Análise de Desempenho de Ambientes de Software para Clusters HPC

Augusto M. Gomes Jr, Calebe de P. Biachini*, Francisco I. Massetto,
Hélio M. de Oliveira, Jean M. Laine, Mohamad M. El Saifi
Edson T. Midorikawa, Liria M. Sato
Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica da Universidade de São Paulo
Av. Prof. Luciano Gualberto, travessa 3, 158, 05508-900
Cidade Universitária, São Paulo, SP
{*augusto.mendes, francisco.massetto, helio.marci, jean.laine,*
mohamad.saifi, edson.midorikawa, liria.sato}*@poli.usp.br*
*calebe@netsite.com.br**

Resumo

Clusters HPC são atualmente a solução mais difundida para aplicações que exigem alto desempenho a baixo custo. Clusters Beowulf empregam software livre e encontram-se disseminados no mundo inteiro. Começam a surgir alternativas que utilizam software comercial e que oferecem desempenhos equivalentes.

As diversas alternativas de ambientes de software para clusters HPC desempenham um papel fundamental para a exploração adequada dos recursos do hardware disponível. Este trabalho procura analisar os recursos oferecidos e o desempenho obtido pela execução de programas de benchmark nos diferentes ambientes de software para uma mesma plataforma de hardware.

Os resultados obtidos mostraram que os vários ambientes analisados são apropriados para atender às necessidades de aplicações de alto desempenho.

1. Introdução

A arquitetura mais difundida atualmente para a obtenção de alto desempenho é a arquitetura de *clusters* HPC [3]. Um *cluster* HPC pode ser descrito como um conjunto de nós de processamento, interligados por um rede de comunicação, onde aplicações paralelas são executadas. Tanto a execução como a comunicação das aplicações paralelas são de responsabilidade de um pacote de *software* composto pelo sistema operacional e de uma biblioteca de comunicação (como o PVM ou MPI).

Os *clusters* Beowulf [18] são um exemplo de arquitetura de alto desempenho bem difundido. Eles se caracterizam pelo uso de *hardware* de prateleira, rede de comunicação interna de alta velocidade, suporte à programação paralela com pacotes adequados (como o PVM e o MPI), sistema operacional e outros *softwares* livres.

Várias instituições de pesquisa no mundo inteiro dispõem de *clusters* do tipo Beowulf. De uma maneira geral, sua configuração padrão é composta por nós de processamento baseados em máquinas do tipo PC com processadores Intel ou AMD; redes internas Fast Ethernet, Gigabit Ethernet ou proprietárias (Myrinet); sistema operacional Linux; bibliotecas de comunicação MPI de implementação livre (LAM-MPI, MPICH); linguagens de programação C, C++ e Fortran com compiladores GNU.

Contudo, alternativas a esta configuração estão sendo experimentadas por algumas instituições de pesquisa. Uma destas alternativas faz uso de *softwares* comerciais: sistema operacional proprietário, implementação MPI e compiladores comerciais. Um exemplo disto são os *clusters* configurados com o sistema operacional Microsoft Windows Server 2003 .Net, o MPI/Pro (implementação da MPI Software Technology) e compiladores Microsoft Visual .Net ou compiladores Intel.

Com o aumento das alternativas de *clusters* HPC torna-se necessária a definição de uma metodologia para avaliação e comparação de desempenho, de forma uniforme e independente de plataforma. Este artigo tem como objetivo analisar as alternativas de *software* de *cluster* HPC disponíveis. Através da execução de alguns programas de *benchmark*, analisa-se as diferenças de desempenho para uma mesma plataforma de *hardware*.

As seções seguintes são organizadas da seguinte forma: a seção 2 apresenta alternativas para avaliação de desempenho de *clusters* HPC e propõe uma forma de comparação

entre eles. A seção seguinte apresenta uma breve análise dos principais sistemas operacionais utilizados em *clusters*, a saber, Linux e Windows. A seção 4 descreve os ambientes de *software* analisados e as aplicações usadas neste trabalho. Os resultados obtidos são analisados a seguir e as conclusões são apresentadas na seção 6.

2. Avaliação de Clusters HPC

A avaliação de desempenho de *clusters* HPC normalmente é efetuada através do uso de programas de *benchmark* [1]. Várias categorias de testes de desempenho podem ser aplicadas para avaliar e comparar estes sistemas: os *benchmarks* de *hardware* ou de baixo nível e os *benchmarks* de aplicação ou paralelos. Os *benchmarks* de *hardware* ou de baixo nível verificam o comportamento e o desempenho de uma parte específica do sistema, como por exemplo, a capacidade de processamento, o funcionamento do sistema de memória ou de transferência da rede. Exemplos desta classe de *benchmarks* são o Lmbench [9], Stream [12] e Netperf [16]. Os *benchmarks* de aplicação ou paralelos são compostos por programas reais ou trechos de aplicações paralelas de diversas naturezas. Por exemplo, o NPB (*NAS Parallel Benchmarks*) [1] contém programas de dinâmica dos fluidos computacional e trechos de programas (*kernels*) para cálculos diversos, como equações de Poisson, autovalores de matrizes esparsas e FFT.

Diversos trabalhos têm sido publicados, cujos resultados procuram verificar o desempenho geral do sistema analisado. Estes trabalhos não realizam nenhuma análise específica sobre a influência dos pacotes de *software* utilizados no desempenho do *cluster*. É exatamente este aspecto que é trabalhado neste artigo: qual a influência no desempenho da adoção de um determinado ambiente de *software*.

Quando se deseja analisar o desempenho do *software* dentro de um sistema computacional, um fator muito importante é a escolha do sistema operacional, pois este é responsável pela gerência dos recursos de *hardware* e de *software*. A próxima seção apresenta as características dos dois principais sistemas operacionais empregados em *clusters* HPC: Linux e Windows.

Para se caracterizar o desempenho de ambientes de *software* de *clusters* HPC é muito importante observar uma condição: a plataforma de *hardware* deve ser a mesma, pois devemos manter as mesmas condições de execução para os sistemas avaliados.

Outro fator essencial é a disponibilidade do código fonte das aplicações utilizadas para a caracterização de desempenho. Como devem ser caracterizados o sistema operacional, o compilador e a biblioteca de comunicação MPI, os programas de *benchmark* devem ser compilados e executados no ambiente de teste em avaliação.

Para evitar interferências no processo de análise, os testes devem garantir um ambiente isolado de perturbações, como por exemplo, a execução de rotinas de gerência de sistema (*crontab* no Linux ou verificação de vírus no Windows), durante a execução dos testes. Um outro exemplo de perturbação é o acesso remoto de um outro usuário e execução de uma outra aplicação.

Como o objetivo deste trabalho é verificar a influência dos componentes de *software* de *clusters* HPC, a metodologia de avaliação tem as seguintes diretivas:

- Uso de programas de *benchmark* de aplicação;
- Adoção de uma mesma plataforma de *hardware* para os testes;
- Isolamento do ambiente de testes de perturbações externas.

As seções seguintes apresentam a aplicação da metodologia usada e os resultados obtidos na análise e comparação de desempenho de várias configurações de *software* de *clusters* HPC.

3. Análises dos Recursos dos SOs Utilizados

Esta seção apresenta alguns recursos existentes nos sistemas operacionais Linux e Windows que podem ser utilizados no desenvolvimento de aplicações paralelas e distribuídas. Tais recursos englobam criação e manipulação de processos e *threads*, intercomunicação, sincronização e como esses mecanismos podem ser utilizados para o desenvolvimento de aplicações paralelas e distribuídas.

3.1. Recursos oferecidos pelo Linux

Em Linux os recursos utilizados para a obtenção de alto desempenho, tanto em máquinas multiprocessadas como em *clusters* de computadores, referem-se ao gerenciamento, comunicação e sincronização entre processos e *threads*. Particularmente, as *threads* em Linux são implementadas através da biblioteca *pthreads* [11].

Na comunicação entre processos, pode-se utilizar recursos de memória compartilhada, passagem de mensagens através de bibliotecas (MPI [8]) ou RPC. Em termos de sincronização, pode-se utilizar semáforos e *mutexes*, onde ambos os recursos são necessários para sincronizar processos e *threads* em uma mesma máquina. As implementações de *mutex* são equivalentes às de semáforos, porém com a diferença que os *mutexes* são binários.

3.2. Recursos oferecidos pelo Windows

Da mesma forma que no Linux, o Windows também proporciona a criação de processos e *threads*, porém com um

elemento a mais, os *fibers* [13]. *Fibers* podem ser comparados às *threads*, com uma diferença básica, no que se refere ao escalonamento, pois os *fibers* são escalonados pela aplicação do usuário, enquanto que as *threads* são escalonadas pelo sistema operacional. Desse modo, é possível haver vários *fibers* associados a uma única *thread* que cuida de seu escalonamento.

Além dos recursos existentes no Linux, tais como passagens de mensagens (MPI [14, 15]), RPC, Pipes e memória compartilhada, há também outros mecanismos de comunicação entre processos. Dentre eles, podem ser citados Clipboard (área de transferência), COM, DataCopy (envio de dados entre aplicações com formato definido), DDE (protocolo para envio de mensagens em qualquer formato), MailSlots (comunicação unidirecional) e Windows Sockets.

Com relação à sincronização entre processos, também existem semáforos e seções críticas, além dos *mutexes* (semáforos binários), Waitable Timers (um objeto de sincronização é sinalizado quando determinado intervalo de tempo é passado), Interlocked Exchange (que permite travar um recurso e realizar atomicamente uma comparação e uma operação) e Events (eventos de sincronização).

4. Ambientes e Aplicações

As atividades de análise e avaliação de desempenho podem ser utilizadas com o objetivo de obter explicações para desempenhos ruins e o que deve ser feito para melhorá-los, bem como efetuar comparações entre sistemas de um mesmo propósito. No entanto, por causa de erros cometidos ao longo das atividades envolvidas neste processo, muitas vezes não alcançamos o resultado desejado inicialmente. Dentre estes erros, podemos citar [6, 17]: falta de um objetivo bem definido; falhas na definição de parâmetros, métricas e *workloads*; escolha equivocada da técnica de avaliação; falhas na análise dos resultados; entre outros.

Nesta seção, apresentamos a configuração do ambiente utilizado nos testes experimentais e uma breve descrição das aplicações analisadas.

4.1. Configuração dos Ambientes de Testes

Os resultados apresentados neste artigo foram obtidos sobre um *cluster* composto de 9 nós interconectados através de uma rede Fast-Ethernet e um switch Intel Express 510T. Cada nó possui 2 processadores Intel Xeon 2 GHz, 512 MBytes de memória e 40 GBytes de HD. Os testes realizados utilizaram esta arquitetura e uma combinação de sistemas operacionais e compiladores. Os pacotes de *software* utilizados em conjunto com a arquitetura descrita anteriormente denominamos de ambiente computacional. Assim, os vários ambientes testados estão listados na tabela 1.

Tabela 1. Ambientes de testes.

Ambiente	SO	Compilador	MPI
A	Win 2003 .NET Server	Visual C++ .NET	MPIPRO
B	Win 2003 .NET Server	Intel C++ 7.1	MPICH
C	Win 2003 .NET Server	Visual C++ .NET	MPICH
D	Win 2003 .NET Server	Intel C++ 7.1	MPIPRO
E	Win 2003 .NET Server	Intel Fortran 7.1	MPICH
F	Linux Red Hat 9.0	Gcc	LAM
G	Linux Red Hat 9.0	Intel C++ 7.1	LAM
H	Linux Red Hat 9.0	Intel Fortran 7.1	LAM

4.2. Aplicações Analisadas

Para análise dos ambientes de *software* adotou-se programas de *benchmark* de aplicação. Do pacote NPB selecionou-se os programas IS e SP porque ambos envolvem cálculos que demandam um grande poder computacional e também efetuam um volume significativo de comunicação entre os processos. Também utilizou-se uma implementação do algoritmo do caixeiro viajante (*TSP - Travelling Salesman Problem*) para a análise de desempenho. Este programa foi escolhido porque ele utiliza recursos de comunicação para os processos distribuídos entre os nós (MPI), e entre os processos de um mesmo nó (memória compartilhada).

4.2.1. IS e SP

O NPB (*NAS Parallel Benchmarcks*) [2] é um conjunto de *benchmarks* destinados a avaliação de desempenho de sistemas computacionais paralelos. Eles foram desenvolvidos pelo programa NAS (*Numerical Aerodinamics Simulation*) do Centro de Pesquisas Ames da NASA. Entre os programas que compõem o NPB encontramos 5 *kernels* paralelos (EP, MG, CG, FT e IS) e 3 simuladores de aplicações de dinâmica de fluidos (LU, BT e SP). Neste trabalho, em particular, os *benchmarcks* SP e IS foram utilizados para avaliar o desempenho das plataformas mencionadas na seção 1. A seguir apresentamos uma breve descrição dos programas escolhidos.

O IS é um *benchmark* que realiza a ordenação de N chaves (números inteiros) em paralelo. Uma característica desta aplicação é que as chaves são geradas por um algoritmo sequencial e distribuídas uniformemente pela memória. É importante ressaltar que esta distribuição inicial tem um grande impacto sobre o desempenho deste *benchmark*. Para ser executado, o IS requer que a quantidade de processos seja uma potência de 2.

Já o SP é uma aplicação cujo o objetivo é resolver 3 conjuntos de sistemas de equações, primeiro em x, depois em y e por último em z. Os sistemas de equações são escalares pentadiagonais, não diagonalmente dominantes. Para resolver estes sistemas o SP utiliza um esquema de multi-

partição. Neste esquema, cada processo é responsável por vários sub-blocos de pontos disjuntos da matriz. Os processos trocam informações durante a execução para solucionar os sistemas de equações. Este *benchmark* necessita de um número quadrado de processos para ser executado.

4.2.2. TSP

Em aplicações paralelas e distribuídas executadas em *clusters* de computadores, um aspecto importante a ser considerado é a existência de informações que influenciam no volume de processamento em cada nó do *cluster* e conseqüentemente no desempenho da sua execução. Em tais aplicações, o critério de parada na execução de uma tarefa depende de uma informação a qual, se for compartilhada entre suas diferentes instâncias em execução pelos nós do *cluster*, faz com que o resultado seja obtido com um volume menor de processamento em cada nó.

O problema do caixeiro viajante [5] (TSP - *The Travelling Salesman Problem*) possui tal característica, uma vez que o caminho mínimo corrente no momento torna-se uma condição de parada e que deve ser compartilhada entre os diversos processos distribuídos. Em casos onde um caminho parcial é maior que o mínimo, a execução da tarefa deve ser imediatamente interrompida. O compartilhamento das informações globais é obtido através da troca de mensagens entre os nós que estão em execução.

Uma aplicação que necessita de uma informação global que deve ser compartilhada por todos os nós do *cluster* deve prover uma cópia local dessa informação em cada nó. Toda alteração desta informação efetuada por um dos nós é propagada entre todos os demais nós do *cluster*. Devido a isso, a solução proposta baseia-se na existência de dois processos MPI em cada nó. Um responsável pela recepção da informação atualizada pelos demais nós restantes e outro responsável pela execução da tarefa. A cópia local da informação global deve ser acessada por ambos os processos (receptor e executor), cuja atualização somente é permitida através do uso de semáforos para acessar a região de memória compartilhada.

Nesta estratégia, um nó mestre é responsável pela criação de uma fila de tarefas e atribuição dessas tarefas aos nós escravos e a ele próprio, como pode ser ilustrado pela Figura 1. Um dos nós do *cluster* é eleito como mestre (1), cuja funcionalidade é criar uma fila (2) com as tarefas que serão executadas por todos os nós, inclusive ele próprio. Sempre que um nó escravo estiver ocioso, ele faz uma requisição de tarefas ao nó mestre (3). Esse nó retira da fila um pacote de tarefas (cujo tamanho pode ser previamente definido) e envia para serem executadas pelo processo executor do nó requisitante (4). O processo passa, então a executar a tarefa (5) e quando houver alguma alteração em uma informação global, ela é enviada pelo executor para os processos receptores dos demais nós (6). Os receptores recebem as informações (7) e atualizam suas cópias locais, armazenando-as em sua área compartilhada com o processo executor (8).

Tabela 2. Resultados dos testes com o IS.

p	NT	Tempo máximo	Tempo mínimo	Desvio padrão	Tempo médio
4	10	270,469	233,128	11,575	252,680
8	10	167,305	158,098	2,578	162,983
16	10	149,142	142,091	2,356	144,844
a) Ambiente D (-O3)					
4	20	269,560	237,311	9,073	251,422
8	20	170,577	158,405	2,911	162,245
16	20	148,765	143,007	1,552	144,753
b) Ambiente D (-O2)					
4	20	351,357	261,417	18,532	277,560
8	20	179,748	172,251	2,249	176,325
16	20	157,628	148,000	2,299	151,527
c) Ambiente A					
4	20	690,221	608,821	19,361	629,431
8	20	133,878	130,336	0,977	131,234
16	20	136,971	127,774	2,533	131,334
d) Ambiente G					
4	20	649,183	609,653	11,372	623,657
8	20	140,947	132,307	1,843	133,058
16	20	139,433	129,434	2,669	132,167
e) Ambiente F					

p = número de processos, NT = número de testes

Dessa forma, o algoritmo do caixeiro viajante utiliza alguns recursos essenciais para a obtenção do alto desempenho no *cluster*. São utilizados pares de processos (um executor e um receptor) em cada nó; para a comunicação entre os pares de processos, há uma variável compartilhada (simulando uma variável global) que é acessada através de um mecanismo de sincronização, neste caso um semáforo. Para a comunicação entre processos distribuídos, utiliza-se o mecanismo de troca de mensagens, através de MPI.

5. Análise dos Resultados

5.1. Testes realizados com o IS e o SP

Os resultados dos testes realizados com os *benchmarks* IS e SP podem ser verificados nas Tabelas 2 e 3. Além dos tempos máximos, mínimos e médios, as tabelas exibem as quantidades de processos (p), o número de execuções e o desvio padrão observado em cada caso. Nas execuções com o IS foram utilizados 4, 8 e 16 processos, enquanto que nos testes com o SP este número variou entre 1, 4, 9 e 16.

Durante as medições, cada nó de processamento do *cluster* recebeu apenas 1 processo, a não ser quando tínhamos $p = 16$ (nestes casos foram alocados dois processos por máquina). As configurações envolvidas nos testes incluíram os sistemas Windows e Linux Red-Hat, três implementações

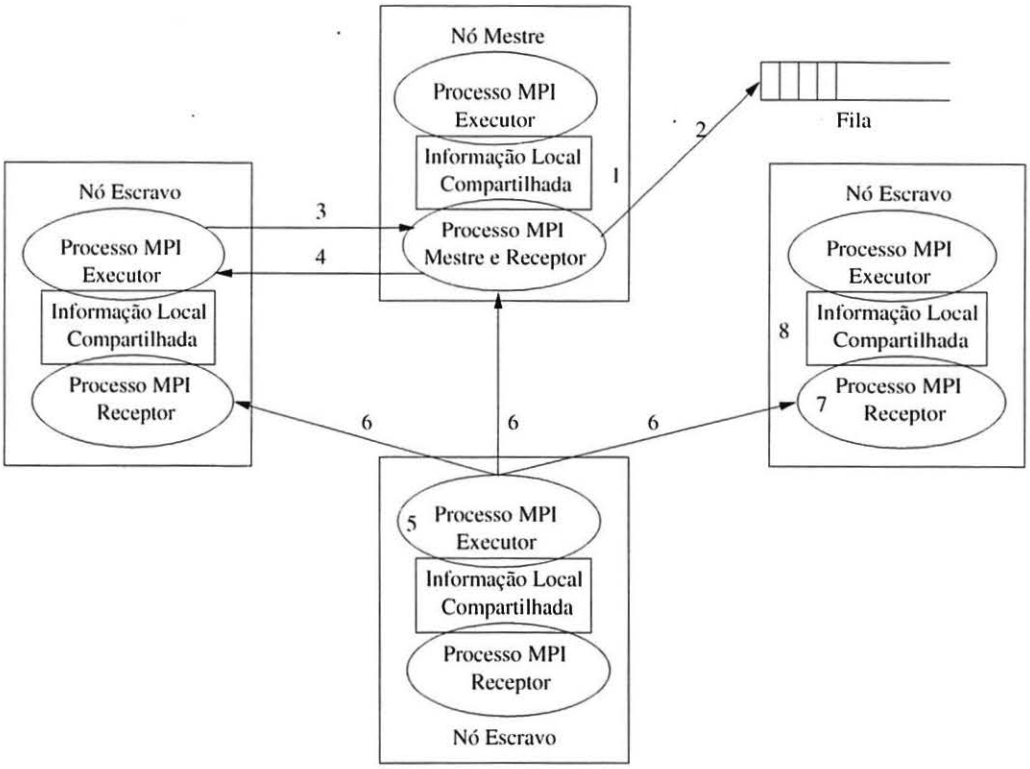


Figura 1. Arquitetura do problema do caixeiro viajante.

Tabela 3. Resultados dos testes com o SP.

p	NT	Tempo máximo	Tempo mínimo	Desvio padrão	Tempo médio
1	15	2198,625	2193,502	1,362	2195,035
4	15	703,773	700,211	0,956	702,337
9	15	623,030	612,635	3,043	618,464
16	15	725,956	713,926	3,028	720,979

a) Ambiente H

1	13	1784,524	1717,923	21,374	1758,687
4	15	1251,482	1067,779	67,241	1143,152
9	8	1472,128	1230,109	83,330	1366,768
16	8	1766,412	1658,364	42,122	1700,358

b) Ambiente E

p = número de processos, NT = número de testes

MPI (MPI-Pro, LAM-MPI e MPICH) e os compiladores Intel, Gcc e Visual C (VC). Em todos os casos, procuramos maximizar o nível de otimização durante a compilação dos programas (considerando o compilador Intel, a Tabela 2 exibe dados coletados com os níveis O2 e O3).

Embora os tempos medidos não tenham apresentado grandes variações, como podemos observar através dos desvios indicados nas Tabelas 2 e 3, procuramos selecionar aqueles valores capazes de expressar melhor o comportamento normal do sistema. Para o cálculo do tempo médio, aplicamos uma política de seleção sobre os dados coletados e determinamos a média entre os valores no intervalo de $m - dv$ e $m + dv$, sendo m e dv a média de todos os tempos medidos e o desvio padrão, respectivamente. Conforme é discutido em outros trabalhos de medição e análise de desempenho [6, 7, 17], em estudos envolvendo a coletagem de tempos de execução convém adotarmos certos cuidados (como, por exemplo, o tratamento dos dados), objetivando a obtenção de resultados significativos.

A Figura 2 ilustra o desempenho do *benchmark* NPB IS executado com a classe C. O comportamento das curvas obtidas em nossos testes foram semelhantes ao apresentado em [10], onde os autores simplesmente mostram como é o desempenho do IS ao ser executado sobre um Cray. Entre as três configurações utilizadas com o Windows e o MPIPro, observamos tempos menores para os testes com o compilador Intel. No Linux, os compiladores Intel e Gcc apresentaram resultados semelhantes. Os testes com o ambiente Windows e o MPI-Pro envolvendo 4 processos demonstraram tempos muito inferiores aos obtidos com o Linux e o LAM-MPI. Contudo, nos demais casos verificamos tempos mais próximos e os resultados coletados no Linux foram melhores.

Considerando os testes executados com o *benchmark* SP com a classe B, podemos analisar o comportamento da aplicação com auxílio da Figura 3. Assim como aconteceu com o IS, o comportamento do SP também ficou próximo do ob-

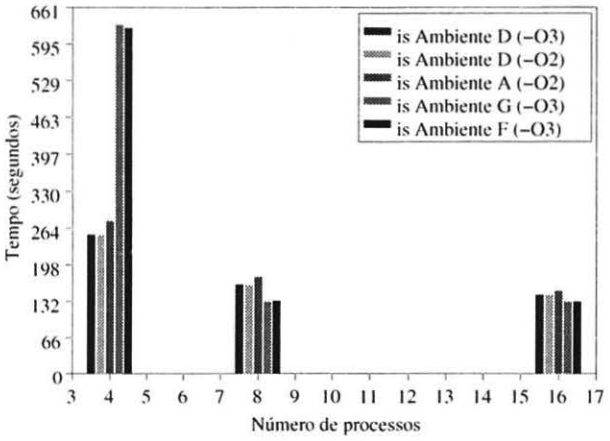


Figura 2. NPB IS com a classe C.

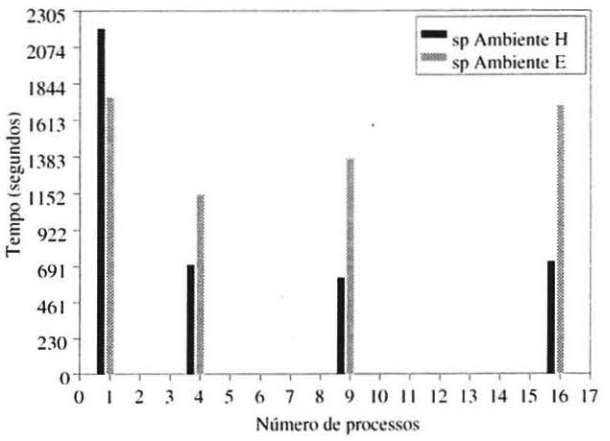


Figura 3. NPB SP com a classe B.

tido em [10]. De forma semelhante ao observado nos testes do IS, os tempos coletados demonstraram desempenhos melhores da configuração Linux-LAM para quantidades maiores de processos e piores quando apenas um processo foi utilizado.

5.2. TSP

Para cada teste do TSP, foram realizadas 7 medições de tempo. O valor médio usado como referência para a comparação representa a média aritmética dos tempos medidos.

O gráfico da Figura 4(a) apresenta os tempos de execução do algoritmo sequencial da aplicação para 18, 20, 21, 22 e 23 cidades. A legenda WseqVC representa o tempo de execução da aplicação utilizando o compilador Visual C++ .NET no sistema operacional Windows; WseqIntel representa o tempo de execução do algoritmo utilizando o compilador Intel no sistema operacional Windows; LseqGCC representa a execução utilizando o compilador GCC no sistema operacional Linux; e LseqIntel representa a execução utilizando o compilador Intel no sistema operacional Linux.

Através desse gráfico, pode-se perceber que os tempos de execução das aplicações WseqVC, WseqIntel e LseqIntel estiveram bastante próximos, porém a aplicação LseqGcc obteve um desempenho bastante inferior às demais. Este resultado já era esperado porque os compiladores da Intel foram desenvolvidos de forma a gerarem códigos eficientes para os seus processadores. O compilador utilizado pelo Visual C++.NET é uma versão do compilador da Intel com algumas alterações específicas para o ambiente Windows.

A Figura 4(b) apresenta os *speedups* dos algoritmos distribuídos das aplicações, utilizando a biblioteca de passagem de mensagens MPI-Pro, o compilador Microsoft Visual C++ .NET em sistema operacional Windows (Ambiente A). As legendas seguem o padrão de número de processos por nó seguida da quantidade de nós em execução. Por exemplo: 2P2N representa 2 processos (1 executor e 1 receptor) por nó, executando em 2 nós; 4P2N representa 4 processos (2 executores e 2 receptores) por nó, executando em 2 nós e assim por diante.

Nesta figura observa-se que quando há uma grande quantidade de processos executando (por exemplo 4 processos por nó executando em 8 nós), o *speedup* torna-se baixo devido à grande quantidade de mensagens trocadas. Isto torna-se acentuado quando o número de cidades é baixo, pois cada nó executa a sua tarefa muito rapidamente, tornando o tempo entre requisições de tarefas curto, e, com isso, aumentando o número de mensagens trafegadas em um pequeno intervalo de tempo.

Também foram realizados testes utilizando o MPICH e o compilador Intel. Netes casos, os resultados foram bastante próximos, confirmando a equivalência no desempenho entre eles. Uma grande diferença notada foi o tempo de inicialização dos processos utilizando MPI-Pro e MPICH. No MPI-Pro houve um atraso em torno de 30 segundos para estabelecer a comunicação remota e inicializar os processos em cada nó, enquanto que no MPICH a inicialização foi imediata.

A Figura 4(c) ilustra os *speedups* obtidos utilizando as mesmas quantidades de processos, porém com as aplicações utilizando a biblioteca de passagem de mensagens LAM-MPI, compilador Intel no sistema operacional Linux (Ambiente G). Pode-se notar que ocorre o mesmo fenômeno visualizado no gráfico da Figura 4(b), com algumas variações.

Os gráficos das Figuras 4(d) e 4(e) ilustram, respectivamente, os tempos de execução e *speedups* de diferentes versões obtidas através de diferentes compilações e alternando o número de processos e número de nós em execução. Nestes casos, a legenda WproVC indica a versão compilada utilizando MPI-Pro, Visual C++ .NET e executado em Windows (Ambiente A); WchVC ilustra o a versão utilizando MPICH, Visual C++ .NET e executando em Win-

dows (Ambiente C); e LlamIntel representa a versão utilizando LAM-MPI, Intel e executando em Linux (Ambiente G). As indicações 2P2N e as demais seguem o padrão de interpretação das figuras anteriores.

Analisando os gráficos, nota-se que os tempos de execução das versões WproVC e WchVC e LlamIntel foram bastante próximos, com ligeira vantagem para as duas primeiras. Com relação ao *speedup*, as três versões apresentam-se equivalentemente eficientes, com exceção da medição para 4 processos por nó executando em 8 nós (4P8N), onde a vantagem das duas primeiras foi significativa com relação à terceira.

6. Conclusões

Este trabalho realizou uma análise das alternativas de *softwares* para *clusters* HPC. Dentre os ambientes analisados, encontram-se tanto soluções com *softwares* livres, como com *softwares* comerciais. A avaliação constou da compilação e execução de alguns programas de *benchmark* de aplicação.

Os testes foram conduzidos utilizando-se a mesma plataforma de *hardware* para que esta análise permitisse uma comparação correta entre os ambientes. Cada alternativa considerada apresentou características próprias. Por exemplo, os compiladores comerciais (Microsoft Visual .Net e Intel) apresentaram um desempenho superior ao compilador gcc.

Os resultados obtidos mostraram que os ambientes analisados apresentaram desempenhos próximos e adequados para atender os requisitos de aplicações de alto desempenho.

Como trabalhos futuros pretende-se completar a análise com outros programas de aplicação. E como os ambientes mostraram desempenhos compatíveis, deverá ser iniciado um trabalho orientado ao desenvolvimento de ambientes de programação para plataformas com ambientes de *software* heterogêneos.

7. Agradecimentos

Os autores gostariam de agradecer a Intel do Brasil e a Microsoft do Brasil pela disponibilização das máquinas e *softwares* utilizados neste trabalho. Sem esta ajuda não seria possível a realização dos testes de desempenho apresentados.

Referências

[1] ANDERSSON, K.-J. et al. An evaluation of the system performance of a Beowulf cluster. Internal Report 2001:4. Uppsala University, Sweden, 2001.

[2] Bailey, D. et al. The NAS Parallel Benchmarks. RNR Technical Report RNR-94-007, March 1994.

[3] BUYYA, R. (ed.), High performance cluster computing: architectures and systems, vol 1. Prentice-Hall, Upper Saddle River, New Jersey, 1999.

[4] Cornell Theory Center. http://www.tc.cornell.edu/hpc/

[5] GOMES Jr, A. et al. A strategy for implementation of applications with global information-dependent execution for clusters of computers. The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications(PDPTA'03). Las Vegas, EUA, Junho 2003.

[6] JAIN, R. **The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling**. New York: John Wiley & Sons, 1991. 685p.

[7] LAINE, J. M. **Desenvolvimento de Modelos para Predição de Desempenho de Programas Paralelos MPI**. 2003. 100p. Dissertação (Mestrado), Departamento de Engenharia da Computação e Sistemas Digitais, Universidade de São Paulo.

[8] LAM Team. LAM/MPI Parallel Computing, MPI General Information, http://www.lam-mpi.org/mpi/

[9] Lmbench - tools for performance analysis. http://www.bitmover.com/lmbench/

[10] LUECKE, G.R. and LI, Y. **The Performance and Scalability of the NAS Parallel Benchmarks on a Cray SV1**. 2002.

[11] MATHEOS Jr, W. **Uma implementação da linguagem paralela CPAR usando modelo de programação threads**. 2002. Dissertação (Mestrado), Departamento de Engenharia da Computação e Sistemas Digitais, Universidade de São Paulo.

[12] McCALPIN, J. D. STREAM: Sustainable Memory Bandwidth in High Performance Computers. http://www.cs.virginia.edu/stream/

[13] Microsoft Development Network. http://msdn.microsoft.com

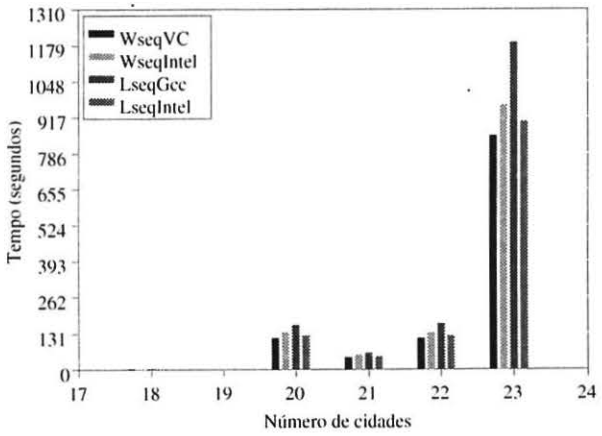
[14] MPI Software Technology. http://www.mpi-softtech.com/

[15] MPICH. A Portable MPI Implementation. http://www-unix.mcs.anl.gov/mpi/mpich/

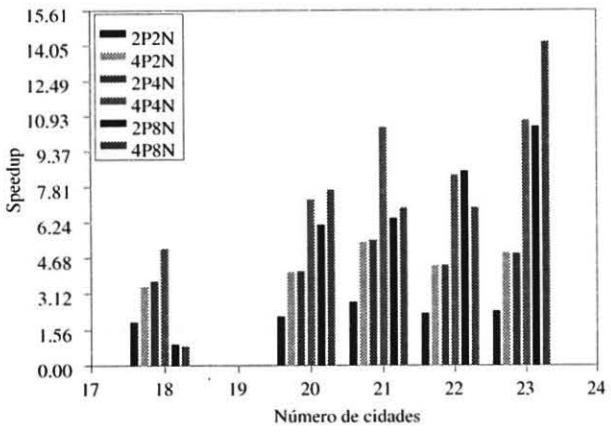
[16] Netperf Homepage http://www.netperf.org/netperf/NetperfPage.html

[17] OLIVEIRA, H. M. **Modelagem e Predição de Desempenho de Primitivas de Comunicação MPI**. 2003. 98p. Dissertação (Mestrado), Departamento de Engenharia da Computação e Sistemas Digitais, Universidade de São Paulo.

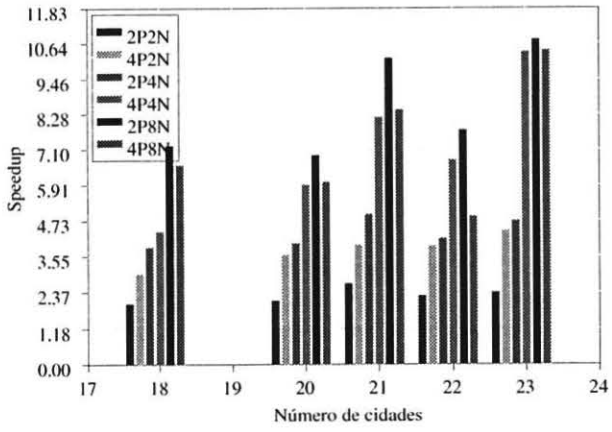
[18] STERLING, T. (ed.), Beowulf cluster computing with Windows, The MIT Press, Cambridge, Massachusetts, 2002.



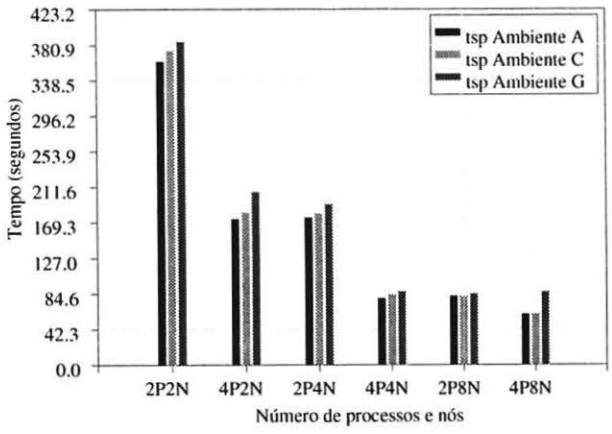
(a) Tempo das versões seqüenciais do TSP



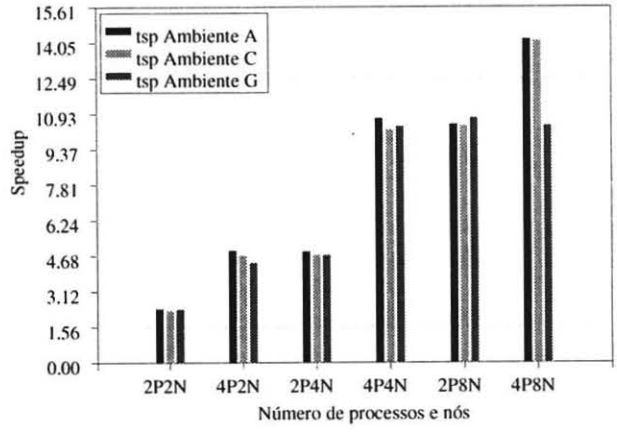
(b) *Speedup* do TSP para o ambiente A



(c) *Speedup* do TSP para o ambiente G



(d) Tempo do TSP para 23 cidades



(e) *Speedup* do TSP para 23 cidades

Figura 4. Resultados obtidos com o TSP