# PARALELIZAÇÃO EM AMBIENTES PVM E MPI DE UMA APLICAÇÃO DO MÉTODO DOS ELEMENTOS FINITOS

**Flávia Romano Villa Verde**
Departamento de Engenharia Mecânica, Universidade de Brasília, flaviarvv@terra.com.br

**Gerson Henrique Pfitscher**
Departamento de Ciência da Computação, Universidade de Brasília, gerson@unb.br

**Dianne Magalhães Viana**
Departamento de Engenharia Mecânica, Universidade de Brasília, diannemv@unb.br

***Resumo***. *Muitas vezes o desenvolvimento de aplicações baseadas no método dos elementos finitos empregando-se a programação paralela pode ser uma forma econômica de se resolver um problema. Neste sentido, com uma rede de computadores disponível, aliada ao uso de softwares livres ou de baixo custo e, ainda, um sistema operacional Linux, o uso da computação paralela em um cluster de microcomputadores passa a ser uma alternativa interessante e possível de ser implementada. Este trabalho tem por objetivo comparar a execução paralela em ambiente MPI e PVM, em um cluster de computadores, de uma aplicação baseada no método dos elementos finitos. Foi desenvolvido um código para solução de problemas estruturais, escrito em linguagem C, que utiliza o método dos gradientes conjugados para solução do sistema de equações. O pré-processamento é feito utilizando-se o programa GID® e para a partição da malha é empregado o METIS®. O sistema de equações é montado conforme a estrutura "block arrowhead". O código é validado e avaliações quanto ao desempenho do mesmo são efetuadas mostrando-se que ganhos significativos em termos de tempo de execução são obtidos.*

***Palavras-chave***: *Método dos Elementos Finitos, MPI, PVM*

## 1. INTRODUÇÃO

O uso do método dos elementos finitos em análise estrutural muitas vezes apresenta um custo computacional elevado em termos de tempo de execução dado o volume de informações a serem processadas durante a solução do sistema de equações. Neste sentido, a programação paralela em *cluster* de microcomputadores passa a ser uma opção viável por possibilitar a redução do tempo computacional a baixos custos, uma vez que utiliza tecnologia já depurada de alta disponibilidade e confiabilidade, arquitetura aberta e softwares de domínio público.

As bibliotecas padrões mais utilizadas na paralelização são o MPI (*Message-Passing Interface*) e o PVM (*Parallel Virtual Machine*). O PVM é mais antigo que o MPI, tendo surgido em 1989 nos laboratórios da *Emory University* e *Oak Ridge National Laboratory*, com o objetivo de criar e executar aplicações paralelas em um hardware já existente. O ambiente PVM tornou-se um padrão devido a sua flexibilidade, pois habilita um conjunto de computadores heterogêneos a comportarem-se como um único computador paralelo virtual.

O MPI teve sua primeira versão publicada em 1994 e atualizada em junho de 1995. Incorporou os modelos de comunicação por troca de mensagens habitualmente descritos na literatura, prevendo tanto a comunicação síncrona quanto a assíncrona, permitindo, em quaisquer dos casos, o uso excelente suporte à comunicação coletiva, que facilita o desenvolvimento de aplicações que necessitam de efetuar freqüentes operações sobre matrizes.

Duas etapas importantes no desenvolvimento de aplicações paralelizadas baseadas no método dos elementos finitos são a decomposição do domínio, que depende da arquitetura da máquina paralela e a solução do sistema de equações, que demanda um maior custo computacional. Atenção deve ser direcionada também aos custos adicionais em comunicação em face do processamento multitarefa. Em um código de elementos finitos paralelizado, as comunicações entre os

processadores podem ser necessárias na distribuição dos dados de entrada entre os processadores, na divisão das informações referentes aos nós compartilhados pelos elementos, na distribuição dos resultados dos produtos de vetores, etc.

Este trabalho apresenta uma implementação de um código paralelizado para uma aplicação do método dos elementos finitos utilizando o método dos gradientes conjugados, em ambiente MPI e PVM. Apesar da análise linear não representar um exemplo de alto custo dentre aplicações do método dos elementos finitos, sabe-se que análises não-lineares mais complexas podem envolver o emprego de algoritmos que executam análises lineares por partes na obtenção da solução final, justificando a paralelização da aplicação em elasticidade linear adotada neste trabalho. Ainda, comparações entre os tempos de execução do código nos ambientes MPI e PVM são apresentadas. Foram utilizadas as implementações mpich do MPI (Gropp et all, 1994) e PVM3 do PVM.

## 2. ESTRUTURA DE DADOS

A discretização do domínio contínuo utilizando o método dos elementos finitos conduz a um sistema de equações algébricas que pode ser escrito como:

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b} \tag{1}$$

onde, **A** é uma matriz simétrica positiva definida, **x** é o vetor de incógnitas nodais e **b** é o vetor de termos independentes.

No presente trabalho, o sistema de equações foi estruturado na forma matricial por blocos ou "*block arrowhead*" de acordo com Jimack and Touheed (2000) e descrita a seguir.

A solução do problema paralelizado requer a divisão do domínio em sub-regiões de acordo com o número de processadores disponíveis na rede e as equações referentes a estas regiões são resolvidas independentemente. Dessa forma, a Figura (1) representa uma malha de elementos finitos triangulares, dividida em subdomínios. Observa-se que graus de liberdade de fronteira são compartilhados por mais de um subdomínio, e graus de liberdade internos (ou privados), são contidos em um único subdomínio.


Figura 1. Partição de uma malha em 5 outras sub-malhas.

A matriz **A** de cada subdomínio computacional é desmembrada em quatro outras que são $\mathbf{A_p}$, $\mathbf{A_s}$, $\mathbf{B_p}$ e $\mathbf{B_p}^T$, como mostra a Fig. (2). $\mathbf{A_p}$ é uma matriz quadrada onde são armazenados somente os valores referentes aos graus de liberdade internos. A matriz $\mathbf{A_s}$ também é quadrada e seu tamanho é definido pelo número de graus de liberdade que estão na fronteira de cada partição. A dimensão da matriz $\mathbf{B_p}$ é função tanto do número de graus de liberdade internos quanto dos de fronteira. Nesta matriz são escritos os valores que relacionam os graus de liberdade privados com os de fronteira. O vetor **b** por sua vez é desmembrado em $\mathbf{b_p}$ e $\mathbf{b_s}$, sendo que o primeiro tem dimensão igual ao número de graus de liberdade internos e o segundo aos de fronteira. O mesmo procedimento é feito para o vetor **x.**

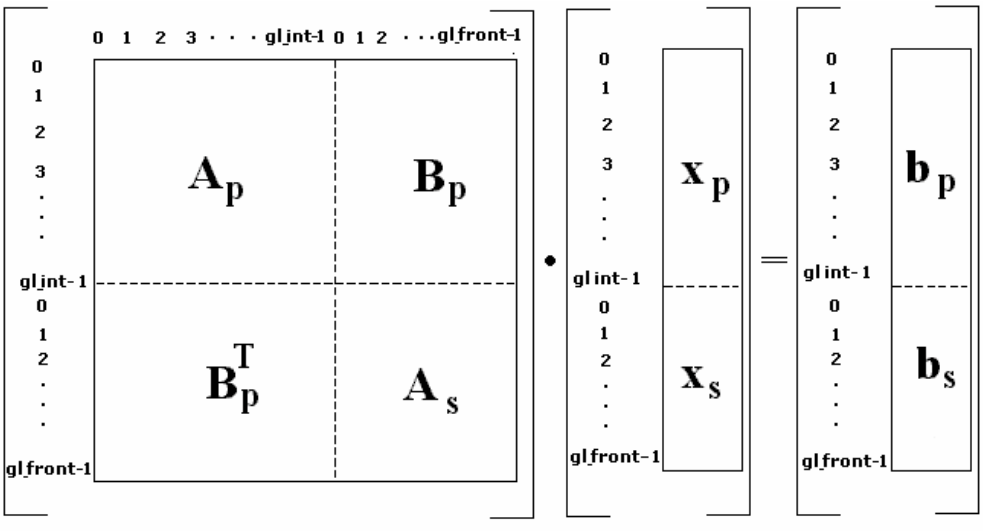



Figura 2. Matriz de um subdomínio montada de acordo com a estrutura "*block arrowhead*".

## 3. DESCRIÇÃO DO CÓDIGO

A etapa de pré-processamento é realizada em uma única máquina com o uso do aplicativo GID® que fornece para o código paralelizado os arquivos de entrada referentes à geometria, condições de contorno e malha, Fig. (3). O corpo do programa realiza a chamada das funções de leitura de dados, particionamento da malha e montagem das matrizes dos elementos.

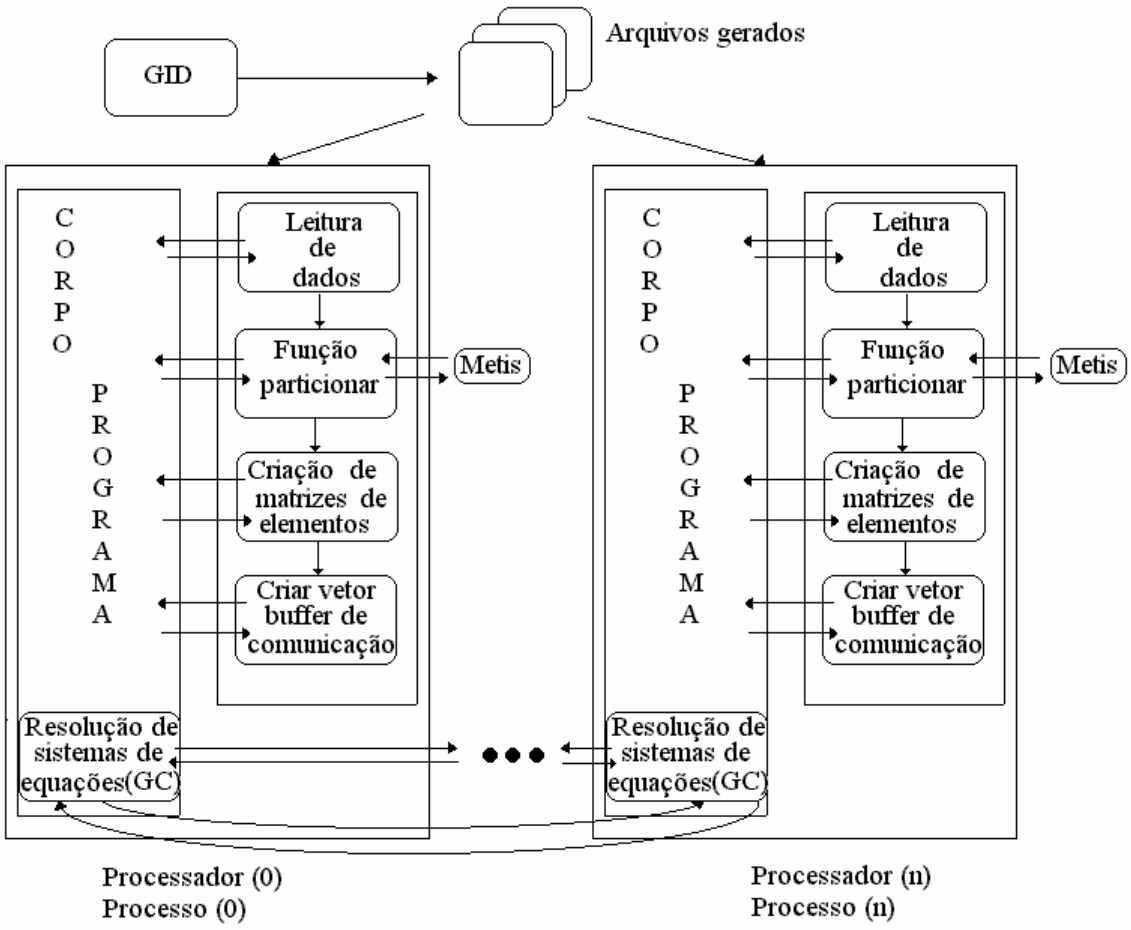



Figura 3. Fluxo de dados no código paralelizado.

O particionamento da malha é realizado através do programa METIS® (George and Vipin, 1998), e o número de partições é igual ao número de processadores utilizados.

A solução do sistema de equações algébricas é feita no próprio corpo do programa através do método dos gradientes conjugados (Golub and Van Loan, 1989), por ser adequado à solução de grandes sistemas de equações e pelo fato do algoritmo paralelizado já se encontrar disponível na literatura (Jimack and Touheed, 2000).

A função gradientes conjugados resolve o sistema de equações montado de acordo com a estrutura de dados "*block arrowhead*" para cada subdomínio. Dentro desta função são chamadas as funções produto interno e atualizar.

A função produto interno calcula o produto interno entre dois vetores distribuídos em paralelo, sendo necessário nesta etapa, uma comunicação global entre os processos. Antes (no caso do PVM) ou durante (no caso do MPI) esta comunicação é realizada uma operação de redução, a qual determina a soma das contribuições do produto interno. Ao término desta(s) operação(ões) todos os processos recebem o valor calculado. Ver Tabela (1) e Tabela (2) para as implementações do código no MPI e no PVM, respectivamente.

A função atualizar faz uso de algumas comunicações ponto a ponto de cada processo com seus vizinhos. Esta função tem por finalidade permitir que as distribuições dos valores nodais na fronteira da partição sejam montadas em cada processo.

A Tabela (1) mostra a seqüência de comandos do código paralelizado com o uso da biblioteca MPI.

Maiores detalhes sobre esta implementação em ambiente MPI podem ser encontrados em Jimack and Touheed (1999).

Tabela 1 – Código paralelizado com o uso da biblioteca MPI

<table>
<tr><th colspan="2">Todas as tarefas:</th></tr>
<tr><td colspan="2">Inicialização das variáveis do programa</td></tr>
<tr><td colspan="2">MPI_Init ( ) – inicialização do MPI</td></tr>
<tr><td colspan="2">MPI_Comm_rank ( ) – cada tarefa obtém o seu rank no comunicador</td></tr>
<tr><td colspan="2">MPI_Comm_size ( ) – cada tarefa obtém o tamanho do grupo (número de tarefas)</td></tr>
<tr><td colspan="2">Chamada da função leitura de dados</td></tr>
<tr><td colspan="2">Chamada da função de particionamento: cada processador executa o METIS®</td></tr>
<tr><td colspan="2">Chamada da função elemento trilinear: cada processador calcula as matrizes de elemento e monta a matriz na estrutura arrowhead associada ao seu subdomínio</td></tr>
<tr><td rowspan="2">Chamada da função gradientes conjugados:</td><td>• Chamada da função produto interno<br>MPI_Allreduce ( ) – Cada processo envia o valor calculado e sobre este valor é realizada a operação de soma, em seguida todos os processos recebem o valor calculado na operação.</td></tr>
<tr><td>• Chamada da função atualização (comunicação)<br>MPI_Send( )<br>MPI_Recv( )</td></tr>
<tr><td colspan="2">MPI_Finalize( ) – Finaliza o MPI</td></tr>
</table>

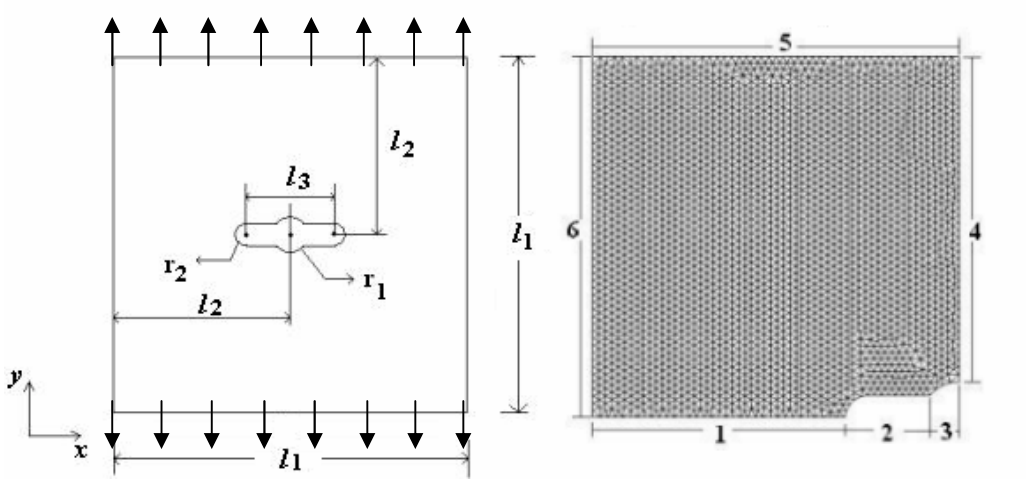
Para facilitar a implementação do código com o uso da biblioteca PVM a partir de um código escrito para MPI, utilizou-se as instâncias das tarefas no lugar dos *ranks* (MPI) ao invés de identificadores de tarefas (*tids)*. Os valores dos *tids* foram apenas utilizados quando requeridos em comandos do PVM (funções *send* e *receive*). A Tabela (2) mostra a seqüência de comandos do código paralelizado com o uso da biblioteca PVM.

Tabela 2 – Código paralelizado com o uso da biblioteca PVM

<table>
<tr><td colspan="3">Todas as tarefas:</td></tr>
<tr><td colspan="3">Inicializar as variáveis do programa</td></tr>
<tr><td colspan="3">Receber número de tarefas (PROCNO).</td></tr>
<tr><td colspan="2">Se Tarefa - instância (rank) = 0, fazer:</td><td>Senão:</td></tr>
<tr><td colspan="2">pvm_spawn ( ) – A tarefa mestre cria PROCNO -1 tarefas, de modo que o grupo tenha PROCNO tarefas.</td><td></td></tr>
<tr><td colspan="3">Todas as tarefas:</td></tr>
<tr><td colspan="3">pvm_barrier("pvm", PROCNO) – sincroniza as tarefas no grupo “pvm” antes de começar a execução dos cálculos. em caso de erro todas as tarefas devem sair do grupo (pvm_lvgroup( )) e finalizar o PVM (pvm_exit( )). PROCNO é o número de tarefas para sincronizar.</td></tr>
<tr><td colspan="3">Chamada da função leitura de dados</td></tr>
<tr><td colspan="3">Chamada da função de particionamento</td></tr>
<tr><td colspan="3">Chamada da função elemento trilinear</td></tr>
<tr><td rowspan="2">Chamada da função gradientes conjugados:</td><td colspan="2">• Chamada da função produto interno<br>(Função Produto Interno)<br>Se Tarefa - instância (rank) = 0, fazer:<br>pvm_reduce(, , , , , , ) operação de redução, na qual a tarefa rank=0 recebe o valor da redução.<br>pvm_initsend( ) –inicializar o buffer para o envio do bcast<br>pvm_pkfloat(, , ) – empacota dados<br>pvm_bcast(,) – A tarefa mestre envia o resultado da redução para todas as outras tarefas do grupo.<br>Senão, outras tarefas:<br>pvm_reduce(, , , , , , ) - operação de redução, na qual a tarefa rank=0 recebe o valor da redução.<br>pvm_recv(, ) – tarefas com rank ≠0 recebem o valor enviado pelo bcast da mestre. É necessário usar o comando pvm_gettid como argumento.<br>pvm_upkfloat(, , ) – desempacota os valores recebidos.</td></tr>
<tr><td colspan="2">• Chamada da função atualização<br>(Função Atualização)<br>Todas as tarefas:<br>pvm_initsend()<br>pvm_pkfloat(, , )<br>pvm_send(,) - é necessário usar o comando pvm_gettid como argumento.<br>pvm_recv(, ) - é necessário usar o comando pvm_gettid como argumento.<br>pvm_upkfloat(, , )</td></tr>
<tr><td colspan="3">Todas as tarefas:</td></tr>
<tr><td colspan="3">pvm_barrier("pvm", PROCNO) – sincroniza a saída das tarefas do grupo “pvm” .</td></tr>
<tr><td colspan="3">pvm_lvgroup(“pvm”) – Tarefas devem sair do grupo “pvm”.</td></tr>
<tr><td colspan="3">pvm_exit( ) –finalizar os comandos pvm</td></tr>
</table>

## 4. VALIDAÇÃO DOS RESULTADOS E ANÁLISE DE DESEMPENHO

Para realizar os testes de validação do código e a análise dos tempos de execução em paralelo foi utilizada uma placa tracionada, representada na Fig. (4a) e o respectivo modelo computacional, Fig. (4b) com a malha de elementos finitos triangulares lineares. As dimensões da placa são: $l_1$ = 0,058 m, $l_2$= 0,0254m, $l_3$ = 0,0127m, $r_1$ = 0,00254m e $r_2$ = 0,0015875m.

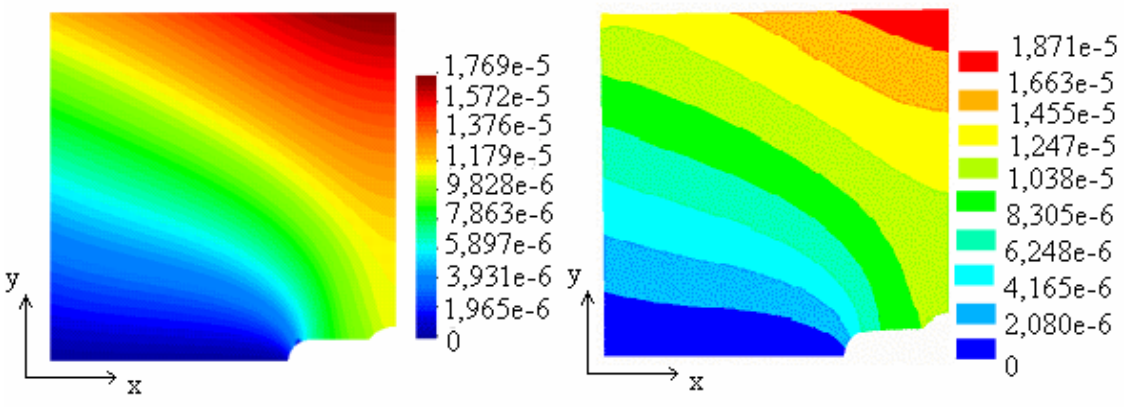



(a) (b)

Figura 4. Geometria da placa; (b) Malha computacional de elementos triangulares utilizada na simulação paralela.

No modelo da Fig. (4b) a placa é tracionada com uma carga de 22.241 N distribuída no contorno 5 e foram aplicadas restrições nos graus de liberdade da direção *y* do contorno 1 e da direção *x* do contorno 4. Considerou-se ainda o módulo de Young, E = 6,89 GPa, o coeficiente de Poisson, $\upsilon$ = 0,35 e a espessura da placa unitária. A malha gerada, resultou em 5.808 graus de liberdade. Os resultados foram obtidos com 311 iterações do método dos gradientes conjugados e uma tolerância de 1x $10^{-6}$.

O programa ANSYS foi utilizado para a verificação dos valores calculados pelo código. Cabe observar que no modelo executado no referido programa, a malha foi refinada localmente sendo empregados elementos finitos triangulares quadráticos. Na Fig. (5) são mostrados os resultados obtidos dos deslocamentos na direção ***y*** para o código paralelizado e para o ANSYS, respectivamente.




(a) (b)

Figura 5. (a) Deslocamentos (m) na direção ***y*** calculados pelo código paralelizado (b) Deslocamentos (m) em ***y*** calculados pelo ANSYS.

A Tabela (3) mostra os resultados dos tempos decorridos do particionamento e do código (obtidos com a função *getttimeofday*), nos ambientes mpich-1.2.5 e pvm3. Os valores apresentados foram obtidos pela execução do código em um *cluster* de PCs composto por nove máquinas Pentium II 350 MHz com 128 MB de memória RAM que utiliza o sistema operacional Linux Debian. As máquinas se comunicam por meio de um *switch* e a rede de interconexão é *ethernet* de 100 Mb/s. As simulações foram realizadas em apenas seis nós (máquinas) do *cluster* e a cada um destes foi atribuído um processo em execução.

Tabela 3. Tempos de execução do código paralelizado em MPI e PVM.

| **Nós/Processos** | **MPI** | | **PVM** | |
|---|---|---|---|---|
| | **Particionamento (s)** | **Código (s)** | **Particionamento (s)** | **Código (s)** |
| **1** | 0,112 | 1.150,56 | 0,119 | 1.140,07 |
| **2** | 0,133 | 320,70 | 0,141 | 314,87 |
| **3** | 0,117 | 156,72 | 0,156 | 151,24 |
| **4** | 0,119 | 90,48 | 0,142 | 91,03 |
| **5** | 0,120 | 60,23 | 0,143 | 65,55 |
| **6** | 0,124 | 43,40 | 0,145 | 52,32 |
| **7** | 0,127 | 34,44 | 0,153 | 42,85 |
| **8** | 0,174 | 28,55 | 0,332 | 60,18 |

Ainda, na Tabela (3), observa-se um acréscimo no tempo de processamento do particionamento para a execução em ambiente MPI com o aumento no número de sub-malhas. O tempo de particionamento é o tempo gasto na execução do arquivo particionar.c, o qual tem como principal escopo particionar a malha de elementos finitos com o auxílio do Metis®. De acordo com os valores apresentados é possível verificar que o tempo de particionameto tende a crescer com o aumento do número de processos em execução, isto porque a complexidade do corte aumenta com o número de subdomínios a serem criados. Verifica-se ainda que na execução no ambiente MPI os tempos de particionamento foram menores do que os obtidos no ambiente PVM.

A Figura (6) mostra a relação entre os tempos de execução do código e o número de processadores de acordo com os valores apresentados na Tab. (3). Para o MPI e o PVM esta relação se mostrou bastante ajustada por meio de uma curva polinomial, onde $\boldsymbol{t}$ é a função de ajuste (tempo) e $R^2$ é o coeficiente de correlação dos valores desta função. Não houve preocupação com a eficiência do método dos gradientes conjugados implementado para resolver o sistema de equações, não tendo sido feito o pré-condicionamento, por exemplo. Melhorias também podem ser implementadas utilizando-se a técnica elemento-por-elemento e paralelizando-se o produto matriz-vetor (Sahu, 1995).

A Figura (7) mostra a curva de ajuste para os valores de *speedup* obtidos para as execuções em ambientes MPI e PVM. Pode-se observar que o *speedup* cresce com o aumento do número de processos para os dois ambientes sendo que, a curva referente ao MPI apresenta um crescimento mais acentuado. Na mesma Figura (7), a tabela mostra o ganho percentual em termos de tempo decorrido de um ambiente em relação ao outro.

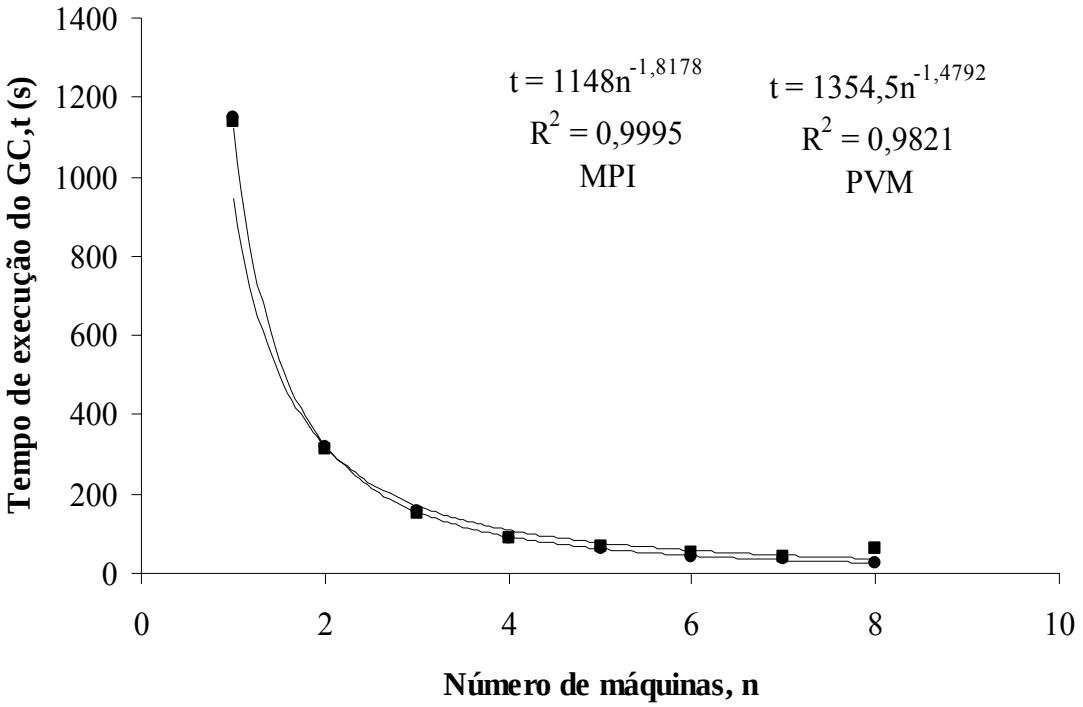
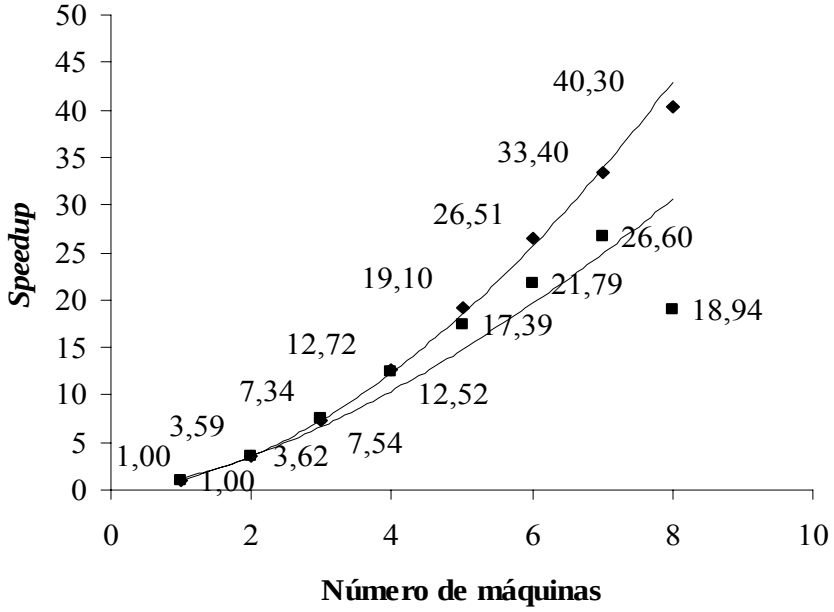



Figura 6. Tempo de execução do código paralelizado versus número de máquinas. ●MPI ■PVM.




| Máquinas | Ganho do MPI sobre PVM (%) |
|---|---|
| **1** | -0,91% |
| **2** | -1,85% |
| **3** | -3,62% |
| **4** | 0,60% |
| **5** | 8,10% |
| **6** | 17,05% |
| **7** | 19,63% |
| **8** | 52,56% |

Figura 7. *Speedup* versus número de máquinas ●MPI ■PVM.

É importante notar que mesmo existindo ganhos significativos na solução paralelizada do programa tanto em MPI quanto em PVM foi observada uma melhor adequação ao ambiente MPI para o caso estudado, principalmente com o aumento do número de máquinas. Este fato está relacionado a um menor tempo gasto na comunicação entre processos para este ambiente. No ambiente MPI são usadas funções de comunicação mais apropriadas aos cálculos realizados nas técnicas de solução existentes, como por exemplo, a função *MPI_Allreduce* empregada na função produto interno do método dos gradientes conjugados. No ambiente PVM esta função foi

substituída por outras seis que são: *pvm_reduce*, *pvm_initsend*, *pvm_pkfloat*, *pvm_bcast*, *pvm_recv* e *pvm_upkfloat*.

Para a solução de um problema de elasticidade utilizando o método dos elementos finitos, mostra-se que podem ser obtidos bons resultados utilizando-se uma rede de computadores já existente sem nenhum custo adicional com hardware ou software. Na realização desta aplicação foi necessário ainda, o uso do pré-processador GID® para descrição da geometria e geração da malha e o uso de um programa para realização do particionamento da malha, no caso, o METIS®. Para aplicações que não necessitam de maiores investimentos no desempenho após a paralelização, o uso do *cluster* de computadores em rede pré-existente é uma alternativa interessante em termos de economia de tempo de processamento.

## 5. AGRADECIMENTOS

Ao LAICO - Laboratório de Circuitos Integrados e Concorrentes do Departamento de Ciência da Computação da Universidade de Brasília pelo uso do *cluster* de PCs.

À Capes pela bolsa de mestrado.

## 6. REFERÊNCIAS

Golub, G.H. and Van Loan, C.F., 1989, "Matrix Computations", 2nd ed., John Hopkins University Press.

George, K., Vipin, K., 1998, "METIS – A Software Package for Partitioning Unstructured Graph, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices", Manual, University of Minnesota, Department of Computer Science.

Gropp, W., Lusk, E. Skjellum, A., 1994, "Using MPI: Portable Parallel Programming with Message-Passing Interface" , MIT Press, Cambridge, Massachusetts.

Jimack, P.K., Touheed, N., 1999, "An Introduction to MPI for Computational Mechanics", in: Parallel and Distributed Processing for Computational Mechanics Systems and Tools, ed. B.H.V. Topping and L. Lamner (Saxe-Coburg Pub.), pp. 24-45.

Jimack, P.K., Touheed, N., 2000, "Developing Parallel Finite Element Software Using MPI", in: High Performance Computing for Computational Mechanics, ed. B.H.V. Topping and L. Lamner (Saxe-Coburg Pub.), pp. 15-38.

Sahu, R., 1995, "Parallelization of Linear FE Structural Analysis in 2D Using MPI", Proceedings of MPI Developers Conference, University of Notre Dame, Notre Dame, Indiana.

# TITLE: PARALLELIZATION IN PVM AND MPI ENVIRONMENT APPLIED TO A FINITE ELEMENT METHOD APPLICATION.

**Código**: 54012

***Abstract:*** *Developing applications to finite element method using distributed computing can be an economic way to solve some problems. For an existing network and disposing some free or low cost softwares and a Linux operational system, the use of cluster computing is an interesting alternative. The aim of this work is the comparison of parallel execution using both MPI and PVM enviroments for a finite element application. Using C language, a code was developed to the solution of structural problems. The conjugate gradient method was employed to solve the resulting set of equations. The mesh generation was done using GID® as a preprocessor. For mesh partition, METIS® was employed. The equation system was assembled in a block arrowhead data structure. The code was validated and performance valuations were done to show the speedup in execution time.*

***Keywords:*** *finite element method, MPI, PVM.*