

Aspectos Gerais da Computação Paralela e do Sistema PVM

Célio Oda Moretti
Túlio Nogueira Bittencourt

*Laboratório de Mecânica Computacional
Departamento de Engenharia de Estruturas e Fundações*

Introdução

Atualmente, a necessidade de sistemas de computação mais rápidos e eficientes é cada vez maior, tanto na área científica como na área comercial. A demanda por computação de alto desempenho em áreas diversas como biomecânica, meteorologia e engenharia vem crescendo bastante nos últimos anos. Existem limites físicos para o aumento da velocidade de um único processador, além do alto custo associado ao desenvolvimento desses processadores mais velozes, o que levou ao surgimento de novas arquiteturas com alto poder computacional. Estas novas arquiteturas integram vários processadores (o número de processadores varia de dezenas a milhares), razoavelmente rápidos, compondo assim uma máquina de alto desempenho. Estas máquinas podem ser classificadas em três tipos:

- Multiprocessadores vetoriais: possuem um pequeno número de processadores vetoriais de alta performance.
- Sistemas MPP (*Massively Parallel Processors*): possuem de centenas a milhares de processadores, com memória distribuída ou compartilhada.
- Rede de estações de trabalho: máquinas interligadas por redes que podem trabalhar como uma única máquina virtual paralela.

Para se aproveitar plenamente o poder computacional dessas máquinas, é necessário o desenvolvimento de modelos de programação, de algoritmos e de ferramentas para estes novos ambientes. Este trabalho se baseia na utilização de estações de trabalho ligadas em rede. Para isso, será utilizada a biblioteca para passagem de mensagens PVM (*Parallel Virtual Machine*) para gerenciar o funcionamento da máquina paralela virtual, formada pelas estações, e serão desenvolvidos algoritmos específicos para esta configuração.

Neste trabalho serão apresentados alguns aspectos gerais sobre programação paralela, modelos de organização da memória e modos de programação. Também será descrita a biblioteca PVM: seu funcionamento e forma de utilização.

Programação Paralela

Nesta seção são apresentados aspectos gerais sobre a programação paralela, com a descrição da terminologia utilizada, modelos de acesso à memória em ambientes distribuídos e modelos de programação mais utilizados em processamento paralelo.

Terminologia

São descritos a seguir alguns termos utilizados em processamento paralelo [Elias,95]:

- Tarefas (ou processos): programas executados concorrentemente, geralmente disparados por um programa mestre. São as principais unidades do processamento paralelo em um ambiente de computação distribuída; comunicam-se através de troca de mensagens.
- Execução seqüencial: execução de um programa em um único processador, com as instruções sendo processadas uma de cada vez.
- Paralelização de código: consiste na transformação de um programa seqüencial em paralelo, com a identificação de porções de código que podem ser executadas independentemente. Exige mudanças no código do programa e, caso necessário, no algoritmo utilizado no programa seqüencial.
- Aceleração (*speed-up*): consiste na comparação entre o tempo de execução do programa em um único processador e o tempo de execução utilizando vários processadores.

$$speed-up = \frac{\text{tempo de execução em 1 processador}}{\text{tempo de execução em vários processadores}}$$

- Sincronização: coordenação necessária entre processos para a troca de informações. Pode ser um fator de decréscimo da eficiência do programa, uma vez que alguns processadores podem ficar inativos, esperando pelo término de outros processos.
- Granularidade: quantidade de processamento realizado por cada processo, em relação à quantidade de comunicação entre processos. Quando os processos executam poucas instruções e necessitam se comunicar muito, diz-se que o programa é muito granular. Quando, ao contrário, os processos executam muitas instruções, com pouca troca de informação, diz-se que o programa é pouco granular. Um programa com granularidade alta necessita de uma maior sincronização que um programa com menor granularidade, o que afeta o tempo de execução do programa.
- Escalabilidade: um sistema computacional paralelo é dito escalável se a aceleração atingida cresce proporcionalmente ao número de processadores utilizados.
- Balanceamento de carga: consiste na distribuição equilibrada de tarefas entre os processadores, de forma a garantir uma execução eficiente do programa paralelo.
- SPMD (*Single Program - Multiple Data*): modelo de programação onde todos os processadores executam o mesmo programa sobre diferentes conjuntos de dados.

Redes Locais de Computadores

Uma rede de computadores é formada pela interconexão de um conjunto de equipamentos processadores (computadores, impressoras,...) através de um sistema de comunicação. O sistema de comunicação é constituído de uma rede de interconexão montada sobre meios físicos de transmissão e um conjunto de regras de comunicação denominadas protocolos [Soares,95]. Uma rede local (*Local Area Network - LAN*) constitui-se de um conjunto de computadores confinado a uma área geográfica limitada por uma faixa de distância de metros a alguns quilômetros [Corrigan,89] [Soares,95].

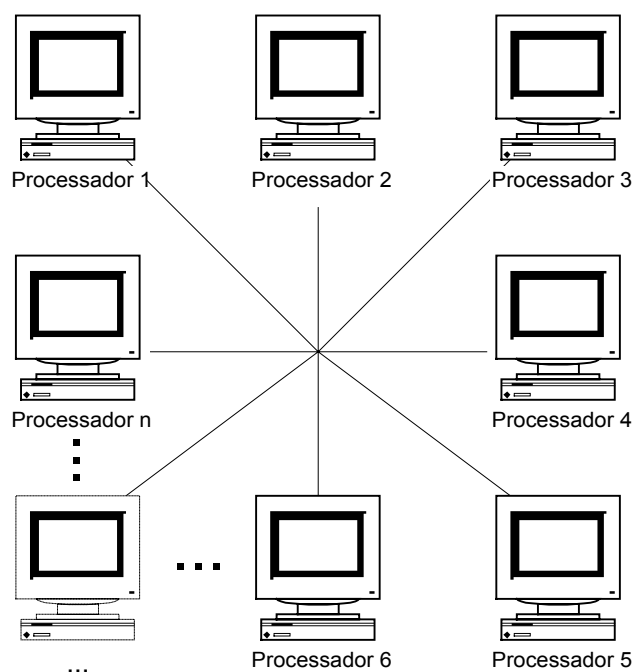


Figura 1 - Rede em estrela

As topologias mais utilizadas em redes locais são: estrela, anel e barra [Soares,95]. Redes em estrela são mais adequadas quando o sistema computacional paralelo baseia-se em um conjunto de processadores secundários comunicando-se com um processador mestre, o qual controla toda a comunicação ocorrida entre os processadores secundários (Figura 1). Redes em anel constituem-se basicamente em um conjunto de processadores interconectados por um anel (Figura 2). Redes em barramento constituem-se em um conjunto de processadores interligados através de um barramento comum a todos os processadores (Figura 3).

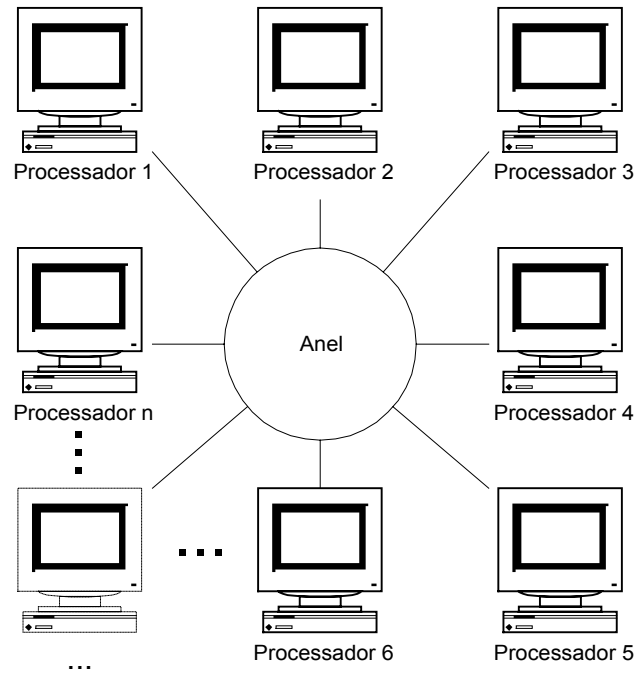


Figura 2 - Rede em anel

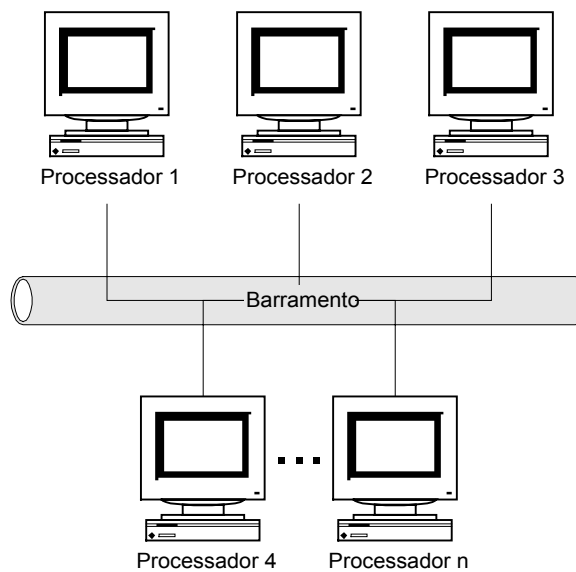


Figura 3 - Rede em barramento

A base de transmissão de dados de uma rede é qualquer meio físico que tenha a capacidade de transportar informações eletromagnéticas (dados) entre os componentes da rede. Os mais utilizados são o par trançado (TP), o cabo coaxial (BNC) e a fibra ótica.

Modelos de organização da memória

Existem dois modelos básicos de organização da memória [Elias,95] [Kumar,94]: memória compartilhada e memória distribuída.

Quando se utiliza o modelo de memória compartilhada, todos os processadores têm acesso total (escrita e leitura) a qualquer área da memória (Figura 4). Neste modelo, a sincronização entre processos é realizada através do controle das operações de escrita e leitura feitas pelos processos. Tem como vantagem a rapidez de acesso aos dados e, como desvantagem, a limitação no número de caminhos entre os processadores e a memória, o que diminui a escalabilidade do sistema.

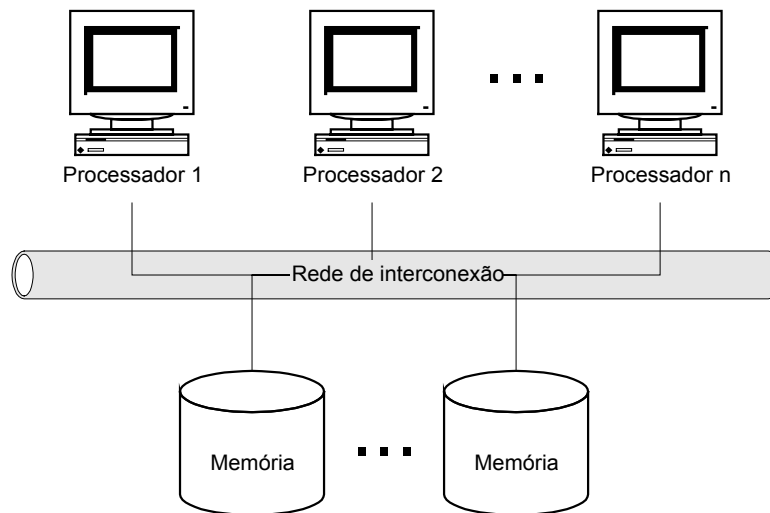


Figura 4 - Arquitetura de memória compartilhada

Uma forma de melhorar a escalabilidade do modelo de memória compartilhada consiste em prover cada processador com uma memória local (Figura 5), onde seriam armazenados o programa a ser executado e os dados exclusivos do processo

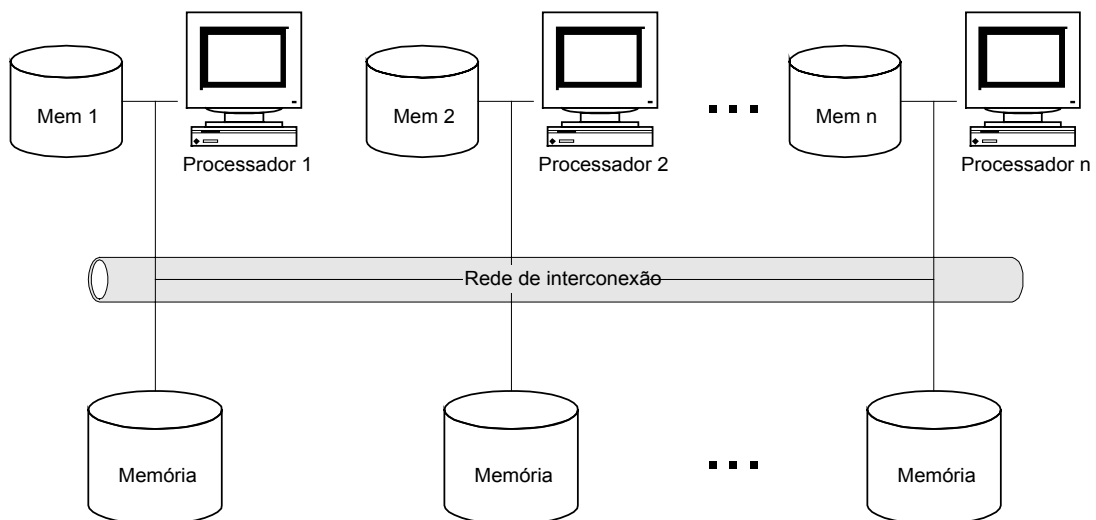


Figura 5 - Arquitetura de memória compartilhada com memória local nos processadores

Uma outra variação desse modelo consiste na eliminação total de compartilhamento físico da memória (Figura 6).

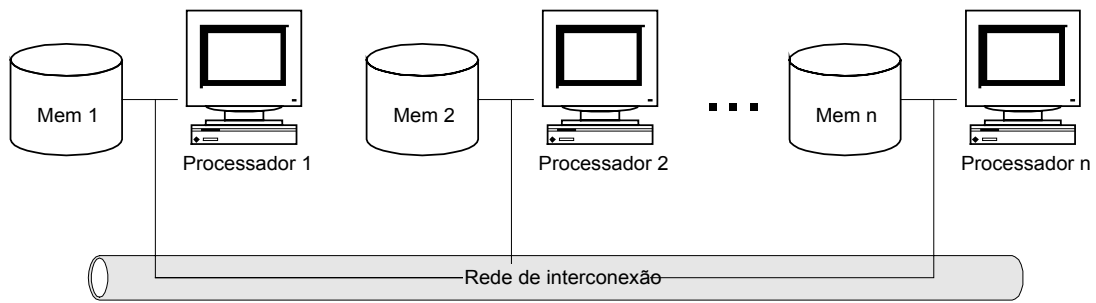


Figura 6 - Arquitetura de memória compartilhada somente com memória local nos processadores

Para se utilizar de maneira correta o modelo de memória compartilhada, deve-se prestar especial atenção no controle sobre o acesso à memória compartilhada pelos processos. Por exemplo, um dado não pode ser alterado por um processo enquanto um outro processo estiver lendo esse mesmo dado.

O outro modelo básico de organização da memória é o modelo de memória distribuída. Neste caso, a memória é distribuída fisicamente entre os processadores, sendo de acesso exclusivo do processador ao qual está associada. As informações são trocadas entre os processadores através do envio de mensagens pela rede de comunicação (Figura 7).

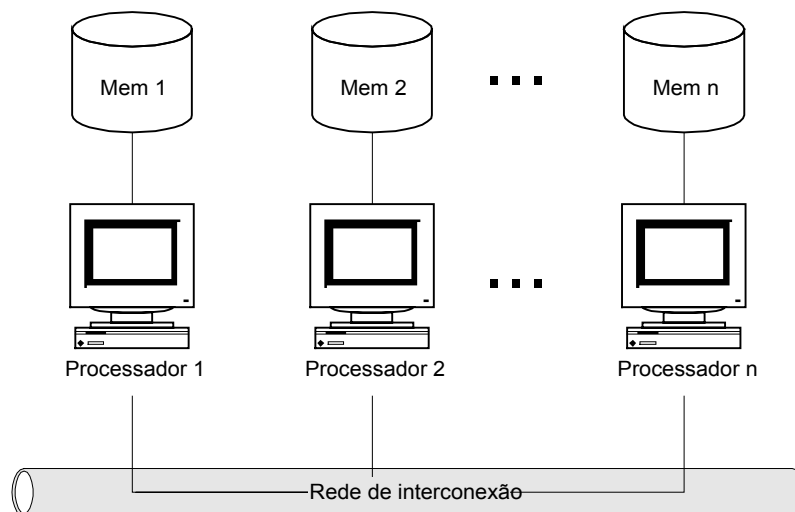


Figura 7 - Arquitetura de memória distribuída

As vantagens do modelo de memória distribuída são a escalabilidade do sistema, uma maior confiança no sistema devido à duplicação dos dados entre os processadores, com menor probabilidade de ocorrência de falhas, e também a facilidade de se incorporar novos processadores ao sistema, podendo-se utilizar conjuntamente processadores de arquiteturas diferentes.

Modos de programação

De maneira geral, os modos de programação paralela podem ser classificados em três modelos [Smith,92]:

- Paralelismo de dados: uma mesma seqüência de instruções é executada por cada processador sobre dados diferentes. Este é o modo de programação adotado na maior parte dos algoritmos implementados neste trabalho.
- Paralelismo de tarefas: o programa é particionado em tarefas cooperativas. Estas tarefas podem executar procedimentos bastante diferentes entre si, sem necessariamente haver uma sincronização entre as tarefas.
- Paralelismo de objetos: neste caso, o paralelismo pode ser realizado de diferentes formas. Por exemplo, uma função membro de um objeto pode chamar uma função pública de outro objeto remoto ou não. Ou um tipo abstrato de dados, como um vetor, pode ser implementado de maneira distribuída.

PVM

O PVM (*Parallel Virtual Machine*) é um conjunto integrado de ferramentas de *software* e bibliotecas que emula uma máquina paralela utilizando computadores interconectados de arquiteturas distintas [Geist,94]. O principal objetivo do PVM é possibilitar a utilização da computação paralela através de um conjunto heterogêneo de computadores ligados em rede.

Princípios

A unidade básica do paralelismo no PVM é a tarefa, uma seqüência independente de instruções, que se comunica com outras tarefas através da troca de mensagens. Não há a necessidade de se executar uma tarefa por processador. O PVM possibilita a execução de múltiplas tarefas em um único processador; ou seja, o modelo computacional do PVM é baseado no processo. É adotado também um modelo explícito de troca de mensagens, onde várias tarefas, cada uma realizando uma parte do trabalho computacional, comunicam-se entre si utilizando-se de funções de envio e recepção de mensagens. O tamanho da mensagem é limitado apenas pela quantidade de memória disponível.

O PVM suporta heterogeneidade de máquinas, redes e aplicações. Permite que um mesmo dado que tenha uma representação diferente em duas máquinas de arquiteturas distintas, possa ser trocado de maneira transparente entre essas máquinas, fazendo a conversão automaticamente.

O conjunto de máquinas pertencentes à máquina virtual paralela pode ser configurado livremente pelo usuário, o qual escolhe quais máquinas deseja utilizar. A adição ou retirada de máquinas pode ser feita tanto antes de se executar o programa paralelo, quanto durante a execução. Além disso, os programas podem ver o conjunto de máquinas apenas como uma coleção de processadores idênticos, ou podem explorar as capacidades específicas de cada processador, enviando as tarefas aos computadores mais apropriados.

Descrição do sistema

O PVM é composto de duas partes. A primeira parte é um *daemon* (programa residente executado em *background*), chamado *pvmd3*, que reside em todos os

computadores pertencentes à máquina virtual. Quando o usuário deseja executar uma aplicação PVM, primeiro deve criar a máquina virtual, inicializando o PVM. A aplicação PVM pode então ser executada de qualquer uma das máquinas.

A segunda parte do sistema é uma biblioteca de rotinas de interface, contendo um conjunto completo de primitivas que são necessárias para a interação entre as tarefas de uma aplicação. Esta biblioteca contém rotinas para a passagem de mensagens (envio e recepção), disparo de processos, coordenação das tarefas e modificação da máquina virtual.

Todas as tarefas são identificadas por um número inteiro chamado *task identifier* (*TID*). Mensagens são enviadas e recebidas utilizando-se esse identificador. Uma vez que esses números devem ser únicos para toda a máquina virtual, eles são definidos pelo *daemon* local, e não por uma escolha do usuário. O PVM possui diversas rotinas que retornam valores de TID, possibilitando a identificação das tarefas pela aplicação do usuário.

O sistema PVM suporta as linguagens C, C++ e Fortran. As rotinas da biblioteca para C e C++ são implementadas como funções, seguindo as convenções gerais da maioria dos sistemas C. Programas escritos em C e C++ acessam as funções da biblioteca PVM ligando-se a um arquivo de biblioteca (`libpvm3.a`). Já para a linguagem Fortran, as rotinas são implementadas como subrotinas, ao invés de funções, e os programas feitos nessa linguagem devem-se ligar a uma outra biblioteca (`libfpvm3.a`).

Paradigma de programação com PVM

O paradigma geral de programação de aplicações utilizando-se o PVM é mostrado a seguir [Geist,94]. Um usuário escreve um ou mais programas seqüenciais em C, C++ ou Fortran que contêm chamadas embutidas à biblioteca PVM. Cada programa corresponde a uma tarefa. Esses programas são compilados para cada arquitetura presente na máquina virtual, e os arquivos resultantes são colocados em um local acessível às máquinas pertencentes ao sistema. Para executar uma aplicação, o usuário geralmente executa uma tarefa (usualmente a tarefa "*master*" ou de inicialização) manualmente a partir de uma máquina do sistema. Este processo dispara outras tarefas, resultando em uma coleção de tarefas ativas, as quais executam seqüências de instruções localmente e trocam mensagens entre si para resolver o problema.

A seguir, é mostrado um exemplo simples que ilustra os conceitos básicos da programação PVM [Geist,94]. Basicamente, esse exemplo consiste em um programa "*master*" (`hello.c` - Figura 8) , executado manualmente, que dispara uma tarefa (`hello_other.c` - Figura 9) e espera pelo retorno de uma mensagem contendo uma *string* (seqüência de caracteres). As rotinas do PVM utilizadas neste exemplo são descritas nas Figuras 8 e 9.

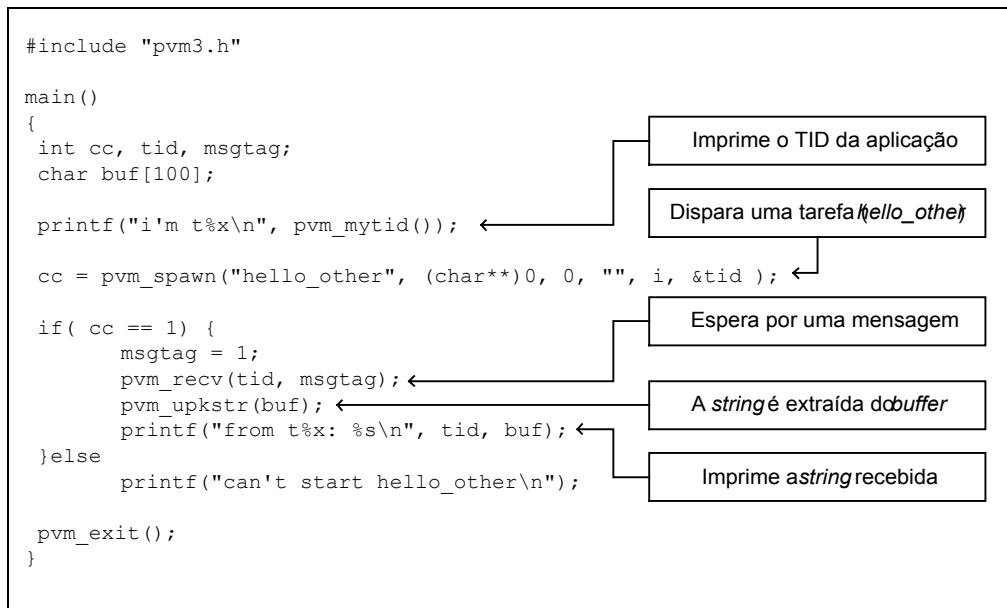


Figura 8 - Programa *hello.c*

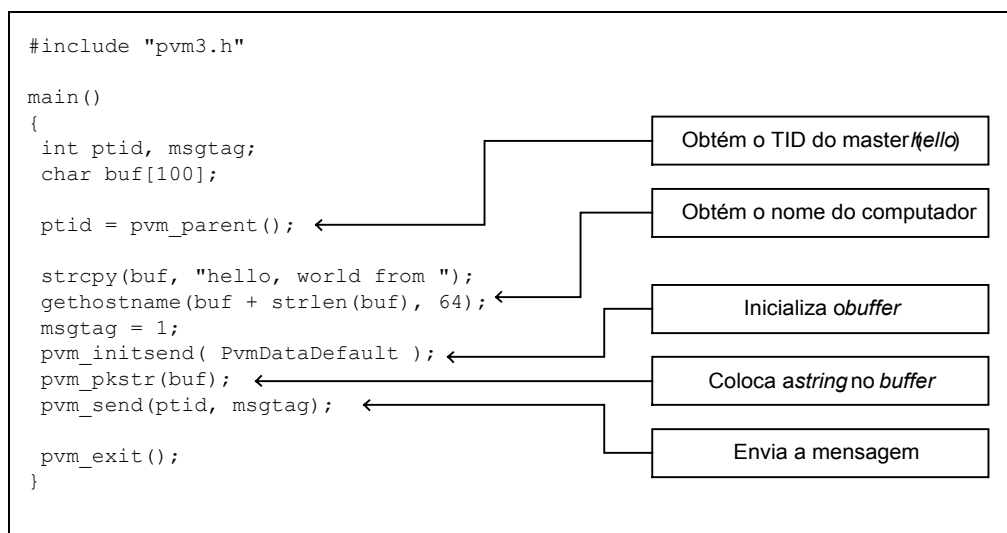


Figura 9 - Programa *hello_other.c*

Resumo

Neste trabalho, foram apresentados aspectos gerais da programação paralela, incluindo a terminologia utilizada, modelos de organização da memória e modos de programação. Também foram apresentados os diferentes tipos de redes locais de computadores. Por fim, os princípios e o funcionamento do PVM foram descritos, incluindo um exemplo simples de programa que ilustra os conceitos básicos da programação PVM.

REFERÊNCIAS BIBLIOGRÁFICAS

[Elias,95] Elias, D., "Introduction to Parallel Programming Concepts", *Workshop on Parallel Programming on the IBM SP*, Cornell Theory Center, 1995. Disponível através da WWW no endereço <http://www.tc.cornell.edu/Edu/Workshop>.

[Soares,95] Soares, L.F.G., Lemos, G., Colcher, S., 1995, "Redes de computadores: das LANs, MANs e WAMs às redes ATM", Campus, Rio de Janeiro, Brasil.

[Corrigan,89] Corrigan, P.H., Guy, A., 1989, "Building local area networks with Novell's Netware", M&T Publishing, Inc., Redwood City, USA.

[Kumar,94] Kumar, V., Grama, A., Gupta, A., Karypis, G., "Introduction to Parallel Computing: Design and Analysis of Parallel Algorithms", Benjamin/Cummings, 1994, Redwood City, USA.

[Smith,92] Smith, P.H., "System Software and Tools for High Performance Computing Environments", 1992. Disponível através da WWW no endereço <http://www.ccsf.caltech.edu/PSTP.html>.

[Geist,94] Geist, A., Beguelin, A., Dongarra, J., Jiang, W., Manchek, R., Sunderman, V., PVM: Parallel Virtual Machine - A User's Guide and Tutorial for Networked Parallel Computing, MIT Press, 1994, Cambridge.

[Johnsson,84] Johnsson, L., "Highly Concurrent Algorithms for Solving Linear Systems of Equations," Elliptic Problem Solver II, Proceedings of the Elliptic Problem Solvers Conference, Naval Postgraduate School, Monterey, California, 10-12 January 1983, edited by Birkhoff, G. and Schoenstadt, A., Academic Press, New York, 1984, pp. 105-126.

[Geist,85] Geist, G. A. and Heath, M. T., "Parallel Cholesky Factorization on a Hypercube Multiprocessor," Oak Ridge National Laboratory, Engineering Physics and Mathematics Division, Mathematical Sciences Section, Oak Ridge, Tennessee, Paper no. ORNL-5190, Distribution category UC-32, 1985.

[Li,86] Li, G. and Coleman, T. F., "A Parallel Triangular Solver for a Hypercube Multiprocessor," Cornell Theory Center Technical Report CTC86TR8, Cornell University, Ithaca, New York, December 1986.

[Hestenes,52] Hestenes, M. e Stiefel, E., "Methods of Conjugate Gradients for Solving Linear Systems," Journal of Research of the National Bureau of Standards, Vol. 49, No. 6, December 1952, Research Paper 2379, pp. 409-436.

[Noble,66] Noble, B., Applied Linear Algebra, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1966.

[Golub,83] Golub, G.H. e Van Loan, C.F., Matrix Computations, The Johns Hopkins University Press, Baltimore, 1983.

[Manteuffel,79] Manteuffel, T.A., "Shifted Incomplete Cholesky Factorization," Sparse Matrix Proceedings 1978, editado por Duff, I.S. e Stewart, G.W., SIAM, Philadelphia, Pennsylvania, 1979, pp. 41-51.

[Jennings,77] Jennings, A. e Malik, G.M., "Partial Elimination," Journal of the Institute of Mathematics and its Applications, Vol. 20, No. 3, November 1977, pp. 307-316.

[Hajjar,87] Hajjar, J.F., "Parallel Processing for Transient Nonlinear Structural Dynamics of Three-Dimensional Framed Structures," Department of Structural Engineering Report, No. 87-5, Cornell University, November 1987.

[Elman,86] Elman, H., "A Stability Analysis of Incomplete LU Factorization," Math. Comp., No.47, pp. 191-218.

[Nasra,90] Al-Nasra, M. e Nguyen, D.T., "An Algorithm for Domain Decomposition in Finite Element Analysis," Computers & Structures, Vol. 39, No. 3/4, pp. 277-289, 1991.

[Farhat,88] Farhat, C., "A Simple and Efficient Automatic FEM Domain Decomposer," Computer & Structures, Vol. 28, No. 5, pp. 579-602, 1988.

[Hsieh,93] Hsieh, S.H., "Parallel Processing for Nonlinear Dynamics Simulations of Structures including Rotating Bladed-Disk Assemblies," Ph.D. dissertation, Cornell University, Ithaca, New York, 1993.

[Karypis,95] Karypis, G. e Kumar, V., "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs," Technical Report TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[Karypis/Kumar,95] Karypis, G. e Kumar, V., "METIS: Unstructured Graph Partitioning and Sparse Matrix Ordering," Department of Computer Science, University of Minnesota, 1995. Disponível através da WWW no endereço <http://www.cs.umn.edu/~karypis/metis/metis.html>.