Análise de Desempenho de Programas Paralelos em Redes de Workstations

Eduardo F. Loures* Guido E. P. Silva Junior Virgílio Almeida†
Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
30161 Belo Horizonte, Brasil
e-mail: virgilio@dcc.ufmg.br

Resumo

Cada vez mais tem crescido o número de aplicações que requer computadores de alto desempenho. Tradicionalmente, essas aplicações são processadas por supercomputadores. Uma alternativa atraente do ponto de vista de custo/desempenho, é o uso simultâneo de várias *workstations*. Este trabalho analisa o desempenho de programas paralelos em execução em uma rede de *workstations*. São identificadas condições onde as execuções paralelas superam as sequenciais. Com o objetivo de aumentar o desempenho do processamento paralelo, o artigo propõe e analisa o uso de *workstations* de diferentes velocidades para minimizar os efeitos negativos das frações sequenciais e da comunicação existentes em um programa paralelo.

Abstract

The number of applications that demand high performance computing systems has increased in the last years. Traditionally, supercomputers have been used to run large applications. An attractive alternative to run those numerically intensive applications seems to be the use of a network of workstations. This paper analyzes the performance of parallel programs executing on a network of workstations. The paper also identifies conditions where the parallel processing outperforms the sequential execution. With the goal of increasing the performance of parallel processing, the paper proposes and analyzes the use of heterogeneous workstations to overcome the bottlenecks imposed by sequential fraction and communication existing in a parallel program.

*Parcialmente apoiado pelo CNPq
†Parcialmente apoiado pela FAPEMIG (contrato TEC-1113/90)

1 Introdução

Cada vez mais tem crescido o número de aplicações que requer computadores de alto desempenho para suas soluções. Essas aplicações envolvem áreas como química, engenharia, biotecnologia, indústrias de petróleo, aeronáutica, aplicações de tempo-real e outras.

A solução para esses problemas numericamente intensivos tem sido tradicionalmente o uso de supercomputadores. Uma solução alternativa, que apresenta uma relação custo/desempenho mais favorável pode ser o uso simultâneo de várias *workstations* para executar a solução de um mesmo problema.

Este artigo analisa o desempenho de aplicações paralelas que executam em uma rede de *workstations*. Ao analisar a execução, busca-se identificar os gargalos que limitam o desempenho de programas paralelos. Além disso, são também apresentados resultados experimentais da utilização de configurações heterogêneas de *workstations*. O uso de processadores de diferentes velocidades formando uma arquitetura paralela heterogênea tem-se mostrado uma forma efetiva para minimizar os gargalos impostos pela fração seqüencial dos algoritmos paralelos.

Inicialmente, o artigo apresenta o ambiente computacional, *hardware* e *software*, usado para implementar e testar o processamento paralelo. A seção 3 descreve as aplicações paralelas analisadas nos experimentos. Os resultados numéricos obtidos estão na seção 4. Finalmente, a seção 5 apresenta as conclusões obtidas com a execução de programas paralelos em redes de *workstations*.

2 Ambiente de Processamento Paralelo

2.1 Ambiente do DCC-UFMG

Basicamente o ambiente computacional do DCC-UFMG é composto de uma variedade de *workstations* interconectadas por uma rede Ethernet de 10 Mbits. Os experimentos analisados usam um conjunto de 10 *workstations* SUN modelos SLC e SPARC-2 com desempenhos nominais de 17 e 29 em MIPS-SPEC e 2 e 4 em MFLOPS respectivamente, interligadas por uma rede local. Esse conjunto de *workstations* foi utilizado de uma maneira compartilhada com os demais usuários da rede.

2.2 Descrição do PVM

O PVM [Sund90] é um *software* que foi desenvolvido para ser utilizado em ambientes distribuídos. O desempenho das aplicações que o utilizam está ligado ao compromisso entre a habilidade de encontrar suficiente paralelismo na aplicação, procurando manter todos os processadores disponíveis ocupados e a granularidade dos processos paralelos obtidos.

Ao nível do usuário o PVM pode ser visto como uma biblioteca de funções que facilita as operações de criação, sincronização e escalonamento dos processos que fazem parte da aplicação paralela [Sund90]. Assim, o PVM integra várias *workstations* como se fossem um único recurso (figura 1) podendo o usuário se abstrair de problemas como, por exemplo, a comunicação entre máquinas que possuem representação de dados diferentes.

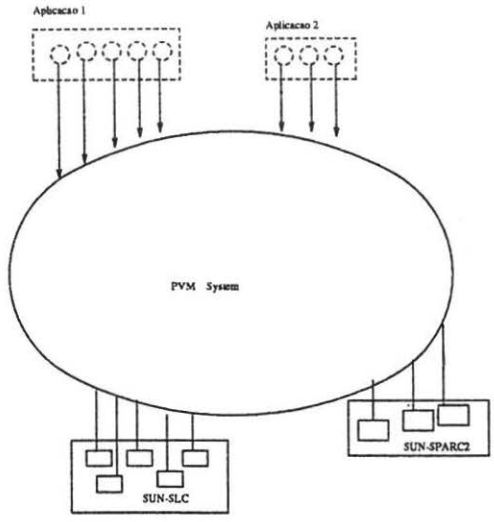

Figura 1: PVM: Parallel Virtual Machine

Uma característica importante do PVM é a variedade de arquiteturas que podem ser integradas para o processamento de uma mesma aplicação paralela. O PVM não coloca nenhuma restrição quanto a arquitetura dos processadores individuais que o utilizam. Este conjunto de máquinas, denominado de *Parallel Virtual Machine*, pode variar enormemente sendo que um nodo, ou seja, um elemento de processamento, pode ser desde uma *workstation* com um único processador até um hipercubo Intel iPSC/2 de 64 nodos.

A comunicação entre processos se dá através de troca de mensagens. O PVM não possui primitivas de compartilhamento de memória, uma vez que foi projetado para uma arquitetura com memória distribuída. Quando os processos que desejam se comunicar estão em máquinas distintas a troca de mensagens ocorre através de uma rede local. As máquinas individuais devem conter suficiente poder computacional e capacidade de memória para executar códigos da aplicação no tamanho desejado.

Os canais de comunicação implementam uma conexão lógica ponto a ponto entre dois processos. Os processos de uma aplicação paralela que necessitam se comunicar são conectados aos pares por uma rede de comunicação composta de canais de comunicação. Um processo possui tantos canais de comunicação quantos são os processos com os quais ele se comunica. Isso faz com que a troca de mensagens entre dois processos, com exceção da primeira, seja feita diretamente, sem o auxílio do núcleo do PVM. A primeira troca de mensagens é feita através do núcleo do PVM, pois é necessário que a partir daí seja criado um único canal de comunicação entre eles.

A eficiência de execução dos processos é obtida pela execução de operações ao nível do usuário, ao invés de executá-las ao nível do núcleo. Isso evita o *overhead* das chamadas ao núcleo, que pode ter um alto custo em relação ao trabalho total requerido [Hwan84]. Além disso, como as operações são executadas dentro do contexto de um determinado programa paralelo, muitas das operações de teste requeridas pelo núcleo de um sistema gerenciador

de processos paralelos são no PVM evitadas deixando os testes para serem realizados pelo usuário em seu programa de aplicação [Sund90].

Em resumo o ambiente do PVM apresenta as seguintes características convenientes para o processamento distribuído:

- Um ambiente de computação geral, flexível e concorrente para redes de máquinas heterogêneas.
- O usuário faz a interface com o PVM através de um conjunto de primitivas chamadas a partir de uma linguagem procedural. Estas primitivas permitem o gerenciamento dos processos, comunicação entre processos, sincronização, e outras funções.
- Provê um consistente e coerente paradigma de programação, no qual pode-se implementar aplicações paralelas. Sendo de fácil uso como extensão de linguagens de programação amplamente difundidas como "C" e "Fortran".

2.3 Como a Aplicação é Iniciada?

No PVM uma instância em execução de um programa denomina-se processo, sendo este identificado pelo nome do programa executável e por um número de instância positivo. Uma mensagem enviada por um processo deve identificar o processo destino, assim toda mensagem carrega consigo a identificação do processo que lhe deu origem.

O ambiente de execução da aplicação paralela se inicia ao ser executado o programa de gerenciamento chamado de "pvmd" (núcleo do PVM). Este primeiro programa dá início a execução dos demais programas "pvmd's" nas máquinas que irão formar o *Parallel Virtual Machine*.

Cada processo "pvmd" tem informações sobre a localização e situação de todos os processos que compõem a aplicação. Em determinado instante, para cada máquina hospedeira da rede, tem-se tantos "pvmd's" ativos quantos são os programas paralelos que associam aquela máquina à sua execução em paralelo. Esse ambiente permite que o "pvmd" possa gerenciar toda a comunicação entre processos do programa paralelo associado a ele.

É responsabilidade do usuário gerar os programas executáveis para cada uma das arquiteturas desejadas para a execução da aplicação. A biblioteca do PVM deve ter sido previamente gerada para todas as arquiteturas onde a aplicação irá executar.

A partir do instante em que os processos foram iniciados, eles passam a ser escalonados na mesma política adotada pelo sistema operacional (da máquina hospedeira) para os demais processos do sistema. É interessante observar que um processo ao utilizar o PVM pode entrar em estado de espera em dois níveis distintos. A nível do usuário o processo pode ser bloqueado ao executar alguma primitiva de sincronização do PVM. Por exemplo, o processo pode executar a primitiva *barrier* de sincronização com um conjunto de outros processos de um mesmo programa paralelo. Além disso, o processo pode ser bloqueado a nível do sistema operacional. Por exemplo, ao solicitar um I/O ou quando da ocorrência de um *page fault*. Em ambos os casos o processador fica disponível para a execução de um outro processo qualquer.

3 Aplicações Paralelas

Esta seção descreve duas aplicações paralelas usadas para avaliar a capacidade de processamento paralelo de uma rede de workstations. São aplicações que se caracterizam por requerer

uma grande quantidade de cálculos numéricos.

3.1 Cálculo dos harmônicos

O principal objetivo desta aplicação é o de testar o comportamento de um algoritmo do tipo mestre/escravo, em um ambiente de computação distribuído e interligado por uma rede local Ethernet, frente a configurações homogêneas e heterogêneas de *hardware*.

O algoritmo paralelo proposto abaixo é do tipo master-slave. Nessa estrutura o processo principal envia tarefas a serem realizadas por outros elementos de processamento (processos filhos), coleta os resultados e toma decisões dos próximos passos a seguir. A alocação de tarefas e o balanceamento do sistema está sob controle do programa. A contenção é reduzida nesse algoritmo, pois toda a troca de mensagens ocorre entre o processo principal e os processos filhos.

O programa paralelo proposto faz a transformação gradual de uma onda através da soma de harmônicos utilizando um ambiente de computação distribuído. Para esse tipo de ambiente, o custo de acessar dados que não estão residentes na memória local causam um sério impacto no desempenho global do algoritmo[Hwan84]. Dessa maneira, evitamos a utilização de qualquer mecanismo de memória compartilhada na implementação da aplicação. Cada mensagem trocada entre processos da aplicação tem o tamanho de 1440 bytes. O tempo de execução de um processo filho para cada mensagem trocada com o processo principal, quando executado em uma SUN-SLC e uma SUN Sparc 2, são de 34,23 segs. e 11,25 segs. respectivamente. O tempo de preparação e envio de uma mensagem da aplicação é de 14785 μseg. Esta aplicação foi implementada mantendo-se os processos filhos com alta granularidade, às custas de um sacrifício no seu grau de paralelismo.

A descrição do algoritmo paralelo é:

```
PROCESSO PRINCIPAL:

        cria os processos filhos;
        enquanto "houver processos filhos livres   e
                  "houver tarefas a serem realizadas" faca
             prepara pedido de calculo de uma tarefa;
             envia pedido para um processo filho livre;
        fim enquanto;

        enquanto "houver processos filhos tratando pedidos" faca
             recebe resultados de um dos processos filho;
             se "ainda houver tarefas a serem realizadas"
                 prepara pedido de calculo de uma tarefa;
                 envia pedido ao processo filho que trouxe
                 os ultimos resultados;
             fimse;
        fim enquanto;

        ------------------------------------
```

```
PROCESSO FILHO:

recebe pedido de calculo do processo principal;
trata o pedido do processo principal;
envia os resultados obtidos ao processo principal;
```

O paralelismo da aplicação proposta pode ser descrita por um grafo, onde os nodos do grafo representam os processos ou tarefas e os arcos representam a dependência de dados e/ou processos. Esta aplicação contém uma única fase paralela de execução. Dentro da fase um mesmo processo filho pode executar mais de um pedido do processo principal. Portanto, o ciclo de processamento consiste de uma fase seqüencial seguida por uma paralela. O ciclo é repetido diversas vezes.

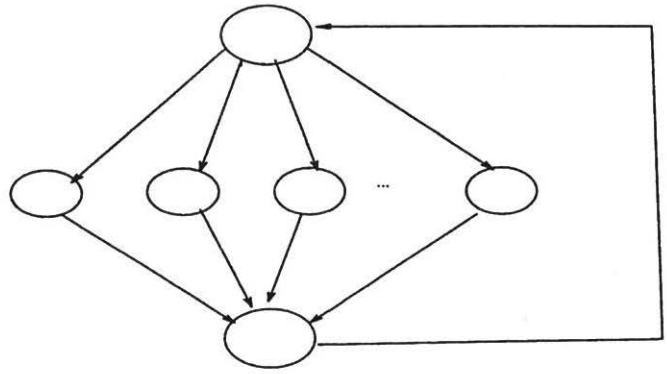

Figura 2: Grafo de Tarefa do tipo Mestre-Escravo

3.2 Problema da equação de Bessel

A equação de Bessel dada por:

$$x^2y'' + xy' + (x^2 - n^2)y = 0 \tag{1}$$

é uma equação diferencial onde n é uma constante e x uma variável independente que aparece em diversas aplicações matemáticas e científicas envolvendo simetria circular.

Para se conseguir a solução dessa equação diferencial não linear tem-se a seguinte expressão :

$$y = J_n(x) \tag{2}$$

onde J é a função de Bessel de ordem n do primeiro tipo.

Para valores de x menores do que aproximadamente 15 as funções de Bessel podem ser calculadas através da série:

$$J_n(x) = \sum_{k=0}^{n} \frac{-1^k}{k!\Gamma_i(n+k+1)}(x/2)^{n+2k} \tag{3}$$

onde

$$\Gamma_i(x) = \sqrt{\frac{2\pi}{x}}x^x e^y \tag{4}$$

$$y = \frac{1}{12x} - \frac{1}{360x^3} - x \tag{5}$$

Pode-se observar acima que o cálculo para cada ponto de uma função de Bessel de ordem n envolve muitas operações de ponto flutuante que é especialmente importante para representar aplicações numericamente intensivas.

Basicamente a aplicação paraleliza o cálculo de cada função de Bessel de ordem n. Sendo que essa paralelização vai seguir um grafo de tarefas em forma de uma árvore binária completa (figura 3).

Em particular não existe comunicação entre processos filhos, apenas entre o pai e os fillhos e entre os nodos-folhas e o programa principal que criou a raiz (figura 3), portanto a aplicação como um todo possui baixa comunicação. O algoritmo é descrito a seguir.

```
PROCESSO PRINCIPAL:

leia o numero total de processos;
registra a hora inicial da aplicacao;
cria processo nodo raiz;
envia nivel 0 e ordem 0 para o nodo raiz;
calcula o numero de nodos folhas;
enquanto "todos os nodos-folha nao sinalizaram que acabaram" faca
      aguarde chegar mensagem do nodo folha;
      incrementa o numero de mensagem recebidas dos nodos-folha;
fim enquanto;

registra a hora final da aplicacao;
```

```
PROCESSO FILHO:

 recebe a ordem da funcao de Bessel que deve ser calculada;
 recebe o nivel na arvore que o seu pai esta;
 recebe o numero total de processos; 10

 calcula a funcao de Bessel de ordem n;
 se   "o numero total de processos ainda nao acabou"
      incrementa o nivel da arvore que foi recebido;
      cria mais dois PROCESSOS FILHOS;
```

```
        envia para esse dois processos :
          qual funcao de Bessel devem calcular
          o numero total de processos
          passa o nivel da arvore que se encontra
    senao
        manda mensagem de aviso para o PROCESSO PRINCIPAL;
        termina;
```

Inicialmente o algoritmo lê o número de processos total da aplicação, ou seja, quantas funções de Bessel devem ser calculadas. 10

A partir daí o PROCESSO PRINCIPAL calcula quantos nodos-folha vão existir na base da árvore, cria o nodo raiz (que por sua vez irá criar os outros nodos da árvore) e fica aguardando enquanto todos os nodos-folhas não enviarem a mensagem que acabaram o seu processamento.

Cada PROCESSO FILHO calcula por sua vez uma função de Bessel de ordem n que lhe é passada como parâmetro. A seguir determina quantas funções de Bessel foram anteriormente calculadas e quantas estão sendo calculadas simultaneamente no seu nível dentro da árvore binária (figura 3). Se o somatório for menor que o número total de processos, cria mais dois processos-filhos e passa qual a ordem n da função de Bessel que terão que calcular, do contrário envia uma mensagem ao PROCESSO PRINCIPAL sinalizando que acabou seu processamento.

Assim, basicamente a comunicação entre os processos se resume a transmissão de três inteiros, sendo portanto uma aplicação de baixa comunicação.

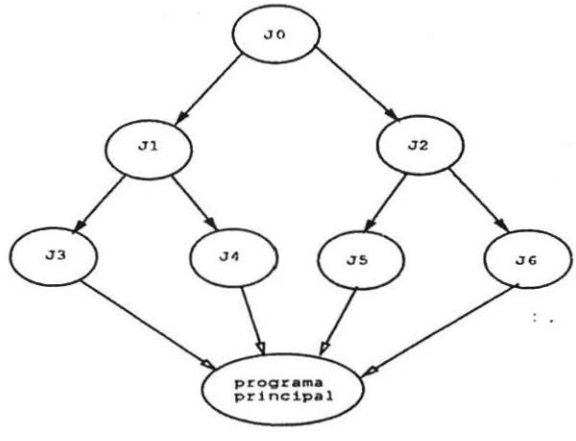

Figura 3: Grafo de tarefa em forma de árvore binária

4 Interpretação dos Resultados

Os experimentos de execução consideram duas classes de configurações de *workstations*. A primeira delas chamada homogênea consiste de n *workstations* idênticas, no que se refere à velocidade computacional. Especificamente nessa configuração são usadas *workstations* SUN SLC. A outra classe denominada heterogênea consiste do uso de $n - p$ *workstations* idênticas e uma *workstation* mais rápida. Menascè & Almeida [MeAl90] mostram que o uso de processadores de diferentes velocidades em uma arquitetura paralela contribuem para diminuir os efeitos da fração seqüencial dos algoritmos e aumentar o *speedup* das aplicações paralelas.

Busca-se analisar configurações homogêneas e heterogêneas que tenham a mesma capacidade total de computação, representada pela quantidade total de MIPS da configuração. No caso dos experimentos, a configuração heterogênea compõe-se de $n-2$ *workstations* SUN-SLC e uma SUN SPARC-2.

Foram realizados diversos experimentos de execução das aplicações paralelas descritas na seção 3 na rede de *workstations* do DCC/UFMG. Esta seção analisa os resultados obtidos, procurando identificar os aspectos que limitam o desempenho das possíveis configurações de *workstations*.

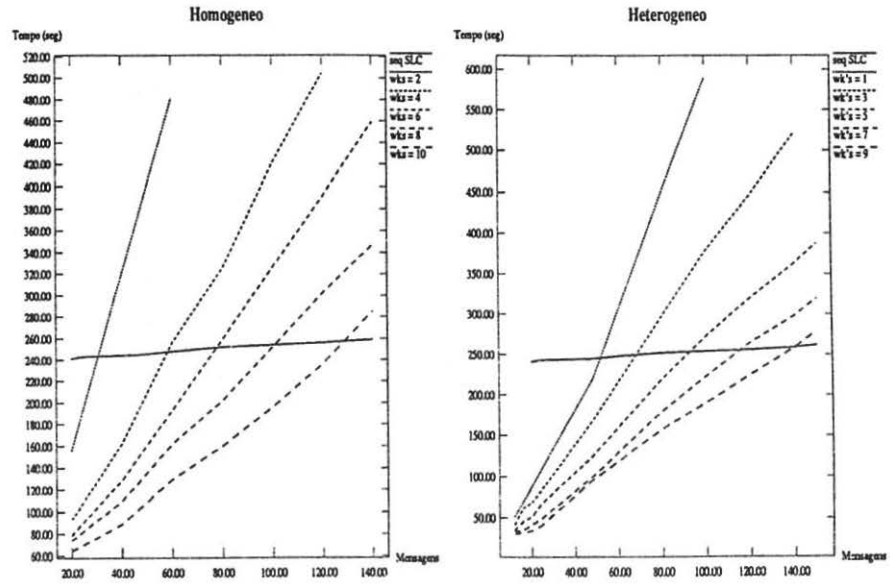

Figura 4: Tempo de Execução x Mensagens Trocadas

Os gráficos da figura 4, onde $n = \{2, 4, 6, 8, 10\}$ e $p = 2$, mostram o tempo de execução para se processar um determinado número de harmônicos em função do número de mensagens trocadas entre todos processos do programa paralelo para as configurações homogêneas e heterogêneas. Além disso os gráficos mostram também o tempo de execução do programa

seqüencial equivalente ao programa paralelo (linha contínua). Apesar de não existir troca de mensagens no programa seqüencial a equivalência foi obtida pela correspondência entre a quantidade de mensagens trocadas e o número de harmônicos calculados. Nestes gráficos, cada processo filho executa em uma *workstation* diferente, e o processo principal executa juntamente com um dos processos filhos.

Os gráficos da figura 4 mostram uma tendência de aumento no tempo de execução do programa paralelo conforme o número de mensagens trocadas entre os processos aumenta. Chega-se a um ponto em que o tempo de execução do programa seqüencial torna-se menor que o do programa paralelo. Isso se explica porque a partir desse ponto o custo de preparação e envio de mensagens do programa paralelo passa a ser significativo em relação ao trabalho computacional, tornando o programa seqüencial mais rápido.

A tabela abaixo mostra os tempos de execução do programa seqüencial em função do número de harmônicos calculados e para fins de comparação com os gráficos da figura 4 mostra também o número equivalente de mensagens trocadas no programa paralelo:

Tempo de execução (seg.)	n. harmônicos	mensagens trocadas
241	10	20
243	12	24
245	24	48
248	30	60
252	40	80
254	50	100
256	60	120
259	70	140

Figura 5: Tempo de execução

Considerando-se o conjunto de todas as curvas, observa-se que quanto menor o grau de paralelismo da aplicação, mais rápido vai ser atingido o ponto onde o *overhead* de comunicação passa a ser mais significativo.

Nos experimentos correspondentes aos resultados apresentados pela figura 4 na configuração heterogênea, o processo principal foi mantido juntamente com um dos processos filhos na máquina mais rápida. Apesar de se utilizar uma configuração com um menor número de máquinas, obtiveram-se melhores tempos de execução para a aplicação paralela na configuração heterogênea do que na configuração homogênea. Isso é explicado porque o processo filho que está na máquina mais rápida consegue executar um número maior de pedidos gerados pelo processo principal, diminuindo a troca de mensagens entre processos em máquinas diferentes. Com isso consegue-se uma redução no custo total de comunicação, resultando em melhores tempos de execução do programa paralelo em uma configuração com uma menor quantidade de *workstations*. Fica então claro, a partir da figura 4, que existe uma região, definida pelo custo de comunicação, onde a execução paralela de uma aplicação em uma rede de *workstations* sugere a execução seqüencial.

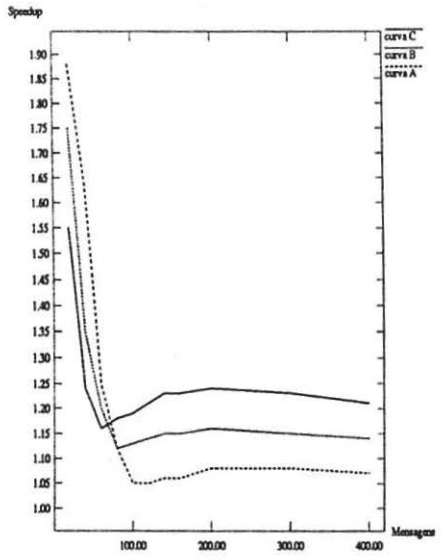

Figura 6: *Speedup* x Mensagens Trocadas

No gráfico da figura 6, o *speedup* representa a razão entre o tempo de execução de uma aplicação em uma configuração homogênea pelo tempo de execução da mesma aplicação utilizando uma configuração heterogênea. Assim, o *speedup* é fator de ganho devido a configuração heterogênea.

Uma configuração heterogênea corresponde à substituição de 2 *workstations* modelo SLC por uma *workstation* modelo SPARC-2. Assim as curvas A, B e C se referem respectivamente à substituição das configurações de 10, 8 e 6 *workstations* idênticas pelas suas correspondentes heterogêneas.

Para uma baixa troca de mensagens entre processos, o ganho de desempenho da configuração heterogênea em relação à configuração homogênea é proporcional a quantidade de máquinas em paralelo. Isso é explicado porque a aplicação paralela começa com uma maior quantidade de processos em paralelo, que ainda não sentiram os efeitos da troca de mensagens entre processsos em máquinas diferentes.

Seja as variáveis tempo de computação e tempo de comunicação denotadas por t_{cp} e t_{cm} respectivamente. Assim,

$$t_{cp} = \sum_{i=1}^{n} (t_{CPU})_i \tag{6}$$

onde, n é a quantidade de *workstations* que participam da aplicação paralela e, t_{CPU} 'e o tempo total de execução ("CPU Time") dos processos da aplicação na *workstation* i.

$$t_{cm} = \sum_{j=1}^{m} (t_{troca})_j \qquad (7)$$

onde, m é a quantidade de mensagens trocadas entre processos da aplicação que utilizam a rede local e, t_{troca} é o tempo de preparação e envio de cada mensagem da aplicação que transita pela rede.

Seja $\varphi = \frac{t_{cp}}{t_{cm}}$. Verificou-se que a razão φ manteve-se constante para várias configurações da rede, conforme tabela da figura 7.

Configuração	φx10^3
Homogênea 6,8 e 10	1.16
Heterogênea 5	1.43
Heterogênea 7	1.33
Heterogênea 9	1.28

Figura 7: Valores de φ

Observa-se que φ na configuração heterogênea decresce com o aumento da quantidade de *workstations* em paralelo. Por exemplo, a configuração heterogênea com 5 *workstations* é aquela que fornece a melhor relação de desempenho entre as demais configurações testadas.

Concluímos, então, que de acordo com quantidade de mensagens trocadas entre os processos da aplicação, uma menor quantidade de *workstations*, distribuídas de maneira que os processos seqüenciais executem na *workstation* mais rápida, pode levar a um maior *speedup*.

Quanto ao problema da equação de Bessel os resultados experimentais mostraram que o tempo de execução do programa em paralelo na configuração heterogênea foi mais lento que na configuração homogênea. O que de fato era previsto já que a razão φ nesse tipo de aplicação é alta, pois o tempo de comunicação (mensagens transferidas entre os processos) é muito menor que o tempo de computação.

Apesar da substituição da configuração homogênea pela heterogênea diminuir o tempo de comunicação entre os processos, esse ganho não foi significativo em relação à diminuição do tempo de computação, pois a razão φ é menor na configuração heterogênea. A diminuição do número de *workstations* diminuiu o paralelismo da aplicação, não trazendo ganhos no que se refere à comunicação e a parte seqüencial do algoritmo. Portanto, não houve ganhos com o uso da configuração heterogênea.

5 Conclusões

O artigo apresenta o desempenho de duas aplicações paralelas processadas por uma rede de *workstations* em um ambiente de execução provido pelo PVM. Dessa forma, foi verificado a viabilidade da alternativa de se executar aplicações numericamente intensivas em várias *workstations*. Os experimentos realizados apontam a necessidade da aplicação ser estruturada na forma de tarefas de grande granularidade (*coarse grain*) como uma maneira de se minimizar o custo da comunicação. O artigo também apresenta uma anállise do uso de configurações heterogêneas como forma de se diminuir os gargalos impostos pela parte seqüencial dos algoritmos e pela comunicação excessiva entre tarefas. O uso da configuração heterogênea se apresenta como um fator promissor para acelerar o desempenho da aplicação paralela.

Referências

[BDGM91] A. Beguelin, J. J. Dongarra, G. A. Geist, R. Manchek e V. S. Sunderan. *"A user's guide to PVM parallel virtual machine". Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, December, 1991.*

[BeSG91] Annamaria Benzoni, Vaidy S. Sunderam, Robert van de Geijn. *"Matrix Factorization on a RISC Workstation Network", High Performance Computing II,1991,p. 207.*

[FJLO88] G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon, D. Walker. *"Solving Problems on Concurrent Processors", Prentice-Hall Internacional Editions, 1988.*

[Hwan84] F. A. Briggs, K. Hwang. *"Computer Architecture and Parallel Processing", MacGraw-Hill, New York, 1984.*

[Mapl85] Creve Maples. *"Analyzing Software Performance in a Multiprocessor Environment", IEEE Software, 1985, p. 50.*

[MaZa89] C. MacCann, J. Zahorjan. *"Processor Scheduling in Shared Memory Multiprocessors". Technical Report 89,09-17 pp, Depto. of Computer Science and Engineering, University of Washington, September 1989.*

[MeAl90] Menascé, D. e Almeida, V., *"Cost-Performance Analysis of Heterogeneity in Supercomputer Architectures", Proc. ACM-IEEE Supercomputing'90 Conference, New York, November 1990.*

[Sund90] V. S. Sunderan. *"PVM: A framework for parallel distributed computing. Concurrency: Practice and Experience", 2(4):315-339 pp, December 1990.*