ESTUDO DE APLICABILIDADE DE SISTEMAS FRACAMENTE ACOPLADOS UTILIZANDO *HARDWARE* DE BAIXO CUSTO

Hilario Viana Bacellar[*]
Matheus de Paula França[**]
Rafael Roberto[***]
Edgar Noda[****]
Jeanne Dobgenski[*****]

RESUMO: Este trabalho apresenta um estudo inicial que verifica o uso de arquiteturas paralelas fracamente acopladas na resolução de problemas tradicionalmente intratáveis pelos sistemas computacionais usuais. Entende-se por sistemas computacionais usuais todo o poder de processamento obtido em uma única máquina, cujo custo é considerado acessível ao público em geral. A solução proposta neste artigo é baseada em uma estrutura de *cluster Beowulf*, implementado com a biblioteca de comunicação paralela MPI em um *hardware* de baixo custo. A heterogeneidade do equipamento utilizado também foi levada em consideração por ser um cenário usual na utilização de máquinas de baixo custo. De forma a validar o modelo proposto são comparados os tempos de execução de tarefas que executam cálculos exaustivos, os quais foram paralelizados em configurações distintas. A expectativa inicial com relação à possibilidade de ganhar tempo computacional com a paralelização dos dados é confirmada assim como a relevância que a escalabilidade de máquinas possui sobre os resultados obtidos.

PALAVRAS CHAVES: Cluster. Beowulf. Processamento paralelo. MPI.

STUDY OF APPLICABILITY OF WEAK SYSTEMS COUPLED USING LOW-COST HARDWARE

ABSTRACT: This paper presents an initial study that checks the use of parallel architectures weakly coupled in resolving problems which are unmanageable by usual computer systems. It is understood by usual computer systems all the power of processing obtained in a single machine, whose cost is considered accessible to the general public. The solution proposed in this paper for high-power processing is based on a

[*] Bacharel em Ciência da Computação. E-mail: hilario_viana@hotmail.com
[**] Bacharel em Ciência da Computação. E-mail: matheus_tux@yahoo.com.br
[***] Bacharel em Ciência da Computação. E-mail: rarob16@yahoo.com.br
[****] Orientador. Mestrado em Engenharia Elétrica e Informática. E-mail: noda.edgar@gmail.com
[*****] Orientadora. Mestrado em Engenharia Elétrica. E-mail: jeanne.dob@gmail.com

Recebido em: Setembro/2008 Aprovado em: Outubro/2008

structure of Beowulf cluster, implemented with the MPI parallel library of communication in a low-cost hardware. In order to validate the proposed model we compared the time of execution of tasks that perform exhaustive calculations, which were paralleled in different configurations. The initial expectation related to the possibility of gaining time with the parallelization of computational data is confirmed as well as the importance that the scalability of machines have on the results. The expectation of using machines at low cost to solve this kind of problem is confirmed, since the service is still available.

KEY WORDS: Cluster. Beowulf. Parallel processing. MPI.

INTRODUÇÃO

Nesses últimos anos é notável a evolução do poder computacional que está disponível para a grande maioria de usuários. Com a atual tendência da indústria no desenvolvimento de processadores com múltiplos núcleos, o processamento paralelo nunca esteve tão acessível. Entretanto, muitas aplicações ainda necessitam de um equipamento muitas vezes superior ao utilizado pela maioria dos usuários comuns. Aplicações como previsões climáticas, simulações de problemas que envolvam um número elevado de variáveis ou mesmo resoluções ótimas de problemas com características exponenciais, ainda demandam arquiteturas multiprocessadas fortemente acopladas, equipamentos classificados como supercomputadores.

Devido ao alto valor a ser investido, os supercomputadores, se tornam inviáveis para a grande maioria dos potenciais usuários interessados nesse tipo de processamento.

Os supercomputadores, que são sistemas fortemente acoplados, apresentam características como compartilhamento do endereçamento de memória e utilização de um sistema operacional homogêneo. Machado (2007) explica que os sistemas fracamente acoplados são caracterizados pela independência de seus nós computacionais sendo que a comunicação entre eles é realizada por uma das tecnologias de redes de computadores. Uma desvantagem de sistemas fracamente acoplados em relação aos supercomputadores é a necessidade de troca de informações através da rede. Esse é um limitador considerável de desempenho se comparado aos supercomputadores que utilizam o mesmo barramento de memória para a troca de informações. Nos casos de tarefas que apresentam uma granulosidade grossa[1] é possível obter um desempenho similar ao de um supercomputador, pois, a utilização da rede não influencia no resultado por haver pouca troca de informação.

[1] Granulosidade grossa: grande quantidade de processamento por ponto de sincronização.

Na atualidade, existem diversos trabalhos que vêm sendo desenvolvidos na área de sistemas distribuídos, no qual, *clusters* de computadores fazem parte. No trabalho realizado por Pitanga (2004) o *cluster* Multipinguim é usado para a programação paralela e é uma implementação do tipo *Beowulf*, cuja finalidade é criar um supercomputador acessível a laboratórios *e* instituições de ensino superior. Adams (2007) apresenta o Microwulf cujo objetivo foi conseguir a menor relação custo/benefício de um *cluster* de alto desempenho, visando quebrar a barreira dos 100 dólares por *gigaflop*[2]. Ao término do projeto, com um gasto de 2570 dólares o *cluster* alcançou a marca de 26,25 *gigaflops*.

OBJETIVOS

Tem-se por objetivo a avaliação do desempenho de sistemas fracamente acoplados utilizando *hardware* de baixo custo ao se utilizar tarefas de cálculo exaustivo e paralelizável. Para isso é necessário a construção de um *cluster* de computadores compatível com programação paralela tendo como foco o processamento massivo de dados.

METODOLOGIA

Com o embasamento da revisão bibliográfica foi possível aprofundar o conhecimento sobre os tipos de *cluster* e suas principais características. Desta forma, foi adotada, inicialmente, a concepção e realização de testes com o *cluster* do tipo de alto desempenho, cujo foco é o processamento massivo de dados.

Para o *cluster* de alto desempenho, foi escolhido o tipo *Beowulf*, utilizando-se o sistema operacional Linux e a biblioteca MPI (*Message Passing Interface*) para a comunicação paralela. O principal motivo da escolha do tipo *Beowulf* se deve a sua flexibilidade em trabalhar tanto em ambientes homogêneos como em heterogêneos. A escolha da biblioteca MPI se deve ao fato de que ela é uma biblioteca de comunicação portátil, estável e relativamente nova, apresentando ainda uma facilidade na obtenção de documentação.

O material utilizado para o desenvolvimento da pesquisa foi alocado conforme a disponibilidade dos recursos no laboratório em que os testes foram realizados.

- Um Sempron 3000 1.8 GHz com 512 Mb de memória RAM (Random Access Memory), 40 Gb de HD e placa de rede 100 Mbits.

[2] G*igaflop*: um bilhão de instruções computacionais de pontos flutuantes por segundo.

- Três Duron 950 Mhz com 512 Mb de memória RAM, 20 Gb de HD e placas de rede 100 Mbits.
- Um *Hub* com 8 portas *ethernet*.

DESENVOLVIMENTO

Para a realização deste trabalho foi necessário o entendimento sobre o funcionamento de *clusters,* como tornar um processo paralelizável e o uso de bibliotecas de comunicação paralelas no suporte desta tarefa. Logo, nesta seção são apresentadas informações básicas sobre esses tópicos.

CLUSTER

Por definição *cluster* é uma arquitetura fracamente acoplada interligando mais de um computador em rede, cujo objetivo é fazer com que todo o processamento da aplicação seja distribuído entre os processadores de forma mais transparente possível. De acordo com Tanenbaum (2007) "sistemas de computação em *cluster* tornaram-se populares quando a razão preço/desempenho de computadores pessoais e estações de trabalho melhorou." Conforme afirma Pitanga (2004), a utilização de *cluster* de computadores tem inúmeras vantagens, entre as quais é possível destacar o alto desempenho, a escalabilidade, tolerância a falhas, baixo custo e independência de fornecedores.

Por essas razões, *clusters* são utilizados em servidores *web*, sistemas de comércio eletrônico, servidores de banco de dados e soluções de *firewall*. Os tipos de *cluster* mais conhecidos são:

- processamento distribuído;
- balanceamento de carga;
- alta disponibilidade e balanceamento de carga;
- alta disponibilidade e tolerância a falhas.

O *cluster Beowulf* não exige uma arquitetura dedicada, tão pouco máquinas homogêneas. A conexão entre os nós pode ser feita por meio de *Ethernet*. Deve haver um ou mais nós mestres (também conhecidos por nós controladores ou *front-end*) para realizar o controle dos nós escravos (*back-end*) - Figura 1. O sistema operacional deve ser baseado em Linux, sendo que o mesmo deve conter todas as ferramentas necessárias para a configuração do *cluster*.

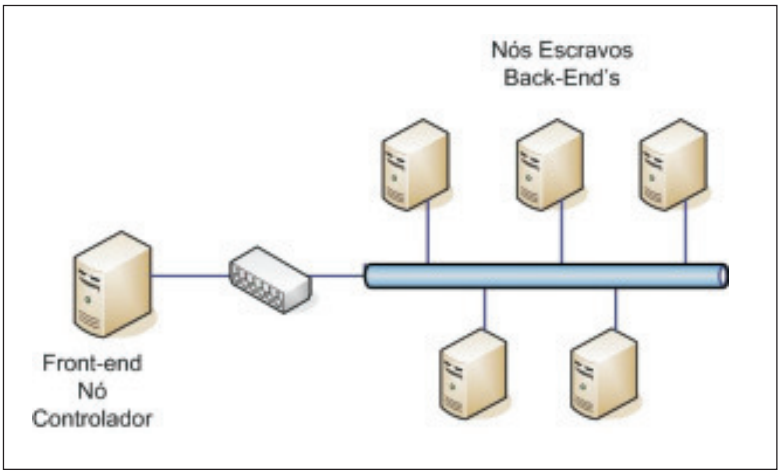

Fig. 1 - Arquitetura *front-end back-end* em *cluster*

O nó mestre é responsável pelo monitoramento das falhas que possivelmente podem ocorrer e pelo direcionamento da carga de processamento, caso haja alguma indisponibilidade.

Os nós escravos estão restritos ao processamento das informações passadas a eles pelo nó mestre e a devolução dos resultados.

BIBLIOTECA DE COMUNICAÇÃO PARALELA

As bibliotecas de comunicação paralela são responsáveis pela comunicação entre os nós do *cluster*. Cada tipo de biblioteca de comunicação tem suas particularidades, ou seja, nelas são implementadas de maneira diferente as soluções para os problemas de comunicação paralela.

Os padrões mais difundidos de comunicação paralela são: PVM (*Parallel Virtual Machine*) e MPI (*Message Passing Interface*).

Segundo Jaquie (1999) o PVM é um conjunto integrado de bibliotecas e de ferramentas de *software*, cuja finalidade é emular um sistema heterogêneo.

O PVM cria um modelo de programação paralela composto de processos seqüenciais assíncronos cooperando através de troca de mensagens. (KRUG; TEODOROWITSCH, 1996)

O ambiente PVM é composto de três partes principais. A primeira parte é o *console*[3] que é usado para montar a máquina paralela virtual, por meio de uma máquina que estará disponível para o programador disparar os processos. A segunda parte é um *daemon* – um programa que roda em todos os nós que formam o *cluster*, responsável pelo controle das tarefas que estão sendo executadas nesses nós. A terceira parte é uma biblioteca das rotinas de interface, que contém um conjunto de primitivas que são necessárias para a cooperação entre tarefas de uma aplicação.

O MPI é uma biblioteca de rotinas que fornecem funcionalidades básicas para que os processos se comuniquem. Segundo Jaquie (1999), o MPI surgiu da necessidade de resolver alguns problemas relacionados a plataformas de portabilidades.

O MPI, diferentemente do PVM, não possui uma máquina virtual paralela. Está centrado no padrão de comunicação de dados para a comunicação paralela por troca de mensagens entre os processos. O MPI oferece diversas rotinas de suporte que ajudam a resolver, ou estruturam melhor a solução para o problema. As rotinas básicas mais conhecidas são definidas a seguir.

- *Rank*: identificação única crescente atribuída pelo sistema, começando de zero até N-1, no qual N é o número de processos.
- *Group*: é o conjunto ordenado de processos, podendo este ser o todo ou parte dele.
- *Communicator*: é o conjunto dos grupos. Permite ao usuário definir módulos encapsulados que fornecem estruturas de comunicação interna.

CONFIGURAÇÃO DO *CLUSTER* IMPLEMENTADO

Partindo do pressuposto que o sistema operacional Linux já esteja instalado, as demais configurações necessárias podem ser divididas em três partes: redes, comunicação SSH (*Secure Shell*) e MPI.

No quadro 1 são apresentadas as configurações de rede utilizadas em cada um dos nós do *cluster*. No quadro 2 está descrita a edição necessária do arquivo /etc/hosts para estabelecer a identificação das máquinas utilizadas.

[3] *console:* interface de comunicação que permite ao ser humano a interação com uma máquina.

HOSTNAME	IP	NETMASK	GATEWAY
Host1.cluster (Server)	192.168.1.1	255.255.255.0	192.168.1.1
Host2.cluster (Client)	192.168.1.2	255.255.255.0	192.168.1.1
Host3.cluster (Client)	192.168.1.3	255.255.255.0	192.168.1.1
Host4.cluster (Client)	192.168.1.4	255.255.255.0	192.168.1.1

Quadro 1 - Configuração de rede das máquinas no *cluster*.

```
#for loopback
127.0.0.1          localhost
#machines
192.168.0.1        host1.cluster    host1
192.168.0.2        host2.cluster    host2
192.168.0.3        host3.cluster    host3
192.168.0.4        host4.cluster    host4
```

Quadro 2 - Arquivo com as configurações de rede

Após realizar as configurações iniciais, utiliza-se o protocolo de comunicação seguro e criptografado SSH para que as máquinas se comuniquem. Porém é necessário configurá-lo para não haver a necessidade de senha. Para isso, deve-se aplicar o comando apresentado no quadro 3 nos nós escravos.

```
ssh-keygen -t rsa
```

Quadro 3 - Comando para geração de chave pública

Uma chave pública é gerada após o comando da Figura 3 e é necessário concatená-la com a chave do servidor do *cluster* (quadro 4).

```
scp /root/.ssh/id_rsa.pub root@servidor:/root/
(Digitar senha)
ssh host2
cat /root/id_rsa.pub >> /root/.ssh/authorized_keys
```

Quadro 4 - Comandos para concatenar as chaves

Esses passos devem ser seguidos para todos os nós pertencentes ao *cluster*. Feito isso, a comunicação sem senha está pronta.

O próximo passo é a instalação da biblioteca MPI. Foi utilizada uma extensão do MPI, mpich-1.2.7p1, cuja configuração é feita ao executar os comandos apresentados no quadro 5.

```
cd mpich-1.2.7p1
./configure —prefix=/usr/local -rsh=ssh
make
make install
```

Quadro 5 - Comandos para configuração e instalação do *Mpich*

A configuração do SSH sem senha e a instalação do *Mpich* devem ser executadas em todas as máquinas.

APLICAÇÃO TESTADA

O paradigma escolhido para ser aplicado neste trabalho foi SPMD (*Single Program Multiple Data)* com enfoque em um problema clássico - paralelismo de dados para a solução da multiplicação de matriz quadrática, o qual necessita de alto poder computacional.

A estratégia utilizada para a divisão das tarefas consiste no modelo mestre-escravo, no qual o nó mestre recebe as informações, divide as tarefas entre os integrantes do *cluster* de forma homogênea, as envia por meio de troca de mensagem MPI e fica aguardando a resposta de cada nó para agrupar as informações e concluir o processo de execução. Na Figura 2 é apresentado esse cenário.

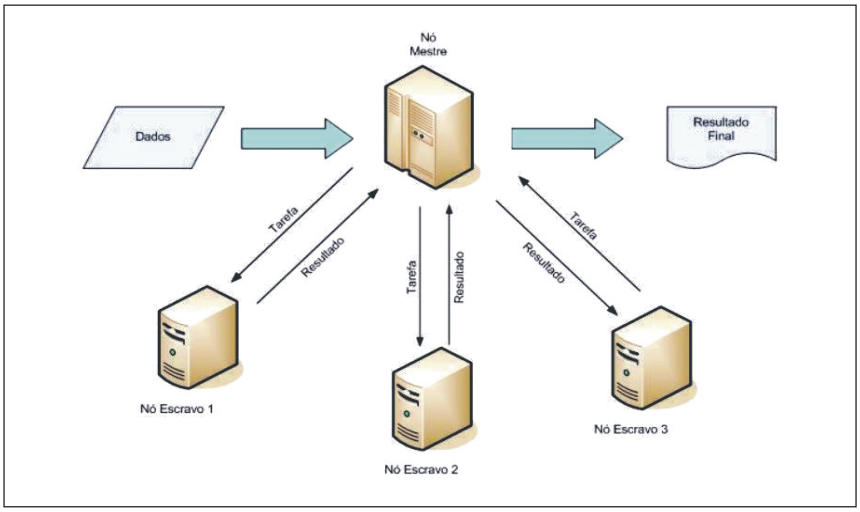

Fig. 2 - Comunicação por troca de mensagem

A aplicação implementada foi programada em C e é baseada no paralelismo explícito de dados, no qual o programador é responsável pelo controle do algoritmo paralelo aplicado. Foi utilizada, como modelo de comunicação, a troca de mensagens da biblioteca MPI para linguagem C.

O processo paralelo tem início no nó mestre que recebe as informações e inicia o processo paralelo. Essa atividade consiste em realizar o *start* do programa em todas as máquinas e, é nesse momento que cada processo recebe sua identificação única dentro do grupo de comunicação que também é criado nesse instante. O grupo de comunicação contém a identificação de todos os processos que foram criados.

Com a conclusão dessa atividade, o processo mestre cria e inicializa as matrizes quadráticas A, B com dados aleatórios do tipo *double* e cria a matriz C. Em seguida, realiza a distribuição das matrizes, em dimensão homogênea, entre os nós escravos e aguarda a resposta.

Por sua vez, o processo escravo aguarda o recebimento dos dados do processo mestre. Cada processo escravo recebe a matriz B e a cada troca de informação uma das colunas da matriz A. Com essas informações, cada processo escravo realiza o cálculo de multiplicação de matriz quadrática e devolve os resultados para ao mestre e finaliza o processo paralelo encerrando a comunicação.

O processo mestre agrupa todos os resultados na matriz C e também encerra o processo paralelo. No quadro 6 é apresentado o pseudocódigo do algoritmo proposto para o modelo mestre-escravo.

```
INICIO
        inicia_processo_paralelo
        Processo_mestre
                cria_iniciliza_matriz(A)
                cria_iniciliza_matriz(B)
                cria_matriz(C)
                distribui_matriz_entre_escravos(A)
                envia_matriz(B)
                recebe_resultado_escravos(C)
        Fim Processo_mestre

        Processo_escravo
                recebe_fragmento_matriz(A)
                recebe_matriz(B)
                realizar_calculo
                envia_resultados_mestre
        Fim Processo_escravo
        Finaliza_processo_paralelo
FIM
```

Hilario Viana Bacellar / Matheus de Paula França / Rafael Roberto / Edgar Noda / Jeanne Dobgenski

Quadro 6 - Pseudocódigo paralelo

RESULTADOS

Os testes realizados consistiram em 10 execuções para cada dimensão N da matriz quadrada avaliada. As matrizes foram geradas aleatoriamente utilizando a função *Rand()* com o tamanho entre [0, 10000] e também foi utilizada uma semente por tempo, assim evitou-se que os valores das matrizes fossem iguais.

No quadro 7 são descritas as configurações dos testes realizados. No quadro 8 são apresentados os tempos médios dos testes e o desvio padrão obtido na variação das 10 execuções para as dimensões de N igual a 1000, 3000 e 5000.

Configuração 1	1 Sempron 3000
Configuração 2	1 Sempron 3000 e 1 Duron 950
Configuração 3	1 Sempron 3000 e 3 Duron 950

Quadro 7 - Configuração dos testes

Tipo de dados: *double*		Configuração 1 tempo (s)	Configuração 2 tempo (s)	Configuração 3 tempo (s)
N = 1000	Média	23,032	44,374	24,097
	Desvio Padrão	0,014	0,015	0,009
N = 3000	Média	2434,157	4568,150	2273,433
	Desvio Padrão	6,868	7,114	6,570
N = 5000	Média	13590,481	22306,736	11024,401
	Desvio Padrão	10,198	12,055	9,406

Quadro 8 - Tempo de execução dos testes

As análises foram realizadas considerando as dimensões das matrizes e as configurações testadas.

MATRIZ N = 1000

Neste teste, a configuração 1 se mostrou mais eficaz, por se tratar de uma amostra pequena de dados. Com relação à configuração 2, ao realizar a paralelização dos dados verificou-se que poder computacional das máquinas influencia nos resultados, pois o tempo total de processamento ficou limitado ao fim da execução da tarefa no Duron 950. Para a configuração 3, o *cluster* teve desempenho similar ao processo seqüencial na configuração 1.

Foi observado que a configuração 2 teve grande aumento no tempo total de execução. Isso ocorreu devido às diferenças entre as máquinas e a necessidade de troca de informações pela rede, causando um atraso.

MATRIZ N = 3000

Neste teste, a configuração 3 teve tempo de execução menor, devido ao fato de que uma matriz dessa ordem apresenta a necessidade de um poder computacional maior, que é atendido com o processamento paralelo. Por sua vez a configuração 1, por processo seqüencial, não consegue suprir essa necessidade de processamento. Foi observada grande ociosidade do *Server* (Sempron 3000) durante a execução da configuração 2, causada pela demasiada demora de conclusão da tarefa no Duron 950.

Esse problema poderá ser resolvido com a otimização do algoritmo proposto. Uma alternativa seria adaptar o algoritmo para balancear a carga de processamento entre o Sempron 3000 e o Duron 950 baseado na capacidade de processamento de cada máquina.

MATRIZ N = 5000

Neste teste, fica evidente que na configuração 2 o tempo total de execução é limitado ao fim da tarefa no Duron 950, pois o poder de processamento do Duron é muito inferior ao do Sempron 3000. O tempo gasto com as trocas de mensagens para paralelizar a execução mais o tempo necessário para o Duron 950 concluir o processamento, aumentaram o tempo total da execução.

A configuração 3 se mostrou aproximadamente 18,75% mais rápida que a configuração 1 e aproximadamente 50,50% mais rápida que a configuração 2. Esses dados comprovam que o *cluster* da configuração 3 para o problema de multiplicação de matriz quadrática, se torna viável para os casos testados aproximadamente a partir de N igual a 3000. É a partir desta dimensão que a configuração 3 começa a ter tempo total de execução menor que os tempo obtidos pelas configurações 1 e 2. O gráfico 1 ilustra a disposição dos tempos apresentados no quadro 8.