Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Unidade Acadêmica de Engenharia Elétrica

CLUSTERS DE ALTO DESEMPENHO - A CLASSE BEOWULF

Gustavo Barbosa Galindo

Relatório referente ao ***Trabalho de Conclusão de Curso*** *submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande.*

Orientador:
Prof. Dr. Bruno Barbosa Albert

Campina Grande, Paraíba, Brasil

Trabalho de Conclusão de Curso

Relatório apresentado à Coordenação do Curso de Engenharia Elétrica da Universidade Federal de Campina Grande em cumprimento às exigências para a obtenção do título de Engenheiro Eletricista.

Gustavo Barbosa Galindo

Aluno

Prof. Dr. Bruno Barbosa Albert

Orientador

Campina Grande, Paraíba, Brasil

Julho de 2011

Resumo

O presente trabalho aborda a construção de *clusters* de alto desempenho, forma de agregar computadores de modo que possam operar em paralelo como se fossem apenas uma grande máquina.

Com o avanço da tecnologia, a computação paralela têm se tornado cada vez mais presente nas empresas de médio e grande porte, além de estar presente em diversas universidades pelo mundo, seja para fins educacionais, seja para pesquisa científica e resolução de problemas matemáticos. Pode-se, também, citar a indústria cinematográfica, onde há a utilização de *clusters* Beowulf para a renderização de filmes e imagens.

Inicialmente será discutida a temática geral envolvendo sistemas operacionais, suas características e sua distribuição. Atualmente, há *clusters* sendo criados tanto em Linux, quanto em ambientes Windows, porém, a classe Beowulf tem como principal característica a utilização de *software* de distribuição livre, portanto a grande aplicação do sistema operacional do pinguim. Ainda no mesmo capítulo, abordaremos a supercomputação e alguns tipos de *cluster*, além de uma rápida visão acerca da computação em grade.

Em seguida, serão mostrados alguns dados sobre a supercomputação no mundo, com os maiores supercomputadores já construídos ao redor do planeta, bem como suas características de *software*, *hardware* e desempenho.

Por fim, a classe Beowulf, sua construção, instalação e configuração serão descritos no texto. Para ilustrar os resultados experimentais, executamos alguns *scripts* de administração do *cluster*, que ilustram o funcionamento da comunicação entre as máquinas e programas matemáticos, que serão aplicados na confirmação da redução do tempo de processamento, além de servirem como exemplos de programação paralela.

Palavras-chaves: *cluster*, supercomputação, Beowulf, MPI.

Índice

Índice de Tabelas

Índice de Figuras

1

Introdução

1.1 O Unix

O Unix, por ser um sistema operacional poderoso, moderno e atual, sendo atualizado constantemente pela comunidade entusiasta, é utilizado universalmente há mais de 30 anos. A cada dia, novas aplicações são criadas nas mais diversas plataformas de hardware. Este continua a ser um exemplo de maturidade e eficiência e vem ganhando espaço em meio a tanta concorrência com os sistemas comerciais. Apesar de existirem várias distribuições pagas, algumas das mais utilizadas são as versões livres, como o Linux.

Na prática, utilizar o Linux é o mesmo que utilizar o Unix, mesmo nas versões pagas. Uma das grandes vantagens do sistema é de ser multitarefa e multiusuário e estar disponível para diversas plataformas de hardware. O Unix é multiusuário, pois permite que vários usuários utilizem o mesmo computador simultaneamente, por meio de terminais remotos. É dito multitarefa por possibilitar que vários programas sejam executados ao mesmo tempo.

Além disso, várias são as possibilidades de aplicações para redes, como sistema de cota de disco, FTP, e-mail, WWW, DNS, execução de programas em *background* etc.

1.2 O Linux

A partir do momento em que o Unix não pôde mais ser utilizado livremente, alguns programadores iniciaram a criação de diversas implementações próprias do sistema operacional, como é o caso do Minix (Tanenbaum, 1986) e do Linux por Linus Torvalds.

Originalmente desenvolvido como passatempo por Linus Torvalds na Universidade de Helsinky, na Finlândia, o Linux foi criado como uma alternativa barata e funcional do Unix para quem não está disposto a pagar o alto preço de um sistema Unix comercial ou não tem um computador suficientemente rápido. O seu desenvolvimento contou com a colaboração de muitos programadores e especialistas através da Internet e foi inspirado no sistema operacional criado por Tanenbaum.

O Linux não é um software de domínio público, mas licenciado sob a licença GPL (GNU Public License), o que garante que seu código fonte permaneça livremente disponível. As pessoas podem cobrar pela cópia do Linux, se desejarem, desde que, com isso, não limitem sua distribuição.

Muitas pessoas vêm trabalhando conjuntamente para continuar o seu desenvolvimento sob a direção de Linus Torvalds, o autor original, e cada uma delas mantém os direitos de copyright sobre o código que escreveram.

O Linux segue o modelo de desenvolvimento aberto e, por isso mesmo, a cada nova versão liberada ao público, é considerado "um produto de qualidade". Atualmente, o número de versão do kernel do Linux consiste em quatro números **A.B.C[.D]**. Atualmente, a versão estável é a 2.6.x. O número **C** muda cada vez que novas características ou drivers forem incluídos. (Pereira, 2008)

Diversas empresas e organizações de voluntários decidiram juntar os programas do Linux em "pacotes"próprios aos quais fornecem suporte. Estes são chamados de distribuições, das quais as mais famosas são Red Hat Enterprise/Fedora Core, Mandriva, Debian, Slackware, SuSe (Novell) e Ubuntu. Esta última sendo a utilizada para a implementação prática no presente trabalho. (Pereira, 2008)

Na verdade, muitos conhecem e divulgam o sistema operacional como sendo apenas Linux, porém o termo correto é GNU/Linux, já que o Linux é apenas o kernel do sistema,

sendo necessário o uso de várias ferramentas para o seu funcionamento. Estas ferramentas são providas pelo projeto GNU, criado por Richard Stallman. Em outras palavras, o citado sistema operacional é a união do kernel Linux com as ferramentas GNU, por isso o termo supracitado.

1.3 GNU/Linux *versus* Windows

O sistema operacional comercial mais conhecido no momento é o Windows, utilizado em massa nos computadores pessoais. Além de ser pago, possui diversas diferenças em relação ao Linux, das quais a mais marcante é o fato deste último ter seu código fonte aberto e desenvolvido por programadores voluntários espalhados por todo o mundo e distribuído sob a licença pública GPL, como anteriormente citado. Enquanto o Windows tem seu desenvolvimento em um ambiente corporativo, com modificações controladas e limitadas.

Além de não precisar pagar para instalar o Linux em qualquer máquina, sua cópia e distribuição são livres. Uma vantagem neste ponto é a flexibilidade que se cria com relação às necessidades do usuário, o que torna suas adaptações e "correções"muito mais rápidas. Ainda, enquanto sistemas comerciais são desenvolvidos em uma empresa, com desenvolvedores contratados, e apenas estes, sistemas de código aberto, como o Linux, possuem milhares de desenvolvedores ao redor do mundo, possibilitando uma resposta mais rápida com relação aos problemas encontrados e às questões de segurança. Com isto, o desenvolvimento do Linux é constante e ininterrupto.

Outra diferença e grande vantagem é o sistema X-Window, fornecedor do ambiente gráfico do Linux. Neste, o gerenciador de janelas, ou interface visual, é criado em um processo separado. Com isso, o usuário pode escolher entre diversos gerenciadores desenvolvidos por diferentes grupos. Alguns exemplos são o Gnome e o KDE.

Desta forma, o Linux se torna um alternativa viável para usuários residenciais e comerciais que não estão dispostos custear a compra de software proprietários para computadores pessoais e, principalmente servidores. Podemos, ainda, citar características marcantes como sua estabilidade, sua robustez e a flexibilidade de plataformas, em contrapartida ao Windows.

1.4 Supercomputação

Com o aumento da quantidade de dados disponibilizados na Internet e com o fluxo cada vez maior de informações, aliado ao crescimento do número de tecnologias ligadas à grande rede e ao maior número de pesquisas que necessitam de processamento computacional, a demanda por um maior poder de processamento é cada vez maior.

A taxa de crescimento do poder de processamento de minicomputadores, mainframes e os tradicionais supercomputadores, têm sido em torno de 20% ao ano, enquanto que, para os microprocessadores, a taxa está numa média de 40%. Outro avanço muito importante foi a invenção das redes locais, ou LAN - *Local Area Network*. (Pitanga, 2008)

O desenvolvimento de máquinas poderosas, como os *mainframes*, se tornou habitual, porém seu preço ainda é elevado, bem como o de seus componentes. Configuração e manutenção ainda são uma realidade distante para aqueles que não são treinados para tal. Universidades, pequenas e médias empresas que necessitam de servidores para a realização de pesquisas científicas ou para que seja assegurado o funcionamento de seus serviços, como um site com grande número de acessos simultâneos, por exemplo, e não possuem verbas suficientemente grandes para tal investimento estão utilizando cada vez mais uma estrutura de agrupamento de computadores, chamada de *cluster*, para a realização dessa tarefa.

Um dos objetivos deste arranjo é a paralelização de suas aplicações, a fim de se obter um menor tempo de resposta. Além disso, apresenta uma série de vantagens em relação aos outros ambientes de alto desempenho e/ou disponibilidade, como a possibilidade de aumento do seu poder de processamento com o acréscimo de outros computadores, ou o seu baixo custo em relação às outras soluções.

1.4.1 Cluster de Computadores

Neste trabalho serão abordados os *clusters* de alto desempenho (HPC - *High Performance Computing*) da classe Beowulf, sua construção, funcionamento e testes de desempenho. Outros dois tipos de *cluster* são bastante utilizados e estudados: os de alta disponibilidade (HA - *High Availability*) e os de balanceamento de carga.

A utilização de *cluster* como solução para computação de alto desempenho é uma reali-

dade. Esta tecnologia pode ser utilizada nos cursos universitários na formação de profissionais de Tecnologia da Informação e ainda como ferramenta de apoio às pesquisas em outras áreas como biologia, química, física, dentre outras, integrando cada vez mais os diversos segmentos da sociedade. Outros exemplos de sua utilização são os servidores da Internet, onde o cluster pode distribuir a carga, aumentando a capacidade de resposta ao usuário e na Computação Gráfica, onde pode-se diminuir o tempo de renderização de imagens e vídeos. Este último sendo a abordagem utilizada neste texto para atestar seu desempenho. (Pitanga, 2008)

Clusters de Alta Disponibilidade

A alta disponibilidade é uma técnica que consiste na configuração de dois ou mais computadores para que passem a trabalhar como um só. A sua vantagem é o monitoramento entre si das máquinas, fazendo com que os demais, em caso de falhas, assumam os serviços que ficaram indisponíveis. Esta forma de agrupamento tem como função essencial deixar um sistema no ar vinte e quatro horas por dia, sete dias por semana, ou que não suportem paradas não planejadas, influenciando diretamente a qualidade do serviço e os prejuízos financeiros que possam acarretar. (Pereira, 2008)

Para tanto existem classes de disponibilidades que devem ser citadas:

- Disponibilidade convencional - encontrada em qualquer computador comum disponível no mercado, sem nenhum recurso extra de *software* ou *hardware* que oculte as eventuais falhas destes. Sua disponibilidade é, em geral, de 99% a 99,9%, significando que em 1 ano de operação o computador pode ficar indisponível por um período de 9 horas a 4 dias (Pereira, 2008);

- Alta disponibilidade - encontrada em computadores mais sofisticados com recursos de detecção, recuperação e ocultamento de falhas. Possuem disponibilidade de 99,99% a 99,9996%. Portanto, em 1 ano de operação, estes podem ficar indisponíveis por um período de um pouco mais de 5 minutos;

- Disponibilidade contínua - presente em computadores ainda mais sofisticados e com recursos de detecção, recuperação e ocultamento de falhas, onde se obtém disponibilidade cada vez mais próxima de 100% (Pereira, 2008).

Nos clusters de alta disponibilidade os equipamentos são usados em conjunto para manter um serviço ou equipamento sempre ativo, replicando serviços e servidores, o que evita máquinas paradas ou ociosas, esperando apenas o outro equipamento ou serviço paralisar, passando as demais a responder por ela normalmente.

Alguns trabalhos em GNU/Linux podem ser citados como referências em alta disponibilidade (Pitanga, 2008):

- LVS - *Linux Virtual Server*, que tem como objetivo o balanceamento de carga e a alta disponibilidade em *clusters* de servidores, criando a imagem de um único servidor "virtual";
- Eddiware - atuando com escalabilidade de servidores *web*, consistindo em um servidor de balanceamento de http, um servidor de DNS aperfeiçoado, usa técnica de duas camadas com o *frontend* e *backend*, fazendo roteamento de tráfego;
- Heartbeat, Mon, DRDB.

Clusters de Alto Desempenho (Beowulf)

Os *clusters* de alto desempenho, como os de classe Beowulf, têm como principal foco o aumento do poder de processamento através da realização da tarefa pelos processadores disponíveis, ou mesmo, a divisão da tarefa por vários destes. Com a inserção de novos computadores a um agrupamento, onde estão presentes um nó principal, ou servidor, e um ou mais nós clientes, ligados por uma rede, pode-se chegar ao nível de desempenho comparável ao de supercomputadores por um custo razoavelmente baixo. É construído com componentes comuns de *hardware*, como qualquer computador capaz de operar com o Linux, placas de rede padrão *Ethernet* e *switches*.

O nó servidor também é a porta de acesso para o mundo exterior. É comum, em grandes máquinas Beowulf, que sejam utilizados mais de um nó servidor e, da mesma forma, outros nós dedicados a tarefas específicas, como consoles ou estações de monitoramento. Em sua maioria, os nós clientes em um sistema Beowulf são computadores simples. Na verdade, quanto mais simples, melhor. Ainda é possível que alguns nós clientes não tenham discos rígidos e, desta forma, não sabem os seus endereços IP até que o servidor os diga quem são.

Na maioria dos casos, os nós clientes não possuem teclados, monitores nem mouses e são acessados via *login* remoto. (Pereira, 2008)

É importante frisar que o Beowulf não é um pacote de *software* especial, uma nova topologia de rede nem uma versão do kernel do Linux modificada. É apenas uma tecnologia de agrupar computadores utilizando o sistema operacional supracitado para formar um supercomputador paralelo virtual.

Normalmente, ao se pensar em processamento de alto desempenho, processamento paralelo ou processamento distribuído, se pensa em grandes máquinas dedicadas e de altíssimo custo. Estas máquinas são de difícil operação e estão alocadas em salas com alta segurança. Nos dias atuais, devido aos *clusters*, os custos foram reduzidos e existem máquinas muito rápidas, o que viabiliza o uso do processamento de alto desempenho na solução de problemas em diversas áreas.

Como todo o *software* necessário para sua criação e operação está disponível na Internet e o *hardware* utilizado é o normalmente disponível em PCs, sua vantagem quanto ao custo é óbvia e a facilidade de se montar um *cluster* é evidente.

Um ponto desfavorável é o de que todo o conhecimento produzido acerca do assunto está espalhado por diversas fontes e cresce rapidamente, o que dificulta, às vezes, um entendimento mais amplo ou atualizado desse cenário.

Cluster de Balanceamento de Carga

Nos clusters de balanceamento de carga, a divisão do trabalho é de forma uniforme, seja entre dois ou mais computadores, seja pelos enlaces de rede, ou discos rígidos. O seu objetivo é a otimização da utilização dos recursos disponíveis, maximizando o desempenho e evitando sobrecarga. Há o balanceamento de armazenamento, onde o ganho é em tempo de acesso aos dados, o balanceamento da rede e o de CPUs.

Estes e os *clusters* de alta disponibilidade compartilham diversos módulos como uma solução para uma variedade de problemas relacionados ao alto tráfego em sites muito acessados, aplicações de missão crítica, paradas programadas, serviços contínuos, entre outros.

1.4.2 Computação em Grade

A computação em grade, ou *Grid Computing*, é um caso particular da computação distribuída, onde se objetiva a resolução de problemas computacionais através da utilização de diferentes recursos distribuídos geograficamente. Uma vez que o *Grid* é orientado essencialmente para aplicações que precisam de uma grande capacidade de cálculos, ou enormes quantidades de dados transmitidos de um lado para o outro, ou as duas coisas. Para o caso da computação distribuída, esta passa a ser um caso em grade a partir do momento em que existe uma infra-estrutura física e lógica que permita coordenar os trabalhos e garantir a qualidade do serviço.

Seu nome se deve às malhas de interligação dos sistemas de energia elétrica (*power grids*), caso em que há a utilização do serviço sem que se saiba onde esta foi gerada, sendo totalmente transparente ao usuário.

Ian Foster, professor do Departamento de Ciências da Computação da Universidade de Chicago, toma dois conceitos de *Grid*: "Compartilhamento de recursos coordenados e resolução de problemas em organizações virtuais multi-institucionais dinâmicas"e "*Grid Computing* são sistemas de suporte à execução de aplicações paralelas que acoplam recursos heterogêneos distribuídos, oferecendo acesso consistente e barato aos recursos, independentemente de sua posição física. A tecnologia de *Grids Computing* possibilita agregar recursos computacionais variados e dispersos em um único 'supercomputador virtual', acelerando a execução de várias aplicações paralelas. *Grids* se tornaram possíveis nos últimos anos devido à grande melhoria em desempenho e redução de custo, tanto em redes de computadores quanto de microprocessadores". (Pitanga, 2008)

Para o Professor Rajkumar Buyya, do Departamento de Ciência da Computação e Engenharia de Software da Universidade de Melbourne, Austrália, a grande diferença entre *Grid* e *Cluster* pode ser simplificada: "Se acontece o compartilhamento de recursos gerenciado por um único sistema global sincronizado e centralizado, então é um *cluster*. Em um *cluster*, todos os nós trabalham cooperativamente em um objetivo comum e o objetivo é a alocação de recursos executada por um gerente centralizado e global. Em *Grid*, cada nó possui seu próprio gerente de recursos e política de alocação".

2

Supercomputação no Mundo

Desde junho de 1993, duas vezes por ano uma lista com os 500 maiores projetos de supercomputação é compilada por uma organização intitulada "*Top 500 Supercomputers Sites*", que descreve os maiores supercomputadores em atividade no mundo. Estão discriminados dados como poder computacional, localização geográfica, arquitetura utilizada, sistema operacional e área de operação do projeto.

Iniciativas como esta são grande interesse para os fabricantes, usuários e possíveis usuários. Estas estatísticas podem facilitar o acesso aos colaboradores e a troca de dados e *software*, bem como uma boa visão do mercado de computação de alta *performance*.

Na tabela 2.2 são listados os 10 maiores supercomputadores em atividade e, na tabela 2.1, os dois projetos brasileiros citados entre os TOP500. Alguns dados foram suprimidos para um melhor entendimento. Em seguida, na figura 2.1, um gráfico que ilustra a situação da supercomputação desde o início da listagem e uma projeção para o futuro.

Tabela 2.1: Discriminação dos 2 projetos brasileiros listados pelo site TOP500. (TOP500, 2011)

Pos.	**Localização**	**Fabricante**	**Rmáx**	**Rpico**	**Arquitetura**	**País**
29	National Institute for Space Research (INPE)	Cray Inc.	205100	258048	MPP	Brazil
116	NACAD/COPPE/UFRJ	Oracle	64630	72396,8	Cluster	Brazil

Tabela 2.2: Discriminação dos 10 primeiros *clusters* listados pelo site TOP500. (TOP500, 2011)

Pos.	**Localização**	**Fabricante**	**Rmáx**	**Rpico**	**Arquitetura**	**País**
1	National Supercomputing Center in Tianjin	NUDT	2566000	4701000	MPP	China
2	DOE/SC/Oak Ridge National Laboratory	Cray Inc.	1759000	2331000	MPP	EUA
3	National Supercomputing Centre in Shenzhen (NSCS)	Dawning	1271000	2984300	Cluster	China
4	GSIC Center, Tokyo Institute of Technology	NEC/HP	1192000	2287630	Cluster	Japão
5	DOE/SC/LBNL/NERSC	Cray Inc.	1054000	1288630	MPP	EUA
6	Commissariat a l'Energie Atomique (CEA)	Bull SA	1050000	1254550	Cluster	França
7	DOE/NNSA/LANL	IBM	1042000	1375780	Cluster	EUA
8	National Institute for Computational Sciences/University of Tennessee	Cray Inc.	831700	1028850	MPP	EUA
9	Forschungszentrum Juelich (FZJ)	IBM	825500	1002700	MPP	Alemanha
10	DOE/NNSA/LANL/SNL	Cray Inc.	816600	1028660	MPP	EUA

Entre os dados listados, está Rmáx, se referindo ao desempenho máximo obtido pelo ambiente de testes LINPACK e Rpico, que seria o desempenho teórico da arquitetura, todos em Megaflops, onde 1 flop equivale a capacidade de realizar 1 cálculo de ponto flutuante por segundo. Ou seja, o supercomputador listado em primeiro lugar, intitulado Tianhe-1A, do NUDT (Universidade Nacional de Tecnologia de Defesa) e situado na cidade de Tianjin, na China, atingindo a marca de 2,566 petaflops, consegue realizar cerca de $2{,}5.10^{15}$ cálculos em ponto flutuante por segundo.

O teste realizado pelo ambiente LINPACK é feito com uma bateria de cálculos, onde é medido sua capacidade na solução de sistemas de equações lineares com um grande número de variáveis. Este *software* foi escolhido pela possibilidade de otimização de forma a ser melhor adequado a cada sistema, afim de que se obtenha o melhor desempenho e pela portabilidade para várias arquiteturas. No entanto, tais modificações não refletem no desempenho geral, mas sim, no desempenho de um sistema dedicado à solução de um denso sistema de equações lineares.

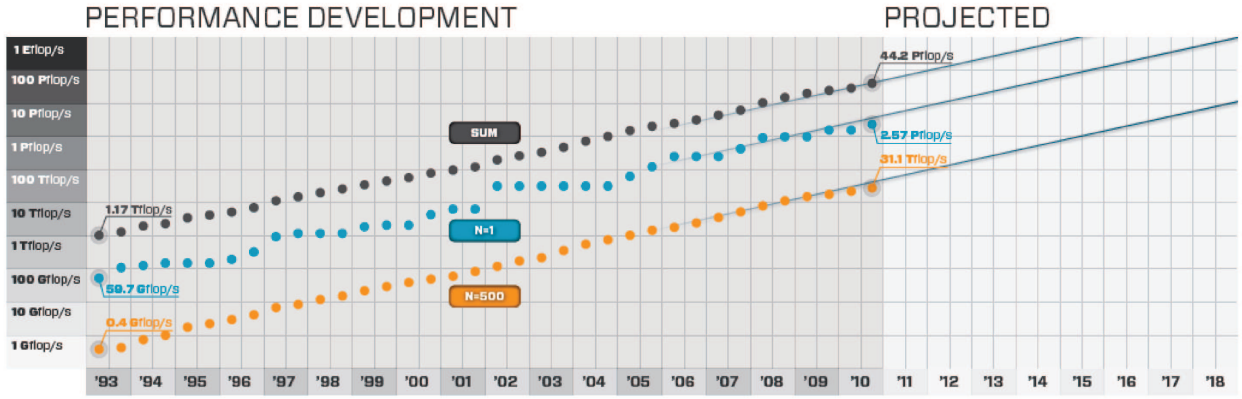

Figura 2.1: Desempenho dos supercomputadores no mundo desde 1993. (TOP500, 2011)

É possível perceber que, em números, o desenvolvimento de supercomputadores no mundo segue uma tendência linear, porém a sua utilização não consegue seguir tal taxa. Segundo Hu Weiwu, chefe de desenvolvimento da série de microchips Loongson, na Academia Chinesa de Ciências (*Chinese Academy of Sciences* - CAS), no final do ano de 2010 seus pesquisadores serão capazes de extrair apenas 10% do seu potencial. Ainda há muitas questões científicas a serem respondidas pelos cálculos que podem ser implementados em supercomputadores, porém ainda faltam bons algorítmos e uma boa coleta de dados para fazê-los funcionar.

O chip Loongson será utilizado pelo primeiro supercomputador chinês criado com *hardware* próprio e este contará com 10.000 unidades deste microchip. Com este número, seu processamento poderá atingir 1 petaflop de poder de processamento. Para tanto, Hu Weiwu aproxima um custo anual de 1 milhão de dólares apenas pela energia elétrica utilizada para mantê-lo funcionando.

Em números, a China é o segundo país em número de supercomputadores em atividade, atingindo a marca de 8,2% dentre os listados pelo TOP500, seguido pela Alemanha, França e Japão, todos com 5,2% destes. Em primeiro lugar, está os Estados Unidos da América, com 54,8%. Tais dados estão listados na figura 2.2 em forma de gráfico.

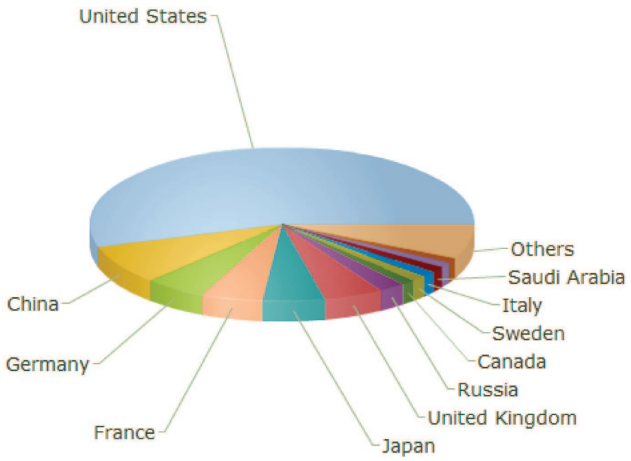

Figura 2.2: Distribuição geográfica dos TOP500. (TOP500, 2011)

Além disso, como ilustrado na figura 2.3, 82,8% utilizam a arquitetura em cluster e 82% são sistemas Linux, ilustrado na figura 2.4.

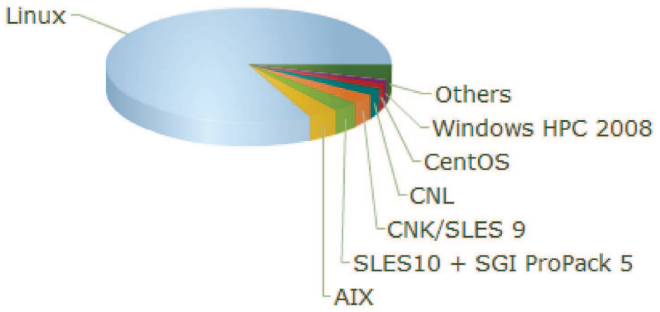

Figura 2.3: Distribuição geográfica dos TOP500. (TOP500, 2011)

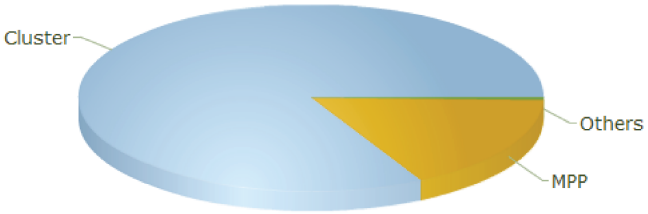

Figura 2.4: Sistemas operacionais utilizados dentre os TOP500. (TOP500, 2011)

2.1 Aplicabilidade dos Supercomputadores

A utilização dos supercomputadores tem se mostrado muito eficiente na resolução de grandes problemas computacionais, não sendo possível sem o uso da computação paralela e do seu grande poder de processamento. Com a necessidade de simulações numéricas de sistemas complexos como tempo, clima e reações químicas, por exemplo, seu desenvolvimento tem sido motivado constantemente.

Como exemplos de aplicações, podem-se citar:

- Bases de Dados - onde pode ser reduzido o tempo de busca em grandes bancos de dados;
- Computação Gráfica - o tempo de processamento pode ser uma limitação para a evolução dos projetos, o que pode ser melhorado com a utilização de *clusters*, diminuindo-se o tempo de renderização de grandes imagens, ou na elaboração de filmes;
- Engenharia Genética - como no projeto Genoma;
- Análise de elementos finitos - para o cálculo de barragens, pontes, aviões etc.

Com o desenvolvimento da Informática, ainda há muitas aplicações a serem criadas e destinadas para uso em *clusters*. Em 1974, ninguém diria que o efeito que o CRAY traria ao desenvolvimento das áreas da física, biologia, química e microeletrônica. A disponibilidade de tais sistemas de alto desempenho fora dos centros de dados tradicionais, onde normalmente são achados os supercomputadores, trará um efeito profundo tanto nas áreas de pesquisa quanto na área empresarial nos próximos anos. (Rocha, 2003)

3

O *Cluster* Beowulf

Com o crescimento do desempenho computacional dos PCs, aliado ao fato de que o mercado destes é maior que o de *workstations*, o preço dos PCs vêm diminuindo.

A computação em *cluster* teve como primeiro projeto o SAGE (*Semi-Automatic Ground Environment*), construído para o NORAD (Comando Norte Americano de Defesa Aeroespacial) pela IBM, em 1962. Consiste em diversos sitemas separados trabalhando de forma cooperativa para monitorar invasões aéreas no continente norte-americano, hoje incluindo também o Canadá. (Pitanga, 2008)

Nos anos 90, o desempenho dos computadores pessoais começou a se equiparar com o de estações de trabalho de alto desempenho e o custo dos equipamentos de rede diminuiu, aliado ao fato da disponibilidade de hardware e da gratuidade do sistema Linux, fizeram com que cientistas encontrassem uma maneira de interligar computadores pessoais para obterem maiores poderes computacionais sem a compra de máquinas caras.

No ano de 1993, a NASA utilizava um pequeno cluster da IBM para simular partidas de naves espaciais. No mesmo ano, surge a primeira lista dos TOP500, sendo o projeto NOW (*Network of Workstations*) da Universidade da Califórnia, na cidade de Berkeley, listado na primeira posição. (Pitanga, 2008)

Em 1994 surgia o *Cluster* Beowulf, criado a partir da necessidade de um equipamento que conseguisse atingir a marca de 1 gigaflop. A aquisição de uma máquina que obtivesse este poder de processamento custaria à agência cerca de 1 milhão de dólares. Devido a isso,

os pesquisadores Thomas Sterling e Donald J. Becker ligaram 16 computadores pessoais com processadores Intel 486 Dx4 a 100 Mhz, utilizando o Linux e uma rede padrão. Atingiu-se a marca dos 70 megaflops, o que na época era o obtido com pequenos supercomputadores comerciais. (Pitanga, 2008)

Este *cluster* se tornou uma ferramenta popular para os cientistas que precisavam de um maior poder computacional para modelagens matemáticas e outras aplicações. O nome Beowulf foi escolhido por causa do herói da literatura inglesa pelos seus criadores.

Seu sistema é construído utilizando-se componentes de *hardware* comuns, placas de rede e *switches*. Além de não possuir nenhum componente desenvolvido especificamente para o seu desenvolvimento, ainda usa software livre, como o Linux, e bibliotecas para comunicação em rede. O nó servidor controla todo o *cluster* e distribui os processos aos nós clientes. No entanto, sua arquitetura é tão sofisticada, robusta e eficiente quanto qualquer outra comercial.

3.1 Pilha de PCs

Pilha de PCs (Pilha de Computadores Pessoais, ou PoPC - *Pile-of*-PCs) é o termo utilizado para descrever uma montagem solta de PCs, ou, um *cluster* de PCs. Como detalhes de sua construção, podem-se citar:

- Uso de componentes disponíveis no mercado tradicional de informática;
- Processadores dedicados às tarefas, em contrapartida à utilização do tempo ocioso das estações;
- Uso da rede local para os nós computacionais.

Ainda, para que a arquitetura seja considerada Beowulf, esta deve obedecer às seguintes características:

- Não utilização de componentes feitos por encomenda;
- Independência de fornecedores de *hardware* e *software*;

- Uso de *software* livre e de código aberto;
- Uso de ferramentas de computação distribuída disponível livremente e com alterações mínimas.

Como vantagens desta construção, podem-se citar a falta de direitos sobre o produto pelo fornecedor e a utilização de componentes diversos, devido à padronização de interfaces. Com a utilização de *software* de licença livre, como Linux e as bibliotecas de passagem de mensagem, PVM e MPI, permitiu-se fazer alterações para que possam ter novas características que facilitem a implementação em diversas áreas e para diversas necessidades.

3.2 *Cluster* de Estações de Trabalho

O *Cluster* de Estações de Trabalho, ou *Cluster of Workstations*, é muito parecido com o Beowulf, sendo constituído de estações de trabalho de alto desempenho, cada um com monitor, teclado e mouse, ligadas por uma rede *Ethernet*. Todas estas máquinas se comunicam entre si. A partir de um nó, pode-se acessar outro, executando-se, assim, tarefas remotamente. Em um COW podemos ter acesso ao teclado em qualquer nó da rede, o que não acontece com o Beowulf, o qual só se tem acesso ao teclado no computador principal. Tendo acesso aos outros nós por acesso remoto, no nosso caso, via SSH - *Secure Shell*. (Pitanga, 2008)

Outra característica do COW é a utilização dos nós como postos de trabalho quando estas não estão participando de um processamento distribuído. No entando, seu funcionamento é igual ao do Beowulf, existindo um computador principal que serve de controlador para a distribuição das tarefas para os nós escravos. (Pitanga, 2008)

4

Implementação de *Clusters* HPC

Como citado inicialmente, este trabalho tem como fim a implementação prática de um ambiente paralelo, um cluster HPC utilizando-se o Linux. Devido à grande variedade de bibliotecas de programação paralela, bem como de envio de mensagens, esta parte não será abordada no presente texto. Em contrapartida, serão citados os ambientes mais utilizados, como o PVM e o MPI. Além das implementações do MPI, as bibliotecas MPICH e LAM/MPI.

O processo de construção e gerenciamento de *clusters* paralelos envolve diversos fatores, desde a escolha e instalação do sistema operacional, definição das ferramentas para a configuração, manutenção, monitoramento e escalonamento das tarefas. (Pitanga, 2008)

Todos os *softwares* necessários para a montagem de uma máquina paralela, como o sistema operacional, os compiladores para as linguagens de programação, além de todos os programas e utilitários que podem ser utilizados para gerenciar, ou automatizar as tarefas, estão disponíveis na *Internet* e gratuitamente. (Pitanga, 2008)

O objetivo da programação paralela é transformar grandes algoritmos complexos em pequenas tarefas que possam ser executadas simultaneamente por vários processadores, reduzindo assim o tempo de processamento, chamado de *wall time clock*. (Pitanga, 2008)

4.1 Bibliotecas de Passagem de Mensagens

Para a utilização do poder de processamento de um Cluster Beowulf, algumas bibliotecas estão disponíveis, como as supracitadas MPI e PVM. Um *cluster* de alto desempenho implementa a comunicação entre os nós utilizando estas bibliotecas, que são um conjunto de rotinas que visam facilitar a comunicação entre os processos e um padrão de mensagem que possibilita ao programador uma interface padronizada para escrever códigos portáteis entre plataformas. (Batista, 2007)

Uma das definições do ambiente de passagem de mensagens que podemos tomar é:

> Método de comunicação baseado no envio e recebimento de mensagens através da rede seguindo as regras do protocolo de comunicação entre os vários processadores que possuam memória própria. Usualmente requer operações cooperativas a serem executadas por cada processo, onde uma operação de envio deverá ter uma operação correspondente de recebimento, ficando a responsabilidade de sincronização entre os processos por parte do programador. (Pitanga, 2008)

4.1.1 PVM - *Parallel Virtual Machine*

PVM é um pacote de *software* que permite a utilização de uma organização de computadores em Unix e/ou Windows em conjunto como se fossem uma grande máquina paralela por uma rede local. Assim, podem ser solucionados grandes problemas computacionais com menor custo. O seu código fonte está disponível na *Internet* e já foi compilado para as mais diversas máquinas, desde *laptops* a CRAYs. (ORNL e CSM, 2010)

É utilizada em diversos locais pelo mundo para a solução de problemas científicos, industriais e médicos, além de ser uma ferramenta educacional para o ensino da programação paralela. (ORNL e CSM, 2010)

O PVM foi um dos primeiros sistemas de *software* a possibilitar que programadores utilizem uma rede de sistemas heterogêneos ou sistemas MPP (*Massively Parallel Processors*) para desenvolver aplicações paralelas sob o conceito de passagem de mensagens. (Pitanga, 2008)

Como características do PVM, podemos citar:

- **Quanto à interoperabilidade** - além da portabilidade, os programas podem ser executados em diversas arquiteturas diferentes;
- **abstração completa** - o PVM permite que a rede seja totalmente heterogênea, administrada como se fosse apenas uma máquina virtual;
- **controle de processos** - com o PVM, é possível iniciar, interromper e controlar processos em tempo de execução;
- **controle de recursos** - totalmente abstrato, graças ao conceito de máquina virtual paralela;
- **tolerância a falhas** - comporta esquemas básicos de notificação de falhas

4.1.2 MPI - *Message Passing Interface*

O MPI, ou Interface de Passagem de Mensagens, é uma biblioteca com funções para troca de mensagens, responsável pela comunicação e sincronização de processos em um *cluster* paralelo. Dessa forma, os processos de um programa paralelo podem ser descritos em uma linguagem de programação sequencial, tal como C ou Fortran. O principal objetivo do MPI é disponibilizar uma interface que seja largamente utilizada no desenvolvimento de programas que utilizem troca de mensagens. (Pitanga, 2008)

Suas funções são a comunicação ponto a ponto, comunicação coletiva, suporte para grupo de processos, para contextos de comunicação e para topologia de processos. Muitas implementações de bibliotecas do padrão MPI têm sido desenvolvidas, sendo algumas proprietárias e outras de código livre, mas todas seguindo o mesmo padrão de funcionamento e uso. (Pitanga, 2008)

O MPI implementa um excelente mecanismo de portabilidade e independência de plataforma computacional. Por exemplo, um código MPI que foi escrito para uma arquitetura e utilizando um sistema operacional específico pode ser portado para outra arquitetura e outro sistema operacional com poucas modificações no código fonte da aplicação.

Implementações do Padrão MPI

O **MPICH**, ou MPICHamaleon, é uma implementação do padrão MPI e utilizada no presente trabalho, advinda do padrão MPI-2, criada pelo Argonne National Laboratory. Esta é disponibilizada livremente e sem restrições comerciais em seu *site* na *internet*. Estaremos utilizando a versão 1.2.7, criada em 2004. Esta versão combina portabilidade, interoperabilidade e alto desempenho.

Sua instalação e configuração são abordados na página 29, na sessão 5.3. Sua versão atual pode ser encontrada no *site* da empresa, disponível em http://www-unix.mcs.anl.gov/mpi/mpich. A versão 1.2.7p1 utilizada aqui está disponível em ftp://ftp.mcs.anl.gov/pub/mpi/mpich-1.2.7p1.tar.gz.

Uma outra implementação do padrão MPI é o **LAM** (*Local Area Multicomputer*), criada pelo *Open Systems Lab*, situado na Universidade de Indiana. Com um *cluster* dedicado ou uma rede de estações de trabalho, pode-se operá-lo como se fosse um único computador paralelo. O ambiente LAM implementa totalmente as funcionalidades do padrão MPI-1 e algumas do MPI-2. Seu código fonte para compilação pode ser baixado em http://www.lam-mpi.org.

5

A Construção do *Cluster* Beowulf

Para a construção do *software* do *cluster*, utilizamos a distribuição Linux Ubuntu 11.04 *Natty Narwhal*, em versão *desktop* 32 bits, já que sua implementação foi apenas para efeito de estudo do seu funcionamento. Esta escolha foi feita baseada na praticidade da instalação de todos os pacotes e para o acesso visual aos aplicativos.

Todas as máquinas têm a mesma configuração de *hardware*. *Desktops* equipados com processador Intel Pentium Dual-Core E2160 (1.8 GHz, 800 MHz FSB, 1 MB L2 Cache), 1 GB de memória RAM, HD com 80 GB de espaço e controladores *Ethernet* 88E8001 *Gigabit Ethernet Controller*.

Para a alocação física do espaço, foi utilizado o LEIG - Laboratório de Ensino Informatizado da Graduação, situado no Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande - UFCG. Na figura 5.1 pode-se ver o ambiente do laboratório e os computadores utilizados para a montagem. Este laboratório é utilizado para que sejam ministradas aulas de algumas disciplinas da graduação em Engenharia Elétrica, como o Laboratório de Princípios de Comunicações e o Laboratório de Sistemas Elétricos.

Figura 5.1: Ambiente do Laboratório de Ensino Informatizado da Graduação. PCs utilizados para o experimento.

Na figura 5.2 podemos ver a arquitetura proposta neste trabalho. São utilizadas 4 máquinas configuradas como segue em todo o decorrer do texto.

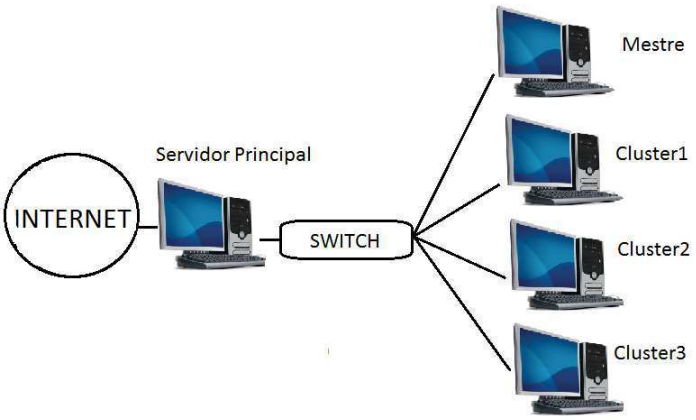

Figura 5.2: Arquitetura Proposta.

Nas sessões 5.1.1 e 5.2.1 são mostrados os passos para se construir um *cluster* como o abordado no presente trabalho. O guia segue desde a instalação e configuração dos serviços necessários à configuração da rede. As seções são divididas em Instalação e Configuração do Mestre, dos Nós e Instalação do MPICH. Foram tomadas algumas referências como base para a instalação e montagem do *cluster*, porém (Pereira, 2008) foi a principal, incluindo-se a sequência do passo a passo.

5.1 Configuração da Máquina Servidor

Instalado o Linux, é escolhida uma máquina para redirecionar os trabalhos e servir os arquivos utilizados no *cluster*, chamada de Servidor. Na nomenclatura de computadores, nomeamos esta como *mestre*. Então, sempre que nos referenciarmos à máquina mestre, esta será a que responderá.

O passo a passo é descrito na seção 5.1.1. No Ubuntu, os pacotes a serem instalados são:

1. **sysv-rc-conf** - que provê um terminal gráfico para que possamos controlar os serviços que estão em funcionamento, selecionando-os como *on* ou *off*. Isto é feito ao gerenciar os *links* em "/etc/rcrunlevel.d"(Packages, 2011f);

2. **openbsd-inetd** - o superservidor de *internet* FreeBSD é especializado na gerência das conexões de rede de entrada, selecionando qual programa deve rodar quando do acontecimento desta conexão (Packages, 2011b);

3. **openssh-server** - uma versão portável do OpenSSH, implementação livre do protocolo *Secure Shell*. SSH é um programa que serve para fazer *login* e executar comandos em máquinas remotas (Packages, 2011d);

4. **nfs-kernel-server** - este é atualmente o servidor NFS recomendado para uso com o Linux. Este pacote contém o suporte em espaço de usuário necessário para uso do servidor NFS de núcleo. O NFS serve principalmente para disponibilizar e montar diretórios via rede (Packages, 2011e);

5. **autofs** - serve para fazer automontagem no Linux (Packages, 2011c);

6. **ntp** - o NTP, *Network Time Protocol*, é usado para manter os relógios dos computadores ajustados por uma sincronização deles através da *Internet* ou de uma rede local (Packages, 2011a).

Ou seus similares. Todos foram instalados via `$ apt-get install <pacote>`. Podemos ver a máquina utilizada como mestre na figura 5.3.

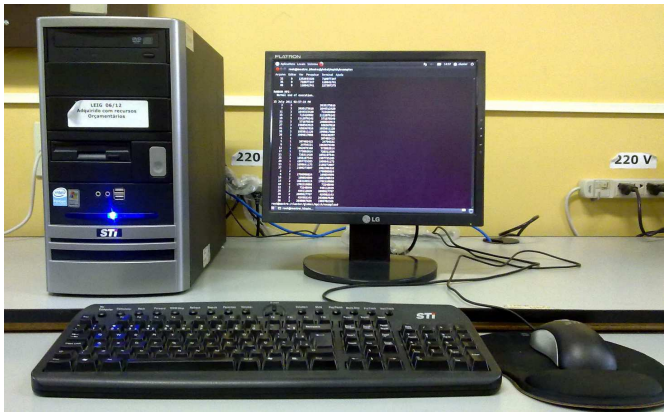

Figura 5.3: Máquina utilizada como nó mestre.

5.1.1 Passos de Configuração do Nó Mestre

A seguir, é mostrado como se devem configurar os serviços instalados para o correto funcionamento do *cluster* que está sendo construído. A ordem dos passos não é obrigatória, apenas foi seguida desta maneira.

Configuração da Rede

No presente caso, não utilizou-se um servidor DHCP para atribuir endereços IP aos computadores ligados à rede, sendo esta configuração feita de forma manual. A cada máquina foi dada um nome e um endereço como segue:

```
10.0.0.1        cluster1
10.0.0.2        cluster2
10.0.0.3        cluster3
10.0.0.6        mestre
```

Sendo, a seguir, passados estes dados ao arquivo /etc/hosts da forma como mostrada. Feito isso, devem-se incluir os nomes das máquinas do *cluster* no arquivo /etc/skel/.rhosts, um a um, em cada linha do arquivo. Por fim, deve-se definir regras de acesso para este arquivo executando o comando abaixo:

```
$ chmod 600 /etc/skel/.rhosts
```

Exportando os Diretórios de Trabalho

São incluídas no arquivo /etc/profile as linhas:

```
#Arquivo /etc/profile (linhas adicionais)
if [ "‘id -u‘-eq 0 ]; then
export PATH=$PATH:/cluster/global/bin:/cluster/global/sbin
else
export PATH=$PATH:/cluster/global/bin
fi
```

Configurando os Protocolos SSH e RSH

São configurados, então, os protocolos de comunicação SSH e RSH. Para a utilização deste último, temos que editar o arquivo /etc/hosts.equiv e /root/.rhosts para podermos fazer login remoto como usuário root. Para o pacote LAM/MPI, que não aceita login como este usuário, o arquivo se encontra em /home/<usuário>/.rhosts.

```
#Arquivo /root/.rhosts
cluster1
cluster2
cluster3
mestre
```

Em seguida, executamos o comando a seguir para mudarmos as regras de acesso ao arquivo supracitado:

```
$ chmod 600 /root/.rhosts
```

Então, editamos o arquivo /etc/securetty e adicionamos rsh e rlogin no final. Adicionamos ao arquivo /etc/inetd.conf as linhas:

```
shell stream tcp nowait root /usr/sbin/tcpd in.rshd
login stream tcp nowait root /usr/sbin/tcpd in.rlogind
```

Executando-se, em seguida, o programa `sysv-rc-conf` no Terminal e marcando a inicialização dos serviços `openbsd-inetd` e `ssh`. No arquivo /etc/ssh/sshd_config devem conter, ou serem descomentadas, as linhas `X11Forwarding yes` e `PermitRootLogin yes`, para que possamos ter acesso ao servidor X dos nós e para que possa ser feito login remoto pelo superusuário root. Por fim, executamos:

```
   $ ssh-keygen -t dsa -b 1024
$ scp /root/.ssh/id_dsa.pub root@<maquina>:/root/.ssh/authorized_keys2
$ invoke-rc.d openbsd-inetd start
$ invoke-rc.d ssh start
```

Onde o primeiro comando pede confirmação de dados e do diretório onde deve ser instalado o arquivo de chave pública de criptografia. Por padrão, deixamos a opção automática. O segundo deve ser executado para cada nó adicionado à montagem para que os programas paralelos possam ser executados sem a necessidade da digitação da senha do usuário em cada vez que for iniciado o processo. Os últimos dois comandos servem para a inicialização dos serviços indicados. Para ter acesso remoto com acesso ao servidor X, deve ser utilizado o comando:

```
$ ssh -X <nome_maquina ou IP_maquina>
```

Servidor NFS

Inicialmente, deve-se incluir ao arquivo /etc/exports, para que seja liberado aos nós da rede o acesso ao diretório remoto via NFS, a linha:

```
/cluster/global 10.0.0.0/255.255.255.0(rw,no_subtree_check)
```

Então, devemos criar os diretórios que serão compartilhados e utilizados para instalação das bibliotecas utilizadas paralelamente e de acesso pelos nós. Para tanto, executaram-se os comandos a seguir:

```
$ mkdir -p /cluster/global
$ cd /cluster/global
$ mkdir bin etc games home include lib libexec sbin share src
$ mkdir share/info share/man
$ cd share/man
$ mkdir man1 man2 man3 man4 man5 man6 man7 man8 man9 mann
$ mkdir man1x man2x man3x man4x man5x man6x man7x man8x man9x
```

Marcando o serviço `nfs-kernel-server` no programa `sysv-rc-conf` e iniciando-o com:

```
$ invoke-rc.d nfs-kernel-server start
```

Scripts de Administração do *Cluster*

Para instalarmos os *scripts* de administração do *cluster*, devemos fazer o *download* dos seus códigos em http://www.novatec.com.br/livros/linuxguiaadm2/linux-guia-adm-sis-2b-ed-mat-apoio.zip e copiá-los para o diretório /usr/local/sbin para que possam ser executados diretamente pelo Terminal.

```
$ cp cap55/beowulf/scripts/* /usr/local/sbin
$ chmod 755 /usr/local/sbin/*
```

Completando sua configuração com a edição do arquivo /etc/beowulf/hostnode, em seguida do arquivo /etc/beowulf/clusternodes com:

```
mestre
```

e

```
cluster1
cluster2
cluster3
```

respectivamente.

O Servidor NTP

Uma configuração básica para o servidor NTP é descrita em seguida, onde não há sincronização externa. Para tanto, edita-se o arquivo /etc/ntp.conf, deixando ele como em:

```
restrict default noquery notrust nomodify
restrict 127.0.0.1
restrict 10.0.0.0 mask 255.255.255.0
fudge 127.127.1.0 stratum 3
server 127.127.1.0
driftfile /var/lib/ntp/ntp.drift
logfile /var/log/ntp.log
authenticate no
```

Deve-se, também, marcar o serviço para inicialização automática e iniciá-lo, como feito pelo comando:

```
$ invoke-rc.d ntp start
```

5.2 Instalação e Configuração dos Nós

Seguem-se basicamente os mesmos passos da instalação do servidor. O único pacote ausente é o *nfs-kernel-server*, já que serão montados diretórios do nó mestre. A seguir, na sessão 5.2.1 são listados os passos para sua configuração. A organização dos computadores montados como nós é mostrada na figura 5.4.

5.2.1 Passos de Configuração dos Nós Escravos

Para a configuração dos nós, devem-se, inicialmente, seguir os passos descritos nas sessões 5.1.1, 5.1.1 e 5.1.1. Em seguida, alguns outros passos são importantes e diferenciados daquela descrita na configuração do nó mestre.

Figura 5.4: Nós do *cluster*.

Autofs

Para automatizar o processo de montagem do diretório de trabalho criado no servidor, deve-se editar os arquivos /etc/auto.master e /etc/auto.cluster, incluindo-se as seguintes linhas:

```
#Arquivo /etc/auto.master
/cluster /etc/auto.cluster -timeout=5

#Arquivo /etc/auto.cluster
global -fstype=nfs 10.0.0.4:/cluster/global
```

NTP

Para a sincronização do relógio da máquina com o do mestre, deve-se executar o comando:

```
sudo crontab -e
```

E adicionar a linha a seguir ao arquivo:

```
01 * * * * ntpdate -su mestre
```

5.3 Instalação e Configuração do MPICH

Depois de feito o *download* dos arquivos fonte do MPICH 1.2.7, temos que compilá-lo e instalar os arquivos. Há, também, pacotes disponíveis para instalação direta via apt-get, mas como utilizaríamos esta versão específica, foi decidido pela instalação manual. Também,

devido a alguns problemas encontrados na sua utilização e implementação com bibliotecas e programas antigos.

Inicialmente, descompactou-se o arquivo mpich-1.2.7p1 para o diretório /mpich-1.2.7p1 e, pelo terminal, entrou neste, para que pudéssemos executar alguns comandos. Os passos seguintes são:

```
$ ./configure -prefix=/cluster/global/mpich
$ make
$ su root -c "make install"
```

Em seguida, editou-se o arquivo "/cluster/global/mpich/share/machines.LINUX"incluindo-se os nomes das máquinas utilizadas no *cluster*, como "mestre", "cluster1", "cluster2"e "cluster3". Um em cada linha do arquivo.

Na figura 5.5 podemos ver a compilação de um programa pelo comando MPICC e na figura 5.6 é mostrado o comando para executar um programa pelo MPIRUN. No segundo caso, é indicado `-np 4`, que passa ao programa que o número de processadores será 4.

```
root@mestre:/cluster/global/mpich/examples# /cluster/global/mpich/bin/mpicc -o heat_mpi /cluster/global/mpich/examples/heat_mp
i.c
```

Figura 5.5: Comando para Compilação de Código Fonte em C - MPICC.

```
root@mestre:/cluster/global/mpich/examples# /cluster/global/mpich/bin/mpirun -np 4 /cluster/global/mpich/examples/random_mpi
```

Figura 5.6: MPIRUN - Comando para Rodar um Programa já Compilado.

Devido à tentativa de instalação da biblioteca LAM/MPI, os *links* para os programas de compilação e de utilização de programas compilados pelo MPICH foram prejudicados, o que nos obrigou a utilizá-los pelo seu caminho completo.

É importante frisar que o MPICH e todas as bibliotecas necessárias, como as de programação, devem ser instalados em todos os nós do *cluster*, o que não é abordado em boa parte dos guias encontrados para acesso pela *Internet*. De fato, isto é abordado nos manuais de instalação e utilização do *software*.

6

Resultados Experimentais

Para validação do nosso *cluster*, foram utilizados alguns programas com código fonte disponíveis livremente e outros *scripts* para administração disponibilizados no *site* da editora Novatec, na sessão do livro Linux - Guia do Administrador do Sistema (Pereira, 2008).

6.1 *Scripts* de Administração

Como já dito, alguns *scripts* de administração do *cluster* estão disponíveis para *download* em . Alguns deles são:

- **palive** - indica quais nós do *cluster* estão disponíveis para utilização - figura 6.3;
- **pmem** - mostra a quantidade de memória RAM utilizada pelo sistema em cada nó - figura 6.1;
- **pload** - exibe a carga de processamento do processador - figura 6.2;
- **ploadu root** - que indica a carga de processamento do usuário root no *cluster* - figura 6.4.

```
A memoria RAM usada pelo sistema e
-----------------------------------
cluster1:       Mem:   1016580k total,   227916k used,   788664
cluster2:       Mem:   1016580k total,   307312k used,   709268
cluster3:       Mem:   1016580k total,   380636k used,   635944
root@mestre:/home/cluster#
```

Figura 6.1: *Script* "pmem".

```
A carga do sistema é:
--------------------
cluster1: 6:31 up  1:48,  0 users,  load average: 0.00, 0.01, 0.02
cluster2: 6:29 up  1:48,  0 users,  load average: 0.00, 0.01, 0.03
cluster3: 6:16 up  1:48,  0 users,  load average: 0.08, 0.03, 0.05
root@mestre:/home/cluster#
```

Figura 6.2: *Script* "pload".

```
root@mestre:/home/cluster# palive

PING cluster1 (10.0.0.1) 56(84) bytes of data.
64 bytes from cluster1 (10.0.0.1): icmp_req=1 ttl=64 time=3.95 ms
64 bytes from cluster1 (10.0.0.1): icmp_req=2 ttl=64 time=0.108 ms

--- cluster1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.108/2.030/3.952/1.922 ms

O cluster1 está ativo

PING cluster2 (10.0.0.2) 56(84) bytes of data.
64 bytes from cluster2 (10.0.0.2): icmp_req=1 ttl=64 time=3.92 ms
64 bytes from cluster2 (10.0.0.2): icmp_req=2 ttl=64 time=0.130 ms

--- cluster2 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.130/2.029/3.928/1.899 ms

O cluster2 está ativo

PING cluster3 (10.0.0.3) 56(84) bytes of data.
64 bytes from cluster3 (10.0.0.3): icmp_req=1 ttl=64 time=3.96 ms
64 bytes from cluster3 (10.0.0.3): icmp_req=2 ttl=64 time=0.109 ms

--- cluster3 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1001ms
rtt min/avg/max/mdev = 0.109/2.034/3.960/1.926 ms

O cluster3 está ativo


Os nos ativos são:
------------------
cluster1
cluster2
cluster3
root@mestre:/home/cluster#
```

Figura 6.3: *Script* "palive".

```
NODE           PID USER        %CPU %MEM  TIME COMMAND
----------------------------------------
cluster1:      1 root    0  0.2   0:00.68 init
cluster2:   2851 root    2  0.1   0:00.01 top
cluster3:      1 root    0  0.2   0:00.69 init
root@mestre:/cluster/global/mpich/examples#
```

Figura 6.4: *Script* "ploadu root".

6.2 Programas Paralelizados

Algumas aplicações foram implementadas e testadas para participar da validação do *cluster*. Foi utilizado um código para cálculo do valor da constante Pi, contida nos arquivos exemplo da biblioteca MPICH. Outros códigos também foram compilados e executados no ambiente, como o para contagem de números primos de 1 a N, o de geração de números aleatórios e o código "Hello, World!", disponíveis em http://people.sc.fsu.edu/ jburkardt/c_src/mpi/mpi.html.

6.2.1 Cálculo de Pi

Para o cálculo do valor de Pi, é disponibilizado um código fonte na própria biblioteca do MPICH, que foi testado para 1 processador (figura 6.5) e para 4 (figura 6.6), sendo distribuídos 4 processos, 1 para cada máquina no *cluster*.

```
root@mestre:/home/cluster# /cluster/global/mpich/bin/mpirun -np 1 /cluster/global/mpich/examples/cpi
Process 0 on mestre
pi is approximately 3.1416009869231254, Error is 0.0000083333333323
wall clock time = 0.000080
```

Figura 6.5: Cálculo do Valor de Pi - 1 Processador.

Como este é um programa com pouco processamento, percebemos uma grande diferença no tempo para 1 ou para 4 processadores, sendo a utilização de apenas a máquina mestre muito mais rápido. Este fato se deve à velocidade da rede local ser muito lenta em relação à velocidade do acesso à memória pelo processador local. O processamento paralelo é muito melhor aproveitado quando se há uma utilização maior de recursos que valha o problema da velocidade da conexão de rede.

```
root@mestre:/home/cluster# /cluster/global/mpich/bin/mpirun -np 4 /cluster/global/mpich/examples/cpi
Process 0 on mestre
Process 1 on cluster1
Process 2 on cluster2
Process 3 on cluster3
pi is approximately 3.1416009869231245, Error is 0.0000083333333314
wall clock time = 0.003307
```

Figura 6.6: Cálculo do Valor de Pi - 4 Processadores.

6.2.2 Número de Primos

O programa PRIME_MPI é um programa em C que conta o número de primos entre 1 e N, utilizando o MPI para realizar o cálculo de forma paralela. O algoritmo é bem simples,

verificando apenas se, para cada número inteiro I, há algum inteiro J menor que o divide. Da mesma forma, separamos os testes entre 1 processador (figura 6.7) e 4 processadores (figura 6.8).

```
root@mestre:/cluster/global/mpich/examples# /cluster/global/mpich/bin/mpirun -np
 1 /cluster/global/mpich/examples/prime_mpi

'PRIME_MPI
  C/MPI version

  An MPI example program to count the number of primes.
  The number of processes is 1

         N        Pi          Time

         1         0        0.000048
         2         1        0.000005
         4         2        0.000005
         8         4        0.000004
        16         6        0.000005
        32        11        0.000006
        64        18        0.000009
       128        31        0.000019
       256        54        0.000052
       512        97        0.000222
      1024       172        0.000727
      2048       309        0.003408
      4096       564        0.011496
      8192      1028        0.029113
     16384      1900        0.146801
     32768      3512        0.581117
     65536      6542        2.160192
    131072     12251       10.137564

PRIME_MPI - Master process:
  Normal end of execution.
```

Figura 6.7: Números Primos - 1 Processador.

```
'PRIME_MPI
  C/MPI version

  An MPI example program to count the number of primes.
  The number of processes is 4

         N        Pi          Time

         1         0        0.003255
         2         1        0.000321
         4         2        0.000311
         8         4        0.000308
        16         6        0.000306
        32        11        0.000306
        64        18        0.000309
       128        31        0.000314
       256        54        0.000336
       512        97        0.000474
      1024       172        0.000844
      2048       309        0.002027
      4096       564        0.006506
      8192      1028        0.019824
     16384      1900        0.053480
     32768      3512        0.194325
     65536      6542        0.721223
    131072     12251        3.486870

PRIME_MPI - Master process:
  Normal end of execution.
```

Figura 6.8: Números Primos - 4 Processadores.

Percebe-se uma que não há ganho de velocidade para um número pequeno de dados, mas quando há um crescimento na carga de processamento, este ganho é bem perceptível, como para o número 131072, onde levou-se mais de 10 segundos para o cálculo com 1 processador e um pouco mais de 3 segundos quando se aumentou para 4 este número.

6.2.3 *Hello, World!*

O programa *Hello, World!* possui um código bem simples e serve para que seja gerada uma saída em texto com a frase "*Hello, World!*"e é bem utilizado no ensino das linguagens de programação, geralmente como o primeiro código que se aprende. Na figura 6.9 podemos vê-lo em ação de forma paralela, com cada uma das máquinas respondendo.

Figura 6.9: "*Hello, World!*- 4 Processadores.

Infelizmente, em seu código não é incluída uma forma de indicar qual processador está respondendo.

6.2.4 O Gerador de Números Aleatórios

O último código testado foi o gerador de números pseudoaleatórios, seguindo o o Método Linear Congruente (Wikipedia, 2011), que é um dos métodos mais antigos e conhecidos para este fim. A teoria por trás deste é simples e de fácil implementação.

Utilização de 1 Processador

Na figura 6.10 é mostrada a formulação utilizada no algoritmo. A partir das figuras 6.11 e 6.12 pode-se verificar o seu funcionamento quando da utilização de um processador.

Na próxima sessão o mesmo programa será executado nas 4 máquinas disponíveis, onde será feito o mesmo cálculo pelo processador da máquina mestre e, em seguida, este será repetido dividindo-se por 4 o número de cálculos a serem efetuados e cada porção será enviada para um dos nós.

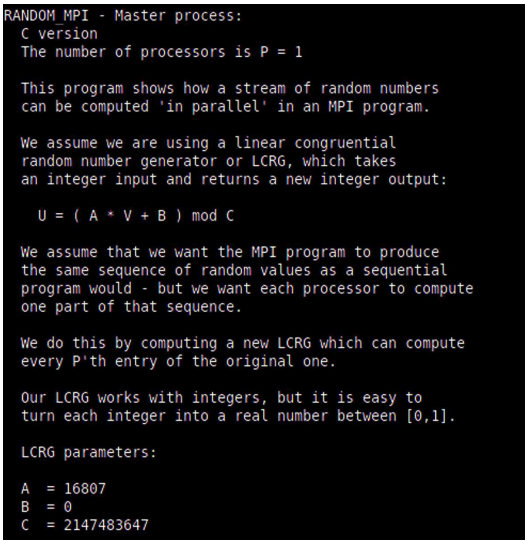

Figura 6.10: O Gerador de Números Aleatórios - 1 Processador.

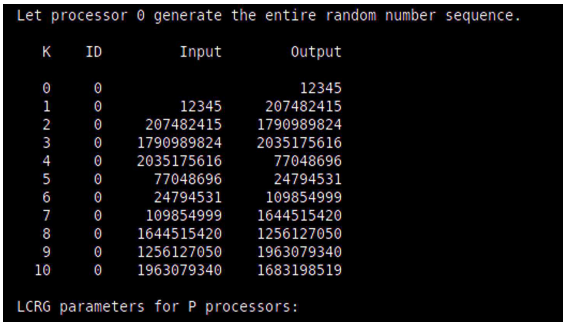

Figura 6.11: Geração dos Números Aleatórios - 1 Processador. Envolvimento do Processador 0.

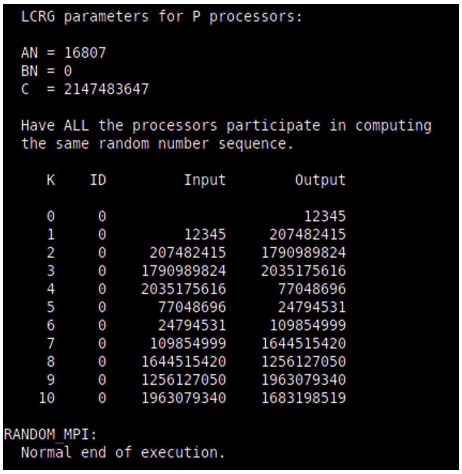

Figura 6.12: O Gerador de Números Aleatórios - 1 Processador. Envolvimento de Todos os Processadores.

Utilização de 4 Processadores

Nas figuras 6.13, 6.14 e 6.15 são ilustrados os casos para 4 processadores, onde, para a geração de 40 números aleatórios, 10 são passados para cada uma das máquinas envolvidas.

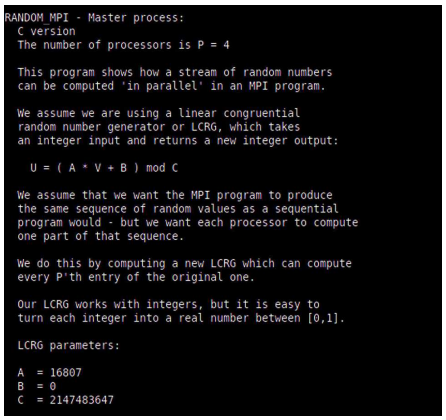

Figura 6.13: O Gerador de Números Aleatórios - 4 Processadores.

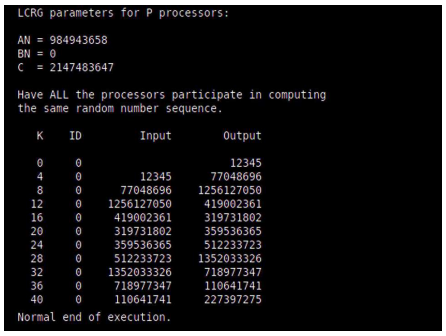

Figura 6.14: Geração dos Números Aleatórios - 4 Processadores. Envolvimento do Nó Mestre.

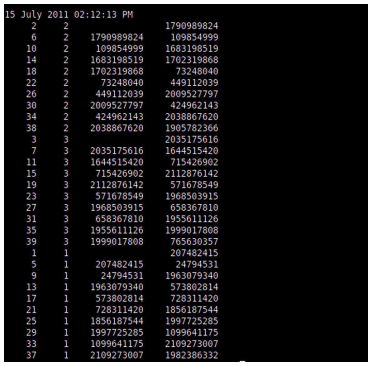

Figura 6.15: O Gerador de Números Aleatórios - 4 Processadores. Envolvimento de Todos os Nós.

É possível, então, perceber a distribuição dos processos entre os nós do *cluster* pela

ferramenta MPI. Quando há um grande número de processadores envolvidos e uma carga pesada de cálculos, esta característica fica mais evidente e é melhor aproveitada, podendo-se, assim, obter um ganho considerável na redução do tempo da tarefa.

7

Considerações Finais

7.1 Conclusões

O estudo dos *clusters* possibilita-nos o entendimento do ambiente paralelo e ainda uma visão geral das bibliotecas envolvidas, das linguagens de programação e dos padrões que já foram desenvolvidos e utilizados. É interessante perceber o crescimento do poder de processamento dos computadores ao longo do tempo e é importante perceber que grandes problemas computacionais não são solucionados apenas por grandes máquinas específicas. Existem outras possibilidades para o estudo e a aplicação de ambientes de capacidade e paralelos, como pela utilização de *hardware* comum e de *software* distribuído livremente.

Clusters Beowulf já foram utilizados para grandes tarefas, como a renderização do filme Titanic (1997), de James Cameron, o que nos faz enxergar a sua aplicabilidade não apenas em tarefas e ambientes científicos ou matemáticos.

Um dos grandes problemas enfrentados pelas bibliotecas livres é que o seu desenvolvimento depende de muitas pessoas que talvez não estejam apenas voltadas para o seu melhoramento, o que faz com que a maioria do material de estudo, dos exemplos disponíveis e dos seus recursos sejam escassos e ultrapassados. Esta foi uma das maiores dificuldades encontradas para a elaboração deste trabalho.

Além do ambiente corporativo, é de grande relevância o aprendizado acerca desta tecnologia por estudantes interessados na programação paralela e na solução de problemas com-

putacionais, visto que o custo envolvido é muito baixo, podendo-se utilizar PCs disponíveis e programas de código livre. Como feito para a elaboração deste texto, para as Universidades a montagem de um *cluster* é uma ideia interessante, do ponto de vista educacional e na redução do tempo de processamento em códigos pesados.

7.2 Propostas Futuras

Uma das bibliotecas disponíveis e que pode ser instalada com grande valia em ambientes acadêmicos, como os de engenharia, é a do POV-RAY, que tem como fim a renderização de imagens a partir do cálculo dos pontos de luz e de reflexão um por um, tomando uma proporção muito realista do cenário.

Além disso, a utilização de padrões mais atuais, como o MPICH-2 e o LAM/MPI, são muito importantes, já que muitos recursos foram melhorados e vários outros foram incluídos desde a versão 1.2.7p1 do MPICH-2. A versão do LAM/MPI continua a mesma desde 2007, porém vários ambientes gráficos podem ser incluídos à sua instalação, como o XMTV (servidor gráfico) e o XMPI (ambiente gráfico para execução e depuração de códigos a partir do MPI).

Uma outra implementação de grande valor é a das ferramentas de teste e de *benchmark* de ambientes paralelos, como a LINPACK, utilizada pelo TOP500.org para apuração do poder computacional de cada um dos sítios onde estão montados grandes máquinas paralelas.

Referências Bibliográficas

Batista, A. C. (2007) Estudo Teórico Sobre Cluster Linux, Monografia de pós-graduação "lato sensu", UFLA, Lavras - Minas Gerais.

ORNL e CSM (2010) PVM: Parallel Virtual Machine, Site de Internet. Disponível em <http://www.csm.ornl.gov/pvm/>. Acessado em julho de 2011.

Packages, U. (2011a) "daemon"e programas utilitários do "Network Time Protocol", Site de Internet. Disponível em <http://packages.ubuntu.com/natty/ntp>. Acessado em março de 2011.

Packages, U. (2011b) The OpenBSD Internet Superserver, Site de Internet. Disponível em <http://packages.ubuntu.com/hardy/openbsd-inetd>. Acessado em março de 2011.

Packages, U. (2011c) Pacote de transição manequim autofs para autofs5, Site de Internet. Disponível em <http://packages.ubuntu.com/natty/autofs>. Acessado em março de 2011.

Packages, U. (2011d) Secure shell (SSH) server, for secure access from remote machines, Site de Internet. Disponível em <http://packages.ubuntu.com/natty/openssh-server>. Acessado em março de 2011.

Packages, U. (2011e) Suporte para o servidor NFS do núcleo ("kernel"), Site de Internet. Disponível em <http://packages.ubuntu.com/natty/nfs-kernel-server>. Acessado em março de 2011.

Packages, U. (2011f) SysV init runlevel configuration tool for the terminal, Site de Internet. Disponível em <http://packages.ubuntu.com/natty/sysv-rc-conf>. Acessado em março de 2011.

Pereira, R. E. (2008) Linux: guia do administrador do sistema, Novatec Editora, 2º edic..

Pitanga, M. (2008) Construindo Supercomputadores com Linux, Brasport, 3º edic..

Rocha, J. M. G. (2003) Cluster Beowulf: Aspectos de Projeto e Implementação, Dissertação de mestrado, UFPA.

TOP500 (2011) TOP 500 Supercomputer Sites, Site de Internet. Disponível em <http://www.top500.org>. Acessado em março de 2011.

Wikipedia (2011) Linear congruential generator, Site de Internet. Disponível em <http://en.wikipedia.org/wiki/Linear_congruential_generator>. Acessado em julho de 2011.

A

Códigos Fonte

A.1 palive

```
#!/bin/bash
/usr/bin/clear
/bin/rm -f  /etc/beowulf/ativos  2>  /dev/null
/bin/touch  /etc/beowulf/ativos

   for node in ‘sed ’/#/d’ /etc/beowulf/clusternodes |cat‘
   do
        ping -c2 $node
echo ""
        if [  $? -gt  0 ]
        then
           echo "O $node nao esta respondendo"
        else
           echo "O $node estÃ¡ ativo"
   echo "$node" >> /etc/beowulf/ativos
   echo ""
fi
```

```
   done
echo ""
echo "Os nos ativos sÃ£o:"
echo "------------------"
cat /etc/beowulf/ativos
```

A.2 pload

```
#!/bin/bash
/usr/bin/clear
/bin/rm -f /etc/beowulf/carga  2>  /dev/null
/bin/touch /etc/beowulf/carga
carga="top -n1 -b |grep \"load average:\"|cut -c 11-"
echo "A carga do sistema Ã©:"
echo "---------------------"
for node in ‘sed ’/#/d’ /etc/beowulf/clusternodes | cat‘
 do
rsh "$node" "$carga" >&1 2>&1|sed "s/^/"‘echo $node|cut -f 1 -d ’.’|cut -c -10‘:" /g"
/etc/beowulf/carga
tail -n 1 /etc/beowulf/carga
 done
```

A.3 ploadu

```
#!/bin/bash
/usr/bin/clear
param1=$1
/bin/rm -f /etc/beowulf/cargau  2>  /dev/null
/bin/touch /etc/beowulf/cargau
carga1="top -n1 -b |grep"
carga2="|head -n1|cut -c -14,45-"
echo "NODE PID USER %CPU %MEM  TIME COMMAND"
```

```
echo "----------------------------------------"

for node in ‘sed ’/#/d’ /etc/beowulf/clusternodes | cat‘
 do
  rsh $node $carga1 $param1 $carga2 >&1 2>&1 | \
  sed "s/^/"‘echo $node|cut -f 1 -d ’.’|cut -c -10‘:" /g" >> /etc/beowulf/cargau
  tail -n 1 /etc/beowulf/cargau
 done
```

A.4 pmem

```
#!/bin/bash
/usr/bin/clear
/bin/rm -f /etc/beowulf/ram  2>  /dev/null
/bin/touch /etc/beowulf/ram
mem="top -n1 -b |grep \"Mem:\"|cut -c -47"
echo "A memoria RAM usada pelo sistema e"
echo "---------------------------------"

for node in ‘sed ’/#/d’ /etc/beowulf/clusternodes | cat‘
 do
  rsh "$node" "$mem" >&1 2>&1|sed "s/^/"‘echo $node|cut -f 1 -d ’.’|cut -c -10‘:"
/g" >> \
  /etc/beowulf/ram
  tail -n 1 /etc/beowulf/ram
 done
```

A.5 Cálculo de Pi - cpi.c

```
#include "mpi.h"
#include <stdio.h>
```

```
#include <math.h>


double f( double );
double f( double a )
{
    return (4.0 / (1.0 + a*a));
}

int main( int argc, char *argv[])
{
    int done = 0, n, myid, numprocs, i;
    double PI25DT = 3.14159265358979323846264 3;
    double mypi, pi, h, sum, x;
    double startwtime = 0.0, endwtime;
    int  namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
    MPI_Comm_rank(MPI_COMM_WORLD,&myid);
    MPI_Get_processor_name(processor_name,&namelen);

    fprintf(stderr,"Process %d on %s\n",
    myid, processor_name);

    n = 0;
    while (!done)
    {
        if (myid == 0)
        {
/*
```

```
            printf("Enter the number of intervals: (0 quits) ");
            scanf("%d",&n);
*/
    if (n==0) n=100; else n=0;

    startwtime = MPI_Wtime();
        }
        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
        if (n == 0)
            done = 1;
        else
        {
            h   = 1.0 / (double) n;
            sum = 0.0;
            for (i = myid + 1; i <= n; i += numprocs)
            {
                x = h * ((double)i - 0.5);
                sum += f(x);
            }
            mypi = h * sum;

            MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

            if (myid == 0)
    {
                printf("pi is approximately %.16f, Error is %.16f\n",
                       pi, fabs(pi - PI25DT));
endwtime = MPI_Wtime();
printf("wall clock time = %f\n",
       endwtime-startwtime);
    }
        }
```

```
    }
    MPI_Finalize();

    return 0;
}
```

A.6 *Hello, World*

```
# include <stdlib.h>
# include <stdio.h>

# include "mpi.h"

int main ( int argc, char *argv[] );

/******************************************************************************/

int main ( int argc, char *argv[] )

/******************************************************************************/
/*
  Purpose:

    MAIN is the main program for HELLO_MPI.

  Discussion:

    This is a simple MPI test program.

    Each process prints out a "Hello, world!" message.

    The master process also prints out a short message.
```

```
  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    30 October 2008

  Author:

    John Burkardt

  Reference:

    William Gropp, Ewing Lusk, Anthony Skjellum,
    Using MPI: Portable Parallel Programming with the
    Message-Passing Interface,
    Second Edition,
    MIT Press, 1999,
    ISBN: 0262571323,
    LC: QA76.642.G76.
*/
{
  int id;
  int ierr;
  int p;
  double wtime;
/*
  Initialize MPI.
*/
  ierr = MPI_Init ( &argc, &argv );
```

```
/*
  Get the number of processes.
*/
  ierr = MPI_Comm_size ( MPI_COMM_WORLD, &p );
/*
  Get the individual process ID.
*/
  ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &id );
/*
  Process 0 prints an introductory message.
*/
  if ( id == 0 )
  {
    wtime = MPI_Wtime ( );

    printf ( "\n" );
    printf ( "HELLO_MPI - Master process:\n" );
    printf ( "  C/MPI version\n" );
    printf ( "  An MPI example program.\n" );
    printf ( "\n" );
    printf ( "  The number of processes is %d.\n", p );
    printf ( "\n" );
  }
/*
  Every process prints a hello.
*/
  printf ( "  Process %d says 'Hello, world!'\n", id );
/*
  Process 0 says goodbye.
*/
  if ( id == 0 )
  {
```

```
    printf ( "\n" );
    printf ( "HELLO_MPI - Master process:\n" );
    printf ( "  Normal end of execution: 'Goodbye, world!'\n" );

    wtime = MPI_Wtime ( ) - wtime;
    printf ( "\n" );
    printf ( "  Elapsed wall clock time = %f seconds.\n", wtime );
  }
/*
  Shut down MPI.
*/
  ierr = MPI_Finalize ( );

  return 0;
}
```

A.7 Cálculo dos Primos - prime_MPI.c

```
# include <stdlib.h>
# include <stdio.h>
# include <math.h>
# include <time.h>

# include "mpi.h"

int main ( int argc, char *argv[] );
int prime_number ( int n, int id, int p );
void timestamp ( void );

/******************************************************************************/

int main ( int argc, char *argv[] )
```

```
/******************************************************************************/
/*
  Purpose:

    MAIN is the main program for PRIME_MPI.

  Discussion:

    This program calls a version of PRIME_NUMBER that includes
    MPI calls for parallel processing.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    07 August 2009

  Author:

    John Burkardt
*/
{
  int i;
  int id;
  int ierr;
  int master = 0;
  int n;
  int n_factor;
  int n_hi;
```

```
  int n_lo;
  int p;
  int primes;
  int primes_part;
  double wtime;

  n_lo = 1;
  n_hi = 131072;
  n_factor = 2;
/*
  Initialize MPI.
*/
  ierr = MPI_Init ( &argc, &argv );
/*
  Get the number of processes.
*/
  ierr = MPI_Comm_size ( MPI_COMM_WORLD, &p );
/*
  Determine this processes's rank.
*/
  ierr = MPI_Comm_rank ( MPI_COMM_WORLD, &id );

  if ( id == master )
  {
    printf ( "\n" );
    printf ( "'PRIME_MPI\n" );
    printf ( "  C/MPI version\n" );
    printf ( "\n" );
    printf ( "  An MPI example program to count the number of primes.\n" );
    printf ( "  The number of processes is %d\n", p );
    printf ( "\n" );
    printf ( "         N        Pi          Time\n" );
```

```
    printf ( "\n" );
  }

  n = n_lo;

  while ( n <= n_hi )
  {
    if ( id == master )
    {
      wtime = MPI_Wtime ( );
    }
    ierr = MPI_Bcast ( &n, 1, MPI_INT, master, MPI_COMM_WORLD );

    primes_part = prime_number ( n, id, p );

    ierr = MPI_Reduce ( &primes_part, &primes, 1, MPI_INT, MPI_SUM, master,
      MPI_COMM_WORLD );

    if ( id == master )
    {
      wtime = MPI_Wtime ( ) - wtime;
      printf ( "  %8d  %8d  %14f\n", n, primes, wtime );
    }
    n = n * n_factor;
  }
/*
  Shut down MPI.
*/
  ierr = MPI_Finalize ( );

  if ( id == master )
  {
```

```
    printf ( "\n");
    printf ( "PRIME_MPI - Master process:\n");
    printf ( "  Normal end of execution.\n");
  }

  return 0;
}
/******************************************************************************/

int prime_number ( int n, int id, int p )

/******************************************************************************/
/*
  Purpose:

    PRIME_NUMBER returns the number of primes between 1 and N.

  Discussion:

    In order to divide the work up evenly among P processors, processor
    ID starts at 2+ID and skips by P.

    A naive algorithm is used.

    Mathematica can return the number of primes less than or equal to N
    by the command PrimePi[N].

                N  PRIME_NUMBER

                1           0
               10           4
              100          25
```

```
            1,000         168
           10,000       1,229
          100,000       9,592
        1,000,000      78,498
       10,000,000     664,579
      100,000,000   5,761,455
    1,000,000,000  50,847,534

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    21 May 2009

  Author:

    John Burkardt

  Parameters:

    Input, int N, the maximum number to check.

    Input, int ID, the ID of this process,
    between 0 and P-1.

    Input, int P, the number of processes.

    Output, int PRIME_NUMBER, the number of prime numbers up to N.
*/
{
```

```
  int i;
  int j;
  int prime;
  int total;

  total = 0;

  for ( i = 2 + id; i <= n; i = i + p )
  {
    prime = 1;
    for ( j = 2; j < i; j++ )
    {
      if ( ( i % j ) == 0 )
      {
        prime = 0;
        break;
      }
    }
    total = total + prime;
  }
  return total;
}
/******************************************************************************/

void timestamp ( void )

/******************************************************************************/
/*
  Purpose:

    TIMESTAMP prints the current YMDHMS date as a time stamp.
```

```
  Example:

    31 May 2001 09:45:54 AM

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    24 September 2003

  Author:

    John Burkardt

  Parameters:

    None
*/
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;

  now = time ( NULL );
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );
```

```
  printf ( "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}
```

A.8 Gerador de Números Pseudoaleatórios

```
# include <stdlib.h>
# include <stdio.h>
# include <time.h>
# include <math.h>

# include "mpi.h"

int main ( int argc, char *argv[] );
int congruence ( int a, int b, int c, int *error );
int i4_gcd ( int i, int j );
int i4_max ( int i1, int i2 );
int i4_min ( int i1, int i2 );
int i4_sign ( int i );
void lcrg_anbn ( int a, int b, int c, int n, int *an, int *bn );
int lcrg_evaluate ( int a, int b, int c, int x );
int power_mod ( int a, int n, int m );
void timestamp ( void );

/******************************************************************************/

int main ( int argc, char *argv[] )
```

```
/******************************************************************************/
/*
  Purpose:

    MAIN is the main program for RANDOM_MPI.

  Discussion:

    This program demonstrates how P processors can generate the same
    sequence of random numbers as 1 processor.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    21 May 2008

  Author:

    John Burkardt

  Reference:

    William Gropp, Ewing Lusk, Anthony Skjellum,
    Using MPI: Portable Parallel Programming with the
    Message-Passing Interface,
    Second Edition,
    MIT Press, 1999,
    ISBN: 0262571323,
    LC: QA76.642.G76.
```

```
*/
{
  int a;
  int an;
  int b;
  int bn;
  int c;
  int error;
  int id;
  int j;
  int k;
  int k_hi;
  int p;
  int u;
  int v;
/*
  Initialize MPI.
*/
  MPI_Init ( &argc, &argv );
/*
  Get the number of processors.
*/
  MPI_Comm_size ( MPI_COMM_WORLD, &p );
/*
  Get the rank of this processor.
*/
  MPI_Comm_rank ( MPI_COMM_WORLD, &id );
/*
  Print a message.
*/
  if ( id == 0 )
  {
```

```
    timestamp ( );
    printf ( "\n" );
    printf ( "RANDOM_MPI - Master process:\n" );
    printf ( "  C version\n" );
    printf ( "  The number of processors is P = %d\n", p );
    printf ( "\n" );
    printf ( "  This program shows how a stream of random numbers\n" );
    printf ( "  can be computed 'in parallel' in an MPI program.\n" );
    printf ( "\n" );
    printf ( "  We assume we are using a linear congruential\n" );
    printf ( "  random number generator or LCRG, which takes\n" );
    printf ( "  an integer input and returns a new integer output:\n" );
    printf ( "\n" );
    printf ( "    U = ( A * V + B ) mod C\n" );
    printf ( "\n" );
    printf ( "  We assume that we want the MPI program to produce\n" );
    printf ( "  the same sequence of random values as a sequential\n" );
    printf ( "  program would - but we want each processor to compute\n" );
    printf ( "  one part of that sequence.\n" );
    printf ( "\n" );
    printf ( "  We do this by computing a new LCRG which can compute\n" );
    printf ( "  every P'th entry of the original one.\n" );
    printf ( "\n" );
    printf ( "  Our LCRG works with integers, but it is easy to\n" );
    printf ( "  turn each integer into a real number between [0,1].\n" );
  }
/*
  A, B and C define the linear congruential random number generator.
*/
  a = 16807;
  b = 0;
  c = 2147483647;
```

```
  if ( id == 0 )
  {
    printf ( "\n" );
    printf ( "  LCRG parameters:\n" );
    printf ( "\n" );
    printf ( "  A  = %d\n", a );
    printf ( "  B  = %d\n", b );
    printf ( "  C  = %d\n", c );
  }

  k_hi = p * 10;
/*
  Processor 0 generates 10 * P random values.
*/
  if ( id == 0 )
  {
    printf ( "\n" );
    printf ( "  Let processor 0 generate the entire random number sequence.\n" );
    printf ( "\n" );
    printf ( "     K    ID         Input        Output\n" );
    printf ( "\n" );

    k = 0;
    v = 12345;
    printf ( "  %4d  %4d                %12d\n", k, id, v );

    for ( k = 1; k <= k_hi; k++ )
    {
      u = v;
      v = lcrg_evaluate ( a, b, c, u );
      printf ( "  %4d  %4d  %12d  %12d\n", k, id, u, v );
```

```
    }
  }
/*
  Processor P now participates by computing the P-th part of the sequence.
*/
  lcrg_anbn ( a, b, c, p, &an, &bn );

  if ( id == 0 )
  {
    printf ( "\n" );
    printf ( "  LCRG parameters for P processors:\n" );
    printf ( "\n" );
    printf ( "  AN = %d\n", an );
    printf ( "  BN = %d\n", bn );
    printf ( "  C  = %d\n", c );
    printf ( "\n" );
    printf ( "  Have ALL the processors participate in computing\n" );
    printf ( "  the same random number sequence.\n" );
    printf ( "\n" );
    printf ( "     K    ID         Input        Output\n" );
    printf ( "\n" );
  }
/*
  Use the basis LCRG to get the ID-th value in the sequence.
*/
  v = 12345;
  for ( j = 1; j <= id; j++ )
  {
    u = v;
    v = lcrg_evaluate ( a, b, c, u );
  }
  k = id;
```

```
  printf ( "  %4d  %4d                %12d\n", k, id, v );
/*
  Now use the "skipping" LCRG to compute the values with indices
  ID, ID+P, ID+2P, ...,
*/
  for ( k = id + p; k <= k_hi; k = k + p )
  {
    u = v;
    v = lcrg_evaluate ( an, bn, c, u );
    printf ( "  %4d  %4d  %12d  %12d\n", k, id, u, v );
  }

  if ( id == 0 )
  {
    printf ( "\n" );
    printf ( "RANDOM_MPI:\n" );
    printf ( "  Normal end of execution.\n" );
    printf ( "\n" );
    timestamp ( );
  }
/*
  Shut down MPI.
*/
  MPI_Finalize ( );

  return 0;
}
/******************************************************************************/

int congruence ( int a, int b, int c, int *error )
```

```
/******************************************************************************/
/*
  Purpose:

    CONGRUENCE solves a congruence of the form A * X = C ( mod B ).

  Discussion:

    A, B and C are given integers.  The equation is solvable if and only
    if the greatest common divisor of A and B also divides C.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    15 November 2004

  Author:

    John Burkardt

  Reference:

    Eric Weisstein, editor,
    CRC Concise Encylopedia of Mathematics,
    CRC Press, 1998, page 446.

  Parameters:

    Input, int A, B, C, the coefficients of the Diophantine equation.
```

```
    Output, int *ERROR, error flag, is 1 if an error occurred..

    Output, int CONGRUENCE, the solution of the Diophantine equation.
    X will be between 0 and B-1.
*/
{
# define N_MAX 100

  int a_copy;
  int a_mag;
  int a_sign;
  int b_copy;
  int b_mag;
  int b_sign;
  int c_copy;
  int g;
  int k;
  int n;
  float norm_new;
  float norm_old;
  int q[N_MAX];
  int swap;
  int temp;
  int x;
  int xnew;
  int y;
  int ynew;
  int z;
/*
  Defaults for output parameters.
*/
```

```
  *error = 0;
  x = 0;
  y = 0;
/*
  Special cases.
*/
  if ( a == 0 && b == 0 && c == 0 )
  {
    x = 0;
    return x;
  }
  else if ( a == 0 && b == 0 && c != 0 )
  {
    *error = 1;
    x = 0;
    return x;
  }
  else if ( a == 0 && b != 0 && c == 0 )
  {
    x = 0;
    return x;
  }
  else if ( a == 0 && b != 0 && c != 0 )
  {
    x = 0;
    if ( ( c % b ) != 0 )
    {
      *error = 2;
    }
    return x;
  }
  else if ( a != 0 && b == 0 && c == 0 )
```

```
  {
    x = 0;
    return x;
  }
  else if ( a != 0 && b == 0 && c != 0 )
  {
    x = c / a;
    if ( ( c % a ) != 0 )
    {
      *error = 3;
      return x;
    }
    return x;
  }
  else if ( a != 0 && b != 0 && c == 0 )
  {
/*  g = i4_gcd ( a, b ); */
/*  x = b / g; */
    x = 0;
    return x;
  }
/*
  Now handle the "general" case: A, B and C are nonzero.

  Step 1: Compute the GCD of A and B, which must also divide C.
*/
  g = i4_gcd ( a, b );

  if ( ( c % g ) != 0 )
  {
    *error = 4;
    return x;
```

```
  }

  a_copy = a / g;
  b_copy = b / g;
  c_copy = c / g;
/*
  Step 2: Split A and B into sign and magnitude.
*/
  a_mag = abs ( a_copy );
  a_sign = i4_sign ( a_copy );
  b_mag = abs ( b_copy );
  b_sign = i4_sign ( b_copy );
/*
  Another special case, A_MAG = 1 or B_MAG = 1.
*/
  if ( a_mag == 1 )
  {
    x = a_sign * c_copy;
    return x;
  }
  else if ( b_mag == 1 )
  {
    x = 0;
    return x;
  }
/*
  Step 3: Produce the Euclidean remainder sequence.
*/
  if ( b_mag <= a_mag )
  {
    swap = 0;
    q[0] = a_mag;
```

```
    q[1] = b_mag;
  }
  else
  {
    swap = 1;
    q[0] = b_mag;
    q[1] = a_mag;
  }

  n = 3;

  for ( ; ; )
  {
    q[n-1] = ( q[n-3] % q[n-2] );

    if ( q[n-1] == 1 )
    {
      break;
    }

    n = n + 1;

    if ( N_MAX < n )
    {
      *error = 1;
      printf ( "\n" );
      printf ( "CONGRUENCE - Fatal error!\n" );
      printf ( "  Exceeded number of iterations.\n" );
      exit ( 1 );
    }
  }
/*
```

```
  Step 4: Now go backwards to solve X * A_MAG + Y * B_MAG = 1.
*/
  y = 0;
  for ( k = n; 2 <= k; k-- )
  {
    x = y;
    y = ( 1 - x * q[k-2] ) / q[k-1];
  }
/*
  Step 5: Undo the swapping.
*/
  if ( swap == 1 )
  {
    z = x;
    x = y;
    y = z;
  }
/*
  Step 6: Now apply signs to X and Y so that X * A + Y * B = 1.
*/
  x = x * a_sign;
/*
  Step 7: Multiply by C, so that X * A + Y * B = C.
*/
  x = x * c_copy;
/*
  Step 8: Now force 0 <= X < B.
*/
  x = x % b;
/*
  Step 9: Force positivity.
*/
```

```
  if ( x < 0 )
  {
    x = x + b;
  }

  return x;
# undef N_MAX
}
/******************************************************************************/

int i4_gcd ( int i, int j )

/******************************************************************************/
/*
  Purpose:

    I4_GCD finds the greatest common divisor of I and J.

  Discussion:

    Only the absolute values of I and J are considered, so that the
    result is always nonnegative.

    If I or J is 0, I4_GCD is returned as max ( 1, abs ( I ), abs ( J ) ).

    If I and J have no common factor, I4_GCD is returned as 1.

    Otherwise, using the Euclidean algorithm, I4_GCD is the
    largest common factor of I and J.

  Licensing:
```

```
    This code is distributed under the GNU LGPL license.

  Modified:

    07 May 2003

  Author:

    John Burkardt

  Parameters:

    Input, int I, J, two numbers whose greatest common divisor
    is desired.

    Output, int I4_GCD, the greatest common divisor of I and J.
*/
{
  int ip;
  int iq;
  int ir;
/*
  Return immediately if either I or J is zero.
*/
  if ( i == 0 )
  {
    return i4_max ( 1, abs ( j ) );
  }
  else if ( j == 0 )
  {
    return i4_max ( 1, abs ( i ) );
  }
```

```
/*
  Set IP to the larger of I and J, IQ to the smaller.
  This way, we can alter IP and IQ as we go.
*/
  ip = i4_max ( abs ( i ), abs ( j ) );
  iq = i4_min ( abs ( i ), abs ( j ) );
/*
  Carry out the Euclidean algorithm.
*/
  for ( ; ; )
  {
    ir = ip % iq;

    if ( ir == 0 )
    {
      break;
    }

    ip = iq;
    iq = ir;
  }

  return iq;
}
/******************************************************************************/

int i4_max ( int i1, int i2 )

/******************************************************************************/
/*
  Purpose:
```

```
    I4_MAX returns the maximum of two I4's.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 August 2006

  Author:

    John Burkardt

  Parameters:

    Input, int I1, I2, are two integers to be compared.

    Output, int I4_MAX, the larger of I1 and I2.
*/
{
  int value;

  if ( i2 < i1 )
  {
    value = i1;
  }
  else
  {
    value = i2;
  }
  return value;
```

```
}
/******************************************************************************/

int i4_min ( int i1, int i2 )

/******************************************************************************/
/*
  Purpose:

    I4_MIN returns the smaller of two I4's.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    29 August 2006

  Author:

    John Burkardt

  Parameters:

    Input, int I1, I2, two integers to be compared.

    Output, int I4_MIN, the smaller of I1 and I2.
*/
{
  int value;
```

```
  if ( i1 < i2 )
  {
    value = i1;
  }
  else
  {
    value = i2;
  }
  return value;
}
/******************************************************************************/

int i4_sign ( int i )

/******************************************************************************/
/*
  Purpose:

    I4_SIGN returns the sign of an I4.

  Discussion:

    The sign of 0 and all positive integers is taken to be +1.
    The sign of all negative integers is -1.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    06 May 2003
```

```
  Author:

    John Burkardt

  Parameters:

    Input, int I, the integer whose sign is desired.

    Output, int I4_SIGN, the sign of I.
*/
{
  int value;

  if ( i < 0 )
  {
    value = - 1;
  }
  else
  {
    value = 1;
  }
  return value;
}
/******************************************************************************/

void lcrg_anbn ( int a, int b, int c, int n, int *an, int *bn )

/******************************************************************************/
/*
  Purpose:
```

```
  LCRG_ANBN computes the "N-th power" of a linear congruential generator.

Discussion:

  We are considering a linear congruential random number generator.
  The LCRG takes as input an integer value called SEED, and returns
  an updated value of SEED,

    SEED(out) = ( a * SEED(in) + b ) mod c.

  and an associated pseudorandom real value

    U = SEED(out) / c.

  In most cases, a user is content to call the LCRG repeatedly, with
  the updating of SEED being taken care of automatically.

  The purpose of this routine is to determine the values of AN and BN
  that describe the LCRG that is equivalent to N applications of the
  original LCRG.

  One use for such a facility would be to do random number computations
  in parallel.  If each of N processors is to compute many random values,
  you can guarantee that they work with distinct random values
  by starting with a single value of SEED, using the original LCRG to generate
  the first N-1 "iterates" of SEED, so that you now have N "seed" values,
  and from now on, applying the N-th power of the LCRG to the seeds.

  If the K-th processor starts from the K-th seed, it will essentially
  be computing every N-th entry of the original random number sequence,
  offset by K.  Thus the individual processors will be using a random
  number stream as good as the original one, and without repeating, and
```

```
without having to communicate.

To evaluate the N-th value of SEED directly, we start by ignoring
the modular arithmetic, and working out the sequence of calculations
as follows:

  SEED(0)   =     SEED.
  SEED(1)   = a * SEED      + b
  SEED(2)   = a * SEED(1)   + b = a^2 * SEED           + a * b + b
  SEED(3)   = a * SEED(2)   + b = a^3 * SEED + a^2 * b + a * b + b
  ...
  SEED(N-1) = a * SEED(N-2) + b

  SEED(N) = a * SEED(N-1) + b = a^N * SEED
                              + ( a^(n-1) + a^(n-2) + ... + a + 1 ) * b

or, using the geometric series,

  SEED(N) = a^N * SEED + ( a^N - 1) / ( a - 1 ) * b
          = AN * SEED + BN

Thus, from any SEED, we can determine the result of N applications of the
original LCRG directly if we can solve

  ( a - 1 ) * BN = ( a^N - 1 ) * b in modular arithmetic,

and evaluate:

  AN = a^N

Licensing:
```

```
  This code is distributed under the GNU LGPL license.

Modified:

  23 April 2008

Author:

  John Burkardt

Reference:

  Barry Wilkinson, Michael Allen,
  Parallel Programming:
  Techniques and Applications Using Networked Workstations and Parallel Computers,
  Prentice Hall,
  ISBN: 0-13-140563-2,
  LC: QA76.642.W54.

Parameters:

  Input, int A, the multiplier for the LCRG.

  Input, int  B, the added value for the LCRG.

  Input, int  C, the base for the modular arithmetic.
  For 32 bit arithmetic, this is often 2^31 - 1, or 2147483647.  It is
  required that 0 < C.

  Input, int N, the "index", or number of times that the
  LCRG is to be applied.  It is required that 0 <= N.
```

```
    Output, int *AN, *BN, the multiplier and added value for
    the LCRG that represent N applications of the original LCRG.
*/
{
  int am1;
  int anm1tb;
  int ierror;

  if ( n < 0 )
  {
    printf ( "\n" );
    printf ( "LCRG_ANBN - Fatal error!\n" );
    printf ( "  Illegal input value of N = %d\n", n );
    exit ( 1 );
  }

  if ( c <= 0 )
  {
    printf ( "\n" );
    printf ( "LCRG_ANBN - Fatal error!\n" );
    printf ( "  Illegal input value of C = %d\n", c );
    exit ( 1 );
  }

  if ( n == 0 )
  {
    *an = 1;
    *bn = 0;
  }
  else if ( n == 1 )
  {
    *an = a;
```

```
    *bn = b;
  }
  else
  {
/*
  Compute A^N.
*/
    *an = power_mod ( a, n, c );
/*
  Solve
    ( a - 1 ) * BN = ( a^N - 1 ) mod B
  for BN.
*/
    am1 = a - 1;
    anm1tb = ( *an - 1 ) * b;

    *bn = congruence ( am1, c, anm1tb, &ierror );

    if ( ierror )
    {
      printf ( "\n" );
      printf ( "LCRG_ANBN - Fatal error!\n" );
      printf ( "  An error occurred in the CONGRUENCE routine.\n" );
      exit ( 1 );
    }
  }

  return;
}
/******************************************************************************/

int lcrg_evaluate ( int a, int b, int c, int x )
```

```
/******************************************************************************/
/*
  Purpose:

    LCRG_EVALUATE evaluates an LCRG, y = ( A * x + B ) mod C.

  Discussion:

    This routine cannot be recommended for production use.  Because we want
    to do modular arithmetic, but the base is not a power of 2, we need to
    use "double precision" integers to keep accuracy.

    If we knew the base C, we could try to avoid overflow while not changing
    precision.

    If the base C was a power of 2, we could rely on the usual properties of
    integer arithmetic on computers, in which overflow bits, which are always
    ignored, don ot actually matter.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    23 April 2008

  Author:

    John Burkardt
```

```
  Parameters:

    Input, int A, the multiplier for the LCRG.

    Input, int B, the added value for the LCRG.

    Input, int C, the base for the modular arithmetic.
    For 32 bit arithmetic, this is often 2^31 - 1, or 2147483647.  It is
    required that 0 < C.

    Input, int X, the value to be processed.

    Output, int LCRG_EVALUATE, the processed value.
*/
{
  long long int a8;
  long long int b8;
  long long int c8;
  long long int x8;
  int y;
  long long int y8;
/*
  To avoid roundoff issues, we need to go to "double precision" integers.
  (Not available on all planets.)
*/
  a8 = ( long long int ) a;
  b8 = ( long long int ) b;
  c8 = ( long long int ) c;
  x8 = ( long long int ) x;

  y8 = ( a8 * x8 + b8 ) % c8;
```

```
  y = ( int ) ( y8 );

  if ( y < 0 )
  {
    y = y + c;
  }

  return y;
}
/******************************************************************************/

int power_mod ( int a, int n, int m )

/******************************************************************************/
/*
  Purpose:

    POWER_MOD computes mod ( A^N, M ).

  Discussion:

    Some programming tricks are used to speed up the computation, and to
    allow computations in which A**N is much too large to store in a
    real word.

    First, for efficiency, the power A**N is computed by determining
    the binary expansion of N, then computing A, A^2, A^4, and so on
    by repeated squaring, and multiplying only those factors that
    contribute to A**N.

    Secondly, the intermediate products are immediately "mod'ed", which
    keeps them small.
```

```
  For instance, to compute mod ( A^13, 11 ), we essentially compute

     13 = 1 + 4 + 8

     A**13 = A * A^4 * A^8

     mod ( A**13, 11 ) = mod ( A, 11 ) * mod ( A^4, 11 ) * mod ( A^8, 11 ).

  Fermat's little theorem says that if P is prime, and A is not divisible
  by P, then ( A^(P-1) - 1 ) is divisible by P.

Licensing:

  This code is distributed under the GNU LGPL license.

Modified:

  16 November 2004

Author:

  John Burkardt

Parameters:

  Input, int A, the base of the expression to be tested.
  A should be nonnegative.

  Input, int N, the power to which the base is raised.
  N should be nonnegative.
```

```
    Input, int M, the divisor against which the expression is tested.
    M should be positive.

    Output, int POWER_MOD, the remainder when A^N is divided by M.
*/
{
  long long int a_square2;
  int d;
  long long int m2;
  int x;
  long long int x2;

  if ( a < 0 )
  {
    return -1;
  }

  if ( m <= 0 )
  {
    return -1;
  }

  if ( n < 0 )
  {
    return -1;
  }
/*
  A_SQUARE contains the successive squares of A.
*/
  a_square2 = ( long long int ) a;
  m2 = ( long long int ) m;
  x2 = ( long long int ) 1;
```

```
  while ( 0 < n )
  {
    d = n % 2;

    if ( d == 1 )
    {
      x2 = ( x2 * a_square2 ) % m2;
    }

    a_square2 = ( a_square2 * a_square2 ) % m2;
    n = ( n - d ) / 2;
  }
/*
  Ensure that 0 <= X.
*/
  while ( x2 < 0 )
  {
    x2 = x2 + m2;
  }

  x = ( int ) x2;

  return x;
}
/******************************************************************************/

void timestamp ( void )

/******************************************************************************/
/*
  Purpose:
```

```
    TIMESTAMP prints the current YMDHMS date as a time stamp.

  Example:

    31 May 2001 09:45:54 AM

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    24 September 2003

  Author:

    John Burkardt

  Parameters:

    None
*/
{
# define TIME_SIZE 40

  static char time_buffer[TIME_SIZE];
  const struct tm *tm;
  size_t len;
  time_t now;

  now = time ( NULL );
```

```
  tm = localtime ( &now );

  len = strftime ( time_buffer, TIME_SIZE, "%d %B %Y %I:%M:%S %p", tm );

  printf ( "%s\n", time_buffer );

  return;
# undef TIME_SIZE
}
```