Arquitetura e Avaliação do Cluster de Alto Desempenho Netuno

Vinícius Silva[1], Cristiana Bentes[3], Sérgio Guedes[2], Gabriel P. Silva[1]
DCC-IM/UFRJ[1], NCE/UFRJ[2], DESC/UERJ[3]
vss@ufrj.br, cris@eng.uerj.br, guedes@nce.ufrj.br, gabriel@dcc.ufrj.br

Resumo

Este artigo apresenta a arquitetura e os resultados da avaliação de desempenho do supercomputador Netuno, um cluster *de alto desempenho recentemente instalado na UFRJ. São apresentados detalhes tanto de sua arquitetura como dos softwares básicos e de* middleware *utilizados na sua construção. Os resultados de avaliação obtidos registram um desempenho de 16,2 TFlops sustentados para o* benchmark *HPL (High Performance Linpack), o que colocou o supercomputador Netuno na 138ª posição na lista Top500 de junho de 2008. Atualmente, o supercomputador Netuno atende diversas instituições de ensino e pesquisa no Brasil, participantes das redes temáticas de pequisa de Geofísica Aplicada e de Oceanografia (REMO), patrocinadas pela Petrobras.*

1. Introdução

Na escolha e configuração de um *cluster* de alto desempenho há uma grande variedade de opções, incluindo-se configuração dos nós, redes de interconexão, sistemas de armazenamento, sistema operacional, compiladores, ferramentas de submissão e *softwares* de gerenciamento, entre outros. O desempenho final do *cluster* irá depender da escolha de um conjunto harmonioso de *hardware* e *software*.

O objetivo deste artigo é apresentar as alternativas e opções arquiteturais para o projeto de um sistema de computação de alto desempenho, tomando-se como referência o projeto do sistema Netuno.

O Netuno é um *cluster* de alto desempenho com 256 nós computacionais, 4 nós para acesso via protocolos *ssh* ou *http*, uma rede de interconexão de alto desempenho do tipo InfiniBand e um sistema de armazenamento paralelo com 30 TB.

Neste artigo abordamos também os detalhes de avaliação do Netuno, que apresentou um desempenho sustentado de 16,2 TFlops para o *benchmark* HPL (High Performance Linpack).

O restante deste artigo está organizado da seguinte maneira: na Seção 2 descrevemos as opções arquiteturais utilizadas; na seção seguinte, são feitas considerações a respeito do sistema operacional, *middlewares* e compiladores utilizados; a Seção 4 apresenta os detalhes do procedimento de avaliação de desempenho do *cluster* Netuno; finalmente, as conclusões e trabalhos futuros são relatados na Seção 6.

2. Opções Arquiteturais

Um *cluster* pode variar imensamente em termos de opções arquiteturais utilizadas: nós computacionais, redes de interconexão e sistemas de armazenamento são as principais delas. Nesta seção, apresentamos as configurações de *hardware* utilizadas neste projeto, com uma justificativa para sua escolha.

2.1. Nós Computacionais

Uma das primeiras escolhas realizadas no Netuno foi a configuração dos nós computacionais, incluindo-se aí quantos e quais processadores a serem utilizados.

2.1.1. Processadores

Na seleção dos processadores, os principais fatores a serem levados em conta são: o número de processadores por nó computacional, a quantidade de núcleos de cada processador, a freqüência de operação, o consumo, a dissipação e, obviamente, o custo.

Uma configuração típica de um *cluster* de alto desempenho pode atingir alguns milhares de núcleos facilmente. A primeira conseqüência disto é um grande impacto na energia consumida e no calor dissipado por todo o equipamento. Portanto, logo após o desempenho, a primeira preocupação em um *cluster* é com o consumo.

Os processadores *multicore* têm a grande vantagem de apresentarem uma relação de consumo versus desempenho muito melhor do que os processadores convencionais, e também um custo por núcleo bem menor. Logo, se estabeleceu como tendência nos *clusters* modernos o uso de um ou mais processadores *multicore* nos nós computacionais.

O projeto do Netuno não foi exceção à regra. Optamos pelo uso em cada nó do Netuno de dois processadores Intel E-5430, cada um com 4 núcleos de 64 bits, com freqüência de trabalho de 2,66 GHz, tecnologia de 45 nm, freqüência de barramento de 1,3 GHZ, memória cache de 12 MB. O desempenho de pico é estimado em 4

operações de ponto flutuante por ciclo de relógio, ou seja, 10,6 GFlops para cada núcleo ou 85,1 GFlops por nó.

O perfil de dissipação térmica do processador escolhido é de 80 W. Outras opções do mesmo fabricante ofereciam um desempenho melhor, mas à custa de um maior consumo e dissipação até 50% maior.

Devemos observar que os processadores modernos possuem formas de controlar dinamicamente a freqüência e a tensão de operação. Eles possuem sensores de temperatura em diversos pontos do encapsulamento. Se for notado um aumento excessivo da temperatura, a freqüência de operação e a tensão são ajustados para que a temperatura máxima do processador não seja ultrapassada. Assim sendo, o investimento adicional em processadores de maior velocidade (e maior consumo) pode ser perdido, caso a potência dissipada pelo sistema esteja acima da capacidade de refrigeração instalada.

2.1.2. Memória

O uso de processadores *multicores* oferece uma série de vantagens, mas exige cuidados, como por exemplo, na definição da quantidade de memória adequada para o funcionamento do nó computacional.

Cada núcleo deve ter uma quantidade de memória suficiente para executar as suas aplicações. Um maior número de núcleos, portanto, tem impacto direto na capacidade de memória a ser instalada em cada nó.

Na época de especificação do sistema Netuno, não havia ainda um conjunto definido de aplicações. Levantamentos realizados indicavam a regra de ouro de pelo menos 1 GB por cada núcleo. Valores menores poderiam resultar em operações excessivas de *swap* de memória virtual, com a conseqüente degradação de desempenho. Optamos, de um modo conservador, pelo uso de 2 GB por núcleo, ou seja, um total de 16 GB de memória principal para cada nó computacional.

O uso de módulos de memória com correção de erro ou paridade são condições também exigidas para equipamentos desse tipo. Utilizamos memórias de freqüência e padrão adequados aos processadores escolhidos: memórias FB-DIMM 667, do tipo *dual rank.*

2.1.3. Armazenamento e Interfaces de E/S

A melhor relação custo/benefício deve ser buscada na definição do tipo de disco a ser utilizado localmente. Em termos de quantidade, o espaço em disco no nó computacional deve ser apenas o suficiente para armazenar uma cópia do sistema operacional e para definir uma área temporária disponível para as aplicações.

Deste modo, cada nó computacional conta com um disco local de 160 GB, do tipo SATA, de baixo custo e desempenho adequado para o uso em aplicações paralelas.

Note que os arquivos de usuário, como aplicações, arquivos de configuração e dados de entrada e saída, devem estar em um sistema de arquivos remoto, acessível a todos os nós de uma maneira uniforme.

Para acesso a esse sistema de arquivos utilizamos uma rede do tipo *ethernet*, com uma interface de 1 Gbps para cada nó computacional, compartilhada entre os todos os processadores (e seus respectivos núcleos). O aumento do número de núcleos em cada nó, nesse caso, é de menor impacto nas demandas para acesso aos sistemas de arquivos remotos, pois apenas uma única cópia do sistema operacional, responsável pelos acessos de E/S, é executada em cada nó computacional.

Já em se tratando de largura de banda disponível para comunicação, o aumento do número de núcleos em cada nó requer um aumento correspondente na largura de banda. Não há como prever *a priori* os padrões de comunicação de cada aplicação, assim sendo, deve-se assegurar o máximo de largura de banda possível. Entre as diversas tecnologias disponíveis, optamos pela inclusão em cada nó de uma interface do tipo HCA para a rede *InfiniBand*, com taxa máxima de transferência de 20 Gbps. Esta interface se destina à comunicação entre processos, como aquela realizada pelas bibliotecas MPI.

Assim, com um total de 8 núcleos para cada nó, o Netuno tem uma largura de banda média de 2,5 Gbps disponível para cada um dos núcleos.

2.2. Redes de Interconexão

Como mencionado anteriormente há duas redes de interconexão no Netuno: uma para comunicação, via troca de mensagens, entre as aplicações em execução nos nós computacionais e outra para acessos de E/S das aplicações ao sistema de arquivos remoto e paralelo.

2.2.1. Rede para Troca de Mensagens

A rede para troca de mensagens entre os nós é importante para um bom desempenho das aplicações paralelas que executam no *cluster,* devendo-se considerar tanto a taxa de transferência máxima por canal, como também a latência, que têm impacto decisivo no tempo total de transmissão das mensagens.

A rede de conexão InfiniBand possui as características acima descritas. O padrão InfiniBand define uma conexão serial padrão (SDR) com 2,5 Gbps em cada sentido e suporta ainda velocidades DDR e QDR, respectivamente de 5 Gbps e 10 Gbps. Os canais usam uma codificação 8B/10B — cada 10 bits transmitidos enviam 8 bits de informação útil — de modo que a taxa de transmissão útil é 4/5 da taxa bruta.

Uma característica muito interessante do padrão InfiniBand é a possibilidade do uso de acesso direto à memória remoto (RDMA) para diminuir a sobrecarga do processador nas operações de envio e recepção de

mensagens, o que também possibilita latências ponto-a-ponto menores que 1 µs.

O uso de RDMA admite também o que é chamado de transferência zero-cópia, permitindo que o adaptador de rede transfira dados diretamente de/para a memória da aplicação (espaço de usuário), eliminando a necessidade de copiar dados entre a memória da aplicação e os *buffers* de dados dos sistema operacional. Como resultado, os dados da aplicação são enviados diretamente para a placa de rede, reduzindo a latência e permitindo uma transferência rápida das mensagens.

Se o padrão de comunicação da aplicação é conhecido, então é possível realizar algumas otimizações no projeto da topologia da rede. O Netuno, contudo, não se destina a um único tipo de aplicação e, portanto, deve prover a maior flexibilidade possível.

A opção adotada no Netuno foi o uso de um s*witch fabric* InfiniBand (CISCO SFS-7024D-X) com 264 portas DDR 4X, permitindo a comunicação simultânea, totalmente sem bloqueio, entre os 256 nós computacionais com uso de conexões em cobre. Este *switch fabric* possibilita o uso de RDMA com latência final (de espaço de usuário para espaço de usuário) menor que 4 µs.

O uso da rede InfiniBand, ao invés de outros tipos de rede, como Infinipath, Myrinet, QsNet e SCI, se justifica por sua relação custo/desempenho e por também ser o tipo de rede de maior crescimento entre os sistemas computacionais de alto desempenho que aparecem relacionados na lista Top500 [1].

2.2.2. Rede de Entrada e Saída

A rede de E/S para acesso aos arquivos remotos possui uma demanda muito menor em termos de latência e taxa de transmissão, já que as aplicações paralelas de interesse fazem tipicamente uso intensivo de processador (*cpu-bound*) e, portanto, realizam relativamente um número menor de operações de E/S.

Por outro lado, essa questão não pode ser tratada com menor importância, já que os supercomputadores consomem e produzem uma quantidade expressiva de dados em um curto espaço de tempo. Segundo Ken Batcher [2], "Um supercomputador é um dispositivo para transformar problemas *cpu-bound* em problemas *I/O-bound*".

Embora a demanda de cada um dos nós computacionais isoladamente possa não ser tão significativa, podem surgir problemas quando todos os nós fazem acesso simultâneo a um único sistema de arquivos. Para evitar este problema, há um *switch* em cada gabinete para onde convergem todas as conexões de 1 Gbps de cada um dos 32 nós computacionais instalados no gabinete. Dos *switches* em cada gabinete, partem duas conexões em fibra ótica, de 10 Gbps cada, que se conectam a um *switch* central de alto desempenho (CISCO 6509), com 16 interfaces em fibra ótica, perfazendo um total de 8 gabinetes.

O sistema de arquivos remoto e paralelo se conecta a este roteador de alta capacidade através de 12 portas *ethernet* de 1 Gbps, totalizando uma taxa máxima de transferência de 1,2 Gbytes por segundo.

Esta opção de ligação do sistema de arquivos remoto foi a melhor possível no momento da especificação e aquisição do equipamento, dado que opções de maior desempenho (10 GE) ainda não estavam disponíveis.

2.3. Sistema de Arquivos Remoto e Distribuído

Uma das maiores dificuldades enfrentadas durante a especificação do Netuno foi a falta de informações para melhor dimensionar o sistema de arquivos compartilhado pelos nós computacionais.

A opção inicial, o uso de um sistema de arquivos distribuído do tipo NAS (Network Attached Storage), acessível remotamente via protocolo NFS (Network File System), foi logo abandonada. Sistemas do tipo NAS não escalam bem, quer em termos de capacidade de armazenamento ou de desempenho. Além disso, o protocolo NFS tem fragilidades e limitações de desempenho que se agravam quando há um grande número de clientes acessando um mesmo servidor/ arquivo.

Para contornar essas limitações, o uso de sistemas de arquivos paralelos distribuídos passou a ser considerado, sendo que, entre as opções disponíveis, avaliamos o uso do PVFS, Lustre e PanFS.

Optamos pelo uso do PanFs pelas características que iremos descrever a seguir. A Figura 1 mostra a arquitetura básica do PanFS.

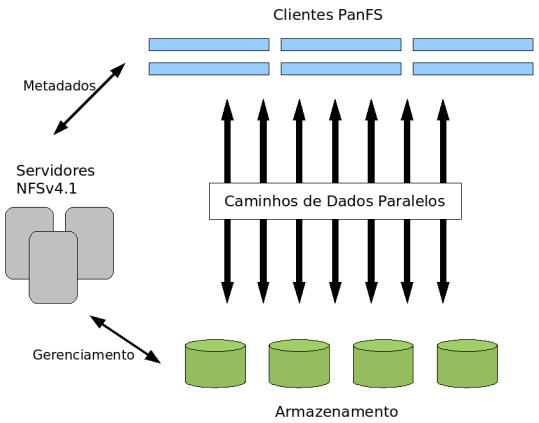

Figura 1: Arquitetura Básica do PanFS

O PanFS é um sistema de arquivos projetado pela Panasas, que tem as características de alta velocidade, globalidade e escalabilidade. Para isso se utiliza de *hardware* dedicado denominado de cartão de

armazenamento. Cada unidade básica de armazenamento, uma gaveta para montagem em gabinete com altura 4U, pode acomodar até 11 cartões, sendo 1 cartão de diretório e 10 cartões de armazenamento

Cada cartão é na realidade um sistema completo, com uma pequena placa mãe, processador, memória, um ou mais discos, interface de rede interna Gigabit. A memória é utilizada também como cache para os dados armazenados nos discos. Há dois tipos de cartões: um cartão de diretório que faz o papel de gerenciador de metadados e um cartão de armazenamento que armazena os dados propriamente ditos.

O PanFs é um sistema de arquivos baseado em objetos, ou seja, os clientes contactam o gerenciador de metadados para realizar um acesso a um arquivo. O gerenciador de metadados envia um mapa da localização dos dados para o cliente. Então o cliente se comunica com os cartões de armazenamento diretamente e em paralelo.

À medida que os dados são escritos nos cartões de armazenamento, eles são quebrados em blocos de 64 KB. Atributos de arquivo, um identificador de objeto e outras informações são adicionadas a cada bloco para formar um objeto. Os objetos são escritos utilizando-se RAID-1 para arquivos com 64 KB ou menos, e RAID-5 para arquivos maiores.

O cliente utiliza o protocolo de acesso DirectFlow, que permite acesso aos cartões de armazenamento diretamente e em paralelo. Este protocolo permite ao cliente enviar pedidos de acesso aos dados de um mesmo arquivo em paralelo, melhorando bastante o desempenho.

Os testes de avaliação que realizamos no Netuno indicam o desempenho de uma única gaveta (4 links Gigabit) com um único cartão de diretório em torno 350 MB/S para leitura de dados.

O cliente DirectFlow da Panasas é muito similar ao cliente NFSv4.1(Parallel NFS), e o código fonte está disponível publicamente. Além do cliente DirectFlow, os clientes podem acessar os arquivos usando NFS e CIFS. Os cartões de diretório atuam como servidores NFS e CIFS.

O PanFS possui ainda uma interface ROM/IO que permite que seja utilizado pelo conjunto de rotinas da biblioteca MPI-I/O.

O sistema adquirido para o Netuno possui 3 gavetas, cada uma com 10 cartões de armazenamento com 2 discos de 500 GB cada, totalizando 30 TB de capacidade de armazenamento e capacidade máxima de transferência de 1,4 GB/S para leitura de arquivos.

3. Opções de *Software*

Apenas a escolha dos componentes de *hardware* de um *cluster* não asseguram o seu correto funcionamento e desempenho adequados. Há necessidade de definição de um sistema operacional, compiladores e *middleware* adequados.

3.1 Sistema Operacional

Há várias opções possíveis na escolha do sistema operacional de um *cluster*. Há versões comerciais e ainda diversas variações do Linux, algumas sob licença e outras gratuitas.

No caso do Netuno, decidimos pela utilização da distribuição CentOS do Linux, versão 5.1, em todos os nós do *cluster*. É uma distribuição gratuita, baseada na distribuição licenciada do RedHat Enterprise Linux, com maturidade e estabilidade suficientes para garantir um funcionamento adequado e sem falhas.

O uso do Linux oferece como grande vantagem a possibilidade de compilar o núcleo de modo personalizado, isto é, apenas com os módulos específicos para o *hardware* dos nós computacionais.

Além da economia de espaço em memória para as aplicações, permite uma maior estabilidade do sistema operacional, pela redução do número de componentes envolvidos.

Um componente importante do sistema operacional é o *driver* que será utilizado para interface com a rede InfiniBand. A inexistência de um *driver* adequado, ou ainda com baixo desempenho, pode comprometer o bom funcionamento do *cluster*.

O *driver* utilizado no Netuno foi aquele fornecido pela OpenFabrics Alliance [3], também conhecido como OFED (OpenFabrics Enterprise Distribution). Esse driver é projetado para realizar a conexão entre os componentes do *cluster*, sejam eles servidores ou sistemas de armazenamento, sendo otimizados para desempenho (alto *throughput*, baixa latência) utilizando RDMA e outras tecnologias disponíveis no adaptador HCA.

A distribuição OFED inclui também um gerenciador de rede InfiniBand (OpenSM), ferramentas de diagnóstico, testes de avaliação de desempenho e suporte para uso da biblioteca MPI com a interface InfiniBand.

3.2 Compiladores e Bibliotecas de Comunicação

Diversos compiladores e bibliotecas de comunicação para o padrão MPI estão disponíveis para utilização em *clusters* com sistemas operacionais do tipo Linux.

Em termos de compiladores para as linguagens C e Fortran podemos encontrar compiladores de uso público, como os compiladores GNU, e as versões proprietárias como Intel e Portland Group.

No *cluster* Netuno todas essas opções estão disponíveis para os usuários já que, de acordo com o tipo de aplicação, uma ou outra versão pode ser mais indicada.

Para os testes de avaliação realizados com o programa HPL, apresentado na Seção 4, foram utilizados os compiladores GNU e Intel. Não foi observada uma variação relevante no desempenho por conta da utilização de um ou outro compilador.

Em termos de bibliotecas de comunicação MPI, estão disponíveis também diversas versões de uso público e algumas soluções proprietárias. Atualmente, encontram-se disponíveis no Netuno as distribuições gratuitas OpenMPI [4], MVAPICH2 [5] e as distribuições licenciadas Intel MPI [6] e Scali MPI [7].

Nos testes realizados há diferenças significativas de desempenho e robustez entre essas diversas versões, sendo que os melhores resultados foram obtidos com a biblioteca OpenMPI.

Os depuradores paralelos de domínio público não apresentam ainda maturidade suficiente para uso em um sistema de produção. Por conta disso optamos pelo uso do depurador paralelo proprietário da TotalView.

3.3 Middleware

Com o conjunto de ferramentas e compiladores relacionados até aqui, já é possível a execução de programas paralelos no *cluster*.

Contudo, quando se trata de em um sistema real, com vários usuários, com diversos tipos de programas e privilégios, e ainda que demanda o gerenciamento de centenas de nós, ferramentas adicionais são necessárias para ordenar e melhorar a utilização dos nós.

Um gerenciador de carga é um conjunto de ferramentas que permite um ou mais usuários compartilhar com eficiência o poder computacional de um *cluster*. No alto nível, ele permite o acesso remoto a esses recursos e coordena as ações para prevenir conflitos entre usuários, melhorar a eficiência do uso *cluster* e gerenciar os nós com defeitos. Utilizando um gerenciador de carga, os usuários conseguem saber o estado geral do *cluster*, submeter novos pedidos, monitorar e controlar pedidos já submetidos, e ainda agendar tarefas para execução com antecedência.

Em conjunto com o gerenciador de carga, normalmente encontra-se um portal de acesso, onde o usuário pode realizar todas essas funções remotamente, sem necessidade de instalar nenhum programa no seu computador e sem necessitar da intervenção do gerente do sistemas.

Entre as diversas opções existentes, de domínio público ou licenciadas, optamos pelo uso no *cluster* Netuno da suíte "Moab Cluster" [8], licenciada, trabalhando em conjunto com o gerenciador de tarefas Torque [9], de domínio público.

Adicionalmente, se faz necessário o uso de ferramentas para gerenciar a infra-estrutura básica do *cluster* garantindo, por exemplo, que todos os nós computacionais estejam consistentes entre si, com a mesma imagem do sistema operacional, incluindo núcleo do sistema operacional, bibliotecas e pacotes instalados.

Uma operação simples, como enviar o mesmo comando para ser executado em todos os nós, pode se tornar uma tarefa complicada sem as ferramentas adequadas.

Mais uma vez, há diversas soluções possíveis, optamos pelo uso no Netuno da ferramenta proprietária Scali Manager, atualmente adquirida e gerenciada pela empresa Platform Computing.

O Netuno possui 2 nós de controle exclusivamente para a execução de todo o *software* de *middleware* descrito nesta seção e fazer a contabilidade e controle dos usuários.

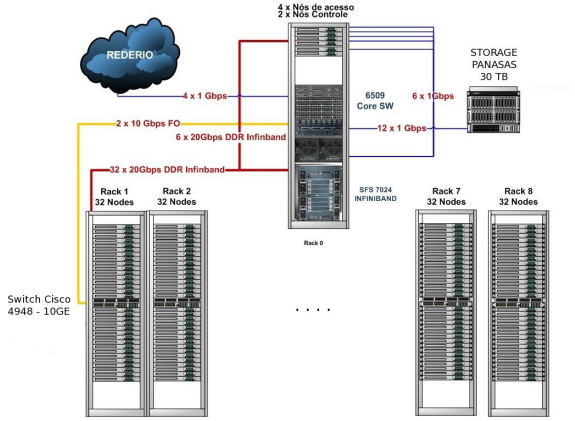

Figura 2: Arquitetura do Netuno

A Figura 2 mostra um diagrama com a configuração do *cluster* Netuno.

4. Avaliação de Desempenho

4.1. O Algoritmo do Benchmark HPL

O *benchmark* **HPL** [10] é a principal ferramenta de avaliação utilizada para compor a lista Top500 [1] que relaciona, a cada 6 meses, os computadores mais rápidos do planeta. O HPL é um programa que resolve um sistema de equações lineares densas (geradas aleatoriamente) em aritmética de precisão dupla (64 bits) em computadores com memória distribuída.

O pacote HPL é um programa de teste com medidas de tempo, que serve tanto para quantificar a **precisão** das soluções obtidas, assim como medir o **tempo gasto** para computá-la. O desempenho máximo que pode ser alcançado na execução deste programa em um sistema paralelo depende de uma grande variedade de parâmetros, que são passados pelo usuário em um arquivo de configuração no momento da sua execução.

O *benchmark* HPL calcula a solução de um sistema de equações lineares $A.x=b$, onde A é uma matriz densa gerada aleatoriamente de dimensão $N \times N$, x e b são vetores de tamanho N. A matriz A é primeiramente fatorada como sendo o produto $A=L.U$, onde L e U representam as matrizes triangulares inferior e superior respectivamente.

Para a estabilidade numérica, o algoritmo de fatoração usa um pivotamento parcial de linha. A solução x é então encontrada em dois passos sucessivos de solução triangular, $L.z=b$ e finalmente $U.x=z$.

Os dados da matriz A que compõem o sistema são distribuídos em uma grade de processos bidimensional de tamanho $P \times Q$ de acordo com um esquema cíclico em bloco. A matriz de coeficientes, de dimensão $N \times N+1$, é primeiro particionada logicamente em $NB \times NB$ blocos, que são ciclicamente distribuídos na grade de processos $P \times Q$. Isto é feito em ambas as dimensões da matriz, garantindo um bom balanceamento de carga, assim como a escalabilidade do algoritmo.

Foi escolhida a implementação mais simples da decomposição (fatoração) de Cholesky, conhecida como *right-looking variant*, para o laço principal da fatoração LU. Isso significa que, a cada iteração do laço, um painel de NB colunas é fatorado, e a submatriz resultante é atualizada. Deste modo, a computação é logicamente particionada com o mesmo tamanho de bloco NB que foi usado para a distribuição dos dados.

A Figura 3 ilustra a distribuição dos blocos da matriz para um conjunto de 6 processos. Na realidade cada elemento A_{ij} representado na figura é uma submatriz de dimensão NB.

	0			1			2	
0	A_{11}	A_{14}	A_{17}	A_{12}	A_{15}	A_{18}	A_{13}	A_{16}
	A_{31}	A_{34}	A_{37}	A_{32}	A_{35}	A_{38}	A_{33}	A_{36}
	A_{51}	A_{54}	A_{57}	A_{52}	A_{55}	A_{58}	A_{53}	A_{56}
	A_{71}	A_{74}	A_{77}	A_{72}	A_{75}	A_{78}	A_{73}	A_{76}
1	A_{21}	A_{24}	A_{27}	A_{22}	A_{25}	A_{28}	A_{23}	A_{26}
	A_{41}	A_{44}	A_{47}	A_{42}	A_{45}	A_{48}	A_{43}	A_{46}
	A_{61}	A_{641}	A_{67}	A_{62}	A_{65}	A_{68}	A_{63}	A_{66}
	A_{81}	A_{84}	A_{87}	A_{82}	A_{85}	A_{88}	A_{83}	A_{86}

Figura 3: Distribuição dos Blocos da Matriz

4.2. Execução do *Benchmark* HPL

A execução do *benchmark* HPL é cercada de diversas possibilidades e variações que incluem desde os compiladores, bibliotecas matemáticas e de comunicação, além dos próprios parâmetros de configuração do programa. Em termos de compiladores, tínhamos disponíveis os compiladores Intel, GCC e PGI (Portland Group).

Além disso, o desempenho do HPL é fortemente dependente da biblioteca BLAS, existindo várias versões que podem ser utilizadas.

A fatoração do *benchmark* HPL requer $(2/3)*N^3$ operações e as duas soluções triangulares totalizam N^2 operações cada, todas de ponto flutuante. A medida que N aumenta, a fase de fatoração domina o tempo de computação.

Os dados relacionados com a i-ésima iteração do algoritmo são apresentados na Figura 4, D é o i-ésimo bloco na diagonal principal, L, U e T são as partes atuais da matriz inferior, superior e restante.

O valor de N é determinado pela quantidade máxima de memória que cada nó computacional pode destinar para o processo HPL antes que comecem acontecer operações de *swap* para o disco. Isso significa que o sistema operacional precisa ser minimizado para permitir mais espaço para a a execução do HPL. Assim sendo, a imagem do sistema operacional para rodar os testes HPL não deve ser a mesma que a da fase de produção do *cluster*.

Assumindo que o tamanho da matriz A é $8*N^2$ bytes, então, se em cada nó computacional deixarmos mais ou menos 20% da memória para o sistema, N pode ser escolhido com a seguinte fórmula [11]:

$$N = \sqrt{(0{,}8 \times NumNos \times MemNo)/8} \quad [1]$$

Onde, $NumNos$ é o número total de nós e $MemNo$ é a capacidade de memória de cada nó do *cluster*.

Se N for muito alto, então o sistema começa a paginar e o desempenho diminui sensivelmente. Se N for muito baixo, mais computação poderia ter sido feita em cada nó.

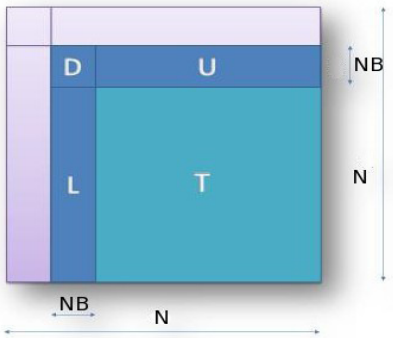

Figura 4: Fatoração da Matriz

Para estimar o valor NB, deve-se iniciar com valores de 32 ou 40, e então tentar fazer NB o maior possível. Uma vez encontrado um bom valor para NB, valores de múltiplos pares de NB, expressos pela fórmula $2*i*NB$, devem dar bons resultados também. Como N e NB são relacionados, é desejável que N seja múltiplo de NB para evitar computações residuais.

Os valores de P e Q estão relacionados ao número de processos, tal que: $P*Q = NumProcessos$. A regra de

ouro é ter $Q > P$ e possivelmente $Q = 2*P$ ou $P = sqrt(N)$.

4.3. Refinando os Parâmetros

Como o desempenho do *benchmark* HPL é dependente do espaço de memória utilizado, devemos obter o maior valor possível de *N* que não cause operações de *swap*. Entretanto, como *N* precisa ser múltiplo de *NB* para evitar o desbalanceamento do trabalho, então é melhor determinar primeiro o maior valor possível para *NB*, para diminuir o custo de comunicação.

Esses testes podem ser feitos com valores menores de *N*, já que o valor ótimo de *NB* tende a manter-se constante [12]. Pode-se iniciar com um valor pequeno de *NB*, como 32 ou 48, e ir aumentando de 16 em 16 até determinar o valor ótimo. Na Tabela 1, mostramos os parâmetros de configuração do HPL com as opções utilizadas na avaliação realizada no *cluster* Netuno.

Tabela 1: Parametros de Configuração do HPL

Linha	Valor	Descrição
5	1	# de tamanhos de problemas (N) a serem lidas
6	655000	N – A dimensão da matriz a ser calculada
7	1	Número de NBs a serem lidos
8	192	NB – O tamanho de NB (número de elementos)
9	0	Mapeamento de Processos (0=Linha,1=Coluna)
10	1	# de grades de processos (P x Q) a serem lidas
11	32	P P * Q = número total de núcleos
12	64	Q P * Q = número total de núcleos
13	16	Threshold – Valor a ser comparado no final da execução para validar os resultados
14	1	# de tipos de painéis de fatoração a serem lidos
15	1	PFACTs (0=left, 1=Crout, 2=Right)
16	1	# de critérios de parada recursivos a serem lidos
17	4	NBMINs (>= 1)
18	1	# de painéis na recursão a serem lidos
19	4	NDIVs
20	1	# de painéis de fatoração recursivos a serem lidos
21	2	RFACTs (0=left, 1=Crout, 2=Right)
22	1	# de tipos de difusão a serem lidos
23	1	BCASTs (0=1rg,1=1rM,2=2rg,3=2rM,4=Lng,5=LnM)
24	1	# de profundidades de lookahead a serem lidas
25	1	DEPTHs (>=0)
26	2	SWAP (0=bin-exch,1=long,2=mix)
27	64	swapping threshold
28	0	Formato de L1 in (0=transposta,1=não-transposta) Use sempre 0
29	0	Formato de U in (0=transposta,1=não-transposta) Use sempre 0
30	1	Equilibration (0=Não,1=Sim) Sempre 1
31	8	Alinhamento de memória double (> 0) (4, 8 ou 16)

4.4. Resultados

Para estimar o tamanho ideal de N utilizamos a Equação 1 e obtivemos *N*=41448 para um apenas um nó e *N*=663177 para todo o *cluster*. Esses valores são aproximações iniciais e os valores ideais são obtidos por tentativa e erro. Para o *cluster* inteiro, realizamos experimentos com *N*=540000, 600000, 650000, 655000. O gráfico da Figura 5 mostra o desempenho em TFlops obtido para esses experimentos. Conforme podemos observar no gráfico, o melhor desempenho foi de 16,2 TFlops, obtido para *N*=655000, para valores maiores, o sistema começa a realizar *swap* e o desempenho cai abruptamente. Isto ocorre porque na medida em que se aumenta o número de nós, a quantidade de memória gasta com os *buffers* de comunicação do MPI também aumenta. Portanto, é preciso deixar um pouco mais do que 20% da memória para o sistema. Para experimentos realizados em apenas um nó do *cluster*, o melhor desempenho obtido foi de 71 GFlops para *N*=44000, ou seja, um pouco maior que o estimado inicialmente.

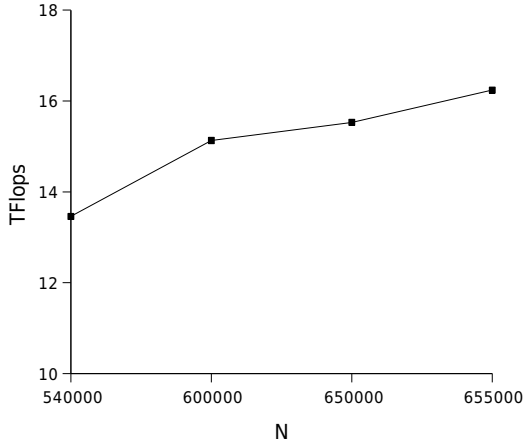

Figura 5: Desempenho x Tamanho da Matriz

Considerando que o desempenho máximo que um nó do *cluster* pode obter é de 85,1 GFlops (conforme visto na seção 2.1.1), a execução para *N*=44000 obteve uma eficiência de 82,3% do valor de pico teórico. Para o *cluster* inteiro o desempenho máximo é dado por $T_{máx}$ = 85,1 GFlops x 256 = 21,7 TFlops. Portanto a execução com *N*=655000 obteve uma eficiência de 74,7%, considerada excelente para *clusters* desse tipo. Só a título de comparação, a primeira máquina na lista Top500, RoadRunner, reportou este ano uma eficiência de 75.9%.

Como o *benchmark* HPL apresenta alta taxa de comunicação entre os processos, avaliamos também o efeito da rede de interconexão e da biblioteca MPI no desempenho final.

Com relação à rede de interconexão, executamos um outro experimento no *cluster* inteiro com *N*=655000, só que utilizando somente a rede *ethernet* para comunicação

entre os nós. Este experimento apontou um desempenho de apenas 8,9 TFlops, ou seja, um eficiência de 42%. Este resultado mostra que a opção arquitetural pelo uso da rede InfiniBand e *drivers* OFED foi fundamental para a exploração em maior grau do potencial de desempenho do *cluster*.

No que se refere à biblioteca MPI empregada, apresentamos na Tabela 2 os resultados de desempenho obtidos para as bibliotecas OpenMPI, Intel MPI e Scali MPI, com diferentes números de nós e parâmetros de entrada.

Conforme podemos observar nesta tabela, a biblioteca OpenMPI apresentou um melhor desempenho do que a biblioteca Intel MPI quando há um número maior de nós. Este resultado comprova a boa adequação da biblioteca OpenMPI para *clusters* muito grandes [13].

Embora a biblioteca comercial ScaliMPI tenha apresentado um bom resultado com até 128 nós, ela não foi utilizada para medir o desempenho final do *cluster*, porque apresentou problemas de execução quando o número de nós ultrapassou este valor.

Tabela 2: Desempenho das bibliotecas OpenMPI e IntelMPI.

Nós	*N*	*NB*	*P*	*Q*	Resultados (GFlops)		
					OpenMPI	Intel MPI	Scali MPI
1	41448	192	2	4	71	72	72
4	82897	192	4	8	282	283	284
8	117234	192	8	8	561	548	570
16	165794	192	8	16	1106	1088	1131
32	234468	192	16	16	2189	2081	2218
64	331588	192	16	32	4248	4097	4365
128	468936	192	32	32	8268	7760	8385

6. Conclusões e Trabalhos Futuros

Procuramos apresentar neste trabalho todos os requisitos, tanto de *hardware* como de *software* necessários à construção e colocação em operação de *cluster* de alto desempenho. Apresentamos também detalhes do algoritmo, da configuração e execução do programa High Performance Linpack (HPL) em um *cluster*, com vistas à sua inclusão dos resultados obtidos na lista Top500.

Nenhuma dessas tarefas pode ser realizada de forma trivial, exigindo do projetista um conhecimento acumulado em muitas áreas da computação. Esperamos com este artigo compartilhar esta experiência e conhecimentos com a comunidade científica, facilitando e estimulando novos projetos deste tipo, fundamentais para o desenvolvimento da área de computação de alto desempenho no país.

Como trabalhos futuros pretendemos estender o processo de avaliação realizado, detalhando aspectos como, por exemplo, o uso de programas com paradigmas mistos de memória compartilhada e distribuída, incluindo outras aplicações de alto desempenho além do HPL e aprofundando a análise de importância da rede de interconexão no desempenho final das aplicações no *cluster*. Além disso, esperamos avaliar com detalhes o sistema de arquivos paralelo em uso e comparar com outras alternativas de domínio público.

7. Referências

[1] TOP500 Supercomputer Site. http://www.top500.org

[2] Batcher, K. E., "Design of a Massively Parallel Processor," *IEEE Transactions on Computers*, Vol. C29, September 1980, 836-840.

[3] OpenFabrics. http://www.openfabrics.org/

[4] Gabriel, E. et alli "Open MPI: goals, concept, and design of a next generation MPI implementation". In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[5] W. Huang , G. Santhanaraman , H.-W. Jin , Q. Gao and D. K. Panda, "Design of High Performance MVAPICH2: MPI2 over InfiniBand", In *Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, p.43-48, May 16-19, 2006

[6] Intel MPI. http://software.intel.com/en-us/intel-mpi-library/

[7] Scali MPI. http://www.platform.com/Products/platform-mpi.

[8] Moab Cluster Suite – Cluster Resources. http:// www.clusterresources.com/products/moab-cluster-suite.php

[9] TORQUE Open-Source Resource Manager. http://www.clusterresources.com/products/torque-resource-manager.php

[10] Petitet, A.; Whaley, R. C.; Dongarra, J.; Cleary, ª "HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers" , Version 2.0, September 10, 2008, http://www.netlib.org/benchmark/hpl/

[11] Pillons, X., "Running HPL on Windows HPC Server 2008". http://windowshpc.net/Pages/Default.aspx.

[12] Wenli, Z.; Jianping, F.; Mingyu, C. "Efficient Determination of Block Size NB for Parallel Linpack Test". *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems* (PDCS 2004).

[13] Shipman, G.M.; Woodall, T.S.; Graham, R.L.; Maccabe, A.B.; Bridges, P.G. "Infiniband Scalability in OpenMPI*" Parallel and Distributed Processing Symposium,* (IPDPS 2006).