Functional and Performance Analyzer, Monitoring, Cluster**

I. Introdução

Diante de atividades que cada vez mais necessitam de alto desempenho, o processamento paralelo tem sido uma alternativa muito interessante em decorrência do limite de velocidade de processamento dos computadores seqüenciais existentes [HWA 98]. Nos últimos anos foram desenvolvidas várias máquinas paralelas, sendo que a arquitetura que tem ganhado um maior espaço, tanto no meio acadêmico quanto no meio comercial, é o cluster ou aglomerado de computadores, devido ao seu alto desempenho e baixo custo. Para que esse aglomerado de computadores trabalhe de forma otimizada, é necessário todo um ambiente de software e hardware, que envolve sistemas operacionais, compiladores, aplicativos, tecnologias de rede, protocolos de comunicação, ambientes de programação etc., e que estes sejam desenvolvidos preocupando-se com os problemas decorrentes do processamento paralelo.

A montagem de clusters nas universidades de todo o mundo para o desenvolvimento de programas paralelos tem se tornado uma prática muito comum devido ao grande volume de informações (tutoriais, cursos, livros etc.) e de softwares gratuitos, desenvolvidos com enfoque no processamento paralelo, que se encontram disponíveis pela internet [BUY 00] [AND 99]. Não só no exterior, mas também na comunidade acadêmica brasileira, existem alguns projetos com o objetivo de se criar *softwares* com interfaces gráficas que ajudem na gerência, monitoramento e desenvolvimento de programas paralelos em clusters, como por exemplo [XAV 00].

Há vários modos de se paralelizar programas em cluster, sendo estes através de compiladores paralelizantes, linguagens paralelas etc., mas a maneira mais utilizada hoje em dia são as bibliotecas com suporte a passagem de mensagens desenvolvidas para linguagens padrão como C, Java e Fortran que já são bastante difundidas. Algumas bibliotecas padrão, como MPI [MPI 01] e PVM [PVM 01], foram criadas para o desenvolvimento de programas paralelos.

Comparado ao processamento seqüencial, o paralelismo introduz novos conceitos que mudam a maneira tradicional de se desenvolver programas. Estamos acostumados a programação utilizando memória compartilhada, onde as variáveis estão localizadas em uma memória local, mas quando utilizamos máquinas paralelas onde existem recursos distribuídos por outras máquinas, é necessário a utilização de passagem de mensagens para requisitar a máquina remota dados que não estão em sua máquina local. Além disso o aluno, usuário ou programador, tem que projetar um código usando alguma metodologia que leve em conta os problemas decorrentes do paralelismo [FOS 95]. Em decorrência dessa multiplicidade de recursos, muitos alunos têm uma certa dificuldade de visualizar e entender este novo paradigma de programação.

Outros fatores importantes no processo de desenvolvimento de um programa paralelo é a validação de seu funcionamento e a análise de desempenho através de testes. Neste momento o aluno se depara com resultados que podem indicar limitações e vantagens do uso de paralelismo. Estes testes podem ser realizados usando benchmarks, ferramentas estatísticas, métodos manuais, softwares de monitoramento, simuladores, ou analisadores de desempenho.

A. Motivação

Durante o nosso projeto de pesquisa atual [FIP 00], que compreende-se na proposta e implementação de um ambiente de sistemas paralelos, dedicados e reconfiguráveis, e também durante a disciplina Processamento Paralelo e Distribuído do Curso de Pós Graduação de Engenharia Elétrica (PPGEE) da PUC Minas, tivemos contato com diversos problemas durante o desenvolvimento e teste de programas paralelos, dentre eles a necessidade de logar em cada máquina localmente, a configuração e transferência de arquivos manualmente, o disparo manual dos *deamons* (PVM), poucos mecanismos de depuração dos programas, falta de um software que fizesse com que cada programa fosse executado isoladamente sem a interferência de outros processos e principalmente dificuldade na validação e avaliação de desempenho dos programas, além de outras dificuldades.

Estes problemas fazem com que os alunos fiquem desmotivados e confusos, pois além de já terem que lidar com as dificuldades que os paradigmas de programação que utilizam passagem de mensagens e variável compartilhada oferecem, o aluno é obrigado a preocupar-se com a gerência explícita do ambiente e também a aprender lidar com diversas ferramentas, para que ele consiga passar por todas as etapas do desenvolvimento e teste de um programa paralelo, perdendo a maioria do seu tempo com atividades secundárias.

Baseado neste contexto, sentimos a necessidade do desenvolvimento de uma ferramenta que ajudasse na configuração, execução e teste de programas paralelos.

B. Objetivo

A ferramenta tem como principais objetivos fazer a análise funcional e de desempenho de programas paralelos e também facilitar a configuração, execução e teste destes em um cluster. O Prober fornecerá ao usuário diversas funcionalidades, dentre elas podemos destacar: I) possibilidade de login remoto; II) transferência automática de arquivos executáveis; III) interface gráfica de configuração dos arquivos de configuração do PVM e MPI IV) escolha do número de iterações do programa a ser executado; V) passagem de parâmetros através de uma interface gráfica e variação destes automática; VI) avaliação do programa usando métricas de desempenho como por exemplo o tempo de execução; VII) geração de gráficos e cálculo de médias; VIII) configuração automática através de arquivo IX) biblioteca de auxílio a medição de tempo e agendamento de tarefas. Todas estas funcionalidades visam simplificar o desenvolvimento e a análise do funcionamento e do desempenho de programas paralelos.

A ferramenta está sendo utilizada na validação do ambiente paralelo do cluster projetado e desenvolvido em nosso projeto atual e será usada para auxiliar no ensino de processamento paralelo [FIP 00].

Neste trabalho, que faz parte de um projeto de iniciação científica, estão sendo descritas todas as funcionalidades propostas para a ferramenta, mas vale ressaltar que ela possui limitações, que serão destacadas, por estar apenas em sua primeira versão. Em trabalhos futuros, como continuidade do projeto em execução, serão implementadas as outras funcionalidades ainda não presentes nesta versão inicial.

C. Apresentação dos tópicos

Este artigo é divido em 4 seções. Na seção II é apresentada a proposta do Prober. Na seção III são descritos os experimentos realizados e os resultados obtidos são analisados. Na última seção são discutidos os resultados, é feita a conclusão baseada nos resultados obtidos com a primeira versão do Prober e os trabalhos futuros são apresentados.

II. PROPOSTA DO PROBER

Durante os últimos dez anos, desde a primeira versão do PVM, depuradores, escalonadores de tarefas e balanceadores de carga (gerenciadores de tarefas), avaliadores de desempenho, benchmarks e monitores vêm sendo desenvolvidos em torno dos padrões PVM e MPI [GEI 00] [MPI 01]. Dentre essas ferramentas podemos destacar os benchmarks Linpack e Mpbench, os gerenciadores de tarefas CONDOR e LSF, os avaliadores de desempenho e também monitores AIMS, VAMPIR e Monito. Muitas dessas ferramentas são encontradas gratuitamente na internet [PTL 01] [PVM 01] [BEN 01] [SOL 00]. Uma ferramenta que gostaríamos de destacar é o XPVM (console gráfico que monitora programas desenvolvidos em PVM). Ele é um software que tem alguma semelhança conceitual com o nosso trabalho por ser um software monitor e também por fazer análises funcionais e de desempenho de programas paralelos [XPV 01].

O Prober é uma ferramenta que analisa o funcionamento e o desempenho de programas paralelos através de

monitoramento de execução e que também auxilia na configuração do ambiente paralelo, pois ele fornece uma interface gráfica que facilita a preparação do ambiente antes da execução do programa e durante a execução ele encarrega-se da coleta de métricas, cálculo de médias, armazenagem destes dados e exibição de gráficos relacionados ao desempenho do programa.

Nesta seção serão descritas todas as funcionalidades da ferramenta, tanto as implementadas quanto as que estão sendo implementadas., pois se trata da primeira versão do Prober.

Em sua primeira versão o Prober foi implementado em ambiente Windows NT usando a linguagem C++, pois é a linguagem mais difundida no meio acadêmico, e compilado com o Borland C++ Builder 4.0 que é o compilador disponível em nossa universidade. Nas versões futuras pretende-se implementá-lo em múltiplas plataformas: Linux, Windows ME/2000 e Solaris , utilizando a linguagem Java por ser mais portável. Utilizamos nesta versão a biblioteca WPVM, uma versão do PVM para Windows implementada pelo *Dependable Systems Group* da Universidade de Coimbra [ALV 95]. Nas próximas versões serão utilizadas outras implementações dos padrões PVM e também do MPI.

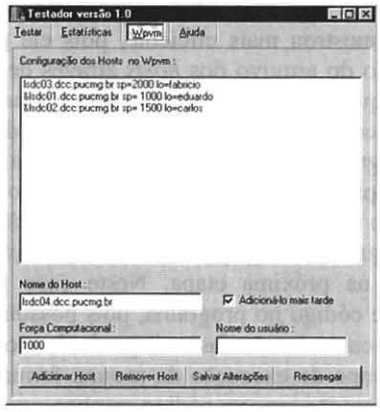

Fig. 1 – Tela de configuração dos hosts

A ferramenta suporta o preenchimento do arquivo de configuração dos *hosts* do WPVM, através da interface gráfica mostrada na Fig.1. Em sua tela principal apresentada na Fig.2 é possível selecionar o número de iterações para que o programa seja testado diversas vezes, minimizando as chances de erro nos cálculos estatísticos. Todas essas configurações podem ser salvas em arquivo, podendo ser carregadas posteriormente. Também é possível programar a seqüência da execução de programas com parâmetros diferenciados.

Ela implementa transferência de arquivos que é feita através da comunicação entre os deamons usando o protocolo TCP/IP. Essa transferência é usada para copiar automaticamente o arquivo executável do programa em teste para as outras máquinas pertencentes ao cluster, antes de sua execução, livrando o usuário deste trabalho.

É possível também agendar tarefas permitindo que os usuários do cluster garantam que apenas um programa paralelo, submetido através do Prober, esteja sendo executados ao mesmo tempo.

O Prober vem acompanhado de uma biblioteca com funções que auxiliam na coleta do tempo de execução em cada bloco de código especificado pelo programador. A função *inicia (int &Bloco)* inicia a contagem do tempo e a função *termina (int &Bloco)* encerra a contagem do tempo. Ao final da execução do programa é gerado um arquivo com os resultados obtidos, que é lido automaticamente pela ferramenta, exibindo-os na tela juntamente com os cálculos estatísticos.

Outra função pertencente a esta biblioteca, é a *pprintf( )* que funciona basicamente como o *printf( )*, só que a saída é gravada em arquivo ao invés de ser exibida na tela. Ao final da execução do programa o conteúdo do arquivo é exibido pelo Prober. Isto faz com que o programa não precise ser interrompido no meio da execução para conferir o seu andamento. Como esta ferramenta é também desenvolvida para alunos que estão aprendendo processamento paralelo, é interessante que eles tenham acesso as mensagens informativas exibidas no decorrer da execução do programa. Para aqueles que preferem ler as mensagens diretamente no processo em execução, foi criada a função *pgetch( )*, que é usada no lugar da função *getch( )* do C usada para paralisar a execução. O *pgetch( )* paralisa os contadores de tempo dos blocos e depois que alguma tecla é pressionada ele continua a contagem dos tempos.

Fig.2 – Tela principal de execução do Prober

Concluindo a apresentação do Prober, como indicado na introdução, ele possui algumas limitações nas suas funcionalidades, pois ele está ainda em sua primeira versão. O agendamento de tarefas ainda é feito manualmente, sem

impedir que outros programas rodem simultaneamente. Algumas funções da biblioteca estão sendo criadas e melhoradas, além de estarem sendo acrescentadas algumas novas funcionalidades estatísticas como cálculo de *speedup* e eficiência . A transferência de arquivos ainda é feita através do compartilhamento de arquivos fornecido pelo Windows NT.

Como um trabalho futuro, mas que já está sendo pensado, o Prober se tornará um gerenciador de tarefas com conceito de imagem única [BUY 97], onde os programas poderão ser escalonados, havendo balanceamento de carga pelo cluster. O agendamento de tarefas será dinâmico e os alunos ou usuários poderão solicitar a execução de seus programas paralelos através da internet, sem ter que se deslocarem até a universidade.

III. Resultados

Existem vários problemas clássicos que envolvem processamento paralelo, como ordenação, grafos, processamento de imagens, cálculos matemáticos etc [BUY 00] . Optamos por um problema relacionado ao cálculo matemático aqui representado pelo cálculo do pi. Este problema foi selecionado para teste e validação do Prober devido a sua simplicidade e seu alto grau de paralelismo.

Realizamos um experimento com a intenção de verificar se o Prober era mais eficiente que o método manual no desenvolvimento e teste de programas paralelos. Estamos, nessa primeira versão, interessados em testar um subconjunto de funcionalidades já implementadas do Prober. Dentre elas estão a interface gráfica que permite a configuração dos *hosts*, a medição automática do tempo de execução, o cálculo de médias e a geração de gráficos automática e também a seleção do número de iterações e parâmetros das execuções dos programas paralelos. Mas nos próximos trabalhos iremos testar as outras funcionalidades ainda não finalizadas, como as funções da biblioteca e a transferência automática de arquivos.

Dividimos o experimento em três etapas distintas: configuração, execução e avaliação dos resultados. A etapa inicial diz respeito ao login nas máquinas, configuração dos *deamons* (PVM), cópia de executáveis e configuração dos arquivos de *hosts*. Na segunda etapa estamos interessados na escolha do número de iterações, armazenamento dos dados e troca de parâmetros a cada execução. A última etapa se concentra na manipulação dos dados obtidos, como construção de gráficos, cálculo das médias e outros valores estatísticos.

Foram escolhidos dois métodos para fins de comparação: o manual e o método baseado no Prober. Primeiramente utilizamos o método manual que é atualmente usado em nossa universidade, onde o aluno precisa entrar com todos os parâmetros de configuração manualmente, executar o programa diversas vezes trocando os parâmetros manualmente e os resultados da medição de tempo são armazenados em *log* ou anotados, para depois serem inseridos em algum programa estatístico. O outro método é baseado no uso do Prober que auxilia o aluno nas três etapas do experimento.

Os programas foram compilados no Borland C++ 5.02 utilizando a biblioteca WPVM [ALV 01]. Tanto o compilador quanto a biblioteca foram escolhidos por serem gratuitos e compatíveis com ambiente Windows NT. Para realização do experimento foi utilizado um cluster composto de 3 Pentium II Celeron A 450 MHz com 64 MB de RAM conectados através de um hub usando uma rede Ethernet/Fast Ethernet 10/100 Mbps.

A. Etapa Nº 1 - Configuração

Na primeira etapa é feita toda a preparação ou configuração do ambiente para a execução do programa paralelo.

Começamos a contar o tempo a partir do momento em que efetuamos os logins nas máquinas. Depois foi feita a cópia do arquivo executável pvmpi.exe para todas as máquinas através do compartilhamento de arquivos oferecido pelo Windows NT. Até este instante tanto com o Prober quanto com o método manual, o tempo transcorrido era o mesmo, mas ao configurar os arquivos dos *hosts*, o Prober se mostrou mais eficiente, pois ele possibilitava a modificação do arquivo dos *hosts* através de uma interface gráfica, o que não ocorreu com o método manual, onde foi necessário editar o arquivo através do bloco de notas.

Mas o grande diferencial nessa etapa, foi o tempo gasto para se implementar, diretamente no código do programa, uma função que gravasse o tempo de execução do programa em um arquivo de *log*, para evitar a anotação manual dos resultados na próxima etapa. Neste caso foi possível a inclusão de código no programa, pois possuíamos o código fonte, mas caso ele não fosse disponível, não seria possível medir o tempo de execução da mesma forma. De qualquer forma a alteração em um código implica que o aluno tenha que ter uma certa experiência em implementação de funções de medição de tempo, impedindo os alunos com menos experiência de realizarem o experimento com o mesmo tempo dos experientes, pois eles terão que ainda aprender como implementar estas funções. O Prober já coleta e armazena estes dados automaticamente.

B. Etapa Nº 2 - Execução

Na segunda etapa preocupamos em avaliar o trabalho que é gasto na execução de um programa paralelo repetidas vezes.

No método manual, foi preciso executar o programa no modo console, pois era necessária a passagem de parâmetros ao executável. A experiência de executar um

programa diversas vezes alterando regularmente seus parâmetros mostrou-se extremamente exaustiva e desmotivadora. Além de se ter que preocupar com a renomeação dos arquivos de saída para evitar a sobreposição de dados. Também ocorreram falhas na execução dos processos devido a submissão de mais de um processo ao mesmo tempo (erro humano), implicando em repetição de medidas. Outra falha encontrada foi no número não exato de execuções do programa realizadas, pois foi necessário contar o número de execuções manualmente.

Este teste poderia ser feito sem a otimização implementada na etapa nº 1, onde ao invés do tempo gasto na execução ser gravado em arquivo, ele fosse mostrado na tela, obrigando a anotar-se os dados manualmente. Mas ao executar o programa desta forma, notou-se um tempo gasto de 15 segundos para cada iteração do programa, entre digitar os parâmetros, executar o programa (programa com tempo de execução menor que 1 segundo) e anotar os valores em um papel ou no bloco de notas. Desta forma seriam gastos no mínimo 25 minutos para cada 100 iterações, como haviam 9 variações de parâmetros, o teste demoraria 3 horas e 45 minutos, sendo totalmente indesejável.

Já com o uso do Prober, bastou apenas escolher o número de iterações, o executável e os parâmetros requeridos através de uma interface gráfica amigável e iniciar a execução do programa. Com o Prober não houveram falhas na execução dos processos.

Em questão de tempo, os dois métodos gastaram tempo equivalente. Mas em relação a facilidade de uso, o Prober foi muito superior.

C. Etapa Nº 3 - Avaliação

Nesta última etapa do experimento, avaliamos como os dois métodos se comportaram no momento do cálculo e geração dos dados estatísticos.

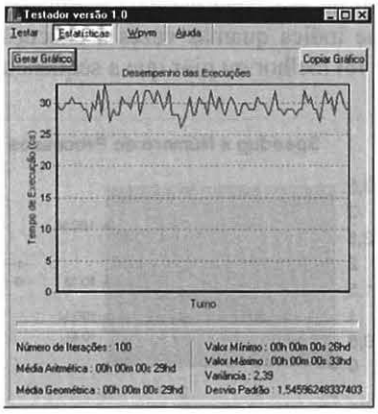

Fig.3 – Tela do Prober com o gráfico resultante de 100 iterações do programa pvmpi com 1 milhão de intervalos.

No método manual, foi necessário utilizar uma ferramenta estatística, no caso o MS Excel, para computar os dados da medição de tempo contidos nos arquivos de log. Os dados foram colados na planilha do Excel e foi feita toda uma configuração para que fossem gerados o gráfico, a média, o valor máximo e o mínimo encontrados na amostra. Esse processo foi repetido para cada amostra (arquivo de *log*).

O Prober foi muito superior ao método manual nesta última etapa, pois a criação dos gráficos (exemplo Fig.3) e dos outros cálculos estatísticos foi imediato, sem ser necessário utilizar-se outras ferramentas auxiliares. Os gráficos obtidos foram muito semelhantes aos gerados no Excel.

D. Análise dos Resultados

O programa pvmpi utilizado neste experimento, foi implementado de tal forma que ele recebesse como parâmetros o número de intervalos, o número de processos e o número de máquinas em que se deseja criar os processos. Foram utilizados 10.000, 100.000, e 1.000.000 intervalos, onde para cada um destes foram utilizados número de processos e máquinas iguais a 1, 2 e 3 simultaneamente. Cada uma dessas 9 variações foi executada 100 vezes tanto no método manual quanto no método utilizando o Prober, para eliminar a margem de erro nas medidas.

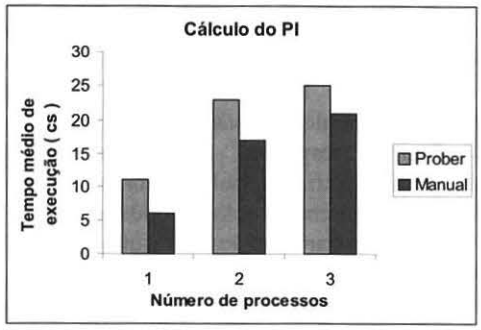

Fig.4 – Execução do pvmpi.exe com 10000 intervalos.

Na Fig.4 podemos observar uma comparação entre o tempo medido pelo Prober e pelo método manual que se baseou em medir o tempo de execução do programa desde o início da sua função principal até o seu final.

Nota-se uma diferença de aproximadamente 50 centésimos de segundo em média entre as medidas de tempo do Prober e do método manual. Esta diferença aparece em função do *overhead* gerado pela criação e término dos processos pelo sistema operacional, pois o Prober consegue medir toda a execução do processo desde a sua criação, enquanto a inclusão de comandos de medição de tempo no código, podem apenas medir o tempo transcorrido após a criação e antes do término do processo.

Através das etapas realizadas, o teste foi capaz de demonstrar que o Prober auxilia na configuração, execução e avaliação de programas paralelos. De acordo com a Fig.5, podemos observar que o Prober fez com que o usuário gastasse menos tempo em cada etapa do processo de teste de um programa paralelo. Utilizando o método manual, foram gastos 51 minutos, desde o login na primeira máquina até o último gráfico mostrado na tela, enquanto com o Prober, foram gastos 21 minutos para executar as mesmas tarefas.

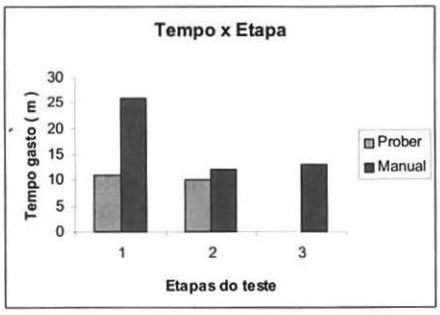

Fig.5 – Tempo gasto em cada etapa do teste

É de grande interesse, a comparação do Prober com algumas outras ferramentas de análise de desempenho e monitoramento de programas paralelos. Tanto que esta pesquisa já foi iniciada através de um questionário, que visa conhecer as ferramentas e ambientes mais utilizados hoje em dia no Brasil para o ensino de processamento paralelo, enviado a alguns professores das universidades brasileiras responsáveis por ministrar aulas relacionadas ao processamento paralelo, mas por enquanto recebemos apenas algumas respostas. Assim que coletarmos as informações necessárias, escolheremos as ferramentas que melhorar se adequarem ao nosso propósito.

Queremos mostrar com estes resultados que o uso do Prober para o desenvolvimento e teste de um programa paralelo foi mais eficaz que o uso do método manual. Em momento algum estivemos interessados em demonstrar que o desempenho de um programa foi melhor ou pior utilizando a ferramenta.

Além do experimento realizado entre a ferramenta e o método manual, foram realizados alguns outros apenas com o Prober no intuito de encontrar dados que nos indicasse em que momento o paralelismo é interessante. Estes experimentos são típicos de um primeiro contato com o paralelismo, onde os alunos se surpreendem com certos resultados. Eles farão parte de uma sessão de demonstrações de execuções de programas do Prober.

No experimento anterior descobrimos que ao utilizar-se intervalos menores que 1 milhão, mesmo que o programa fosse executado em 3 máquinas paralelamente, o tempo gasto ainda era maior que o tempo de executá-lo seqüencialmente. Então resolvemos testá-lo com intervalos de 10 e 100 milhões usando 1, 2 e 3 processos executando tanto concorrentemente quanto em paralelo.

De acordo com a Fig.6, utilizando-se os intervalos de 10 e 100 milhões, pudemos notar que o tempo gasto na execução em paralelo foi menor do que o tempo na execução seqüencial. Isso se deve a grande quantidade de intervalos que cada processo teve que processar, ou seja, quando utiliza-se um número pequeno de intervalos, cada processo faz pouco processamento não compensando o custo da criação do processo associado ao *overhead* na rede para a troca de mensagens entre o processo mestre e os escravos e a fração seqüencial, que fazem com que o desempenho do programa seja ruim. Quando se utiliza um grande número de intervalos (maiores que 10 milhões), cada processo faz muito processamento, tornando o tempo de criação e o *overhead* da rede muito pequenos perto do tempo útil de execução do programa.

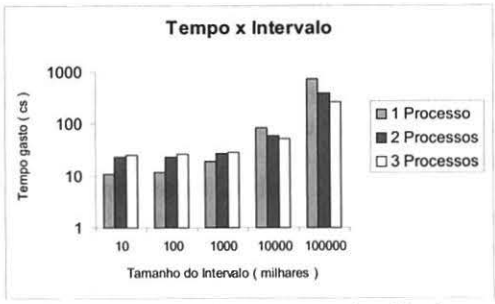

Fig.6 - Tempo de execução do cálculo do pi utilizando diversos intervalos

Estes resultados são de grande importância para os alunos que estão começando a aprender o processamento paralelo, pois ele mostra que o paralelismo pode não ser interessante em certas situações, enquanto que em outras ele pode aumentar em muito o desempenho do programa.

Uma forma muito interessante de identificar o ganho em tempo obtido na execução paralela é através do cálculo do *speedup,* que indica quantas vezes a execução paralela ou concorrente foi melhor ou pior que a seqüencial.

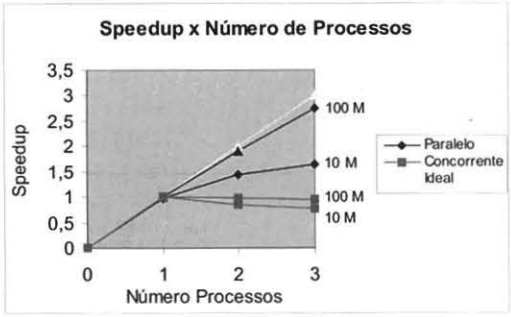

Fig.7 – *Speedup* das execuções do cálculo do pi utilizando 10 e 100 milhões de intervalos

Na Fig.7 mostramos um gráfico que indica o *speedup* executando-se o programa de cálculo do pi com número de intervalos iguais a 10 e 100 milhões de forma paralela e concorrente. O gráfico mostra que executando concorrentemente, com ambos os números de intervalos, o *speedup* nas execuções foi diminuindo a medida que se aumentava o número de processos. Essa diminuição ocorreu devido ao uso de apenas um processador, sendo assim, existiu pseudo-paralelismo na execução dos processos, ou seja, a cada instante apenas um processo esteve rodando no processador. Já na execução em paralelo, percebemos que a medida que se aumentava o número de processos, o *speedup* aumentava, sendo que para o intervalo de 100 milhões o *speedup* aproximou-se do ideal. Então constatamos na prática que o problema do cálculo do pi é altamente paralelizável.

O Prober foi usado para obter todos os tempos de execução necessários para o cálculo do *speedup*. Apesar de ainda não estar implementado nesta primeira versão, o cálculo do *speedup* também é muito importante para os alunos enxergarem o desempenho de um programa paralelo.

IV. Conclusão

O processamento paralelo é cada dia mais necessário para a resolução de problemas computacionais de grande demanda e complexidade e por isso deve ser ensinado em todas as universidades para que os alunos tenham contato com os problemas do paralelismo e as técnicas que possibilitam a solução destes. Para isso é interessante que sejam desenvolvidos programas paralelos e ferramentas que mostrem aos alunos que o processamento paralelo é realmente aplicável.

Neste trabalho constatamos que além dos problemas referentes ao paralelismo, existem uma série de procedimentos e problemas que fazem com que o aluno tenha que lidar com diversos softwares, perdendo mais tempo e consequentemente desanimando-se.

O método manual usado hoje em dia mostrou-se exaustivo e frustrante. Diante deste problema, criou-se o Prober. Neste trabalho comprovamos a utilidade desta nova ferramenta que analisa o funcionamento, o desempenho, e facilita as etapas de desenvolvimento e teste de um programa paralelo.

O Prober ainda está dando na sua fase inicial, mas os resultados obtidos já são de grande motivação para continuar o seu desenvolvimento.

Atualmente estamos implementando as funções da biblioteca, acrescentando novas funcionalidades e melhorando a interface.

Em trabalhos futuros, pretendemos implementar algumas novas opções na geração dos gráficos, armazenamento dos dados relativos a análise do desempenho através de recursos de banco de dados, criação de deamons que irão monitorar os processos que estiverem executando na máquina local a ele, e que transmitirão os dados coletados para um deamon central onde serão armazenados, uma interface gráfica para o controle do agendamento de tarefas e a disponibilização de programas de demonstração.

Como principais contribuições deste trabalho, podemos destacar a disponibilização de uma ferramenta para todo o meio acadêmico brasileiro e que auxiliará no ensino da disciplina de processamento paralelo da PUC-Minas. Ele também serviu como formação de recursos humanos e base para nos motivar a participar de mais um projeto aprovado pelo FIP/CNPQ que será iniciado no segundo semestre deste ano [FIP 01].

Agradecimentos

Gostaríamos de agradecer ao Instituto de Informática, à Pró-Reitoria de Pesquisa e Pós-Graduação (PROPPG), PROBIC, ao CNPq por nos conceder bolsas de pesquisa, aos amigos e companheiros de trabalho do LSDC e ao nosso orientador que sempre nos motivou.

Referências Bibliográficas

[ALV 95] Alves, A. ,Silva, L. , Carreira, J. , Silva., J. "WPVM: Parallel Computing for the People" , Proceedings of HPCN'95, High Performance Computing and Networking Conference, in Springer Verlag lecture Notes in Computer Science, pp 582-587, Milan, Italy 1995.

[AND 99] Andersen, P. "The Texas Tech Tornado Cluster: A Linux/MPI Cluster For Parallel Programming Education And Research", ACM CrossRoads Eletronic Magazine 1999.

[BEN 01] Cluster Benchmarks Web Page (http://www.di.unito.it/~mino/cluster/benchmarks/)

[BUY 00] Buyya, R., Apon, A., Jin, H., Mache, J. "Cluster Computing in the Classroom: Topics, Guidelines and Experiences", 2000.

[BUY 97] Buyya, R. "Single System Image: Need, Approaches and Supporting HPC Systems" , International Conference on Parallel and Distributed Processing Techniques nad Applications, USA, 1997.

[DON 97] Dongarra, J. , Browne, S. , London, K. "Review of Performance Analysis Tools for MPI Parallel Programs" , 1997.
(http://www.cs.utk.edu/~browne/perftools-review/)

[FIP 00] Martins, C. "Proposta e Implementação de um Ambiente de Desenvolvimento e Prototipação Rápida de Sistemas Computacionais: Paralelos, Dedicados e Reconfiguráveis" Projeto aprovado pelo FIP/CNPQ, 2000.

[FIP 01] Martins, C. , Góes, L., Ramos, L. "Proposta e Implementação de um Aglomerado de Computadores usando Redes de Comunicação de Dados de Alto Desempenho e Disponibilizando uma Imagem de Sistema Único" Projeto aprovado pelo FIP/CNPQ, 2001.

[FOS 95] Foster. I "Designing and Building Parallel Programs", On-line book 1995. (http://www-unix.mcs.anl.gov/dbpp/text/book.html)

[GEI 00] Geist, A. "PVM and MPI: What Else Is Needed for Cluster Computing?", 7th European PVM/MPI Users' Group Meeting , 2000.

[HWA 98] Hwang, K.; Xu, Z. "Scalable Parallel Computing:Technology, Architecture, Programming", Mcgraw-Hill, 1998.

[MPI 01] MPI – Message Passing Interface (http://www-unix.mcs.anl.gov/mpi/)

[PTL 01] PTLib – Parallel Tools Library (http://www.nhse.org/ptlib/)

[PVM 01] PVM – Parallel Virtual Machine (http://www.epm.ornl.gov/pvm/pvm_home.html)

[SOL 00] Solsona, F. , Giné, F. , Lérida, J. , Hernández, P. , Luque E. "Monito: A Communication Monitoring Tool for a PVM-Linux Environment", 7th European PVM/MPI Users' Group Meeting , 2000.

[XAV 00] Xavier, E. , Travieso, G. "Interface para Gerenciamento e Uso de Clusters para Processamento Paralelo, 1° Workshop em Sistemas Computacionais de Alto Desempenho, São Paulo 2000.

[XPV 01] XPVM: A Graphical Console and Monitor for PVM - (http://www.netlib.org/utk/icl/xpvm/xpvm.html)