X Simpósio Brasileiro de Arquitetura de Computadores

Balanceamento de Carga em um Ambiente MPI Distribuído

Mario A.R. Dantas
Departamento da Ciência da Computação
UnB - Universidade de Brasília
E-mail: mario@cic.unb.br

Abstract

The MPI (Message-Passing Interface) is an effort from many organizations (universities, laboratories and industry) to define standard message-passing environment. As a standard, the MPI specification was not designed as a comprehensive parallel programming environment. Therefore, no efficient mechanism exists to provide an enhanced parallel tasks allocation. In this paper, we present an approach that helps the balance of parallel tasks executing more efficient with a better scheduling on distributed systems.

O ambiente MPI (Message-Passing Interface) é um padrão de troca de mensagens, o qual foi definido de comum acordo por muitas organizações na área acadêmica (universidades e laboratórios de pesquisas) e indústria (empresas fabricantes de computadores). Uma vez que o padrão foi definido a nível sintático e semântico de biblioteca (e não como um amplo ambiente paralelo) a especificação não contém nenhum mecanismo elaborado para a alocação eficiente de tarefas paralelas num cenário distribuído de *workstations*. Neste trabalho, apresentamos uma abordagem que auxilia de uma forma mais eficaz no escalonamento de tarefas paralelas, através do balanceamento de carga em ambiente distribuído.

Keywords

Sistemas Paralelos, Redes de *Workstations*, MPI, Sistemas Distribuídos.

I. INTRODUÇÃO

Avanços na tecnologia de microprocessadores em conjunto com redes de alta velocidade (exemplos são ATM e Myrinet), têm criado a possibilidade de utilização dos ambientes distribuídos, tais como rede de *workstations*, como configurações paralelas. Resultados empíricos de sucesso destas configurações foram demonstradas no prêmio Gordon Bell [1], [2], onde a categoria de preço-desempenho foi vencida por ambientes de *workstation clusters*. Em outras palavras, os trabalhos vencedores provaram que a existência de um grande número de *workstations* conectados por uma LAN pode representar um ambiente atrativo para a execução de complexas aplicações numericas.

Aplicações científicas e de engenharia são exemplos típicos de programas que consomem grandes recursos computacionais e memória, porque estas duas áreas endereçam a solução de problemas frequentemente utilizando-se de um conjunto complexo de cálculos matemáticos. Desta forma, neste trabalho consideramos um algoritmo paralelo de eliminação por Gauss e outro paralelo de multiplicação de matrizes como aplicações candidatas ao uso de um sistema paralelo distribuído. O objetivo principal é a melhoria de desempenho das aplicações, representada pela diminuição no tempo de execução dos algoritmos.

2

Na ausência de um padrão de biblioteca para o paradigma de troca de mensagem, várias universidades e instituições diversas desenvolveram seus próprios pacotes (exemplos são o PVM [3], IBM-EUI [4] e Zipcode [5]). Um padrão conhecido como **MPI (Message-Passing Interface)**, foi apresentado em 1994 [6] por um grupo de pesquisadores (MPI-Forum) representando muitas das organizações que já haviam desenvolvido algum pacote de troca de mensagem. Desta maneira, o MPI-Forum apresentou um padrão *de facto* para a abordagem de sistema de troca de mensagem, para a elaboração de bibliotecas, e aplicações, para ambientes de memória distribuída. A vantagem mais evidente do MPI é um conjunto definido de sintaxe e semântica que todos possam implementar e assim ganhar a portabilidade necessária para a execução em ambientes heterogêneos.

Computação em configurações de redes de *workstations* usando MPI como ambiente de programação paralelo, pode resultar em baixo desempenho das aplicações paralelas uma vez que nestas configurações não existêm originalmente nenhum mecanismo de *scheduler* (escalonamento de tarefas). Consequentemente, embora uma configuração disponha, por exemplo, de recursos disponíveis para a execução de um vasto número de aplicações complexas, estas podem apresentar um mau desempenho devido a falta de algumas considerações de balanceamento de carga no ambiente.

Uma abordagem alternativa é necessária para prover a confiabilidade requerida pelos programadores de aplicações para a execução de seus programas paralelos MPI numa configuração de rede de *workstations*. Como resultado, neste trabalho apresentamos uma solução de alto nível, ou seja a nível de usuário, que objetiva a melhoria na execução de programas MPI paralelos em *clusters de workstations* (redes de *workstations*). Nossa abordagem é denominada de *Selective MPI (S-MPI)*. O S-MPI implementa um conjunto de funções de *scheduler* monitorando a carga de *workstations*, compilando (ou recompilando) códigos paralelos, procurando casar a necessidade de processos com os recursos existentes e dinamicamente selecionado a melhor configuração paralela para a execução de uma aplicação MPI.

O presente trabalho é organizado da seguinte maneira. Na seção II, apresentamos o ambiente paralelo distribuído e na seção III abordamos os algoritmos MPI paralelos. Conceitos importantes da arquitetura S-MPI são apresentados na seção IV em conjunto com resultados experimentais do ambiente. Finalmente, na seção V discutimos algumas conclusões sobre o trabalho.

II. O AMBIENTE PARALELO DISTRIBUÍDO

Configurações que consideramos neste trabalho, são formadas por *workstations* heterogêneas SUN e Silicion Graphics. Isto significa dizer que, as *workstations* consideradas possuem diferentes processadores, quantidade de memória e sistema operacional. Usamos o pacote de software *mpich* [7] como implementação do padrão MPI. Por esta razão, a partir deste ponto faremos referência ao *mpich* como MPI. Importante mencionar que as *workstations* consideradas são de uso público e que possuem ciclos livres e memória disponíveis para a execução de programas paralelos. Os sistemas operacionais e as características de hardware das *workstations* são apresentados pelas tabelas I e II.

Dezenove *workstations* formam o cluster SGI (tabela I). Três destas *workstations* tinham processador IP20 com clock de 100MHz. As outras dezesseis tinham um processador de 33MHz e 32MBytes de memória, mas mesmo com esta configuração a quantidade de *workstations*

SGI CLUSTER

Machine Architecture	**IP12 - IP20**
Clock(MHz)	**33 - 100**
Data/Instruction Cache (KB)	**32 - 1024**
Memory(MB)	**32 - 48**
Operating System	**IRIX 5.3**
MPI version	***mpich* 1.0.12**

TABLE I
CARATERÍSTICAS DO CLUSTER SGI.

podem ser consideradas para a execução de inúmeras programas paralelos.

SUN CLUSTER

Machine Architecture	**SUN4m**
Clock(MHz)	**70 - 50**
Data/Instruction Cache (KB)	**64**
Memory(MB)	**48 - 16**
Operating System	**SOLARIS 5.3**
MPI version	***mpich* 1.0.12**

TABLE II
CARATERÍSTICAS DO CLUSTER SUN.

O cluster de SUN (tabela II), a qual foi formada com sete *workstations*, possui duas *workstations* com processadores de 50MHz e apenas 16MBytes de memória. Como notamos durante nossa medidas, a inadequada quantidade de memória destas duas *workstations* causaram constante uso da área de swap, o que ocasiona na degradação de desempenho das aplicações. Consequentemente, cuidados especiais devem ser tomados quando da consideração de *workstations* como estas para a formação do cluster paralelo.

Redes de interconexão são conexões de alta velocidade que ligam processadores e unidades de memória em computadores com memória compartilhada. Considerando um cluster de *workstations* como ambiente paralelo, a tecnologia da LAN usada representa a função equivalente das redes de interconexão. Desta forma, é desejável o uso de LANs de alta velocidade para a obtenção do mesmo nível de desempenho na comunicação das aplicações. Todavia, devido a falta de acesso a um LAN de alta velocidade, resolvemos considerar a convencional Ethernet como rede de interconexão do ambiente. Embora compreendamos que o elevado nível de colisões na LAN Ethernet (como por exemplo do tráfego do NFS) pode tornar inadequada a configuração para programação de programas MPI paralelos.

III. ALGORITMOS MPI PARALELOS

Nesta seção descrevemos o projeto, implementação e teste das duas aplicações com diferentes percentuais entre computação e comunicação, que desenvolvemos usando MPI.

Nosso principal objetivo é criar um quadro para uma comparação efetiva do mecanismo proposto neste trabalho para melhorar o desempenho de aplicações MPI paralelas em redes de *workstations*. Outro ponto importante desta seção, é apresentar aspectos tais como a metodologia usada para paralelizar um aplicação, o desempenho de códigos MPI quanto ao custo-efetividade e parâmetros de portabilidade.

Sistemas de equações aparecem num grande número de aplicações científicas e de engenharia. Frequentemente esta parte do código é responsável por requerer a maior quantidade de ciclos de CPU. Como notamos anteriormente [8], ferramentas de analise de software, tais como Forge90 [9] e ida [10] são importantes para auxílio na paralelização de aplicações.

As versões paralelas de nossos códigos foram implementadas usando a abordagem conhecida como *single program multiple data (SPMD)* e minimizando tanto quanto foi possível a comunicação entre *workstations* (devido ao elevado custo de comunicação na LAN Ethernet). Na abordagem SPMD, idênticas cópias do código executam independentemente em diferente *workstations*, trabalhando em diferentes conjuntos de dados. Todavia, quando necessário processos se comunicam usando primitivas do MPI.

```
1. Begin Parallel GE
2. IF (my_id = master)
3.    scatter matrix rows                          (* Scatter *)
4.    for i <--1 to (n-1) do
5.       find the pivot element                    (* Fpivot *)
6.       the owner of the pivot row broadcasts it  (* Bpivot *)
7.       perform elimination using the pivot       (* Elim *)
8.    endfor
9.    gather matrix                                (* Gather *)
10. ELSE
11.   receive rows from master                     (* Recv  *)
12.   for i <--1 to (n-1) do
13.      find the pivot element                    (* Fpivot *)
14.      the owner of the pivot row broadcasts it  (* Bpivot *)
15.      perform elimination using the pivot       (* Elim *)
16.   endfor
17.   send processed rows to master                (* Send *)
18. End
```

Fig. 1. A abordagem Single Program Multiple Data.

O pseudocódigo paralelo para solução do sistema de equações, usando a técnica de redução do elemento pivot por linha, é representada na figura 1. A implementação paralela foi projetada considerando-se cinco mensagens. Na primeira, a *workstation* mestre envia `m` linhas para `n` *workstations* escravas que foram configuradas como parte do ambiente paralelo. Após cada *workstation* escrava ter recebido suas linhas, elas processam suas linhas e calculam o elemento pivot para a primeira iteração.

A segunda mensagem é a transmissão do elemento pivot para a *workstation* mestre. O nó mestre recebe todos os elementos pivot e obtém o maior (após comparar cada elemento pivot recebido). Após ter obtido o maior elemento pivot, a *workstation* mestre faz uma operação de multicast para as *workstations* escravas com a identificação da *workstation* que possui o maior elemento pivot. A *workstation* vencedora, ou seja aquela que possuí o elemento pivot, executa uma primitiva MPI de multicast com todos os elementos da linha para que esta seja processada por toda a máquina paralela distribuída. Esta operação é repetida desde o segundo passo até que o número do sistema de equações seja igual a um. A comunicação final é efetuada por todas as *workstations* escravas enviando suas linhas processadas. A *workstation* mestre recolhe todas as linhas processadas pelas *workstations* escravas e monta a matrix que será utilizada

para fazer a substituição final (*backward substitution*).

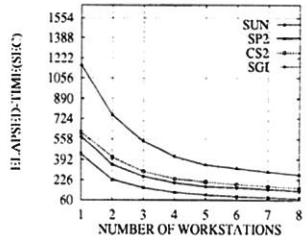

Fig. 2. Elapsed-time para resolver 1000 equações.

A figura 2 ilustra o significante decréscimo no *elapsed-time* em dois clusters, do código sequencial (um processador) em comparação com a versão paralela (de dois até oito *workstations*) para resolver um sistema de 1000 equações. Em outras palavras, esta figura mostra que existe um ganho real da versão paralela quando comparada com a sequencial. Executamos também nosso código no computadores paralelos IBM-SP2 e Meiko-CS2. É importante observar que o mesmo programa foi executado nestas configurações heterogêneas sem a necessidade de troca de uma simples linha. Este fato demonstra a excelente portabilidade do código MPI. Outro aspecto demonstrado nesta figura é o parametro custo-eficiência do ambiente de rede de *workstations* como configuração paralela. O desempenho do cluster SUN com oito *workstations* (que já existiam em nossa instalação sem nenhum custo adicional) é quase equivalente ao desempenho de quatro nós do IBM-SP2.O cluster SGI obtém um desempenho de quatro nós do computador paralelo Meiko-CS2.

```
Matrix initialisation
IF (my_id = master)
  process elements      (* Compute *)
  receive processed -
  elements from slave   (*Master_Gather *)
  save output           (* I/O *)
ELSE
  process elements      (* Compute *)
  send processed -
  elements to master    (* Slave_Gather *)
ENDIF
```

Fig. 3. Pseudocódigo da multiplicação de matrizes.

A multiplicação de matrizes é um componente essencial em inúmeras aplicações. Vários algoritmos com diferentes graus de complexidade estão disponíveis na literatura(exemplos [11], [12]). Estes algoritmos variam desde algoritmos convencionais até aqueles projetados para arquite-

turas específicas. Decidimos implementar nosso algoritmo de multiplicação de matrizes de uma forma SPMD como mostrado na figura 3. Esta abordagem é atrativa para a redução dos altos custos de comunicação num ambiente onde uma rede de interconexão não dedicada (em nosso caso a Ethernet) é empregada. Em adição, apresentamos nossa abordagem de multiplicação de matrizes para demostrar como operam algumas primitivas do MPI e que são apresentadas na seção de resultados experimentais. É essencial mencionar que as operações *Master_Gather* and *Slave_Gather* são as mesmas primitivas MPI (MPI_Gather). Representamos estas separadamente somente para melhor visualização da comunicação entre mestre-escravo nas figuras.

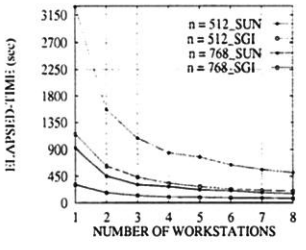

Fig. 4. Elapsed-time do algoritmo de multiplicação de matrizes nos clusters SUN e SGI.

A figura 4 demonstra a queda acentuada no tempo de execução do algoritmo paralelo de multiplexação de matrizes(dois a oito *workstations*) comparado com a versão sequencial (uma *workstation*) executando problemas de diferentes tamanhos. Desta forma, esta figura ilustra o sucesso da implementação MPI paralela em comparação com a versão sequencial.

O decréscimo no tempo de execução das duas aplicações paralelas no ambiente de *workstations* comparado com a implementação sequencial confirmam a eficiência de desempenho da abordagem MPI. A vantagem do ambiente de *workstation* como um ambiente paralelo, em relação a dois computadores paralelos convencionais (SP-2 e Meiko-CS2), também foram mostrados nesta seção.

IV. A ARQUITETURA S-MPI

A especificação MPI deliberadamente não contém nenhuma informação explícita a cerca de como um processo deve ser criado e gerenciado. O MPI-Forum decidiu ignorar estes aspectos, porque estes são aplicados de maneiras distintas em muitos sistemas paralelos. No modelo computacional representado pela abordagem MPI, o ambiente de execução de um processo é representado pelas funções de *escalonamento do job*, gerência do processo e a biblioteca de *troca de mensagens*. Uma solução alternativa é necessária para extender o MPI para configurações de *cluster de workstations*, uma vez que este ambiente não dispõe de função nativa de gerência de execução de processos.

Esta seção apresenta uma pesquisa em como extender o MPI com uma melhor função de *escalonamento de jobs* para executar aplicações paralelas com mais eficiência em redes de *workstations*. Ambientes heterogêneos, onde um grande número de *workstations* com diferentes processadores e quantidade de memória estão disponíveis para serem agrupadas como uma

configuração paralela, provêem um bom exemplo da importância do mecanismo proposto.

Um ambiente o qual denominamos de **Selective-MPI (S-MPI)** foi projetado para endereçar o problema de escalonamento. Em contraste com alguns pacotes de gerenciamento distribuído, os quais proclamam suporte para o MPI e requerem uma complexa operação de configuração usando privilégios de *super-user*, esta proposta é um modelo dito *lightweight* o qual co-existe de maneira uniforme com os conceitos existentes do MPI. Nós acreditamos que um novo ambiente deve possuir a mesma interface amigável do padrão. É necessário prover ao usuário MPI todas as facilidades atrativas do ambiente existente e melhorar o uso de recursos disponíveis sem adicionar níveis extras de complexidades. Decisões administrativas a cerca de qual a ferramenta de gerenciamento de recursos é mais apropriada para a organização e o mandatório envolvimento de adminstradores do sistema operacional Unix, são dois exemplos de complexidades indesejáveis.

A figura 5, ilustra a arquitetura do ambiente S-MPI comparada com a especificação padrão MPI. A interface existente entre os programadores de aplicação e o MPI em uma rede de *workstations* é representada pela interação de comandos de um *Shell* do Unix. O usuário precisa somente editar manualmente os arquivos *machines.xxxx* ou *procgroup* (estes permitem o programador especificar computadores, número de processos e nome da aplicação em ambientes homogêneos e heterogêneos respectivamente) e então submeter a aplicação.

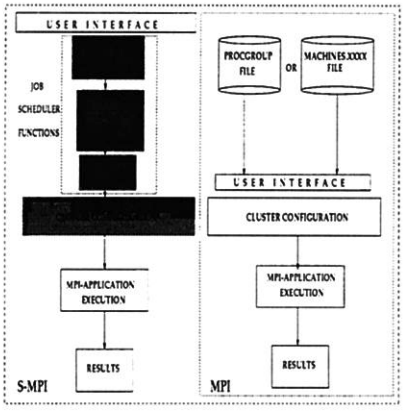

Fig. 5. Diferenças nas arquiteturas do S-MPI e MPI.

A abordagem S-MPI compreende funções de um escalonador de tarefas em alto nível, através da monitoração da carga das *workstations*, compilando automaticamente o código paralelo, checando os requisitos dos processos com os recursos disponíveis e finalmente selecionando dinamicamente um configuração apropriada para a excecução de aplicações paralelas. Desenvolvemos os seguintes módulos utilizando ferramentas existentes no próprio Unix para implementar o *escalonador de jobs*:

- Monitores de Load-CPU - estes monitores são ferramentas auxiliares necessárias para prover

o ambiente S-MPI com checagens regulares da carga de trabalho das *workstations*, aglutinando informação sobre a utilização de carga e CPU das máquinas. Descrevemos esta função dos monitores como *balanceamento seletivo de carga*. Esta abordagem cria no ambiente S-MPI um hibrido dinâmico/estático mecanismo, porque (1) em estilo semelhante ao balanceamento dinâmico, este considera a carga das *workstations* antes da submissão de processos: (2) por outro lado a consideração das *workstations* é efetuada antes da aplicação começar a executar.

- TRANSCEND - aplicações paralelas que executam em arquitetura de *workstations* diferentes têm a necessidade de compilação (e recompilação) em todos os potenciais ambientes. Este procedimento causa um trabalho excessivo por causa da constante necessidade de conexão a estes ambientes (ou *workstations*) requeridos. Este fato pode levar o programador da aplicação paralela a erros, pois o mesmo pode se esquecer de executar a compilação numa determinada configuração. Por este motivo, implementamos um módulo chamado de **TRANSCEND** (**TRANS**parent **C**ode g**EN**eration **D**ispatch), o qual automaticamente compila aplicações paralelas nas diferentes variações de Unix e ainda fornece uma idéia aproximada da quantidade de memória requerida por cada arquitetura.
- Monitor de Recursos - este reporta a quantidade de memória disponível e compõe com a memória requerida pelo código do executável criado pelo módulo **TRANSCEND**. Em adição, este monitor contribui para o S-MPI mecanismo considerar somente *workstations* com suficiente recursos disponíveis para executar uma aplicação.
- Configuração Automática do Cluster (Automatic cluster configuration) - O procedimento adotado pelo MPI para configurar clusters homogêneos de *workstations* é o uso do arquivo *machines.xxxx*. Este arquivo contém o nome (ou endereço IP) da *workstation*, um nome por registro. Não existe consideração sobre a carga das máquinas ou disponibilidade no momento da execução do código paralelo. Configurações heterogêneas são formadas usando o arquivo *procgroup*. Este arquivo é manualmente editado com o nome da *workstation*, área onde se encontra o executável e identificação remota do usuário. Adotamos o mecanismo de *procgroup* para o ambiente S-MPI, pois este permite a configuração de *clusters* homogêneos e heterogêneos. Depois de criar o arquivo *procgroup* dinamicamente, executamos a função *mpirun* com o apropriado número de *workstations* e nome da aplicação paralela.

A. RESULTADOS EXPERIMENTAIS

Nesta seção, apresentamos resultados dos nossos algoritmos paralelos implementados usando o sistema S-MPI. Utilizamos o pacote MPE (MultiProgramming Environment) [13] junto com a ferramenta gráfica Upshot [14] para visualização de alguns detalhes de execução (Upshot é um software de visualização X que pode ser usado para a análise *post-mortem* dos arquivos logfiles os quais foram criados pela biblioteca MPE). Nossas medidas foram tomadas durante períodos convencionais de trabalho, usando os dois clusters, e eles representam a execução mais rápida de vinte execuções.

A primeira configuração considerada para a execução dos aplicativos paralelos foi formada com uma *workstation* mestre de menor poder computacional comparada com as outras quinze *workstations* escravas. Nesta configuração temos a mesma *workstation* mestre para os ambientes MPI e S-MPI. Esta configuração foi idealizada, pois é esperado que o programador de uma aplicação execute tarefas paralelas de suas menos potentes *workstations*. A figura 6 mostra o

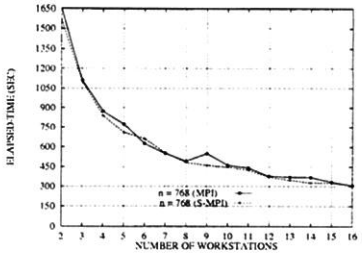

Fig. 6. Multiplicação de matrizes com uma *workstation* mestre com menor poder computacional.

elapsed-time do algoritmo de multiplicação de matrizes nos ambientes MPI and S-MPI para resolver um exemplo de 768 x 768. Esta figura indica que aparentemente os dois ambientes tiveram um desempenho semelhante. Então, para melhor entender o procedimento destes ambientes nós utilizamos a ferramenta Upshot para examinar em detalhes alguns aspectos deste gráfico.

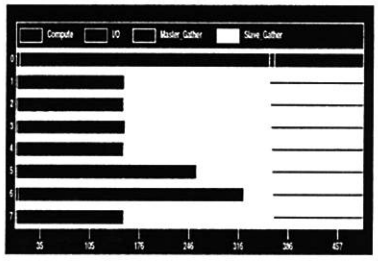

Fig. 7. Detalhe do algoritmo de multiplicação de matrizes no ambiente MPI.

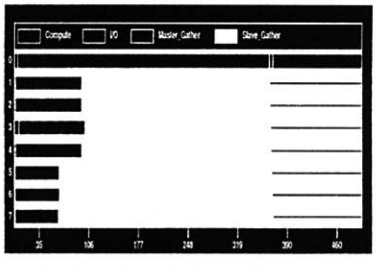

Fig. 8. Detalhe do algoritmo de multiplicação de matrizes no ambiente S-MPI

As figuras 7 e 8 ilustram uma saída Upshot para o exemplo considerando oito *workstations*. A figura 7 apresenta os resultados do MPI, onde o nó mestre (processo 0) gasta significantemente mais tempo comparado com os nós escravos (1 to 7). A figura 8 mostra que no ambiente S-MPI, melhores *workstations* escravas foram selecionadas minimizando o elapsed-time, onde a *workstation* mestre gasta aproximadamente a mesma quantidade de tempo da configuração MPI. Este fato, nos leva a concluir que nenhuma tentativa de melhoria da aplicação nesta configuração será bem-sucessida por causa da limitação da *workstation* mestre, a qual representa um gargalo para a configuração.

Nossa próxima configuração foi caracterizada pelo uso de diferentes *workstations* mestres e escravas para os ambientes MPI e S-MPI. Para o ambiente MPI, uma *workstation* SGI de 33MHz foi escolhida como nó mestre, por causa de medições anteriores que demonstraram (após alguns meses de monitoração) ser esta *workstation* interessante para executar aplicação devido a baixa carga desta *workstation*. Em contraste, a *workstation* mestre para o ambiente S-MPI foi selecionada analisando as cargas das máquinas antes de configurar o ambiente para executar uma aplicação paralela.

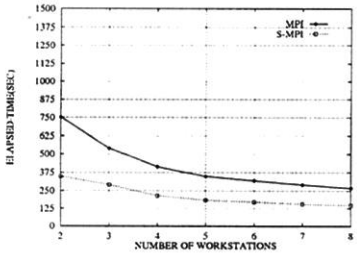

Fig. 9. O algoritmo de Gauss para resolver 1000 equações nos ambientes MPI e S-MPI.

A figura 9 mostra a melhora de desempenho da abordagem S-MPI comparada com o MPI para resolver um sistema de 1000 equações (nós usamos oito *workstations* por causa da grande quantidade de memória necessária e as subsequentes reclamações dos usuários das *workstations* quando executamos a aplicação para resolver problemas deste tamanho). O algoritmo de Gauss demonstrou uma redução acentuada no elapsed-time de cerca de quarenta e cinco porcento na configuração S-MPI quando comparada ao MPI. O uso de mestres distintos foi o fator responsável por esta diferença. este resultado era esperado, como notamos no primeiro experimento, a *workstation* mestre usualmente é uma peça fundamental no desempenho da configuração.

A tendência de melhoria de desempenho do ambiente S-MPI executando o algoritmo de multiplicação de matrizes é apresentada na figura 10. Esta figura demonstra que o S-MPI obteve um melhor desempenho do que o ambiente MPI. É ainda importante notar que a diferença entre os dois ambientes é de trinta e oito porcento até *workstations*. Após este número de máquinas a diferença de desempenho é menor devido a falta de *workstations* mais *potentes* na configuração.

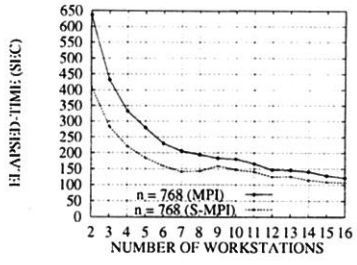

Fig. 10. Elapsed-time do algoritmo de multiplicação de matrizes nos ambientes MPI e S-MPI.

B. FACILIDADES ADICIONAIS

O desempenho da abordagem S-MPI minimizando o elapsed-time das aplicações confirmaram a efetividade do sistema de escalonamento MPI paralelo para configurações de *workstation clusters*. Em adição a este aspecto, alguns conceitos identificados durante a fase de projeto mostraram uma especial significância durante os experimentos. Estes aspectos do S-MPI incluem:

- *Menos interferência*: contrastando com o MPI, onde nenhuma consideração de carga existe (levando algumas vezes a interferência das aplicações paralelas com o usuário da *workstation*), a abordagem S-MPI provou ser menos intrusiva nas *workstations* públicas. Este fato é importante quando executamos aplicações paralelas em ambientes de *workstations* não dedicados. Recursos computacionais podem ser eficientemente agrupados sem uma interferência significante junto ao usuário de uma dada *workstation* (ou pelo menos com um nível menor de interferência). Este estágio foi alcançado selecionando máquinas moderamente carregadas para a configuração paralela.
- *Previsão de execução* : é importante notar que uma certa aplicação rodando no ambiente MPI pode apresentar resultados de tempo de execução variando numa larga faixa de tempo. Em contraste, as aplicações rodando sob o ambiente S-MPI apresentaram resultados bastante similares quando considerando o mesmo tamanho de problema para o mesmo número de *workstations*.
- *Configuração dinâmica*: quando durante a execução de uma aplicação paralela numa configuração MPI uma (ou mais) das *workstations* mudar substancialmente sua carga, basta que o usuário resubmeta sua aplicação que uma nova configuração dinamicamente será escolhida. Por outro lado, um ambiente MPI convencional obriga que o usuário descubra dentre as várias máquinas do arquivo *machines.xxxx* aquela que mudou sua carga. Esta operação consome um certo tempo e não existe garantia que ao final teremos a solução para o problema. Outra *workstation* pode mudar sua carga durante o período de pesquisa no arquivo *machines.xxxx*.
- *Tolerância a falha*: o mecanismo projetado para o S-MPI para compilar automaticamente as aplicações (TRANSCEND), provou-se ser uma ferramenta útil para a geração de um código paralelo mais confiável. Durante os experimentos não ocorreu nenhum erro relacionado a mistura de diferentes versões de código. Estas mensagens são recebidas com frequêcia

quando diversos sistemas são envolvidos e o programador da aplicação deve se recordar da compilação da implementação paralela em cada ambiente.

V. CONCLUSÕES

Neste trabalho apresentamos conceitos e experiências da abordagem S-MPI. O sistema endereça uma extensão da função de escalonamento do padrão de troca de mensagens MPI em configurações de rede de *workstations*, executando mais eficientemente aplicações paralelas. O sistema adapta-se de maneira transparente com os conceitos existentes do MPI e preserva as características originais do ambiente de software. Em adição, o S-MPI prove os programadores da aplicação com uma configuração mais transparente e eficiente para a submissão de códigos paralelas sem níveis adicionais de complexidade ou treinamento, quando comparamos com alguns pacotes de gerenciamento de sistemas distribuídos.

Avaliamos as novas facilidades do novo ambiente, através da execução de duas aplicações paralelas que foram comparadas com a execução numa configuração MPI convencional. A redução significante no elpased-time das aplicações indica que o S-MPI prove um melhor ambiente de escalonamento mais eficiente para a execução de programas paralelos em rede de *workstations*. Outros aspectos tais como menos interferência junto aos usuários das *workstations*, previsão de tempo de execução e configuração dinâmica do *cluster* se mostraram facilidades importantes durante a fase de experimentos.

REFERENCES

[1] Michael Heath Alan H. Karp and Al Geist, "Judges' Summary 1994 Gordon Bell Prize Winners", *IEEE Computer - Special Report*, vol. 28, no. 1, pp. 68–74, 1995.

[2] Michael Heath Alan H. Karp and Al Geist, "Judges' Summary 1995 Gordon Bell Prize Winners", *IEEE Computer - Special Report*, vol. 29, no. 1, pp. 79–85, 1996.

[3] Adam Beguelin Al Geist, Jack Dongarra and Vaidy Sunderam, *PVM : User's Guide and Reference Manual (Version 3.3)*, Oak Ridge National Laboratory Technical Report ORNL/TM-12187, Oak Ridge, USA, May 1994.

[4] D. Frye et al, "An external user interface for scalable parallel systems", *IBM - Technical Report*, May 1992.

[5] A. Skjellum and A. Leung, "Zipcode : a portable multicomputer communication library atop the reactive kernel", *Proceedings of the Fifth Distributed Memory Concurrent Computing Conference*, pp. 767–776, 1990.

[6] MPI-Forum, "MPI: A Message-Passing Interface Standard", *International Journal of Supercomputer Application*, vol. 8, no. 3-4, 1994.

[7] William Gropp Patrick Bridges, Nathan Doss, "Users' Guide to mpich, a Portable Implementation of MPI", *Argonne National Laboratory-http://www.mcs.anl.gov*, 1994.

[8] M.A.R. Dantas and E.J. Zaluska, "Improving the Performance of a Petroleum Reservoir Model on Workstation Clusters using MPI", *Proceedings of the High Performance Computing 1996, Grand Challenges in Computer Simulation, New Orleans, USA*, pp. 72–77, April 1996.

[9] Richard Friedman, *Forge90 Baseline System - User's Guide*, Applied Parallel Research,Inc, Placerville , CA, 1992.

[10] John Merlin, "ida User's Guide", *Electronics and Computer Science Internal Report, University of Southampton*, 1993.

[11] Ellis Horowitz and Sartaj Sahni, *Fundamentals of Computer Algorithms*, Pitman Publishing Limited, London, 1978.

[12] Michael J Quinn, *Parallel Computing - Theory and Practice*, McGraw-Hill, Singapore, 1994.

[13] Ewing Lusk William Gropp and Anthony Skjellem, *Using MPI - Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, Massachusetts, 1994.

[14] Virginia Herrarte and Ewing Lusk, "Studying parallel program behavior with *upshot*", *Argonne National Laboratory - Technical Report ANL-91/15*, 1991.