UNIVERSIDADE DE SÃO PAULO
INSTITUTO DE FÍSICA DE SÃO CARLOS
DEPARTAMENTO DE FÍSICA E INFORMÁTICA

Aparecido Luciano Breviglieri Joioso

Aplicação de computação em grade a simulações computacionais de estruturas semicondutoras

São Carlos
2008

Aparecido Luciano Breviglieri Joioso

Aplicação de computação em grade a simulações computacionais de estruturas semicondutoras

Dissertação apresentada ao Instituto de Física de São Carlos da Universidade de São Carlos para obtenção do título de Mestre em Ciências Física Aplicada.

Área de Concentração: Física Computacional

Orientador: Prof. Dr. Guilherme Matos Sipahi

São Carlos
2008

A Elisete, Amanda e Tássia, esposa e filhas amadas e aos meus pais por tudo que doaram de suas vidas pela minha educação.
Obrigado a todos vocês!

Agradecimentos

- Ao amigo e professor Guilherme pela confiança e orientação para a conclusão deste trabalho.

- À minha esposa Elisete, sempre paciente apesar do meu mau humor em alguns dias difíceis.

- Aos meus amigos: Flávia, João, Kakuda, Sabá e Savério pelo companheirismo, amizade e risadas que sempre me alegraram.

- À Flávia, pela ajuda com as correções do texto.

- Aos amigos do Laboratório de Física Computacional, Felipe, Ivan, Marcel e Tioli pelas dicas e por seus valiosos préstimos que muito me ajudaram a concluir esse trabalho.

- Finalmente, a Deus, pois não importa o que se faça, desde que seja feito com amor e boa vontade, Ele está sempre ao lado de quem faz sua parte.

Resumo

Neste trabalho foi avaliada a utilização da *grid computing* em aspectos importantes para simulações em Física Computacional. Em particular, para aplicações de diagonalização de matrizes de grande porte.

O projeto de código aberto *Globus Toolkit* foi utilizado para comparar o desempenho da biblioteca paralela de álgebra linear ScaLAPACK em duas versões baseadas na biblioteca de passagem de mensagens, a versão tradicional MPICH e a versão desenvolvida para um ambiente de *grid computing* MPICH-G2.

Várias simulações com diagonalização de matrizes complexas de diversos tamanhos foram realizadas. Para um sistema com uma matriz de tamanho 8000 x 8000 distribuída em 8 processos, nos nós de 64 bits foi alcançado um *speedup* de 7,71 com o MPICH-G2. Este *speedup* é muito próximo do ideal que, neste caso, seria igual a 8. Foi constatado também que a arquitetura de 64 bits tem melhor desempenho que a de 32 bits nas simulações executadas para este tipo de aplicação.

Palavras chaves: *Grid*, MPICH-G2, *Globus Toolkit*, ScaLAPACK, simulações computacionais.

Abstract

This work evaluates the use of grid computing in essential issues related to Computational Physics simulations. In particular, for applications with large scale matrix diagonalization.

The Globus Toolkit open source project was used to compare the performance of the linear algebra parallel library ScaLAPACK in two different versions based on the message passing library, the traditional version MPICH and its version developed for a grid computing environment MPICH-G2.

Several simulations within large scale diagonalization of complex matrix were performed. A 7.71 speedup was reached with the MPICH-G2 for a 8000 x 8000 size matrix distributed in 8 processes on 64 bits nodes. This was very close to the ideal speedup, that would be in this case, 8. It was also evidenced that the 64 bits architecture has better performance than the 32 bits on the performed simulations for this kind of application.

Keywords: Grid, MPICH-G2, Globus Toolkit, ScaLAPACK, computational simulations.

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Lista de Gráficos

Sumário

CAPÍTULO 1

Introdução

1.1 MOTIVAÇÃO

A necessidade de poder computacional cresce dia-a-dia diante de novas pesquisas, exigindo sempre maiores capacidades de processamento, armazenamento e memória. As limitações físicas e lógicas no desenvolvimento de computadores levam os desenvolvedores em busca de novas tecnologias. Outra dificuldade é o fato de que computadores com alta capacidade computacional oneram entidades de pesquisa, as quais nem sempre dispõem de recursos suficientes para tais investimentos.

Diante deste cenário, uma tecnologia vem emergindo desde 1997, ganhando força não só em meios acadêmicos como empresariais. A *Grid Computing* ou Computação em Grade tem como filosofia o compartilhamento de recursos computacionais geograficamente dispersos. A *grid computing* não surgiu para substituir os grandes computadores paralelos e *clusters Beowulf*, pois, pelo menos, uma diferença entre essas duas tecnologias é determinante: a disponibilidade dos recursos. Neste contexto, aproveitar o potencial de computadores sem a necessidade de alocá-los fisicamente e, em conseqüência, tendo despesas reduzidas para mantê-los é no mínimo interessante.

1.2 OBJETIVOS

O Laboratório de Física Computacional (LFC) do Instituto de Física de São Carlos (IFSC) estuda materiais semicondutores realizando simulações computacionais com o auxílio

de um programa paralelo desenvolvido no próprio laboratório. Para a realização dessas simulações, esse programa efetua cálculos com matrizes de tamanhos muito grandes que podem ocupar *gigabytes* da memória RAM de um computador.

O objetivo deste trabalho foi estudar a utilização da *grid computing* e um dos seus representantes mais importantes, o *Globus Toolkit* [1], ferramenta utilizada para a construção de *grids*. Mais especificamente, o estudo foi direcionado à comparação entre o desempenho da biblioteca de passagem de mensagens desenvolvida para *grids* (MPICH-G2) com a sua versão para *cluster* (MPICH) em tarefas específicas.

Para realizar tal comparação foi utilizada a biblioteca de álgebra linear ScaLAPACK e um de seus programas-exemplo que efetua cálculos com matrizes hermitianas, aumentando o tamanho da matriz e o número de processos em que essa matriz foi distribuída em cada tomada de tempo e comparando os resultados do MPICH-G2 com o MPICH.

1.3 ORGANIZAÇÃO DESTE TRABALHO

Além da Introdução esta dissertação está organizada nos seguintes capítulos:

- **Capítulo 2** – O que é *"Grid Computing"*

  Apresenta a filosofia da *grid computing* através de uma abordagem geral sobre o assunto e da definição de sua arquitetura.

- **Capítulo 3** – *Globus Toolkit*

  Apresenta o projeto *Globus Toolkit* e seus principais componentes.

- **Capítulo 4** – MPICH-G2 e ScaLAPACK

  Apresenta as ferramentas utilizadas para a realização dos testes de desempenho.

- **Capítulo 5** – Metodologia para a execução dos testes

  Mostra como foi estruturado o Grid LFC e como foram realizados os testes de desempenho.

- **Capítulo 6** – Resultados e Discussões

  Apresenta os resultados obtidos nos testes de comparação de desempenho entre o MPICH-G2 e o MPICH.

- **Capítulo 7** – Conclusões

  Apresenta a conclusão do trabalho e perspectivas para trabalhos futuros.

CAPÍTULO 2

O que é *"Grid Computing"*

Várias definições são encontradas na literatura, no entanto, não há nenhuma que seja consenso. Segundo Ian Foster e Carl Kesselman, *Grid* é uma proposta de infra-estrutura de *software* e *hardware* para a integração de recursos computacionais, dados e pessoas geograficamente dispersas formando um ambiente colaborativo de trabalho [2]. Barker e Buyya consideram que a população da internet e a disponibilidade de computadores poderosos de alta velocidade a baixo custo fornecem a oportunidade tecnológica de usar redes como um recurso computacional unificado [3]. Finalmente, Krauter define *Grid* como um sistema computacional de rede que pode escalar ambientes do tamanho da internet com máquinas distribuídas através de múltiplas organizações e domínios administrativos [4].

Pode-se definir *Grid Computing*, mesclando as três definições acima, como sendo uma infra-estrutura de computadores e/ou qualquer tipo de recursos interconectados a uma rede, seja ela de baixa ou alta velocidade, compartilhando estes recursos entre si, todos eles sujeitos a determinadas políticas de acesso e uso, podendo ser altamente escalável, fornecendo alto desempenho e capacidade ao um custo menor do que um sistema de supercomputação.

2.1 VISÃO GERAL SOBRE *GRIDS*

O termo *grid* surgiu nos Estados Unidos de uma analogia com a rede de distribuição de eletricidade *(power grids)*. Como a energia elétrica não pode ser armazenada, ela deve ser consumida, ou será perdida. O usuário da energia elétrica a consome sem nem se preocupar

com onde ela foi produzida. Com a computação ocorre algo similar, pois não é possível armazenar os ciclos de processamento da CPU. Desta forma, enquanto o computador não está sendo utilizado, poderia efetuar processamento de qualquer tipo de informação a fim de aproveitar os ciclos ociosos da CPU, não havendo também necessidade do usuário saber onde exatamente seu processo está sendo executado. De uma maneira mais ampla, o termo *grid* pode significar compartilhamento de recursos, sejam esses recursos computadores ou qualquer tipo de dispositivo conectado à internet, sendo usado por qualquer pessoa, em qualquer local do planeta, como ilustra a Figura 2.1.

Figura 2.1 - Visão geral sobre *Grid*

O usuário que submete algum tipo de processo ao *grid*, visualizará tudo como sendo um supercomputador não importando onde os dados estão sendo processados. No caso de algum tipo de processamento paralelo, também não é relevante saber em quais recursos do *grid* estão sendo processadas as várias "partes" de seus cálculos. Ao final de todo processamento, o resultado será devolvido como se tudo tivesse sido executado em seu próprio computador.

Para controlar o acesso a todos os recursos de um *grid* e o modo como os dados são processados e compartilhados é necessária uma camada de *software* localizada entre a aplicação do usuário e o sistema operacional, chamada *middleware*.

O *grid* pode oferecer recursos que tenham dependências com relação a outros, cabendo a ele gerenciá-las. Como exemplo, pode-se citar um determinado tipo de execução que dependa de alguma classe de equipamento específico que pode não estar disponível no momento, ou seja, pode ser inviável efetuar algum processamento em algum recurso por falta de algum tipo de requisito, como memória, disco ou poder de processamento. Esses recursos são alocados por um gerenciador de recursos chamado *broker*. O funcionamento deste gerenciador de recursos é ilustrado pela Figura 2.2. Todos os recursos disponíveis são registrados pelo serviço de informação. Quando um cliente submete um processo, ele consulta o *broker* que verifica a disponibilidade e a capacidade de execução desses recursos e os aloca para o cliente efetuar a execução do seu processo.

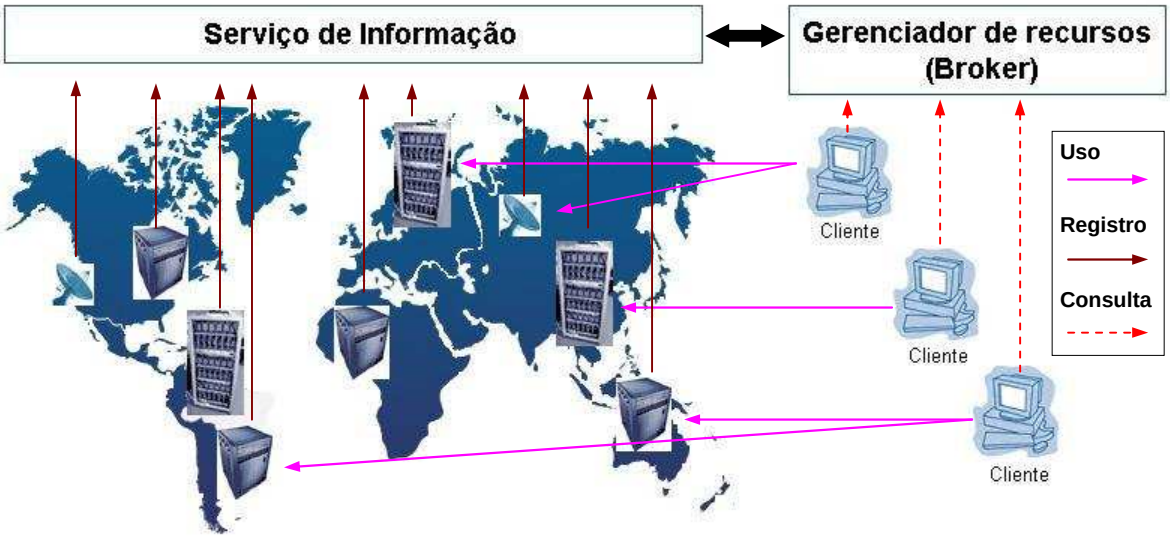

Figura 2.2 - Funcionamento do gerenciador de recursos (*broker*)

O *middleware*, de um *grid*, é todo o conjunto de *software* capaz de habilitar um *grid* e têm como principais funções organizar todos os recursos disponíveis a fim de iniciar processos nos recursos e migrar, quando necessário, para um outro recurso que se julgue mais apropriado para a execução de determinados processos. Além disto, ele estabelece a disponibilidade momentânea dos recursos e avalia se um dado processo terá condições de ser

executado em determinados recursos do *grid*. O *middleware* também deve informar aos usuários o estado de seus processos. Assim, pode-se afirmar que além do *broker*, o serviço de informação é parte integrante do *middleware*.

2.2 ORGANIZAÇÕES VIRTUAIS (VO'S)

Como a função de um *grid* é a reunião de recursos computacionais para solução de tarefas comuns, é necessário formar grupos que estejam dispostos a doar o tempo de processamento ocioso para a resolução de problemas. Tais grupos são denominados de organizações virtuais (*Virtual Organization* – VO). Uma VO pode ser um *cluster* ou um conjunto de computadores ou ainda ambos, estando sob um mesmo domínio administrativo, gerenciados por determinadas políticas que, entre outras coisas, determinam o que será compartilhado, como será compartilhado e com quem serão compartilhados os seus recursos. É importante citar que a união de VO's através dos vários modelos de redes (LAN – *Local Area Network*, MAN – *Metropolitan Area Network* e WAN – *Wide Area Network*) [5] é o que realmente forma um *grid* na sua forma plena e conceitual. Essas VO's podem estar geograficamente dispersas, dando a impressão de que a execução dos processos se dá em um supercomputador virtual. A Figura 2.3 exemplifica um *grid* com suas diversas VO's.

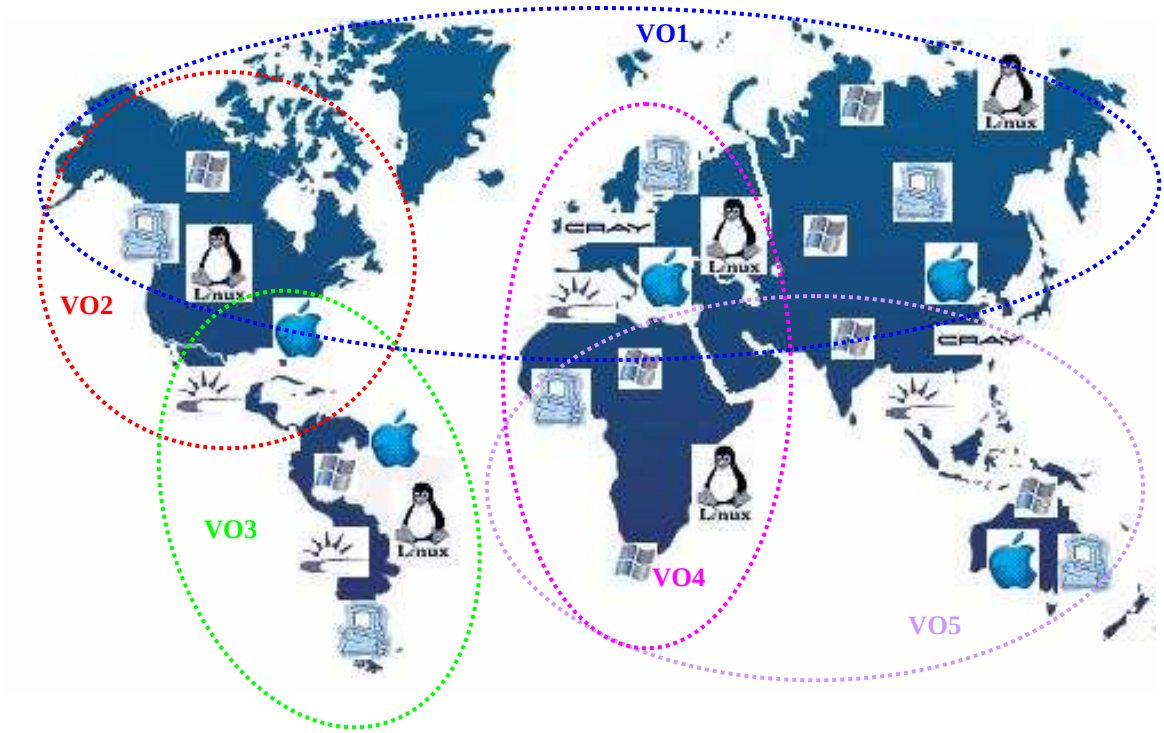

Figura 2.3 - Exemplo de um *grid* e suas VO's

2.3 CLUSTERS x GRIDS

De certa forma um *cluster* tem objetivos semelhantes a um *grid*. No entanto, a quantidade de recursos computacionais reunidas em um *cluster*, às vezes, não é suficiente para efetuar o cálculo de alguns tipos de problemas, como por exemplo: simulação de terremotos para análise de dados, simulação climática para predição do tempo, estudo do câncer, pesquisas espaciais, simulação de cálculos para semicondutores, simulação de ondas, entre outros.

Outras características importantes que diferenciam um *grid* de um *cluster* são a heterogeneidade do *hardware* e a disponibilidade dos recursos computacionais. Em um *cluster* geralmente temos vários computadores com o mesmo tipo de arquitetura, processador e memória, todos interconectados por uma rede de alta velocidade que tem toda sua banda dedicada para prover a comunicação entre os seus nós. Já em um *grid* cada recurso tem sua própria arquitetura, processador e memória e, além disso, esses recursos podem estar

geograficamente separados e interconectados através de uma rede de baixa velocidade sujeita a qualquer tipo de interferência causada por vários motivos, prejudicando assim, a troca de informações entre os recursos participantes do *grid*. Além de todas essas variáveis, também se deve considerar que cada computador (ou conjunto de computadores) que compõe o *grid* está sujeito a políticas de segurança e acesso que podem em certos momentos, inviabilizar o acesso aos recursos.

Os tipos de aplicações que podem ser executadas em um *cluster* e em um *grid* também podem ter características diferentes, uma vez que em *clusters* as aplicações típicas são aquelas com forte acoplamento, isto é, cada processo depende de outro para continuar a execução. Já em um *grid* este tipo de aplicação não é recomendável pelos vários fatores citados anteriormente. Desta forma, as aplicações ideais a serem executadas em um *grid* são aquelas com fraco acoplamento onde a dependência entre os processos seja mínima.

2.4 GRIDS FORA DO MUNDO ACADÊMICO

Grids computacionais nasceram da necessidade de computação de alto desempenho, principalmente nos meios acadêmicos. Depois, este conceito ganhou importância entre as grandes empresas de TI (Tecnologia da Informação) e, atualmente, vários produtos têm sido lançados com esta filosofia. A Tabela 2.1 apresenta uma listagem de algumas das grandes empresas de TI e suas soluções para *grids*, enquanto a Tabela 2.2 apresenta grandes empresas nacionais e multinacionais que estão empregando em alguma de suas áreas de atuação soluções baseadas em *Grid Computing*.

Tabela 2.1 - Empresas e suas soluções para *Grid*

Empresa	Produto
Sun	Sun Grid Engine
HP	Grid Storage Solutions
Oracle	Oracle Database 10g
Avaki	Sybase Avaki Studio for Data Grids
BPL Global	Smart Grid Systems
IBM	Grid Solution for Data Intensive Computing Grid and Grow Grid Medical Archive Solution
Data Synapse	GRIDServer GRIDesign
SPARSI	VirtualCore
NEC	ExpressCluster
Platform	Platform Symphony Platform Globus Toolkit Platform Enterprise Grid Orchestrator
United Devices	Grid MP

Tabela 2.2 - Aplicações de "*Grid Computing*" utilizadas em grandes empresas

Empresa	Área de atuação	Produto/Aplicação
GRSA (Grupo de Soluções em alimentação)	Alimentação	Oracle Enterprise 10 G
Lehman Brothers	Financeiro	Processamento
Fujitsu	Tecnologia	Globus Toolkit
Petrobras	Petróleo	Processamento

2.5 GRANDES PROJETOS QUE USAM A FILOSOFIA DA "GRID COMPUTING"

A idéia de usar computadores como um *grid* ganhou notoriedade pelo projeto da Universidade da Califórnia (*Berkeley*) e da NASA chamado de SETI@Home (SETI – *Search*

for ExtraTerrestrial Intelligence) que tinha como objetivo descobrir vida extraterrestre inteligente. Ficção à parte, este projeto utiliza o seguinte princípio: o usuário de forma totalmente opcional e deliberada pode instalar um *screensaver* em seu computador a fim de doar os ciclos ociosos da CPU e contribuir com o projeto. Este *screensaver* abre uma conexão com os servidores do projeto localizados em *Berkeley* para baixar uma unidade de dados e, após a obtenção dos dados, fecha a conexão. Sempre que o *screensaver* entra em operação ele efetua os cálculos necessários. Ao final dos cálculos abre uma nova conexão com os servidores enviando os dados calculados e obtendo uma nova unidade de dados para calcular. A Tabela 2.3 lista uma série de projetos com a mesma filosofia do SETI@Home bem como suas áreas de aplicação.

Tabela 2.3 - Alguns projetos mundiais que usam a filosofia da *"Grid Computing"*

Projeto	Objetivo
SETI@Home	Procura de vida extraterrestre inteligente
ChessBrain	Supercomputador virtual para jogo de xadrez
Climateprediction.net	Previsão do tempo
Distributed.net	Decifrar chaves de criptografia
FigthAIDS@Home	Pesquisas sobre AIDS
Find-a-drug	Pesquisar doenças com alto impacto na saúde
Folderol	Análise de dados do projeto Genoma
Folding@Home	Biofísica molecular
Mersenne.org	Pesquisa de números primos do tipo Mersenne
ZetaGrid	Verificação da hipótese de Riemann

É claro que uma tecnologia que permita que vários computadores se interconectem das mais variadas localizações e com os mais variados tipos de usuários leva à preocupação com alguns aspectos fundamentais [6] para a execução de suas aplicações, como ilustra a Figura 2.4.

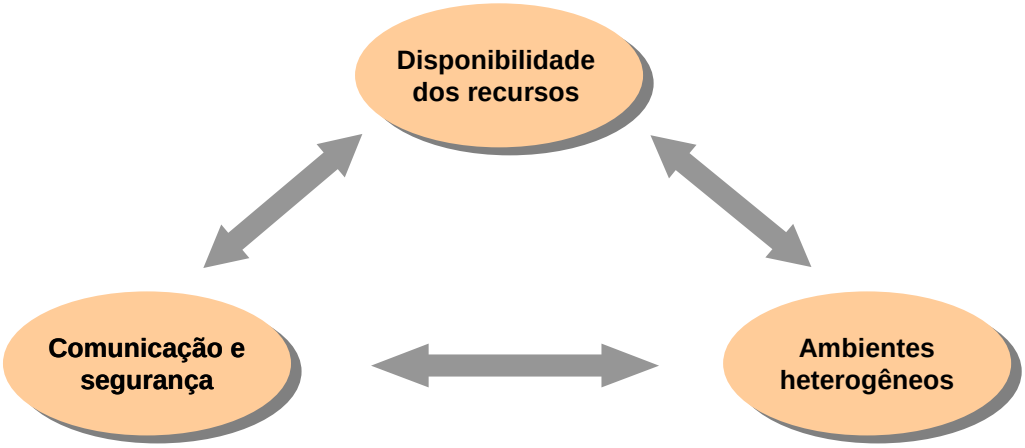

Figura 2.4 - Aspectos fundamentais para aplicações executadas em um *grid*

a) **Comunicação e segurança:** os dados que trafegam entre os componentes do *grid* devem ser protegidos. Para isso devem ser usadas técnicas de criptografia na transferência desses dados. Além disso, processos executados em um *grid* assumem um método de comunicação assíncrono, ou seja, a dependência entre os processos não deve comprometer a execução do processo como um todo;

b) **Ambientes heterogêneos:** os programas que são executados nos clientes devem ter versões para os vários tipos de arquiteturas e sistemas operacionais que compõem o *grid*, como por exemplo: as arquiteturas 32 ou 64 bits, MIPS, CISC, RISC, SPARC; e os sistemas operacionais Linux, Unix, Windows, entre outros;

c) **Disponibilidade dos recursos:** como os recursos do *grid* podem ser usados quando ociosos, não se pode prever quando os resultados esperados serão concluídos.

2.6 ARQUITETURA DE UM GRID

A arquitetura de um *grid* [7] é baseada no conceito de camadas onde cada uma provê funções específicas e tem uma forte relação com as camadas do protocolo Internet (TCP/IP) [8] como ilustra a Figura 2.5. Também é conhecida pelo modelo da ampulheta (*hourglass model*), onde o gargalo é ocupado pelas camadas de **Recursos** e **Conectividade** porque devem definir um conjunto reduzido de métodos e protocolos como TCP e HTTP que serão usados nas aplicações das camadas **Coletiva** e **Aplicação** e, conseqüentemente, facilitando o compartilhamento de recursos com um melhor aproveitamento da camada **Estrutura**. O modelo da ampulheta está representado pela Figura 2.6.

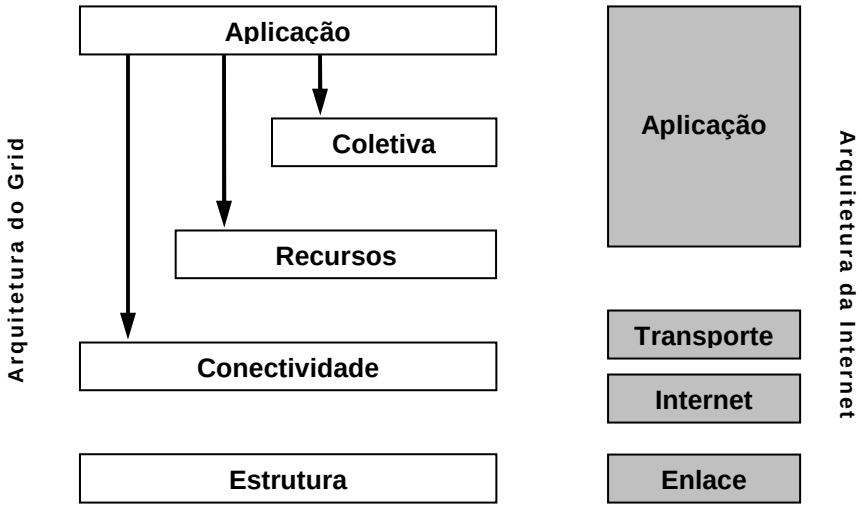

Figura 2.5 - Relação entre os modelos arquiteturais em camadas do *Grid* e Internet

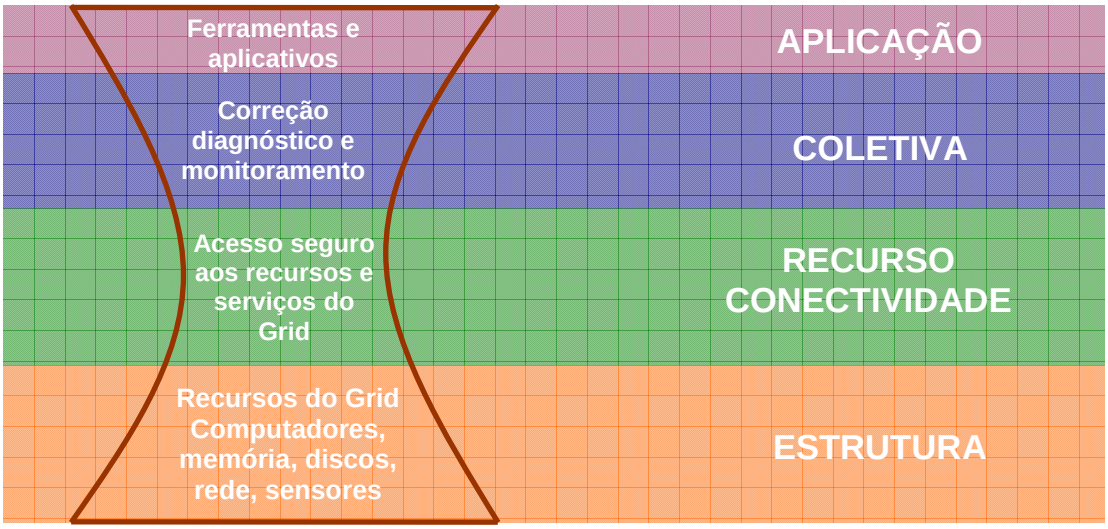

Figura 2.6 - Modelo da ampulheta (*Hourglass model*)

As camadas mais altas do modelo são direcionadas ao usuário e em suas aplicações, enquanto que as camadas mais baixas são direcionadas ao *hardware* em geral.

A seguir são definidas as camadas da arquitetura de um *grid*.

2.6.1 CAMADA DE ESTRUTURA (controle local dos recursos)

Esta camada é responsável pelos recursos do *grid* propriamente ditos: computadores, antenas, sensores, ou seja, tudo que possa compor um *grid*. Além disto, implementa os mecanismos para gerenciar e controlar o acesso aos recursos, computacionais, sistemas de armazenamento, redes, repositório de códigos e catálogos. Tais recursos podem ser uma entidade lógica controlada por protocolos próprios, como um *file system* distribuído (NFS), ou por um sistema gerenciador do *cluster*.

A seguir são listados alguns tipos de recursos e as características que fazem parte da camada de estrutura:

- **Recursos computacionais:** São os mecanismos necessários para iniciar os programas no *grid*, monitorar e controlar a execução dos processos. Além disto, o *grid* implementa mecanismos de descoberta que determinam

características de *hardware* e *software*, como disponibilidade, carga e enfileiramento de processos;

- **Recursos de armazenamento:** São os mecanismos responsáveis pelo armazenamento de arquivos no *grid*, como por exemplo, os dados que alimentam a execução de algum processo que pode migrar por qualquer motivo para outro recurso. Obviamente, os dados de entradas para esse processo devem migrar juntamente com o processo. Mecanismos que verificam o espaço disponível nos discos e a largura de banda para sua transferência também devem controlar tais operações;
- **Recursos de rede:** Esses recursos devem controlar a priorização e reservas das transferências de dados bem como o estado da rede;
- **Repositório de códigos:** É uma forma especializada de recurso de armazenamento que controla as versões dos objetos e códigos, como por exemplo, o CVS (*Concurrent Version System*);
- **Catálogos:** É uma forma especializada de recurso de armazenamento que controla operações de consultas e atualização, comumente usados por banco de dados.

2.6.2 CAMADA DE CONECTIVIDADE (comunicação fácil e segura)

Esta camada é responsável pelos protocolos de comunicação e autenticação utilizados para efetuar as transações de dados no *grid*. Os protocolos de comunicação são responsáveis pela troca de dados entre os recursos da camada de **Estrutura** e os protocolos de autenticação são responsáveis pelos mecanismos de criptografia para verificação de usuários e recursos do *grid*.

A comunicação é feita através dos protocolos da pilha TCP/IP, ou seja, IP e ICMP para endereçamento, TCP e UDP para transporte e OSPF, RSVP e DNS para roteamento e gerência de nomes.

Com relação à segurança na transação dos dados no *grid* os padrões existentes de criptografia, como X.509, SSL/TLS, CA (*Certificate Authority*), devem ser utilizados, pois possuem grande adaptabilidade e fácil integração com sistemas de *grids*.

Para a segurança dos dados em um *grid* os requisitos mais relevantes são:

- **Assinatura única:** o usuário quando conectado a um *grid* deve ter acesso a todos os seus recursos sem precisar efetuar várias autenticações. Por esse motivo as políticas de senha e admissão de usuários em um *grid* devem ter características que mantenham a sua segurança;
- **Delegação:** as permissões concedidas a um usuário em um determinado recurso do *grid* devem ser delegadas para todos os recursos onde existam processos deste mesmo usuário sendo executados;
- **Integração com soluções de segurança locais:** cada recurso do *grid* implementa suas próprias soluções de segurança. Portanto, os seus protocolos de segurança devem ser capazes de operar com essas várias soluções;
- **Confiabilidade baseada no usuário:** os usuários devem ser capazes de utilizar os recursos de um *grid* simultaneamente, sem necessitar a intervenção dos administradores de seus vários recursos componentes. Para tanto, as políticas de segurança e delegação devem ser bem definidas e aplicadas.

2.6.3 CAMADA DE RECURSOS (compartilhamento de recursos)

Esta camada é responsável pelo compartilhamento e interação dos recursos do *grid*, definindo como serão os acessos aos seus recursos individuais, e juntamente com a camada de

conectividade formam o gargalo do modelo da ampulheta comentado anteriormente. Essa definição pode ser feita através de dois tipos de protocolos explicados a seguir:

- **Protocolos de informação**: responsáveis por obter informações dos recursos do *grid*, tais como: memória, processador, uso atual da CPU etc;
- **Protocolos de gerência**: responsáveis por negociar o acesso aos recursos compartilhados do *grid*, tais como, criação de processos, movimentação de dados e migração de processos.

2.6.4 CAMADA COLETIVA (organização de vários recursos)

Esta camada é responsável pela coordenação dos vários recursos compartilhados pela camada de **Recursos**, não se preocupando com um recurso isolado do *grid*, mas sim com uma coleção de recursos do *grid*. Para controlar toda coleção de recursos uma série de serviços pode ser implementada, como mostra a Tabela 2.4.

Tabela 2.4 - Serviços da camada coletiva

Serviços	Descrição
Serviço de diretórios	Permite que os usuários participantes de uma VO façam pesquisas de recursos através de algumas características como: tipo, disponibilidade, carga atual e etc
Serviço de co-alocação, escalonamento e negociação de recursos *(broker)*	Permite que os usuários participantes de uma VO requisitem um ou mais recursos para rodar suas aplicações
Serviço de monitoração e diagnóstico	Oferece suporte a monitoração dos recursos da VO na ocorrência de falhas, sobrecarga entre outros
Serviço de replicação de dados	Oferece suporte aos recursos de armazenamento da VO de forma a melhorar a performance no acesso aos dados
Ambientes de programação *Grid-enabled*	Utilizam modelos e bibliotecas pré-definidos para as várias áreas do *grid*, como por exemplo: MPICH-G2
Sistemas de *workflows*	Utilizados para descrição, uso e gerência de múltiplos recursos
Serviço de descoberta de software	Descobre e seleciona a melhor implementação de software e plataforma de execução para solução de um determinado problema
Servidores de autorização comunitários	Reforçam as políticas de acessos dos membros do *grid* aos seus recursos
Serviços comunitários para *accounting* e *payment*	Reúnem informações de uso dos recursos do *grid* a fim de taxar comunidades de usuários limitando o uso desses recursos
Serviços de suporte e colaboração distribuída	Executam a troca de informações de forma coordenada entre grandes comunidades de usuários de forma síncrona ou assíncrona

2.6.5 CAMADA DE APLICAÇÃO (execução de programas)

Nesta camada são executadas as mais diferentes aplicações dos usuários de um *grid*. É a camada de interação com o usuário, aquela que efetivamente ele "enxerga". Nela podem ser implementadas formas de acesso como portais, descoberta e gerenciamento de recursos e ferramentas de desenvolvimento para suportar as aplicações aptas a executar no *grid*.

2.7 NOTA SOBRE A ARQUITETURA DE UM GRID

Alguns artigos dividem a arquitetura de um *grid* em quatro camadas, que são as seguintes: Rede, Recursos, *Middleware* e Aplicação, como ilustrado pela Figura 2.7 [9].

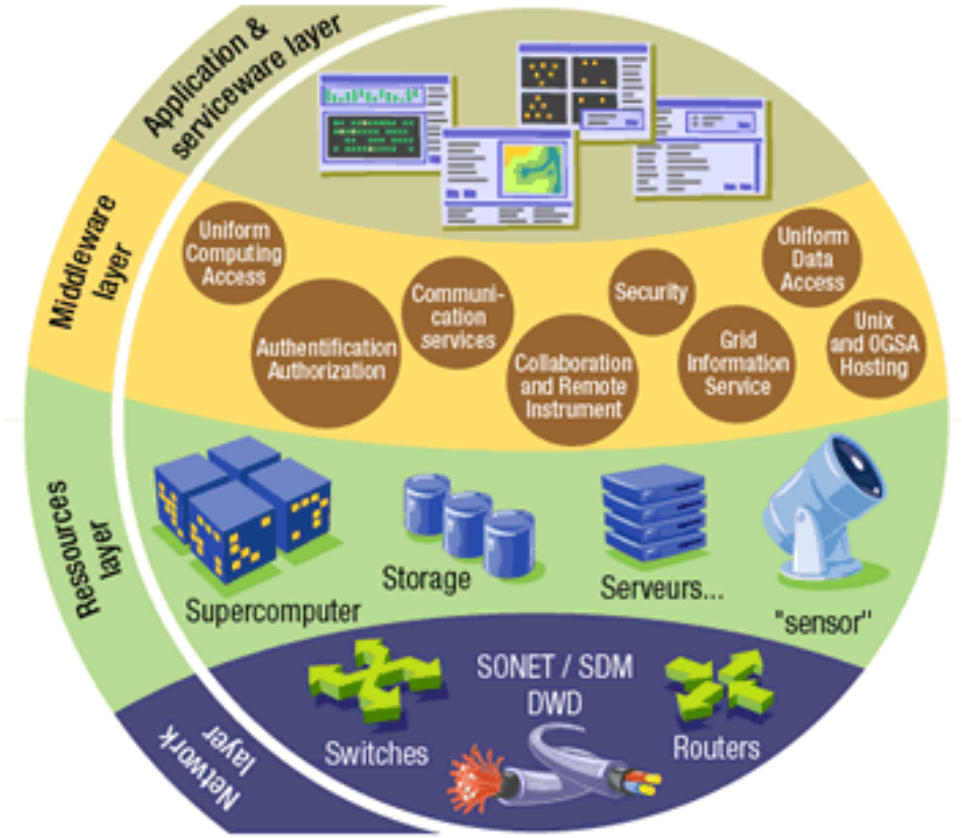

Figura 2.7 - Arquitetura de um grid sob o conceito de 4 camadas – extraído de http://gridcafe.web.cern.ch/gridcafe/gridatwork/architecture.html

A camada de Rede tem as mesmas características da camada de Estrutura que é a interconexão dos recursos do *grid*, através de vários dispositivos como, roteadores, *switches* e estruturas de cabeamento.

A camada de Recursos contém os dispositivos participantes do *grid*, como computadores, dispositivos de armazenamento, sensores, telescópios, microscópios etc, ou seja, qualquer dispositivo que possa estar conectado a rede.

A camada de *Middleware* contém ferramentas capazes de interligar todos os recursos do *grid*, ou seja, é ela que contém todas as soluções de *softwares* para prover segurança,

autenticação, compartilhamento, comunicação entre os componentes do *grid*. Devido a isso, pode ser considerada o cérebro do *grid*, porque tem a função de unir e gerenciar os seus vários recursos através de seus protocolos.

A camada de Aplicação é aquela na qual as aplicações dos usuários são executadas no *grid*.

CAPÍTULO 3

Globus Toolkit

O *Globus Toolkit* é um conjunto de ferramentas *open source* especialmente desenhadas para habilitar *Grids*, desenvolvida pelo consórcio *Globus Alliance* [10]. Esse *Toolkit* permite o compartilhamento de processadores, memória e discos, entre outros, através de múltiplas organizações com segurança e sem sacrificar a autonomia local dos recursos. Inclui ainda bibliotecas para a monitoração, descoberta e gerência de recursos, descoberta de falhas e movimentação de dados no *grid*.

3.1 UM POUCO DA HISTÓRIA DO GLOBUS

Em 1994, Rick Stevens, diretor da divisão de matemática e ciência da computação do Laboratório Nacional de Argonne, e Tom DeFanti, diretor do laboratório de visualização eletrônica da Universidade de Illinois, interligaram 11 redes e 17 locais nos Estados Unidos e Canadá e criando um *grid* nacional chamado de *"I-WAY"* [11]. Durante a Conferência de Supercomputação de 1996 (SC'96) um grupo de cientistas, liderados por Ian Foster, lançou uma série de protocolos que permitiam aos usuários do *"I-WAY"* executar suas aplicações em computadores espalhados através dos Estados Unidos [12]. O projeto Globus nasceu em 1996 em parceria com o Laboratório Nacional de Argonne, ISI *(Information Sciences Institute)* da Universidade da Califórnia e a Universidade de Chicago. Em 1997 foi lançada a versão 1.0 do *Globus Toolkit* que hoje se encontra na versão 4.0.6. Atualmente o projeto Globus é chamado de *Globus Alliance* e inclui várias outras universidades e centros de pesquisa [13].

Desde 2000, várias companhias desenvolvem produtos baseados no *Globus Toolkit*, dentre elas valem destacar: Avaki (desenvolvedora do banco de dados Sybase), Fujitsu, Hewlett-Packard, IBM, NEC, Oracle, Sun e United Devices (desenvolvedora de soluções para *clusters* e *grids*).

3.2 PROJETOS QUE UTILIZAM O GLOBUS TOOLKIT

Atualmente os membros participantes da *Globus Alliance* têm vários projetos nas áreas de *e-Busines* e *e-Science*. A Tabela 3.1 [14] apresenta uma lista destes projetos que se utilizam do *Globus Toolkit*.

Tabela 3.1 - Projetos que utilizam o *Globus Toolkit*

Área	Projeto	Finalidade
Astronomia	*Sloan Digital Sky Survey*	Mapeamento digital do universo
	National Virtual Observatory	Análise astronômica de dados
Química	CMCS	Base de dados para pesquisa em várias áreas da química
Engenharia Civil	NEES	Simulações de terremotos
Estudos climáticos	LEAD	Pesquisas atmosféricas
	Earth System Grid	Integração de vários laboratórios nos EUA para pesquisas climáticas
Ciência da Computação	Condor	Escalonador de processos
	GridLab	Desenvolvimento de ferramentas para Grid
	GriPhyN	Desenvolvimento de ferramentas para aplicações em física e engenharia
	OGCE	Ferramenta para criação de *Web-services*
	CoG	Desenvolvimento de *Web-portals* para aplicações java e python
	VGrADS	Criação de ferramentas para o desenvolvimento de aplicações voltadas para grid
Ecologia	LTER	Aplicações em várias áreas da ecologia
	SEEK	*Data grid* para acesso a uma base de dados ecológica
Geologia	GEON	*Web-service* aplicado à geologia
	SCEC	Estudos de terremotos da Califórnia
Medicina	BIRN	Simulações e pesquisas em várias áreas da medicina
Oceanografia	LOOKING	*Data grid* para pesquisas oceanográficas
Física	FusionGrid	Utilização das ferramentas do globus para análise, visualização e autenticação no apoio a pesquisas na área de fusão magnética
	Particle Physics Data Grid	Aplicação do globus para estudo da Física de partículas

3.3 COMPONENTES IMPORTANTES DO GLOBUS TOOLKIT

Como já comentado anteriormente, o *Globus Toolkit* consiste de uma série de ferramentas e protocolos que são responsáveis pelo funcionamento de toda a solução de *grid* suportada pelo globus. Cada um desses protocolos faz parte de uma das 5 camadas da arquitetura de um *grid*. Dentre esses protocolos vale destacar os seguintes:

3.3.1 GRID SECURITY INFRASTRUCTURE (GSI)

Este serviço faz parte da camada de conectividade e é responsável pela autenticação de usuários no *grid*, bem como pela delegação das permissões do usuário aos recursos que serão utilizados para executar uma aplicação. O GSI [15] utiliza-se do protocolo SSL (*Secure Sockets Layer*) para efetuar a autenticação de usuários sendo, portanto, baseado no esquema de chaves públicas e privadas para a encriptação dos dados que trafegam no *grid*. No *Globus Toolkit*, por padrão, após autenticado, um usuário tem 12 horas para submeter processos ao *grid*. Passadas as 12 horas, é necessário efetuar uma nova autenticação para submeter novos processos. Vale ressaltar que se a aplicação do usuário levar mais de 12 horas para ser finalizada ela não será interrompida. O Quadro 3.1 mostra o comando que efetua a autenticação de usuário no globus: **grid-proxy-init**.

```
joioso@corto1:~$ grid-proxy-init
Your identity: /O=Grid/OU=GlobusTest/OU=simpleCA-
corto1.cluster/OU=cluster/CN=Aparecido Luciano Breviglieri Joioso
Enter GRID pass phrase for this identity:
Creating proxy ............................ Done
      Your proxy is valid until: Sun Mar  4 05:40:08 2007
joioso@corto1:~$
```

Quadro 3.1 - Comando grid-proxy-init para autenticação de usuários do globus

3.3.2 GRID RESOURCE ACCESS AND MANAGEMENT (GRAM)

O GRAM é sem dúvida um dos componentes mais importantes do *Globus Toolkit*, sendo o responsável por tornar possível a submissão de processos locais ou remotos. Para isso ele apresenta características que permitem criar um ambiente para: a execução de um processo, o controle da entrada e saída de arquivos de dados utilizados para a execução desse processo, a submissão de um processo ao escalonador local, o monitoramento de processos, o envio de avisos de notificação caso um processo passe de um estado para outro e a utilização das saídas **stdout/stderr** durante a execução de um processo.

A Figura 3.1 ilustra como é controlada a submissão de um processo e os componentes do GRAM que são responsáveis por cada etapa do ciclo de vida deste processo. Quando o usuário submete uma requisição ela é analisada pelo *Gatekeeper* (porteiro) que faz o gerenciamento desta requisição. Ele tem as funções de efetuar a autenticação entre o usuário e determinar os recursos que serão alocados e criar um *Job Manager* que fará a submissão do processo para os recursos do *grid*, bem como o controle dos estados dos processos desta requisição [16].

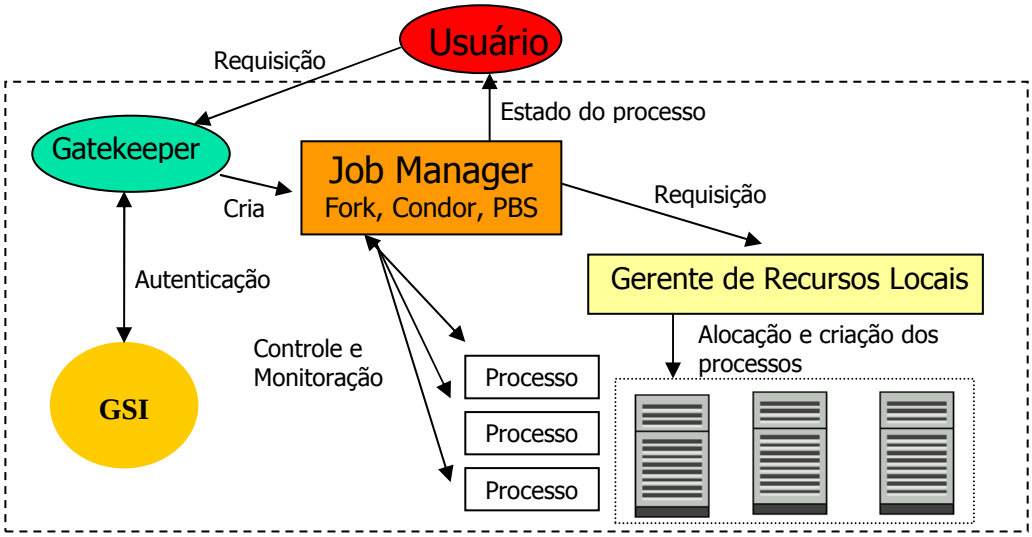

Figura 3.1 - *Grid Resource Access and Management* - GRAM

O Quadro 3.2 mostra o comando **globusrun-ws** que pode ser utilizado para fazer a submissão de uma requisição ao *grid*. Neste caso o comando apenas solicita a data ao nó local. Vale ressaltar que os estados pelo qual essa requisição passa também são mostrados como saída deste comando.

```
joioso@corto1:~$ globusrun-ws -submit -c /bin/date
Submitting job...Done.
Job ID: uuid:2d4f3e9c-a2b9-11dc-98b1-000c766580cf
Termination time: 12/05/2007 22:35 GMT
Current job state: Active
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
joioso@corto1:~$
```

Quadro 3.2 - Comando globusrun-ws para exemplificar a submissão de um processo simples

Em casos de requisições com tempos de resposta mais longos, os estados dos processos desta requisição podem ser acompanhados através do comando **globusrun-ws –state <JobId>.** Os estados que eles podem ocupar estão descritos na Tabela 3.2 e a Figura 3.2 apresenta o diagrama de transição de processos GRAM.

Tabela 3.2 - Estados dos processos GRAM

Estados	Descrição
Unsubmitted	Processo ainda não submetido
StageIn	Processo está aguardando que os arquivos de dados e o executável sejam carregados
Pending	O escalonador local ainda não agendou a execução do processo
Active	Processo em execução
Suspended	A execução do processo foi suspensa
StageOut	Execução do processo finalizada, arquivos de saída sendo transferidos para os meios de armazenamento de destino
CleanUp	Execução do processo e *StageOut* finalizados, limpando tarefas restantes
Done	Processo finalizado com sucesso
Failed	Falha na execução do processo

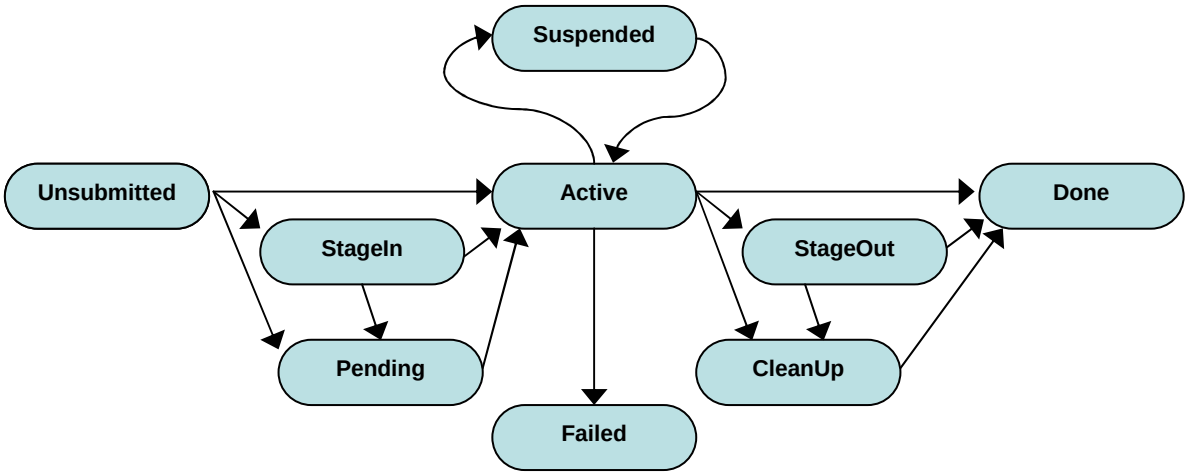

Figura 3.2 - Diagrama de transição de processos GRAM

De acordo com a Figura 3.2, quando uma requisição é submetida ela pode passar para o estado de *Active* assim que todas as dependências para a sua execução estejam satisfeitas. No entanto, ela pode passar para o estado de *StageIn* se o processo estiver aguardando que seja feita alguma transferência de dados para o recurso que irá processá-la ou, ainda, para o estado *Pending* se o *scheduler* local ainda não tiver agendado sua execução. Quando em *Active*, a requisição pode ser suspensa por estar aguardando que algum outro processo dependente finalize. Então ela ocupará o estado *Suspended* ou falhará por algum problema com o recurso que a está processando, ocupando o estado de *Failed*. Antes de finalizar o processamento da requisição, o processo pode passar para o estado de *StageOut*, que significa que algum arquivo de finalização da requisição está sendo gravado e transferido para o nó de onde a requisição foi disparada. Em seguida todos os processos que foram abertos para o processamento da requisição são finalizados e a requisição pode ocupar o estado de *CleanUp*. Quando toda a requisição é processada com sucesso ela passa para o estado *Done*.

Existem duas formas de fazer submissão de processos no *Globus Toolkit*, uma delas foi exemplificada no Quadro 3.2, que é a forma através de comandos, outra forma que certamente é a mais utilizada para requisições mais complexas é através de *batchs*, para isso o GRAM utiliza-se de duas linguagens de *batch scripts*, a RSL (*Resource Specification*

Language) – que será apresentado na seção 4.1.2 do Capítulo 4 - e a *XML-based Job Description Language*.

O Quadro 3.3 exemplifica um arquivo de submissão XML e sua respectiva execução com o comando **globusrun-ws**.

```
joioso@corto1:~$ cat sub.xml
<job>
    <executable>rsl_teste</executable>
    <directory>${GLOBUS_USER_HOME}</directory>
    <argument>Grava na</argument>
    <argument>STDOUT</argument>
    <stdout>${GLOBUS_USER_HOME}/stdout</stdout>
    <stderr>${GLOBUS_USER_HOME}/stderr</stderr>
    <fileStageIn>
        <transfer>
            <sourceUrl>gsiftp://corto2.cluster:2811/bin/echo</sourc
eUrl>
            <destinationUrl>file:///${GLOBUS_USER_HOME}/rsl_teste</
destinationUrl>
        </transfer>
    </fileStageIn>
    <fileCleanUp>
        <deletion>
            <file>file:///${GLOBUS_USER_HOME}/rsl_teste</file>
        </deletion>
    </fileCleanUp>
</job>

joioso@corto1:~$ globusrun-ws -submit -S -f sub.xml
Delegating user credentials...Done.
Submitting job...Done.
Job ID: uuid:dad5d07e-a383-11dc-83f4-000c766580cf
Termination time: 12/06/2007 22:46 GMT
Current job state: StageIn
Current job state: Active
Current job state: CleanUp
Current job state: Done
Destroying job...Done.
Cleaning up any delegated credentials...Done.
joioso@corto1:~$ cat ~/stdout
Grava na STDOUT
```

Quadro 3.3 - Exemplo de arquivo de submissão XML

O arquivo **sub.xml** basicamente copia o comando de sistema **/bin/echo** do *host* **corto2.cluster** e o grava na área local do usuário joioso com o nome de **rsl_teste**. Em seguida, o *script* executa o comando **rsl_test** utilizando os argumentos de entrada "Grava na STDOUT", apagando em seguida o arquivo **rsl_teste**. Como é baseado em XML, os argumentos para execução do *batch* **sub.xml** estão entre *tags* da metalinguagem suportada pelo GRAM. A Tabela 3.3 descreve o significado de cada *tag*.

Tabela 3.3 - *Tags* de uma *XML-based Job Description Language*

Tag	Descrição
<job> </job>	Indica o início e fim do *batch* XML
<executable> </executable>	Nome do arquivo executável
<directory> </directory>	Diretório onde se encontra o executável
<argument> </argument>	Qualquer argumento para execução do programa
<stdout> </stdout>	Nome do arquivo de saída para armazenar a saída do programa
<stderr> </stderr>	Nome do arquivo de saída para armazenar erros na execução
<fileStageIn> </fileStageIn>	Especifica uma lista de comandos que será executado pelo processo
<tranfer> </tranfer>	Especifica uma seção de transferência de arquivos
<sourceURL> </sourceURL>	Especifica o arquivo que será copiado
<destinationURL> </destinationURL>	Especifica o destino para o qual o arquivo será copiado
<fileCleanUp> </fileCleanUp>	Especifica a lista de arquivos que serão apagados após a execução o *job*
<deletion> </deletion>	Especifica o arquivo que será deletado

3.3.3 GRIDFTP

O GridFTP é o serviço responsável pela movimentação de dados do *Globus Toolkit*. Este é um serviço de FTP muito mais robusto, capaz de mover grandes quantidades de dados,

de forma mais rápida, segura e confiável do que o serviço de FTP comum, devido ao fato do arquivo poder estar distribuído em vários recursos do grid, podendo ser transferido simultaneamente. Esse serviço é comumente instalado de forma integrada com toda a solução do *Globus Toolkit*, mas também pode ser instalado de forma independente.

O GridFTP *server* opera na porta 2811 por *default*, e pode ser configurado através do comando **globus-gridftp-server**. Já do lado do cliente, o GridFTP pode ser invocado diretamente com o comando **globus-url-copy** como aparece no Quadro 3.4. Neste exemplo o arquivo **transf_gridftp** localizado no *host* **corto1.cluster** é transferido para o *host* **corto2.cluster** no diretório **/tmp**. O GridFTP também pode ser invocado através de um arquivo RSL ou XML como no Quadro 3.3 da página 42.

```
joioso@corto1:~$ globus-url-copy
file:///home/joioso/transf_gridftp gsiftp://corto2.cluster/tmp/
joioso@corto1:~$ ssh corto2.cluster ls /tmp
transf_gridftp
joioso@corto1:~$
```

Quadro 3.4 - Comando de linha globus-url-copy

CAPÍTULO 4

MPICH-G2 e ScaLAPACK

Este capítulo apresenta uma visão geral sobre as bibliotecas de programação paralela utilizadas neste trabalho: a biblioteca de passagem de mensagens MPI, focando na sua versão para o *Globus Toolkit,* e a biblioteca ScaLAPACK, que é utilizada para cálculos de álgebra linear.

No LFC trabalhamos com uma aplicação que tem como parte mais pesada de processamento a diagonalização de matrizes hamiltonianas de vários gigabytes, e que portanto, necessitam de dezenas de gigabytes de memória RAM para serem processadas. Esta memória geralmente não está disponível em *clusters* acadêmicos, que em média não ultrapassam 10 ou 15 nós. Esta aplicação nos levou ao estudo da *Grid Computing*, mesmo sabendo que é uma aplicação atípica devido à grande comunicação existente entre os processos. Este trabalho foi proposto para estudar o comportamento do MPICH-G2 em um ambiente controlado, ou seja, um ambiente de *cluster*, onde temos uma rede de interconexão de alta velocidade.

Atualmente a aplicação utilizada no LFC faz uso da biblioteca ScaLAPACK para efetuar essa diagonalização. Então, para realizar nossos testes nós utilizamos um programa de exemplo distribuído no site da biblioteca ScaLAPACK que utiliza a mesma rotina de diagonalização que a aplicação do LFC, e que foi modificado para o intuito.

4.1 MPI (MESSAGE PASSING INTERFACE)

O MPI *(Message Passing Interface)* é um padrão para o desenvolvimento de bibliotecas de programação paralela baseada na passagem de mensagens que teve sua primeira versão publicada em 1994 no *MPI Standard 1.0* [17]. Atualmente este padrão se encontra na versão 2, publicado no *MPI Standard 2.0* [18] em 1997. Apesar da versão 2 deste padrão ter sido publicada em 1997 apenas recentemente os desenvolvedores desta biblioteca lançaram versões compatíveis com este padrão.

4.1.1 MPICH

O MPICH [19] é a biblioteca de passagem de mensagens desenvolvida pelo *Argonne National Laboratory* baseada no modelo *"message passing"* e na implementação padrão do MPI. Neste padrão um conjunto de processos tem acesso à memória local, a comunicação é baseada no envio e recebimento de mensagens e a transferência entre processos requer operações de cooperação entre cada processo. Além disto existem operações coletivas que coordenam grupos de processos, como por exemplo, operações de *broadcast*, *scatter* e *gather*.

O MPICH suporta vários tipos de máquinas paralelas. No entanto, no meio acadêmico sua utilização mais comum é em *clusters* do tipo *Beowulf* que são máquinas do tipo PC com sistema operacional Linux.

O *device* de comunicação padrão do MPICH é o **ch_p4**, (*ch=channel, p4=Portable Programs for Parallel Processors*) [20], cujas principais características deste device são:

- P4 roda em Sun/SunOS, Sun/Solaris, Solaris86, Cray, HP, Dec 5000, Dec Alpha, Next, IBM RS6000, Linux86, FreeBSD, IBM3090, SGI e outras;
- Utiliza um *socket* de processo para processo, para a comunicação entre processos que não estão no mesmo nó, ou os processos se comunicam via

memória compartilhada para processos no mesmo nó (isto quando compilado com a opção *"-comm=shared"*);

- São iniciados sub-processos chamados de *"listeners"* para auxiliar o estabelecimento de comunicações de processos para processos;
- Capaz de disparar aplicações MPI em arquiteturas heterogêneas.

A Figura 4.1 ilustra como ocorre a passagens de mensagens no MPICH.

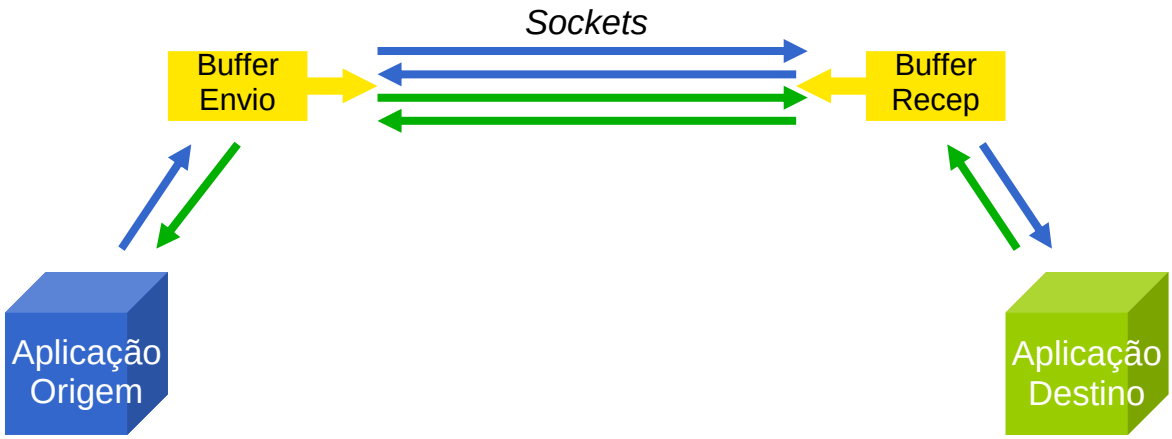

Figura 4.1 - Diagrama de passagem de mensagens do MPICH

De acordo com a Figura 4.1, quando um dado for enviado da aplicação de origem para a aplicação de destino, ela passa por um *buffer* de envio e é enviada através de um *socket* unidirecional para um *buffer* de recepção da aplicação de destino que entrega a mensagem para a aplicação de destino.

4.1.2 MPICH-G2

O MPICH-G2 [21] [22] é uma versão do MPI implementada para suportar *grids*, que permite a execução de programas MPI através de vários computadores em diferentes domínios espalhados pela internet utilizando exatamente o mesmo código MPI da programação de *clusters* convencionais. O MPICH-G2 é uma extensão da implementação do

MPICH desenvolvida para operar juntamente com o *Globus Toolkit*, e é baseado no MPICH 1.2.x.

O *device* de comunicação padrão do MPICH-G2 é o **globus2**. A versão anterior do MPICH-G2 é a MPICH-G, cujo *device* de comunicação é o **globus**. A principal diferença entre estes dois *devices* está na maneira como a comunicação está implementada. No MPICH-G todo o esquema de comunicação está associado a uma biblioteca de comunicação chamada **Nexus**, já para o MPICH-G2 a biblioteca de comunicação utilizada é a **globus_io**.

No MPICH-G e no MPICH a passagem de mensagens ocorre da aplicação de origem para um *buffer* de envio, que envia para um *buffer* de recebimento e só depois é passada para a aplicação de destino, já na versão MPICH-G2 este *buffer* intermediário foi retirado e a passagem de mensagens ocorre diretamente da aplicação de origem para aplicação de destino, como ilustra a Figura 4.2. Outro fator importante, melhorado na versão MPICH-G2 foi o uso mais eficiente nos *sockets* de comunicação. No MPICH-G e MPICH eram abertos dois pares de *sockets* para comunicação entre dois nós. Esses canais transmitiam dados em apenas uma direção, enquanto que no MPICH-G2 apenas um par de *sockets* é criado e são utilizados para comunicações bidirecionais.

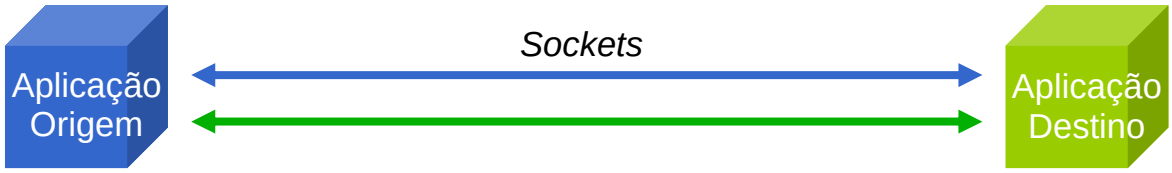

Figura 4.2 - Diagrama de passagem de mensagens do MPICH-G2

A biblioteca **globus_io** também utiliza a estrutura de autenticação do GSI (*Grid Security Infrastructure*) para criar canais de comunicação seguros e do GRAM (*Grid Resource Access and Management*) para efetuar o *handshaking* (estabelecimento da comunicação) entre cliente e *gatekeeper* [23].

O MPICH-G2 utiliza a RSL (*Resource Specification Language*) para efetuar a submissão dos processos MPI no globus e baseado nesta linguagem o MPICH-G2 invoca o DUROC (*Dynamically Update Request Online Coallocator*), que é uma biblioteca de co-alocação dinâmica de recursos distribuídos, capaz de escalonar e iniciar a aplicação nos vários computadores descritos na RSL.

O DUROC utiliza o GRAM para iniciar e gerenciar todas as sub-computações MPI de modo que, para cada sub-computação, o DUROC gera uma requisição de autenticação de usuário ao GRAM que interage com o escalonador local para iniciar os processos**.**

O Quadro 4.1 exemplifica um arquivo de submissão RSL que está sendo utilizado nas simulações da aplicação de diagonalização de matrizes do LFC. Neste exemplo o executável **sample_pcheevx_call_32_4P_11000.exe** será distribuído entre 4 nós (corto1.cluster até corto4.cluster), já a Tabela 4.1 descreve a função de cada parâmetro desta RSL.

```
+
( &(resourceManagerContact="corto1.cluster")
   (count=1)
   (label="subjob 0")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
         (LD_LIBRARY_PATH /usr/local/globus-4.0.4/lib/))
   (directory="/home/joioso")
   (executable="/home/joioso/sample_pcheevx_call_32_4P_11000.exe")
)
( &(resourceManagerContact="corto2.cluster")
   (count=1)
   (label="subjob 1")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
         (LD_LIBRARY_PATH /usr/local/globus-4.0.4/lib/))
   (directory="/home/joioso")

(executable="/home/joioso/grid/sample_pcheevx_call_32_4P_11000.exe")
)
( &(resourceManagerContact="corto3.cluster")
   (count=1)
   (label="subjob 2")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 2)
         (LD_LIBRARY_PATH /usr/local/globus-4.0.4/lib/))
   (directory="/home/joioso ")
   (executable="/home/joioso/sample_pcheevx_call_32_4P_11000.exe")
)
( &(resourceManagerContact="corto4.cluster")
   (count=1)
   (label="subjob 3")
   (environment=(GLOBUS_DUROC_SUBJOB_INDEX 3)
         (LD_LIBRARY_PATH /usr/local/globus-4.0.4/lib/))
   (directory="/home/joioso ")
   (executable="/home/joioso/sample_pcheevx_call_32_4P_11000.exe")
)
```

Quadro 4.1 - Exemplo de arquivo de submissão RSL

Tabela 4.1 - Parâmetros do arquivo RSL

Parâmetro	Descrição
resourceManagerContact	Indica em qual recurso o processo será disparado
Count	Número de processos disparados para esse **resourceManagerContact**
Label	Identificação do processo
enviroment	Definição de variáveis de ambiente para a execução do processo
directory	Diretório onde se encontra o executável
executable	Nome do arquivo executável

A Figura 4.3 ilustra o funcionamento do MPICH-G2. A permissão para executar no *grid* é solicitada pelo comando **grid-proxy-init**, que consulta o GSI para obter as credenciais de autenticação. Uma vez autenticado um *job manager* é criado para analisar o *script* RSL que é produzido pelo comando **mpirun**. Baseado nas informações da RSL, o MPICH-G2 invoca o DUROC para agendar e iniciar os processos nos recursos descritos na RSL. Para cada processo o DUROC faz uma requisição ao GRAM de cada recurso, que por sua vez interage com o *scheduler* local para iniciar os processos. Durante a execução do *job* DUROC e GRAM interagem para monitorar a execução da aplicação [24].

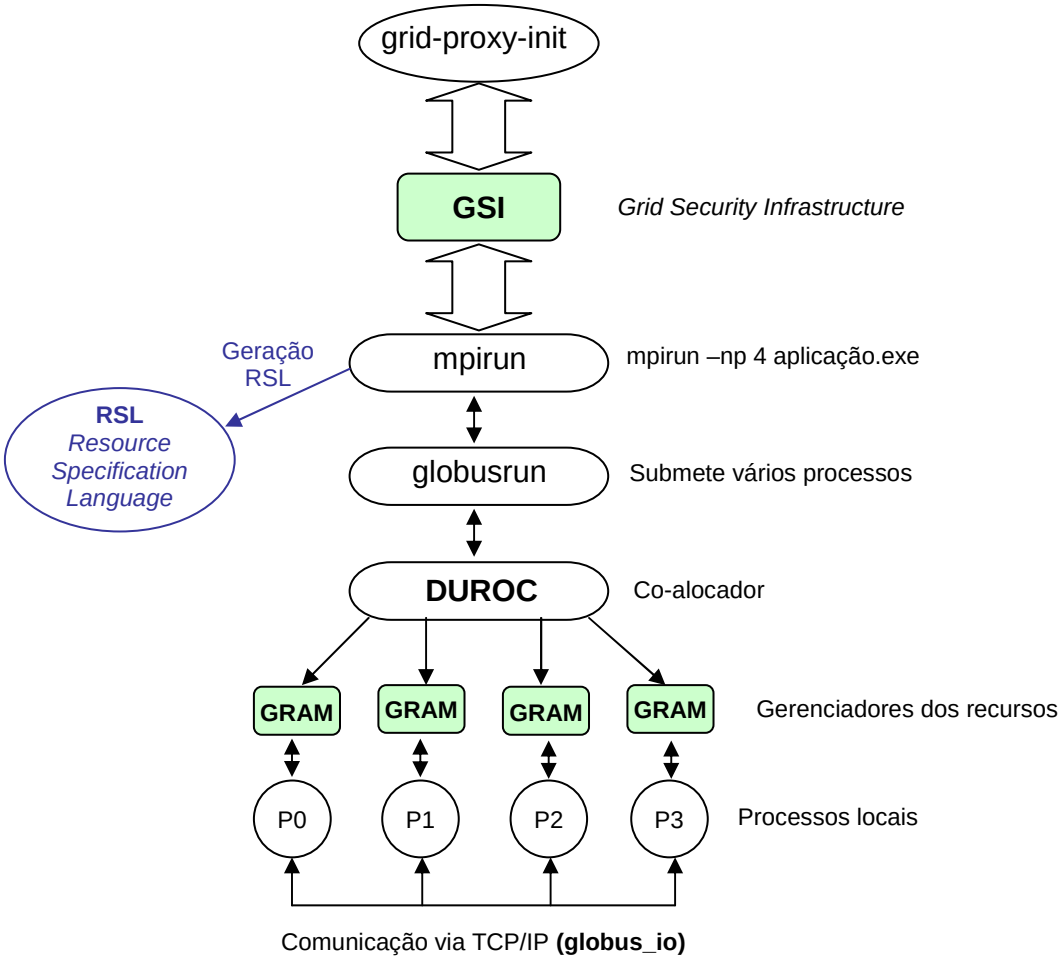

Figura 4.3 - Funcionamento do MPICH-G2

4.2 SCALAPACK

O ScaLAPACK (*Scalable Linear Algebra Package*) [25] é uma biblioteca paralela utilizada para cálculos de rotinas de álgebra linear de alto desempenho para computadores de memória distribuída. O ScaLAPACK descende da biblioteca LAPACK (*Linear Algebra Package*), também utilizada para cálculos de rotinas de álgebra linear, sem, no entanto aproveitar o paralelismo que pode ser oferecido por um *cluster* ou *grid*.

O ScaLAPACK divide a operação sobre matrizes em tarefas mais elementares. Ao invés de trabalhar na matriz como um todo, esta é dividida em sub-matrizes, que então podem ser distribuídas em vários processos.

O ScaLAPACK distribui a matriz em vários processos realizando um decomposição bloco cíclica da matriz original. Desta forma, se temos uma matriz *N x N* ela deve ser decomposta em matrizes de blocos *MB x NB*. A Figura 4.4 ilustra este tipo de decomposição, para o caso de uma matriz 5 x 5, mapeada em uma grade de processos 2 x 2 com blocos de tamanho 2 x 2.

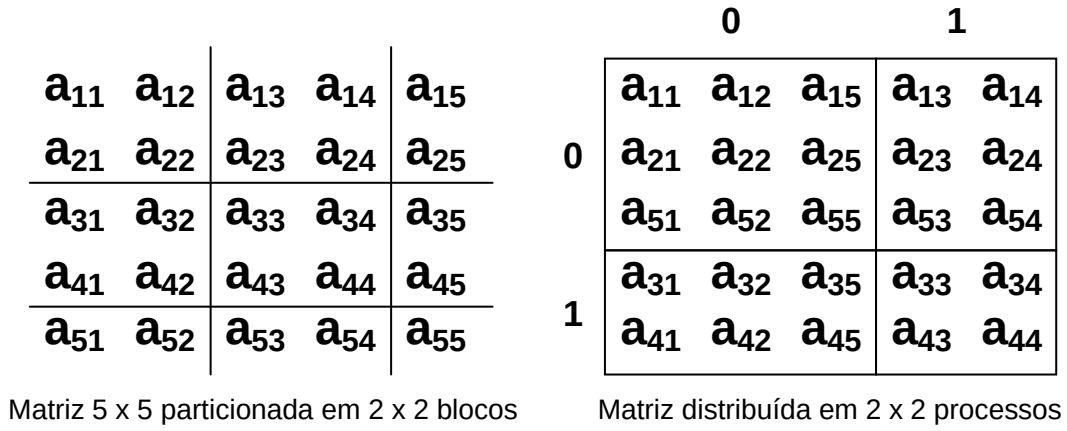

Figura 4.4 - Matriz 5 x 5 decomposta em blocos 2 x 2 mapeada em 2 x 2 processos – extraída de http://www.netlib.org/scalapack/slug/node76.html

O ScaLAPACK tem várias rotinas implementadas para a resolução de vários tipos de problemas de álgebra linear. Todas essas rotinas têm um padrão de nomeação no seguinte formato, PXYYZZZ, onde o caractere P indica que é uma rotina do ScaLAPACK o caractere X representa o tipo de dados, os caracteres YY indicam o tipo da matriz e os caracteres ZZZ indicam o operação da rotina. A Tabela 4.2 apresenta os tipos de dados. Já no Apêndice A, temos a relação de todos os tipos de matrizes suportados e as operações possíveis para as rotinas do ScaLAPACK.

Tabela 4.2 - Tipos de dados representados pelo segundo caractere (X)

Caracter X	Tipo de dado
S	*Real*
D	*Double presicion*
C	*Complex*
Z	*Double complex*

O nome da rotina **PCHEEVX**, por exemplo, indica que esta é uma rotina do ScaLAPACK que opera sobre o tipo de dados complexo e calcula autovalores e autovetores de uma matriz hermitiana.

Para a realização dos testes de desempenho foi utilizado um programa-exemplo do ScaLAPACK que faz uso da rotina PCHEEVX. Essa rotina faz uso de outras rotinas do ScaLAPACK que têm funções distintas. Dentre elas destacam-se 3, porque são as rotinas que utilizam a parte mais significante do tempo de processamento e da comunicação: a rotina PCHENTRD, que reduz a matriz hermitiana para a forma tridiagonal, a rotina PSSTEBZ, que calcula os autovalores de uma matriz tridiagonal simétrica utilizando o método de bisecção, e a rotina PCSTEIN, que calcula o autovetores de uma matriz tridiagonal utilizando iteração inversa.

O ScaLAPACK utiliza outras 2 bibliotecas para executar suas funções: o BLAS (*Basic Linear Algebra Subprograms*) que realiza de forma otimizada operações mais simples, como por exemplo, a multiplicação de matrizes e a BLACS (*Basic Linear Algebra Communication Subprograms*) [26], que executa de forma otimizada a comunicação de dados entre os processos com o auxílio de uma biblioteca de passagens de mensagens como o MPICH.

CAPÍTULO 5

Metodologia para a execução dos testes

O LFC alia o estudo teórico de semicondutores ao desenvolvimento de modelos e simulações computacionais para o estudo das propriedades eletrônicas de sistemas tais como nanoestruturas semicondutoras (fios e pontos quânticos), dispositivos spintrônicos e eletrônicos e novos materiais semicondutores. Esse estudo envolve o uso de técnicas computacionais modernas, como por exemplo, o uso de processamento paralelo.

Atualmente o LFC utiliza um programa para a realização de seus estudos em que a parte que exige computação de alto desempenho é justamente a diagonalização de matrizes hamiltonianas. Esse programa tem uma versão seqüencial na qual apenas um nó fica responsável por todos os cálculos e uma versão distribuída, que utiliza as bibliotecas MPICH e ScaLAPACK.

O estudo aqui apresentado avalia o uso de *grid computing* para a obtenção de resultados para a aplicação distribuída, já que é considerada atípica para a execução em ambientes de *grid*.

5.1 CONFIGURAÇÃO DO SISTEMA PARA A EXECUÇÃO DOS TESTES

Os testes de comparação de desempenho das bibliotecas MPICH e MPICH-G2 foram executados no *cluster* do LFC do IFSC que atualmente conta com 4 máquinas de 64 bits, 4 máquinas de 32 bits e 1 *switch gigabitethernet* para interconexão de todas as máquinas. A

configuração física de interconexão desse *cluster* é ilustrada na Figura 5.1 e a configuração de *hardware* das máquinas está descrita na Tabela 5.1.

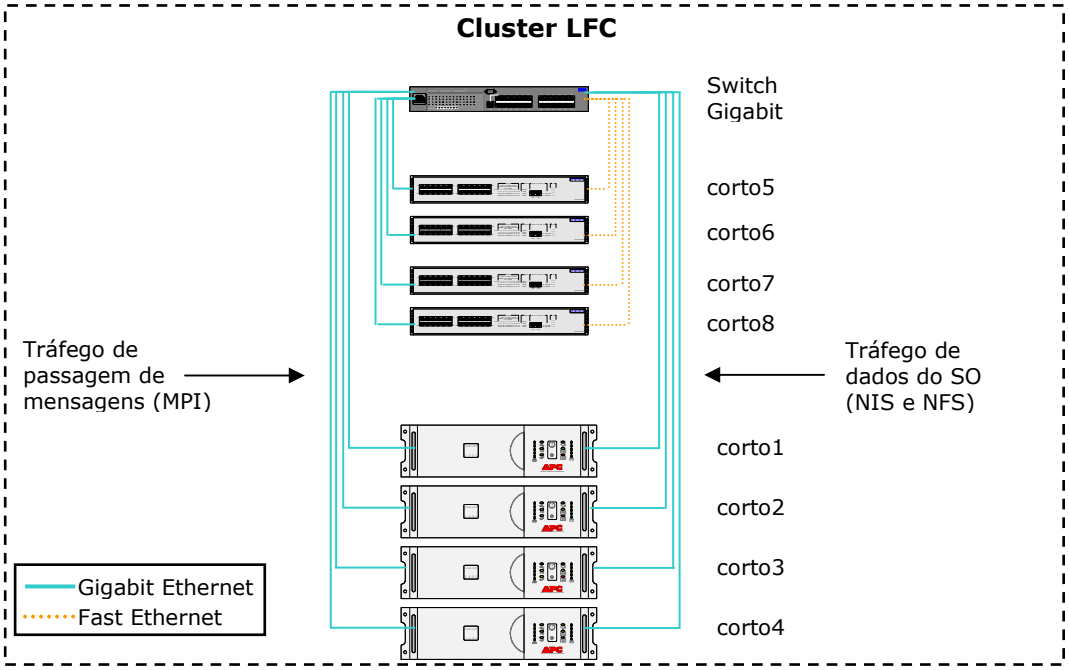

Figura 5.1 – *Cluster* do Laboratório de Física Computacional

Tabela 5.1 - Configuração de *hardware* do *cluster* LFC

Qtde	Proc/Clock	Memória Cache	Memória RAM	Placa de Rede	Nome do Host
4	AMD Athlon 64 X2 Dual Core 4200+ 2210MHz	512Kb	2GB	2 Gigabit Ethernet	CORTO1 CORTO2 CORTO3 CORTO4
4	2 x Pentium III 1133MHz	512Kb	2GB	1 Gigabit Ethernet 1 Fast Ethernet	CORTO5 CORTO6 CORTO7 CORTO8

O *cluster* LFC é denominado CORTO e todos os *hosts* pertencentes ao *cluster* têm os seus respectivos nomes CORTO1 até CORTO8. As primeiras 4 máquinas são de 64 bits e as últimas 4 são de 32 bits. A CORTO1 é o nó principal do *cluster* que tem um disco de 200GB instalado contendo as áreas de usuários que estão exportadas via NIS. Além disso, o disco é também exportado via NFS para todos os outros *hosts*. Como pode ser visto na Tabela 5.1

todas as máquinas são bi-processadas, portanto cada uma pode rodar 2 processos simultaneamente. Desta forma, pode haver um total de 16 processos rodando simultaneamente sem concorrência. Além disto as 4 máquinas de 64 bits possuem 2 placas de rede *GigabitEthernet* (1000Mbit/s) e as 4 máquinas de 32bits tem uma placa de rede *FastEthernet* (100Mbit/s) e uma placa de rede *GigabitEthernet* (1000Mbit/s). O *cluster* está configurado de forma que toda a comunicação do sistema operacional, como por exemplo, NIS e NFS, seja enviada através da placa de rede *FastEthernet* e as passagens de mensagens MPI sejam enviadas através da placa de rede *GigabitEthernet* no caso das máquinas de 32 bits.

Todos os 8 *hosts* do laboratório têm instalado o sistema operacional Linux com a distribuição Debian 4r0 *kernel* 2.6.22 e o *Globus Toolkit* versão 4.0.4.

O compilador utilizado no LFC é o da empresa, *The Portland Group Inc.* (PGI) que é um conjunto de compiladores desenvolvidos para computação de alto desempenho. A *The Portland Group Inc*. desenvolve os compiladores Fortran, C e C++, para diversas arquiteturas e sistemas operacionais.

5.2 PROGRAMA UTILIZADO PARA OS TESTES

O programa utilizado para a execução dos testes de desempenho é uma modificação ao exemplo em Fortran disponível no site http://www.netlib.org/scalapack/examples chamado **sample_pcheevx_call.f**. Esse programa resolve problemas de autovetores e autovalores de matrizes hermitianas e utiliza a rotina PCHEEVX do ScaLAPACK. Esta é a mesma rotina utilizado pela aplicação do LFC. O Apêndice B apresenta uma listagem completa desse programa com as modificações que foram realizadas para a execução dos testes.

O Quadro 5.1 apresenta as variáveis, **MAXN**, **NB**, **NPROW**, **NPCOL** do código **sample_pcheevx_call.f** que foram modificadas a fim de aumentar o tamanho da matriz calculada e o número de processos em que ela foi distribuída. A variável **MAXN** é o tamanho da matriz, **NB** é o tamanho do bloco da decomposição bloco cíclica da matriz e as variáveis

NPROW e **NPCOL** representam em quantas linhas e colunas respectivamente os processos foram distribuídos. Neste exemplo, do Quadro 5.1, temos **NPROW** multiplicado por **NPCOL** contabilizando um total de 6 processos a serem distribuídos entre os nós que irão realizar os cálculos.

```
...
*     .. Parameters ..
      INTEGER            LWORK, MAXN, LIWORK
      REAL               ZERO
      PARAMETER          (LWORK = 10000000, MAXN=11000, LIWORK =
70000, ZERO = 0.0E+0 )
...
*     Set up the problem
*
      N = MAXN
      NB = 8
      NPROW = 3
      NPCOL = 2
*
...
```

Quadro 5.1 - Variáveis modificadas para a realização dos testes

Para os testes realizados, a variável **MAXN** variou nos seguintes valores: 1000, 2000, 4000, 6000, 8000, 10000, 11000. Não foi possível ultrapassar 11000 para o valor máximo de MAXN devido à limitação física de memória dos nós do *cluster*. O tamanho do bloco (**NB**) foi sempre mantido em 8 e as variáveis **NPROW** e **NPCOL** variaram nos valores que estão representados pela Tabela 5.2.

Tabela 5.2 - Número de processos em que as matrizes foram divididas

NPROW x NPCOL	Processos
1 x 1	1
2 x 2	4
3 x 2	6
4 x 2	8
3 x 3	9
4 x 3	12

O número de processos foi limitado em 12 uma vez que 2 nós do cluster apresentaram problemas durante a execução dos testes.

5.3 INSTALAÇÃO DAS BIBLIOTECAS UTILIZADAS PARA OS TESTES DE DESEMPENHO

O objetivo principal deste trabalho foi avaliar o desempenho em um ambiente controlado da biblioteca de passagem de mensagens construída para *grids* (MPICH-G2) com sua versão para *clusters* (MPICH) utilizando a biblioteca de álgebra linear ScaLAPACK. No entanto, o ambiente é heterogêneo com relação à arquitetura de *hardware* uma vez que temos um *cluster* misto com máquinas de 32 e 64 bits.

Como o cluster do LFC é misto, o ambiente de teste foi configurado de forma igual nas máquinas CORTO1 e CORTO5 (64 e 32 bits), com a única diferença de utilizar os compiladores compatíveis para cada arquitetura. Desta forma foi possível submeter nossa aplicação de forma separada, ou seja, disparando da CORTO1 apenas para os nós de 64 bits e da CORTO5 apenas para os nós de 32 bits.

5.3.1 AMBIENTE 1 - CLUSTER

Nos nós CORTO1 e CORTO5 o MPICH foi compilado com o compilador da PGI, utilizando as opções de compilação sugeridas no site da própria PGI as quais são apresentadas no Quadro 5.2. É importante citar que não foi feita nenhuma configuração sobre o *device* de comunicação utilizado pelo MPICH nas passagens de mensagens, portanto a compilação assume o *device* default que é o **ch_p4**. Além das opções sugeridas no site da PGI foram adicionadas as *flags* **enable-f77** e **enable-f90modules** que habilitam os compiladores MPI para o **F77** e **F90**.

```
env CFLAGS="-fast" CXXFLAGS="-fast" FFLAGS="-fast" F90FLAGS="-fast"
LDFLAGS="-fast" OPTFLAGS="-fast"  CC="pgcc" CXX="pgCC"  F90="pgf90"
FC="pgf77"  CPP="pgCC -E"  ./configure --prefix=/opt/mpich-1.2.7-pgi64-patch
--enable-f77 --enable-f90modules
```

Quadro 5.2 - Opções de compilação utilizados na compilação do MPICH do *cluster* do LFC sugeridas no site da PGI

Para a compilação das bibliotecas BLACS e ScaLAPACK foi utilizado o compilador MPICH compilado para este ambiente.

5.3.2 AMBIENTE 2 - GLOBUS

Da mesma forma que no ambiente 1, o MPICH-G2 foi compilado nos *hosts* CORTO1 e CORTO5 com as mesmas opções sugeridas no site da PGI mais a especificação do *device* de comunicação do globus **globus2**. O Quadro 5.3 apresenta a linha de configuração do MPICH-G2. A opção **with-device** é utilizada para passar como parâmetro o *device* de comunicação do globus.

```
Env CFLAGS="-fast" CXXFLAGS="-fast" FFLAGS="-fast" F90FLAGS="-fast"
LDFLAGS="-fast" OPTFLAGS="-fast"  CC="pgcc" CXX="pgCC"  F90="pgf90"
FC="pgf77"  CPP="pgCC -E"  ./configure --prefix=/opt/mpich-1.2.7-
grid --with-device=globus2:-flavor=gcc64dbg --enable-f77 --enable-
f90modules
```

Quadro 5.3 - Opções de compilação utilizados na compilação do MPICH-G2 do grid do LFC

Para a compilação das bibliotecas BLACS e ScaLAPACK foi utilizado o compilador MPICH-G2 compilado para este ambiente.

O *Globus Toolkit* foi instalado em todas as 8 máquinas do *cluster*, a instalação foi realizada conforme instruções do Apêndice C.

5.4 COMPILAÇÃO DOS EXECUTÁVEIS

Foram compilados 42 códigos executáveis em Fortran 90 de 32 bits para cada um dos 2 ambientes, aumentando o tamanho da matriz a ser calculada e o número de processos em que ela foi distribuída. Em ambos os casos o arquivo *makefile* utilizado foi igual, apenas alterando o *path* das bibliotecas de *linkagem*, referentes à versão *cluster* e *grid*. O Quadro 5.4 apresenta o arquivo *makefile* para a versão *cluster*, e o Quadro 5.5 apresenta o arquivo *makefile* referente a versão do *grid*.

```
F90 = /opt/mpich-1.2.7-pgi32/bin/mpif90
FLAGS = -g -tp piii -lc -Minform=inform -Minfo -Mcache_align -
       Mbounds -lpthread  -lgcc -lgcc_eh
       /opt/pgi/linux86/7.0/lib/libpgc.a -Bstatic

MATHPATH =  -L/opt/share/lib32-pgi
USRLIBPATH = -L/usr/lib/
MATHLIB = -lblacsCinit -lblacsF77init -lblacs -lscalapack -lblacs -
          llapack -lblas -lg2c -lc -lgcc -lgcc_eh

OUTPUT_1 = sample_pcheevx_call.exe
FILES_1  = sample_pcheevx_call.f

all: $(OUTPUT_1)

$(OUTPUT_1) : $(FILES_1)
             $(F90) -o $(OUTPUT_1) $(FLAGS) $(FILES_1) $(MATHPATH)
             $(USRLIBPATH) $(MATHLIB) $(MATHLIB)
```

Quadro 5.4 - Arquivo *makefile* utilizado para compilar a versão *cluster*

```
F90 = /opt/mpich-1.2.7-pgi32-g2a/bin/mpif90
FLAGS = -g -tp piii -lc -Minform=inform -Minfo -Mcache_align -
       Mbounds -lpthread  -lgcc -lgcc_eh
       /opt/pgi/linux86/7.0/lib/libpgc.a -Bstatic

MATHPATH =  -L/opt/share/lib32-pgi-g2
USRLIBPATH = -L/usr/lib/
MATHLIB = -lblacsCinit -lblacsF77init -lblacs -lscalapack -lblacs -
           llapack -lblas -lg2c -lc -lgcc -lgcc_eh

OUTPUT_1 = sample_pcheevx_call.exe
FILES_1  = sample_pcheevx_call.f

all: $(OUTPUT_1)

$(OUTPUT_1) : $(FILES_1)
             $(F90) -o $(OUTPUT_1) $(FLAGS) $(FILES_1) $(MATHPATH)
             $(USRLIBPATH) $(MATHLIB) $(MATHLIB)
```

Quadro 5.5 - Arquivo *makefile* utilizado para compilar a versão *grid*

Os executáveis foram nomeados no seguinte formato **sample_pcheevx_call_32_nP_MAXN.exe**, onde **sample_pcheevx_call** é o nome principal do executável, 32 se refere a uma compilação para a arquitetura de 32 bits, **nP** é o número de processos em que a matriz é distribuída e **MAXN** é o tamanho da matriz. Por exemplo, para o executável **sample_pcheevx_call_32_8P_10000.exe**, tem-se uma execução para arquitetura de 32 bits em que a matriz de tamanho 10000 será distribuída em 8 processos.

Os testes foram realizados apenas para códigos compilados para a arquitetura de 32 bits, uma vez que a execução dos testes para o programa compilado na plataforma de 64 bits acusou o seguinte erro: ***0 - MPI_ADDRESS : Address of location given to MPI_ADDRESS does not fit in Fortran integer***. Esse erro, segundo o site da PGI, está relacionado com a forma de endereçamento da memória de 32 para 64 bits, já que o FORTRAN utiliza um tipo de dados inteiro que não é grande o suficiente para endereçar a memória em 64 [27]. No próprio site há um *patch* do MPICH para as versões de 64 bits que foi aplicado. Mesmo assim o erro persistiu.

5.5 SUBMISSÃO DOS PROCESSOS

Nos dois ambientes, cada executável foi submetido 5 vezes e o tempo de cada execução foi medido utilizando o comando do sistema **time –pv –o <arquivo_saida.dat>**. Mais precisamente, a informação de tempo de execução é dada no formato **hh:mm:ss** na linha *Elapsed (wall clock) time.* O tempo de execução foi transformado em segundos para a plotagem dos gráficos dos resultados obtidos. O Quadro 5.6 apresenta o resultado do comando após o término da execução.

```
Command being timed: "/opt/mpich-1.2.7-pgi64-g2a/bin/mpirun -
                          globusrsl sub12p_in64-32_8000.rsl"
User time (seconds): 0.46
System time (seconds): 0.07
Percent of CPU this job got: 0%
Elapsed (wall clock) time (h:mm:ss or m:ss): 27:01.43
Average shared text size (kbytes): 0
Average unshared data size (kbytes): 0
Average stack size (kbytes): 0
Average total size (kbytes): 0
Maximum resident set size (kbytes): 0
Average resident set size (kbytes): 0
Major (requiring I/O) page faults: 0
Minor (reclaiming a frame) page faults: 8607
Voluntary context switches: 3581
Involuntary context switches: 66
Swaps: 0
File system inputs: 0
File system outputs: 0
Socket messages sent: 0
Socket messages received: 0
Signals delivered: 0
Page size (bytes): 4096
Exit status: 0
```

Quadro 5.6 - Saída do comando time

A maneira como os processos MPI são disparados diferem do ambiente 1 (*cluster*) para o ambiente 2 (*grid*). No *cluster*, a forma para disparar os processos MPI é a tradicionalmente utilizada (Quadro 5.7), já no ambiente do *grid* é necessário criar o arquivo de submissão de processos RSL (*Resource Specification Language*) e em seguida dispará-lo. Para criar o arquivo RSL o comando utilizado é o **mpirun,** seguido da opção **–dumprsl** e para disparar os processos o comando é o **mpirun,** seguido da opção **–globusrsl.** O Quadro 5.8 apresenta o comando de criação da RSL e o comando para disparar o processo no *grid*. O arquivo RSL já foi apresentado no Quadro 4.1 do Capítulo 4.

```
/usr/bin/time -pv -o tempo_9P_10000-1.txt /opt/mpich-1.2.7-pgi64-
patch/bin/mpirun -machinefile machines2 -np 9
sample_pcheevx_call_32_9P_10000.exe
```

Quadro 5.7 - Submissão clássica de processos MPI

O Quadro 5.7 apresenta o método clássico de submissão de processos MPI e a linha completa de comando pelo qual foi iniciado o processo MPI, juntamente com o comando **time** utilizado para a coleta do tempo de execução. As opções do comando **time –pv** são, respectivamente, *portability* (conformidade com o padrão POSIX) e *verbose* (mostrar todas as informações de saída do comando). A opção **–o** redireciona para um arquivo de saída, que no caso do exemplo é **tempo_9P_10000-1.txt**. Já as opções do comando **mpirun** são **–machinefile <lista_hosts>**, em que **lista_hosts** é um arquivo com o nome dos nós onde os processos serão disparados; **-np X**, em que **X** é o número de processos que serão disparados entre os nós e por fim o nome do arquivo executável.

```
mpirun -dumprsl –machinefile machines2 -np 8
sample_pcheevx_call_32_8P_10000.exe > sub8p_in32_10000.rsl

/usr/bin/time -apv -o tempo_8p_32_10000-1.txt /opt/mpich-1.2.7-
pgi32-g2a/bin/mpirun -globusrsl sub8p_in32_10000.rsl
```

Quadro 5.8 - Criação do arquivos RSL e submissão dos processo no *grid*

No Quadro 5.8, o comando **mpirun –dumprsl** redireciona a saída da execução para o arquivo **sub8p_in32_10000.rsl** e em seguida o processo é disparado no *grid* através do comando **mpirun –globusrsl** que recebe como entrada o mesmo arquivo **sub8p_in32_10000.rsl**.

Como citado anteriormente os testes foram executados 5 vezes para cada um dos 42 executáveis gerados nos 2 ambientes. Ao final de cada bateria de 5 execuções calculamos a média aritmética e este foi o tempo considerado em nossos resultados. A Tabela 5.3, apresenta como exemplo, as tomadas de tempos para um executável de 6 processos MPICH-G2 nas

máquinas de 64 bits. MAXN é o tamanho da matriz, as linhas R1 a R5 são os tempos em segundos de cada execução, a linha MPICH-G2 é a média aritmética das 5 execuções para cada tamanho de matriz.

Tabela 5.3 - Exemplo de tomada de tempo para 6 processos com MPICH-G2

	MAXN				
	1000	**2000**	**4000**	**6000**	**8000**
R1	25,01	82,65	566,52	465,78	1040,52
R2	21,95	82,44	568,11	465,59	1040,01
R3	21,99	82,45	556,08	473,69	1040,17
R4	21,89	82,67	556,27	465,65	1047,62
R5	21,96	82,61	556,25	465,57	1050,43
MPICH-G2	**22,56**	**82,56**	**560,65**	**467,26**	**1043,75**
Desvio Padrão	**±1,37**	**±0,11**	**±6,11**	**±3,60**	**±4,92**

Para os executáveis com 4, 6 e 8 processos foram executados testes nas máquinas de 64 e 32 bits separadamente. Para o executável **sample_pcheevx_call_32_4P_1000.exe,** por exemplo, o programa foi executado disparando do nó CORTO1 e distribuindo 1 processo em cada um dos nós de CORTO1 até CORTO4 e para o mesmo programa os processos foram submetidos disparando do nó CORTO5 e distribuindo 1 processo em cada um dos nós de CORTO5 até CORTO8. Já para os executáveis com 9 e 12 processos os programas foram executados sempre disparando do nó CORTO1 e distribuindo os processos nas duas arquiteturas.

A Tabela 5.4 apresenta o nome de cada um dos executáveis e como eles foram distribuídos entre os nós. Vale lembrar que cada um dos nós é bi-processado, portanto eles suportam até dois processos sem concorrência.

Tabela 5.4 - Distribuição dos executáveis entre os nós

Executável	**Processos 64 bits**	**Processos 32 bits**
sample_pcheevx_call_32_4P_MAXN.exe	1 em CORTO1 1 em CORTO2 1 em CORTO3 1 em CORTO4	1 em CORTO5 1 em CORTO6 1 em CORTO7 1 em CORTO8
sample_pcheevx_call_32_6P_MAXN.exe	2 em CORTO1 2 em CORTO2 1 em CORTO3 1 em CORTO4	2 em CORTO5 2 em CORTO6 1 em CORTO7 1 em CORTO8
sample_pcheevx_call_32_8P_MAXN.exe	2 em CORTO1 2 em CORTO2 2 em CORTO3 2 em CORTO4	2 em CORTO5 2 em CORTO6 2 em CORTO7 2 em CORTO8
sample_pcheevx_call_32_8P_MAXN.exe *Concorrência*	4 em CORTO1 4 em CORTO2	4 em CORTO5 4 em CORTO7
sample_pcheevx_call_32_9P_MAXN.exe *Concorrência*	3 em CORTO1 3 em CORTO2 3 em CORTO3	3 em CORTO5 3 em CORTO7 3 em CORTO8
	Processos 64 e 32 bits (ambiente misto)	
sample_pcheevx_call_32_9P_MAXN.exe	2 em CORTO1 2 em CORTO2 1 em CORTO3 2 em CORTO5 2 em CORTO7	
sample_pcheevx_call_32_12P_MAXN.exe	2 em CORTO1 2 em CORTO2 2 em CORTO3 2 em CORTO4 2 em CORTO7 2 em CORTO8	

5.6 PROCEDIMENTO PARA A COLETA DE BYTES ENVIADOS E RECEBIDOS ENTRE OS PROCESSOS

Após a realização dos testes de desempenho e de posse do tempo médio da execução de cada uma das aplicações para cada tamanho de matriz, foi efetuada mais uma rodada com alguns executáveis a fim de obter informações sobre a quantidade de bytes recebidos e enviados entre os processos. Para tanto, foram utilizados dois *scripts*: nomeados **rede.sh** e

monitor-rede.sh. O *script* **rede.sh,** apresentado no Quadro 5.9, faz a leitura do diretório **/proc/net/dev** coletando o número de bytes recebidos e enviados pelas placas de rede do computador naquele instante e escrevendo na saída padrão.

```
#!/bin/sh
awk '
/'$1':/ {
$0=substr($0,index($0,":")+1);
print $1," ",$9
}
' /proc/net/dev
```

Quadro 5.9 - *Script* rede.sh

O *script* **monitor-rede.sh** realiza um *loop* a cada 1 segundo chamando o *script* **rede.sh** e escrevendo na saída padrão o número de bytes recebidos e enviados pelo dispositivo selecionado segundo a segundo. Para os testes realizados, variou-se o *loop* pelo tempo médio calculado na execução da aplicação (ver Tabela 5.3) desviando a saída para um arquivo. Por exemplo, se a aplicação tem um tempo médio de 15 segundos para ser executada, o FOR do *script* **monitor-rede.sh** varia de 1 até 15 coletando, assim, 15 iterações para os bytes recebidos e enviados no arquivo de saída. O *script* **monitor-rede.sh** é apresentado pelo Quadro 5.10 e um exemplo do arquivo de saída deste *script* é apresentado pelo Quadro 5.11, no qual a primeira coluna é referente aos bytes recebidos e a segunda os bytes enviados. Note que, neste exemplo, as 15 iterações correspondem a 15 linhas mostradas no arquivo.

```
#!/bin/bash
for a in `seq 1 15`; do
    comando=`rede.sh eth1`
    echo $comando
    sleep 1;
Done
```

Quadro 5.10 - Script monitor-rede.sh

```
18814007159 19638327877
18814010207 19638333301
18814596127 19638369736
18814677105 19638401263
18822592224 19649309760
18828486672 19657146617
18829730201 19658676466
18829730201 19658676466
18829730201 19658676466
18829730355 19658676596
18829739875 19658676596
18829760159 19658676666
18832433517 19663937162
18835780135 19667267992
18835780229 19667268062
```

Quadro 5.11 - Saída da execução do script monitor-rede.sh

Para o exemplo do Quadro 5.11, da primeira coluna foi subtraída a segunda linha da primeira, a terceira da segunda e assim sucessivamente até o final do arquivo obtendo, assim, o número de bytes recebidos em cada segundo da execução da aplicação. O mesmo procedimento foi realizado para a segunda coluna, obtendo o número de bytes enviados em cada segundo da execução da aplicação.

CAPÍTULO 6

Resultados e Discussões

Para este estudo, foram analisados dois tipos de casos distintos. No primeiro tipo, todos os processos foram submetidos sem a condição de concorrência, ou seja, cada núcleo de processamento executou apenas um processo. Esses resultados são apresentados na seção 6.1. A seção 6.2 apresenta os resultados na qual a condição de concorrência foi simulada, com pelo menos um núcleo de processamento executando dois processos ao mesmo tempo.

Os tempos são apresentados em segundos e representam a média aritmética de 5 execuções para cada sistema com mesmo número de processadores e mesmo tamanho de matriz.

6.1 TESTES REALIZADOS SEM CONCORRÊNCIA

As tomadas de tempo para uma distribuição de matriz com 1, 4, 6 e 8 processos foram obtidas separadamente para os nós de 32 e 64 bits. As tomadas de tempo com apenas 1 processo foram realizadas apenas para a utilização nos cálculos do *Speedup*. Nos gráficos apresentados, elas não foram relacionadas. Como o *cluster* contém apenas 8 núcleos de processamento de cada tipo, testes com 9 e 12 processos puderam ser feitos apenas em um conjunto misto de núcleos de processamento de 32 e 64 bits. A Tabela 6.1 a seguir apresenta os resultados obtidos. Os campos em branco na tabela não foram calculados devido aos limites de memória. Quando a matriz é distribuída em poucos processos eles ultrapassam a quantidade de memória disponível por nó para realização dos cálculos.

Tabela 6.1 - Tomada de tempo (s) entre MPICH-G2 e MPICH distribuídos em 1, 4, 6, 8, 9 e 12 processos, sem concorrência entre os processos

Proces-sos	Arqui-tetura		MAXN						
			1000	2000	4000	6000	8000	10000	11000
1	32 bits	MPICH-G2	53,10	358,16					
		MPICH	58,98	461,58					
	64 bits	MPICH-G2	21,15	141,38					
		MPICH	28,71	223,82					
4	32 bits	MPICH-G2	33,39	116,51	1521,92	1755,24			
		MPICH	29,82	228,72	1804,05	2123,20			
	64 bits	MPICH-G2	21,52	81,91	573,70	656,87			
		MPICH	14,87	105,9	806,89	1214,98			
6	32 bits	MPICH-G2	44,40	209,60	1516,68	1427,23	3283,63		
		MPICH	32,24	219,31	1678,27	1738,81	3608,09		
	64 bits	MPICH-G2	22,56	82,56	560,65	467,26	1043,75		
		MPICH	15,78	102,94	736,48	873,16	1972,73		
8	32 bits	MPICH-G2	46,22	208,93	1428,96	1091,91	2488,82	4718,33	5619,67
		MPICH	31,68	209,38	1573,07	1270,86	3033,36	5814,67	7882,67
	64 bits	MPICH-G2	22,66	83,12	537,37	365,27	811,74	1534,52	2016,21
		MPICH	16,54	95,54	684,80	682,34	1541,62	2926,90	3864,33
9	32/64 bits	MPICH-G2	32,18	113,65	708,95	999,80	2319,50	4449,00	5879,00
		MPICH	21,33	109,64	794,85	1127,08	2624,00	5247,00	7165,00
12	32/64 bits	MPICH-G2	32,71	105,88	646,41	718,86	1620,57	3074,37	4039,00
		MPICH	18,12	105,28	746,64	893,89	2031,05	3874,33	5117,67

Apesar da velocidade de comunicação das placas de rede nos nós de 32 e 64 bits serem iguais (1Gbit/s), foi utilizado o programa NETPERF [28] que realiza testes de *benchmark* a fim de medir a performance da rede. Foram realizadas várias medidas e verificou-se que para os nós de 64 bits, o *throughput* médio das placas de rede ficou em torno de 900 Mbits/s, enquanto que para os nós de 32 bits, o *throughput* médio das placas de rede, ficou em torno de 650 Mbits/s, ou seja, o aproveitamento da comunicação *gigabit* dos nós de 64bits é 28% melhor do que os nós de 32 bits. O melhor aproveitamento nos nós de 64 bits está relacionado com o barramento da placa de rede, uma vez que nos nós de 64 bits a placa de rede estão conectadas ao barramento PCI-*Express* e nos de 32 bits ao barramento PCI.

Em todos os testes realizados, o tempo de execução dos nós de 64 bits foi menor que os dos de 32 bits, o que é consistente, uma vez que o *throughput* médio dos nós de 64 bits é

maior e os núcleos de processamento dos nós de 64 bits encontram-se pelo menos duas gerações a frente dos de 32 bits.

O MPICH teve melhor desempenho apenas no caso de matrizes 1000 x 1000, ou 2000 x 2000 para 9 e 12 processos (neste último caso com um ganho quase insignificante). Já para outros tamanhos de matrizes, o MPICH-G2 teve um desempenho melhor. Este mesmo comportamento ocorreu para todas as aplicações testadas nestas condições e ele pode ser comprovado segundo os testes realizados pelos próprios desenvolvedores da biblioteca MPICH-G2 [29]. Eles realizaram testes no qual variavam o tamanho das mensagens enviadas de 1KB até 1MB entre 2 máquinas SUN, onde a primeira se conectava à rede através de uma placa de rede *gigabit* e a segunda através de uma placa de rede *fastethernet*. Os testes realizados comparavam o desempenho da comunicação TCP/IP entre as bibliotecas MPICH-G2, o MPICH-G e o MPICH. Os gráficos, a seguir, apresentam os resultados desses testes, mostrando no gráfico da Figura 6.1 que, entre os tamanhos de mensagens de 15KB e 20KB, o desempenho do MPICH-G2 ultrapassa o do MPICH. Já no gráfico da Figura 6.2, nota-se uma tendência do desempenho do MPICH-G2 ser melhor quando comparado ao do MPICH, para tamanhos maiores de mensagens.

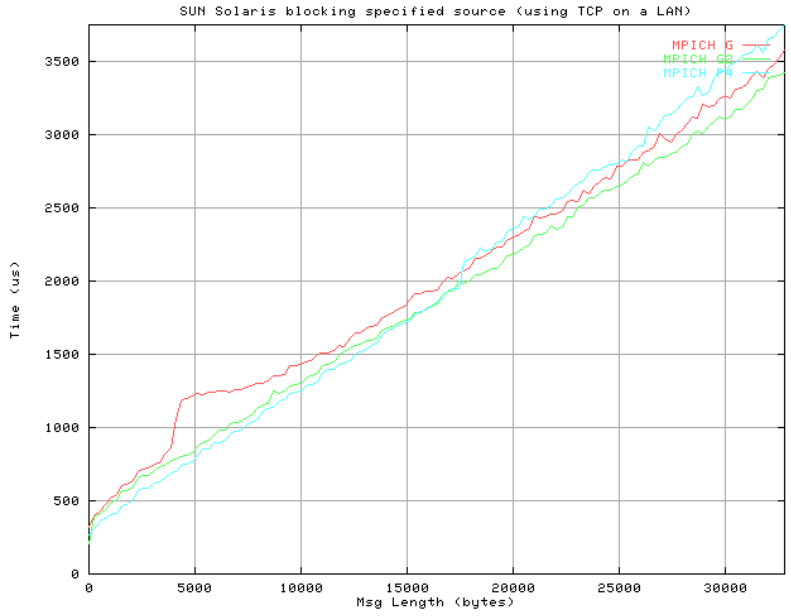

Figura 6.1 – Performance do TCP/IP entre MPICH-G2, MPICH-G e MPICH - Tamanho das mensagens em relação ao tempo (Tamanho das mensagens de 0KB à 30KB) – Retirado do site do MPICH-G2 (http://www3.niu.edu/mpi/)

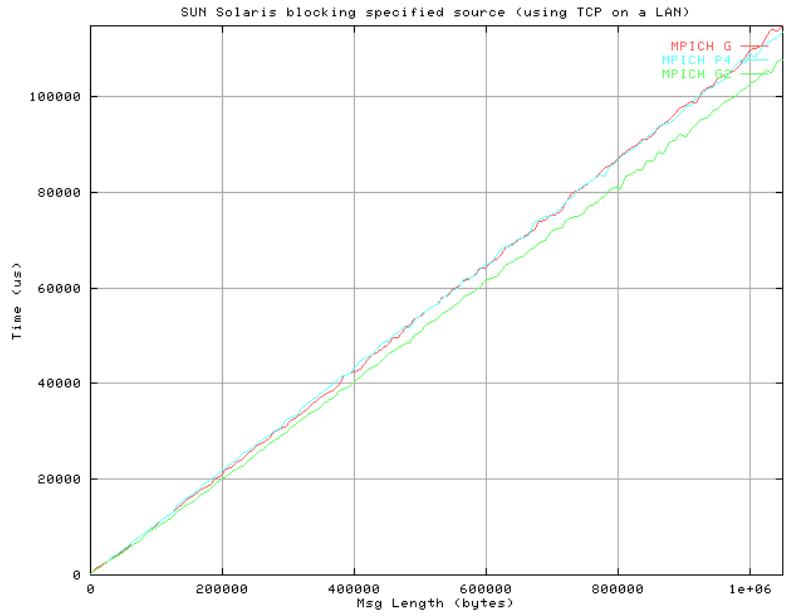

Figura 6.2 - Performance do TCP/IP entre MPICH-G2, MPICH-G e MPICH - Tamanho das mensagens em relação ao tempo (Tamanho das mensagens de 0KB à 1MB) – Retirado do site do MPICH-G2 (http://www3.niu.edu/mpi/)

O Gráfico 6.1 compara os tempos de processamento da aplicação com o MPICH-G2 e com o MPICH nas duas arquiteturas e matrizes distribuídas em 4, 6 e 8 processos.

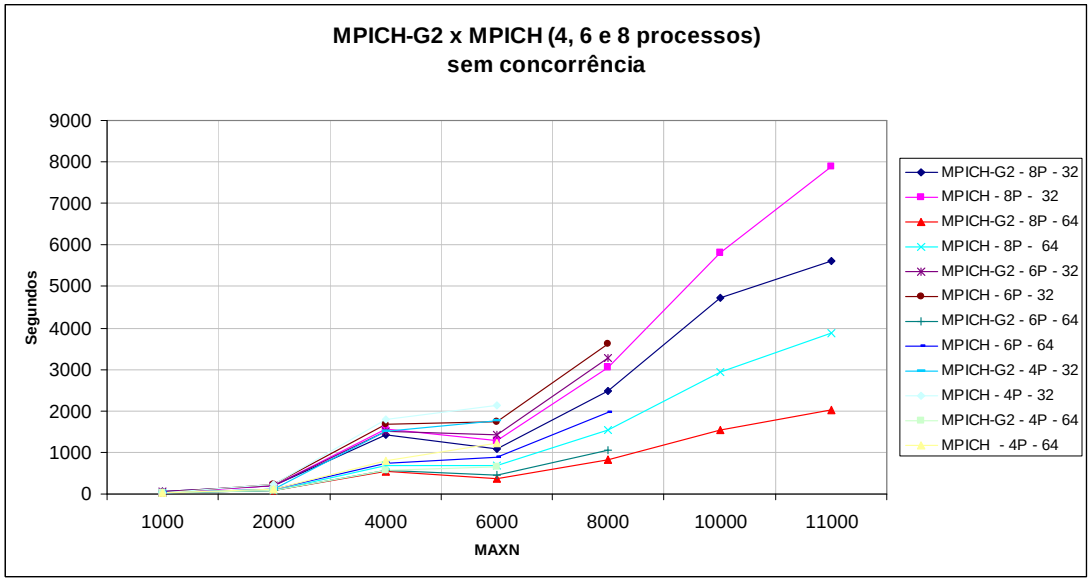

Gráfico 6.1 - Comparação do desempenho do MPICH-G2 com o MPICH para as arquiteturas de 64 e 32 bits e para a matriz distribuída em 4, 6 e 8 processos sem concorrência

Do Gráfico 6.1 pode-se afirmar que para o tipo de aplicação proposta, e para matrizes maiores que 2000, o melhor desempenho foi obtido com o MPICH-G2 em todos os casos, e para o MPICH-G2 sendo executado nos nós de 64 bits o tempo de execução caiu em média 50% para matrizes a partir do tamanho 6000 em relação ao MPICH.

O melhor desempenho do MPICH-G2 em um ambiente sem concorrência pode ser explicado com o auxílio dos gráficos a seguir, nos quais foram coletados os totais de bytes por segundo recebidos e enviados entre os nós que participaram do processamento. Para uma melhor compreensão dos gráficos foram retirados os totais de bytes enviados entre os nós, pois apresentam o mesmo comportamento dos totais de bytes recebidos.

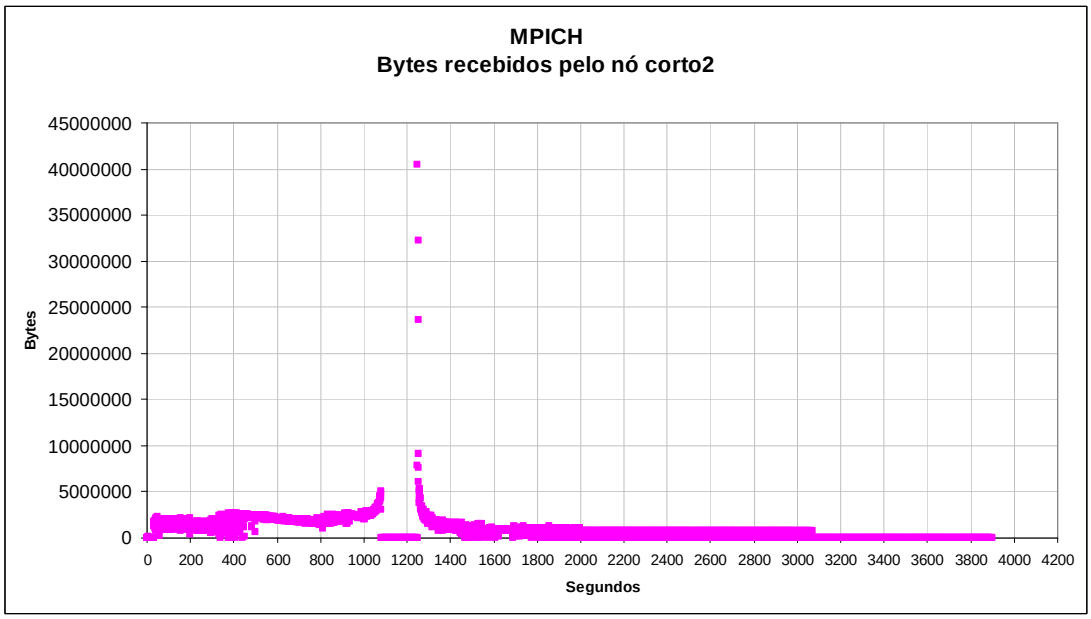

Gráfico 6.2 - Bytes recebidos pelo nó corto2 para uma distribuição de matriz em 8 processos de tamanho 11000 x 11000 para o MPICH

Para entender o Gráfico 6.2, é necessário entender como a rotina PCHEEVX do ScaLAPACK funciona. Como já foi explicado anteriormente, essa rotina calcula autovalores e autovetores de uma matriz hermitiana. Para isso, em um primeiro momento, ela realiza a tridiagonalização da matriz com a comunicação otimizada devido à distribuição bloco cíclica. Essa tridiagonalização dura até o primeiro pico do gráfico. O *gap* é o momento com baixa comunicação, no qual os nós estão calculando os autovalores. No segundo pico é iniciada a determinação dos autovetores, e a partir deste ponto a comunicação cai até a finalização de todos os cálculos.

Ainda com o Gráfico 6.2, nota-se que a parte mais consistente da comunicação está situada de 0 até em torno de 1100 segundos, quando ocorre o primeiro pico, que é o momento em que todos os nós estão realizando a tridiagonalização da matriz. Em seguida, no *gap*, (entre 1100 a 1250) segundos é realizado o cálculo dos autovalores. Do segundo pico em diante, é iniciada a determinação dos autovetores e a finalização dos cálculos. Este

comportamento aconteceu para todos os testes realizados para a situação na qual não houve concorrência entre os processos.

A seguir, são apresentados dois gráficos, nos quais a aplicação é executada com 8 processos para uma matriz de tamanho 11000 x 11000: o Gráfico 6.3 mostra o comportamento com o MPICH-G2 e o Gráfico 6.4 com o MPICH. Nesses gráficos, foram limitadas as escalas do eixo Y (Bytes) em 10MB para obter melhor condição de visualização e também foi mantida apenas a comunicação de um nó, pois todos os nós apresentam o mesmo comportamento na comunicação.

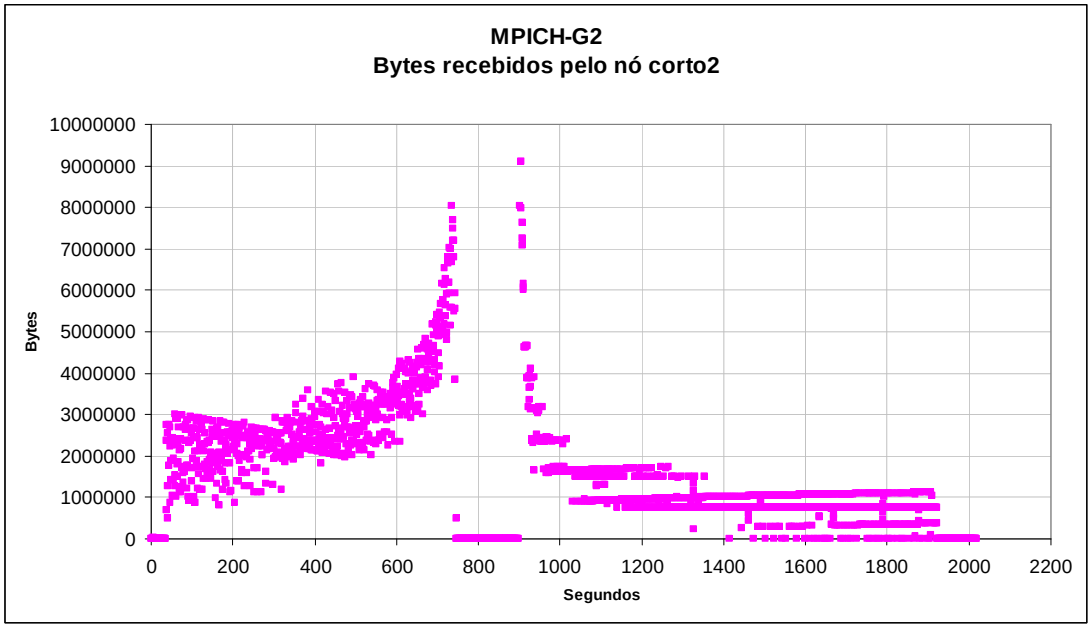

Gráfico 6.3 - Bytes recebidos pelo nó corto2 para uma distribuição de matriz em 8 processos de tamanho 11000 x 11000 para o MPICH-G2 (Eixo Y limitado a 10MB)

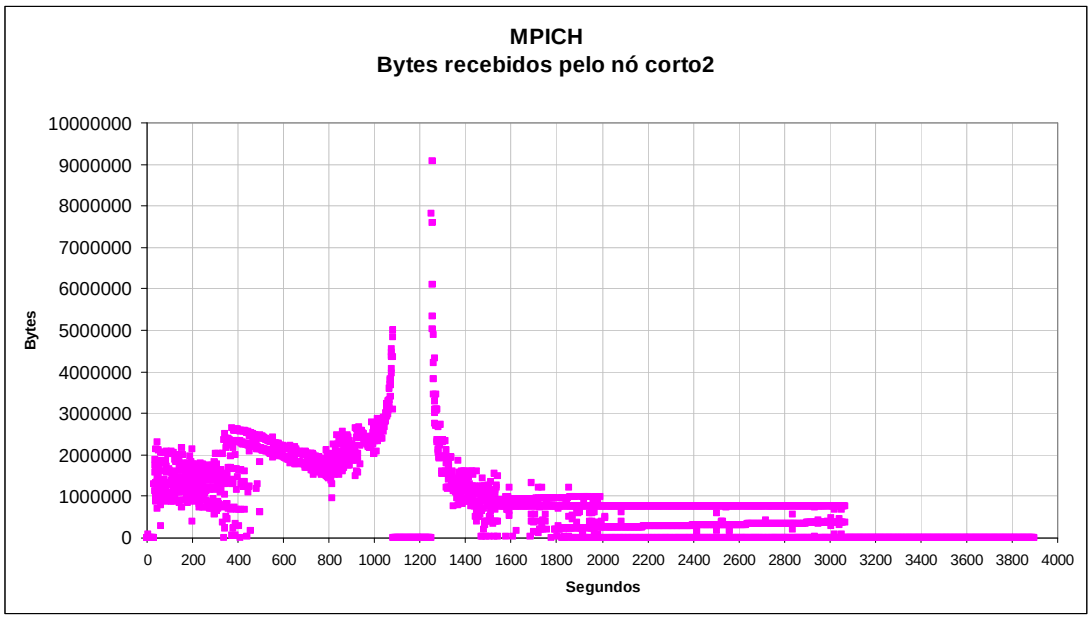

Gráfico 6.4 - Bytes recebidos pelo nó corto2 para uma distribuição de matriz em 8 processos de tamanho 11000 x 11000 para o MPICH (Eixo Y limitado a 10MB)

Comparando-se os gráficos 6.3 e 6.4, nota-se que o total de bytes por segundo recebidos pelo nó na tridiagonalização da matriz com o MPICH-G2 é maior, ficando entre 3MB e 4MB, enquanto que para o MPICH o total de bytes por segundo fica entre 2MB e menos de 3MB. Devido a isto, o MPICH-G2 realiza a tridiagonalização da matriz em um tempo menor que o MPICH. Já o *gap*, tem tempos de execução parecidos (variando em torno do intervalo de 750 a 950 segundos para o MPICH-G2, e 1050 a 1250 segundos para o MPICH) ou seja, o cálculo dos autovalores leva em média 200 segundos em ambos os ambientes para esta situação. Isto acontece porque a comunicação nesta fase não é importante, apenas a capacidade de processamento de cada núcleo. Do segundo pico de comunicação até o final dos cálculos (no qual são determinados os autovetores) a comunicação do MPICH-G2 ainda se mantém em um patamar mais elevado, variando de 1MB a 2MB, contra 1MB do MPICH.

Baseado nestes dados: o maior número de bytes por segundos comunicados pelo MPICH-G2 nas fases inicial e final, e igual na fase do *gap*, pode-se supor que, para este tipo

de aplicação, o MPICH-G2 utiliza a comunicação de maneira mais eficiente que o MPICH. Isto foi afirmado, segundo comunicação pessoal [30] com um dos desenvolvedores do MPICH-G2, onde ele explica que o MPICH utiliza um *buffer* intermediário para as operações de *send* e *receive*, e no MPICH-G2 esse *buffer* foi retirado.

O Gráfico 6.5, representa uma distribuição de matriz com 9 e 12 processos de forma simultânea nos nós de 32 e 64 bits. Também vale notar que, para os processos disparados em uma arquitetura mista, o MPICH tem melhor desempenho para calcular uma matriz de tamanho 1000 x 1000. Quanto a matrizes de tamanho 2000 x 2000, o desempenho do MPICH também é melhor para esta arquitetura mista (ver Tabela 6.1). A diferença de tempo, no entanto, é muito pequena. Isto, como já foi comentado está provavelmente relacionado com o tamanho dos pacotes MPI. Já para todos os outros tamanhos de matrizes calculados o MPICH-G2 obteve melhores resultado.

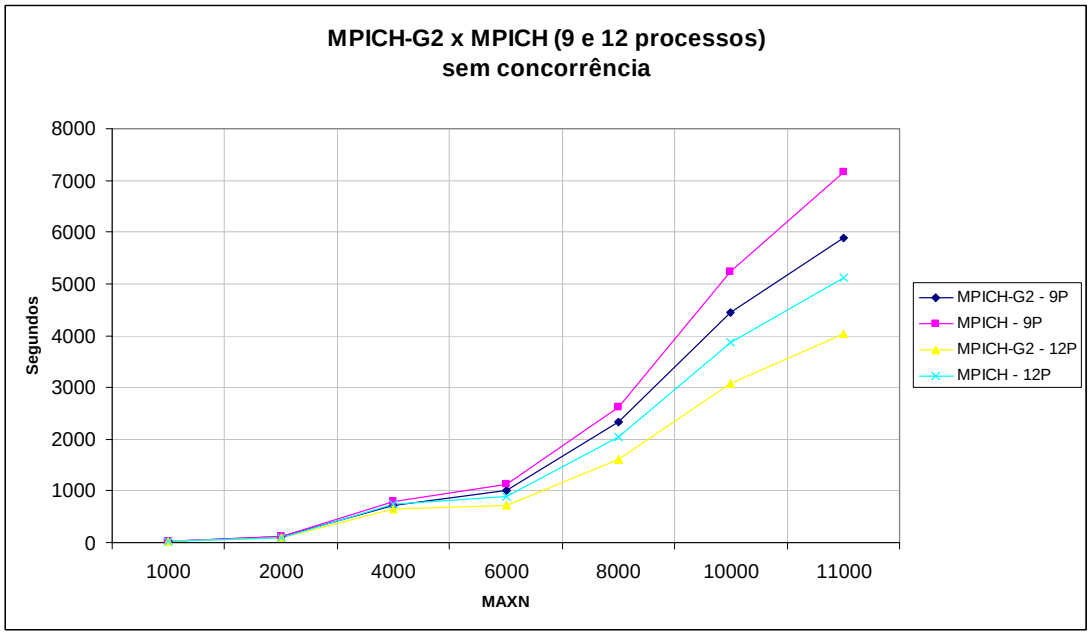

Gráfico 6.5 - Comparação do desempenho do MPICH-G2 com o MPICH para a matriz distribuída de forma mista nos nós de 32 e 64 bits para 9 e 12 processos

Com o auxílio do Gráfico 6.5, pode-se afirmar que, para o tipo de aplicação proposta e para os processos distribuídos de forma simultânea nos nós de 32 e 64 bits, o melhor

desempenho foi obtido com o MPICH-G2, no entanto esta melhor performance ficou em torno de 20%. Nesta situação, tanto o desempenho do MPICH quanto o do MPICH-G2 ficaram limitados ao desempenho dos nós de 32 bits, uma vez que o *throughput* alcançado pelas placas de rede é menor e o próprio núcleo de processamento também é mais lento.

Comparando o melhor caso das duas situações, ou seja, a aplicação sendo executada com 8 processos em nós de 64 bits, com a aplicação sendo executada com 12 processos em nós de 32 e 64 bits simultaneamente, pode-se afirmar de acordo com os testes realizados, que para o tipo de aplicação proposta o melhor desempenho é obtido nos nós de 64 bits distribuindo a matriz em 8 processos. Como pode ser observado com o auxílio do Gráfico 6.6. Isto pode ser explicado, em nosso sistema, pelo fato dos processos serem executados mais rapidamente nos nós de 64 bits que nos de 32 bits, e quando submetemos nas duas arquiteturas os nós de 64 bits terminam a execução de seus processos e ficam aguardando que os nós de 32 bits terminem a execução de seus processos. Como os nós de 64 bits são 2 a 3 vezes mais rápidos que os de 32 bits, os processos de 32 terminam muito defasados.

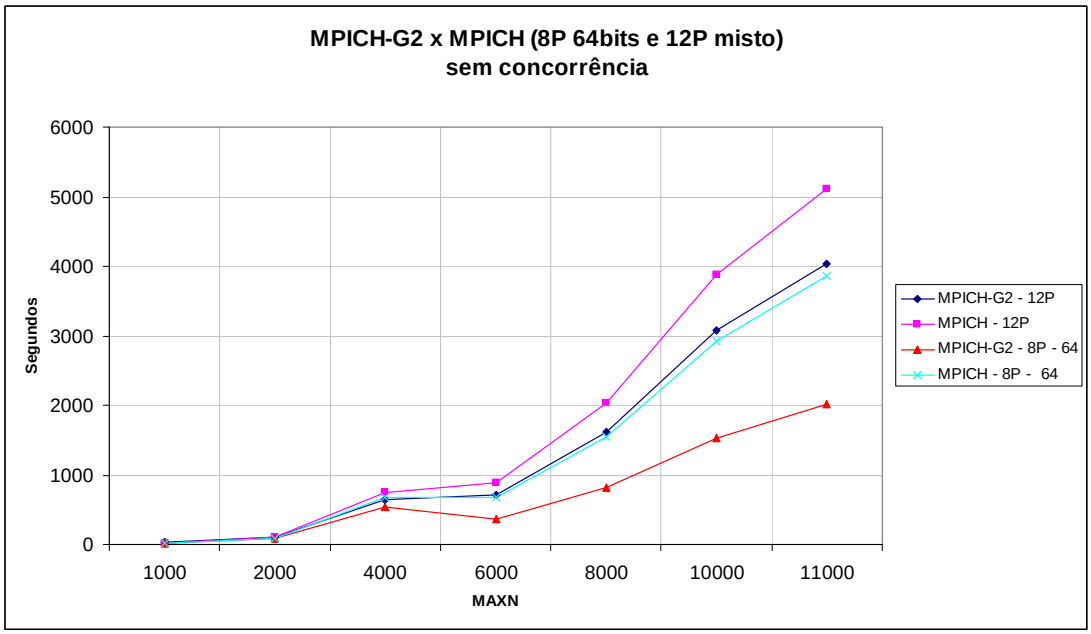

Gráfico 6.6 - Comparação do desempenho entre os melhores resultados obtidos nos nós de 64 bits com 8 processos e nos nós mistos com 12 processos

A seguir é apresentada a Tabela 6.2, que calcula qual a razão entre os desempenhos do MPICH e do MPICH-G2, para todos os testes no qual aplicação foi executada sem a condição de concorrência entre os processos. Para tal, os tempos de diagonalização do MPICH foram divididos pelos tempos do MPICH-G2.

Tabela 6.2 – Razão entre o MPICH e o MPICH-G2

	MAXN						
	1000	**2000**	**4000**	**6000**	**8000**	**10000**	**11000**
4P - 64	0,69	1,29	1,41	1,85			
6P - 64	0,70	1,25	1,31	1,87	1,89		
8P - 64	0,73	1,15	1,27	1,87	1,90	1,91	1,92
9P - Misto	0,66	0,96	1,12	1,13	1,13	1,18	1,22
12P - Misto	0,55	0,99	1,16	1,24	1,25	1,26	1,27
4P - 32	0,89	1,96	1,19	1,21			
6P - 32	0,73	1,05	1,11	1,22	1,10		
8P - 32	0,69	1,00	1,10	1,16	1,22	1,23	1,40

Com o auxílio da Tabela 6.2, é verificado que o desempenho do MPICH-G2 melhora à medida que o tamanho da matriz aumenta, exceto nos casos da matriz de tamanho 1000 x 1000 em todas as distribuições de processo, e 2000 x 2000, para a matriz distribuída em 9 e 12 processos nos nós de 32 e 64 bits.

Com o Gráfico 6.7, temos uma representação visual do melhor desempenho do MPICH-G2, e também se nota que para o tipo de aplicação testada o melhor desempenho do MPICH-G2 é obtido para as aplicações onde a matriz é distribuída em 6 e 8 processos nos nós de 64 bits.

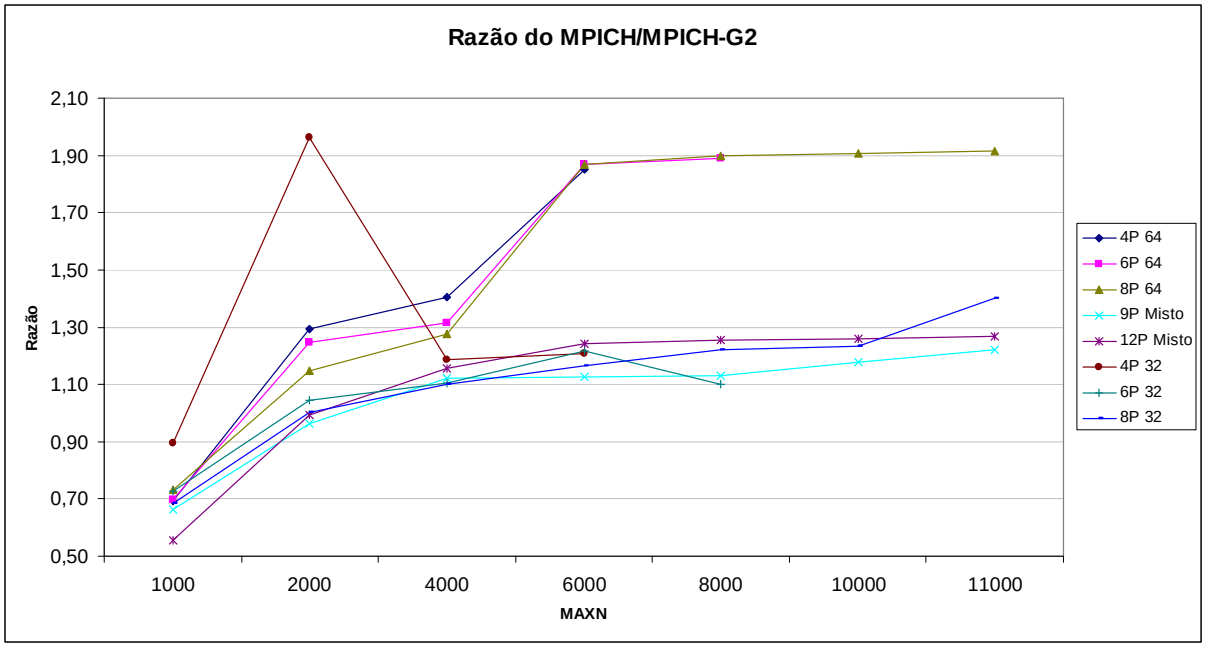

Gráfico 6.7 - Razão entre o MPICH e o MPICH-G2

Além do cálculo da razão entre o MPICH-G2 e o MPICH, foi calculado o *Speedup* [31] entre os as aplicações executadas. O *Speedup* é uma métrica muito utilizada para medir performance de aplicações paralelas.

O *Speedup* é calculado pela expressão:

$$Sp = T1 / Tp$$

T1 é o tempo que a aplicação leva para ser executada com um processo e **Tp** é o tempo em que esta mesma aplicação leva para ser executada com **n** processos.

Como a aplicação com 1 processo, só calculou os tamanhos de matrizes 1000 e 2000, foi utilizado o menor tempo de cada um dos casos, como base de **T1** para calcular o *Speedup*. Esse tempo foi o do MPICH-G2 em 64 bits para as matrizes de tamanho 1000 e 2000, que respectivamente terminaram a execução em 21,15 e 141,38 segundos. Já para o tamanho de matriz 4000 e 6000, como não havia uma medida de tempo com um processador foi utilizado o menor tempo de execução com 4 processos, multiplicando este tempo por 4, como base de **T1**, ou seja, o tempo do MPICH-G2 em 64 bits, que foi respectivamente de 573,70 e 656,87

segundos. Para uma matriz de tamanho 8000, foi utilizado o menor tempo de execução com 6 processos, multiplicando este tempo por 6, que é o tempo do MPICH-G2 em 64 bits (1043,7 segundos). Por fim, para as matrizes de tamanho 10000 e 11000, foram utilizados os menores tempos de execução com 8 processos, que foram os tempos do MPICH-G2 em 64 bits, multiplicados por 8 (1534,52 e 2016,21 segundos respectivamente).

A Tabela 6.3 apresenta os *Speedups* calculados para todos os casos. O Gráfico 6.8 apresenta os *Speedups* para os nós de 64 bits e o Gráfico 6.9 apresenta os *Speedups* para os nós de 32 bits e misto (32 e 64 bits).

Tabela 6.3 – *Speedup* para a aplicação executada sem concorrência

Speedup	1000	2000	4000	6000	8000	10000	11000
MPICH-G2 - 4P - 64	0,98	1,73					
MPICH - 4P – 64	1,42	1,34					
MPICH-G2 - 4P - 32	0,63	1,21					
MPICH - 4P – 32	0,71	0,62					
MPICH-G2 - 6P - 64	0,94	1,71	4,09	5,62			
MPICH - 6P – 64	1,34	1,37	3,12	3,01			
MPICH-G2 - 6P - 32	0,48	0,67	1,51	1,84			
MPICH - 6P – 32	0,66	0,64	1,37	1,51			
MPICH-G2 - 8P - 64	0,93	1,70	4,27	7,19	7,71		
MPICH - 8P – 64	1,28	1,48	3,35	3,85	4,06		
MPICH-G2 - 8P - 32	0,46	0,68	1,61	2,41	2,52		
MPICH - 8P – 32	0,67	0,68	1,46	2,07	2,06		
MPICH-G2 - 9P - M	0,66	1,24	3,24	2,63	2,70	2,76	2,74
MPICH - 9P – M	0,99	1,29	2,89	2,33	2,39	2,34	2,25
MPICH-G2 - 12P - M	0,65	1,34	3,55	3,66	3,86	3,99	3,99
MPICH - 12P – M	1,17	1,34	3,07	2,94	3,08	3,17	3,15

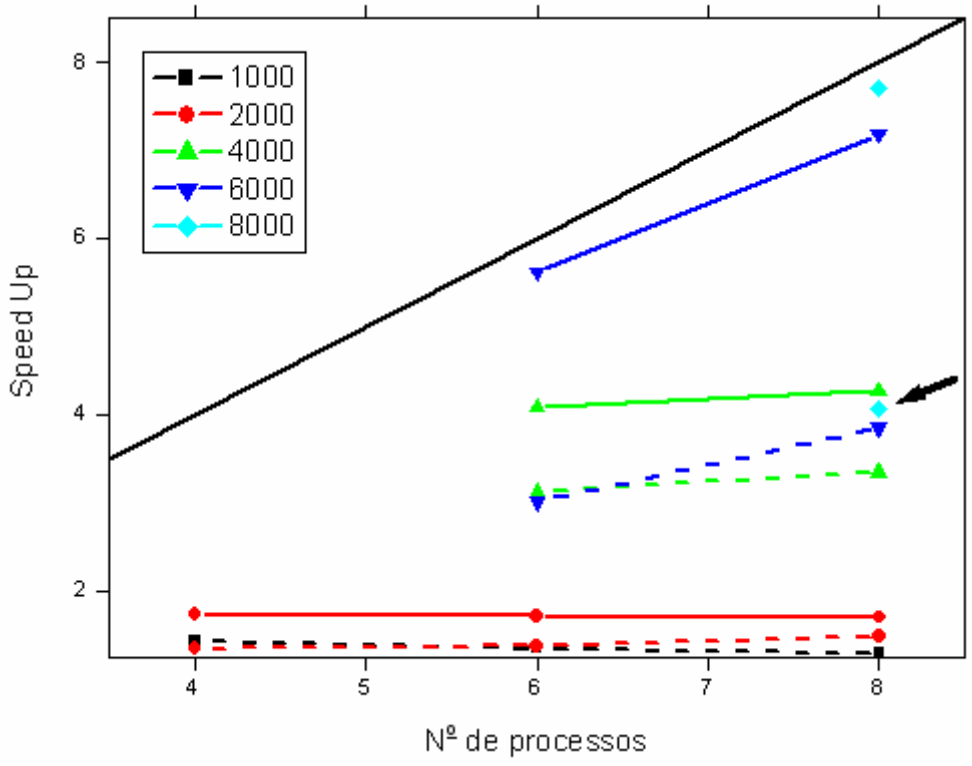

Gráfico 6.8 - *Speedup* para a aplicação executada sem concorrência em 64bits, ponto destacado *Speedup* para MPICH com matriz de tamanho 8000

A Tabela 6.3 e o Gráfico 6.8, afirmam que o melhor *Speedup* foi alcançado com o MPICH-G2 em 64 bits com a matriz distribuída em 8 processos, visto que ela tem a melhor evolução do *Speedup* e para o tamanho de matriz 8000 o *Speedup* está muito próximo do ideal, pois para 8 processos o *Speedup* ideal seria 8 e o *Speedup* calculado ficou em 7,71, contra um *Speedup* de apenas 4,06 da mesma situação com o MPICH.

O *Speedup* do MPICH-G2 com a matriz distribuída em 6 processos nos nós de 64 bits também é muito bom ficando em 5,62, contra 3,01 do MPICH, no entanto, o tamanho máximo de matriz que foi possível calcular distribuindo em 6 processo foi de 8000.

Para matrizes de tamanhos 1000 e 2000, tanto para o MPICH-G2 quanto para o MPICH, não existe ganho do *Speedup* conforme o número de processos aumenta mostrando assim, que este tipo de aplicação não é ideal para ser executado com sistemas pequenos.

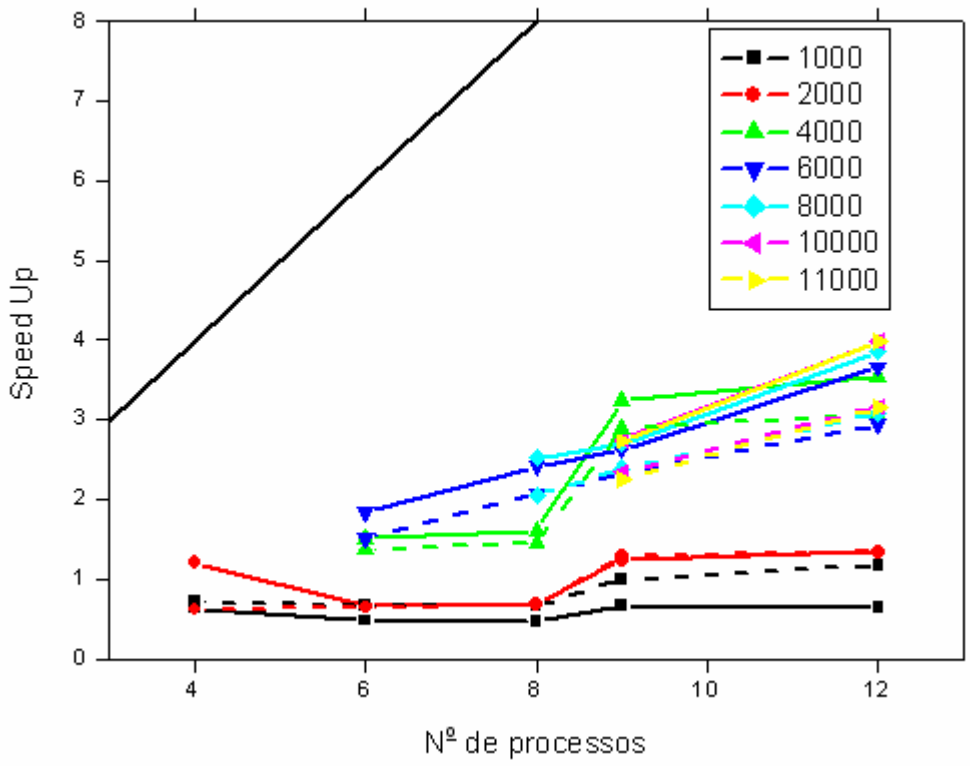

Gráfico 6.9 - *Speedup* para a aplicação executada sem concorrência em 32bits

Com o auxílio do Gráfico 6.9, é verificado que os *Speedup* para os nós de 32 bits não são tão bons quanto os para os de 64 bits, mesmo para quando os processos foram executados apenas nos nós de 32 bits. Isto está relacionado com a comunicação mais lenta dos nós de 32 bits. De qualquer forma, também pode-se afirmar que os melhores *Speedups* estão com o MPICH-G2, mostrando que ele utiliza a comunicação de forma mais eficiente que o MPICH, mas nem tanto assim, pois o ganho para estes casos é de em torno de 20%.

O MPICH-G2 alcançou melhores resultados que o MPICH, o que já foi apresentado pelos gráficos do *Speedup* e Razão, devido a biblioteca de comunicação do MPICH-G2 (**globus_io**) conseguir utilizar de maneira mais eficiente a comunicação entre os nós de 64bits, devido ao melhor aproveitamento do *throughput* das placas de rede e à questão relacionada com o *buffer* intermediário que o MPICH utiliza para as operações de *send* e *receive*. Desta forma pode-se concluir, que mesmo para um ambiente de *cluster*, ou seja, um

ambiente homogêneo com rede e recursos controlados e a disposição em 100% do tempo, o MPICH-G2 mostrou ter resultados melhores que o MPICH, e ainda, pode-se afirmar que o desempenho do MPICH-G2 melhora em quase 2 vezes em relação ao MPICH quando utilizados em máquinas de 64 bits para o tipo de aplicação testada com matrizes maiores.

6.2 TESTES REALIZADOS COM CONCORRÊNCIA

Os testes realizados com a condição de concorrência também foram executados nas arquiteturas de 32 e 64 bits separadamente, mas apenas para as matrizes sendo distribuídas em 8 e 9 processos.

Para os testes com 8 processos, foram disparados 4 processos em cada um dos nós CORTO1 e CORTO2 para 64 bits e CORTO5 e CORTO7 para 32 bits. Sendo assim, cada núcleo de processamento ficou responsável pelo cálculo de 2 processos. Com o comando do sistema **top** foi verificado que cada processo ocupou 50% de cada núcleo de processamento. Já para os testes com 9 processos, foram disparados 3 processos em cada um dos nós CORTO1, CORTO2 e CORTO3 para 64 bits e CORTO5, CORTO7 e CORTO8 para 32 bits, sendo assim 1 núcleo de processamento ficou responsável pelo cálculo de 1 processo e o outro dividiu o processamento entre 2 processos. Com o comando do sistema **top** foi verificado que um núcleo de processamento ocupou 100% de sua capacidade para 1 processo e o outro ocupou 50% para cada processo. A Tabela 6.4 apresenta os resultados obtidos nestas condições.

Tabela 6.4 - Tomada de tempo (s) entre MPICH-G2 e MPICH distribuídos em 8 e 9 processos e com concorrência entre os processos

Proces-sos	Arquite-tura		MAXN						
			1000	2000	4000	6000	8000	10000	11000
8	32 bits	MPICH-G2	1193,25	2593,54	7140,00	8241,00	12999,00	19232,00	23240,00
		MPICH	37,12	260,46	1921,27	2521,44	5880,00	11767,00	15587,00
	64 bits	MPICH-G2	1141,40	2229,99	5277,00	6789,00	9814,00	12898,00	14972,00
		MPICH	18,06	118,23	837,90	1256,20	2285,16	5632,00	7482,00
9	32 bits	MPICH-G2	762,73	1669,83	5164,80	6007,20	9540,40	14230,33	17170,00
		MPICH	34,94	145,15	1790,52	2016,12	4444,00	8345,25	11001,33
	64 bits	MPICH-G2	717,90	1552,66	3705,18	4659,20	7084,00	9590,80	10952,00
		MPICH	16,73	108,28	783,68	1035,28	2382,52	4408,00	5726,20

Com o auxílio da Tabela 6.4 , observa-se que o desempenho do MPICH é muito superior ao do MPICH-G2, e com isso pode-se afirmar que para o tipo de aplicação testada, quando submetido a condições de concorrência, o MPICH-G2 não tem bom desempenho e, portanto sua utilização não é recomendada.

O Gráfico 6.10, compara o desempenho do MPICH-G2 com o MPICH, e nota-se que para os casos onde a matriz foi distribuída em 8 e 9 processos para a arquitetura de 64 bits, e 9 processos para 32 bits, o MPICH obteve melhores resultados.

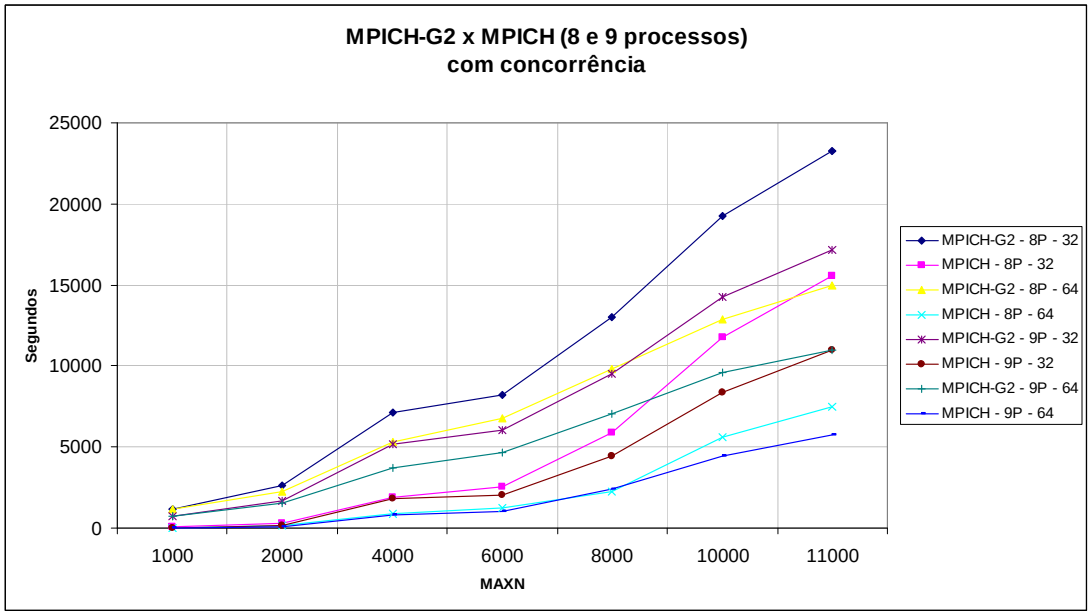

Gráfico 6.10 - Comparação do desempenho do MPICH-G2 com o MPICH para as arquiteturas de 64 e 32 bits e para a matriz distribuída em 8 e 9 processos com a condição de concorrência

Este comportamento pode ser explicado, pelos gráficos nos quais foram coletados os totais de bytes por segundo recebidos e enviados entre os nós que participaram do processamento. Para uma melhor compreensão desses gráficos foram retirados os totais de bytes enviados entre os nós, pois apresentam o mesmo comportamento do número de bytes recebidos. Para o Gráfico 6.11, a escala do eixo Y (Bytes) foi limitada em 5MB, e para o Gráfico 6.12, a escala do eixo Y foi limitada em 2MB. Os gráficos 6.13 e 6.14 são os mesmos, no entanto sem alterações nas escalas para a visualização dos momentos dos picos.

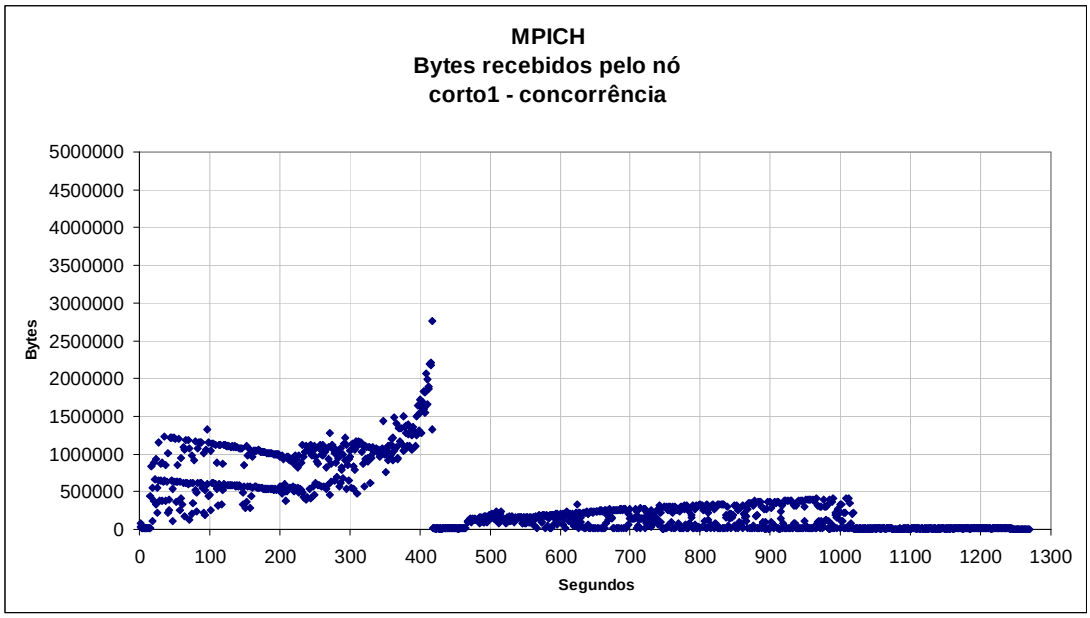

Gráfico 6.11 - Bytes recebidos pelo nó corto1 para uma distribuição de matriz em 8 processos de tamanho 6000 x 6000 com a condição de concorrência para o MPICH

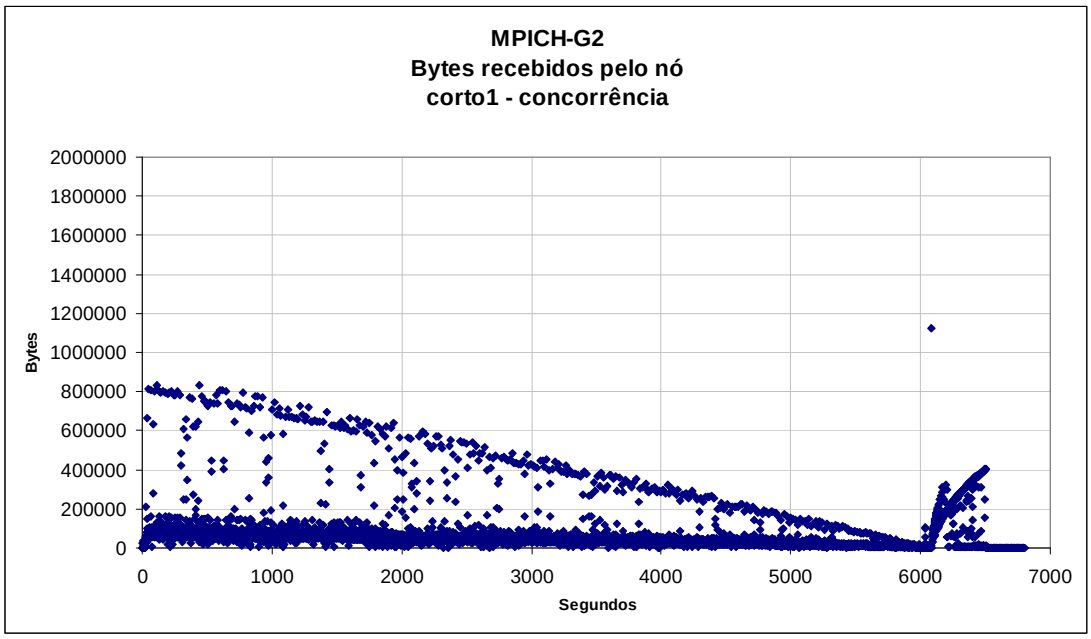

Gráfico 6.12 - Bytes recebidos pelo nó corto1 para uma distribuição de matriz em 8 processos de tamanho 6000 x 6000 com a condição de concorrência para o MPICH-G2

O MPICH, mesmo na condição de concorrência, tem o mesmo comportamento de quando executado sem a condição de concorrência, ou seja, na tridiagonalização da matriz ocorre uma maior comunicação entre os nós até o primeiro pico. Já o *gap*, é o momento sem grande comunicação onde são calculados os autovalores. Do segundo pico em diante (ver Gráfico 6.13), os autovetores são calculados. Já para o MPICH-G2, o comportamento da comunicação é muito diferente. Na tridiagonalização da matriz o tempo de comunicação entre os nós é muito maior, mas, no entanto, o total de bytes por segundos recebidos entre eles é menor e diminui durante este processo. Para este tipo de comportamento é provável que os processos fiquem "chaveando" entre si, consumindo a maior parte do tempo com os processos que ficam em espera. Devido à escala do Gráfico 6.14 a visualização do *gap* ficou dificultada mas, com o auxílio do programa onde os gráficos foram plotados, foi observado que ele ocorre entre 5900 e 6000 segundos. Na fase do cálculo dos autovetores o comportamento foi parecido. A comunicação em ambos os casos ficou em torno de 400B/s e o tempo de

finalização após o *gap* do MPICH-G2 foi de 6300 a 6800 segundos e o do MPICH foi de 420 até 1270 segundos.

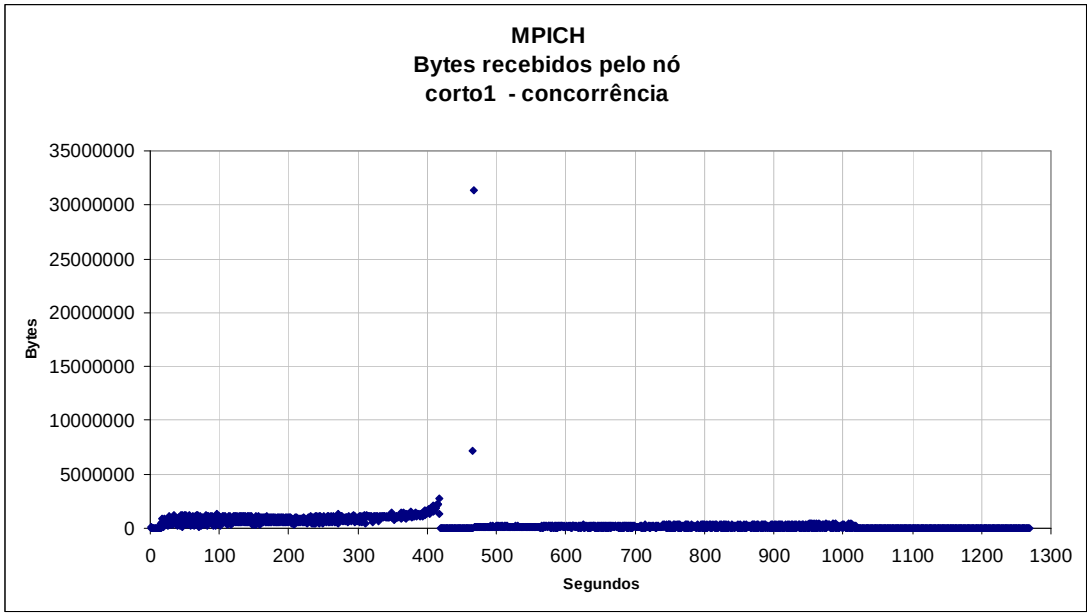

Gráfico 6.13 - Bytes recebidos pelo nó corto1 para uma distribuição de matriz em 8 processos de tamanho 6000 x 6000 com a condição de concorrência para o MPICH mantendo a escala original

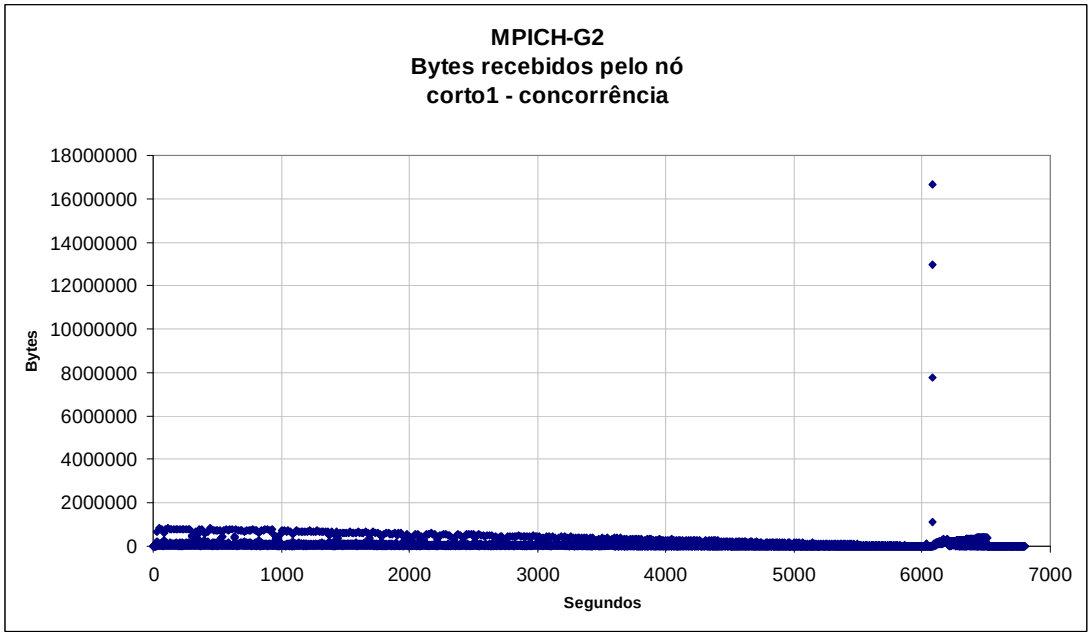

Gráfico 6.14 - Bytes recebidos pelo nó corto1 para uma distribuição de matriz em 8 processos de tamanho 6000 x 6000 com a condição de concorrência para o MPICH-G2 mantendo a escala original

O MPICH mantém o comportamento semelhante na execução da aplicação nas condições sem concorrência e com concorrência. Como esperado, o desempenho da aplicação no ambiente sem concorrência é melhor. Isto pode ser observado pela Tabela 6.5, que apresenta os tempos que o MPICH leva para calcular a matriz distribuída em 8 processos, com e sem concorrência para os ambientes de 64 e 32 bits. Nesta mesma tabela é apresentada a razão entre a execução da situação com concorrência e sem concorrência. O qual pode ser visto na linha Razão da Tabela 6.5 a seguir.

Tabela 6.5 - Comparação do desempenho do MPICH nas condições sem e com concorrência para uma matriz distribuída em 8 processos nas arquiteturas de 64 e 32 bits

		MAXN						
		1000	2000	4000	6000	8000	10000	11000
64 bits	MPICH - s/ conc	16,54	95,54	684,80	682,34	1541,62	2926,90	3864,33
	MPICH - c/ conc	18,06	118,23	837,90	1256,20	2285,16	5632,00	7482,00
	Razão	1,09	1,24	1,22	1,84	1,48	1,92	1,94
32 bits	MPICH - s/ conc	31,68	209,38	1573,07	1270,86	3033,36	5814,67	7882,67
	MPICH - c/ conc	37,12	260,46	1921,27	2521,44	5880,00	11767,00	15587,00
	Razão	1,17	1,24	1,22	1,98	1,94	2,02	1,98
64 bits	MPICH-G2 - s/ conc	22,66	83,12	537,36	365,26	811,73	1534,52	2016,21
	MPICH-G2 - c/ conc	1141,40	2229,99	5277,00	6789,00	9814,00	12898,00	14972,00
	Razão	50,37	26,83	9,82	18,59	12,09	8,41	7,43
32 bits	MPICH-G2 - s/ conc	46,22	208,93	1428,96	1091,91	2488,82	4718,33	5619,67
	MPICH-G2 - c/ conc	1193,25	2593,54	7140,00	8241,00	12999,00	19232,00	23240,00
	Razão	25,81	12,41	5,00	7,55	5,22	4,08	4,14

Antes de iniciar a análise seguinte, devemos levar em consideração que a aplicação quando executada com 1 processo em 1 núcleo de processamento e leva X segundos, quando executada com 2 processos em 1 núcleo de processamento deve levar um tempo pelo menos 2 vezes maior, porque o núcleo de processamento deve compartilhar os recursos entre os 2 processos. Nota-se que, para o MPICH conforme o tamanho da matriz aumenta esta relação de 2 vezes aparece para as matrizes de tamanho 10000 e 11000 em 64 bits e a partir de 6000 para 32 bits, nos casos das matrizes menores onde esta relação não acontece. Isto, provavelmente está relacionado com o fato de matrizes menores sofrerem um impacto maior

pela latência do sistema, dependendo menos da capacidade de processamento de cada núcleo de processamento. Já para o MPICH-G2 apesar do tempo de execução total ser muito maior, nota-se que para tamanhos maiores de matrizes, o desempenho melhora em relação à condição de execução sem concorrência, como pode ser observado na Tabela 6.5.

Para a condição de concorrência pode-se afirmar que o melhor desempenho do MPICH em relação ao MPICH-G2 deve-se ao fato de que na fase de triadiagonalização da matriz o MPICH-G2 leva um tempo maior que o MPICH.

Em relação ao péssimo desempenho do MPICH-G2 nesta situação, pode-se afirmar que a biblioteca de comunicação (**globus_io**), não apresenta uma maneira eficiente de realizar a comunicação entre os processos que concorrem pelo mesmo núcleo de processamento, e que o MPICH-G2 terá melhor desempenho para o tipo de aplicação testada, quando os processos a serem distribuídos forem submetidos em apenas um núcleo de processamento.

CAPÍTULO 7

Conclusões

A utilização da *grid computing* é uma maneira de diminuir os gastos que instituições acadêmicas, e até mesmo empresas privadas tem com seus parques computacionais, uma vez que os recursos utilizados para a computação não precisam estar alocados fisicamente em suas dependências. Mesmo assim, deve-se tomar cuidado com os dados que serão computados, levando em consideração a segurança e a disponibilidade dos recursos.

A *grid computing*, provavelmente, não irá substituir os grandes computadores paralelos, mas vem junto com eles contribuir como mais uma opção de processamento distribuído.

Neste trabalho, foi comparado o desempenho das bibliotecas para programação paralela MPICH-G2 e o MPICH, utilizando a biblioteca de álgebra linear ScaLAPACK, e uma de suas rotinas de diagonalização de matrizes e determinação de autovetores e autovalores (PCHEEVX). Esta rotina é utilizada pelo Laboratório de Física Computacional (LFC) do IFSC na sua aplicação, que realiza cálculos de estruturas semicondutoras.

Nos testes realizados, foram observados dois comportamentos distintos: o primeiro, no qual o desempenho do MPICH-G2 é melhor em relação ao do MPICH, e o segundo, esta situação se inverte. Para esses comportamentos, foi observado que o MPICH-G2 alcançava melhores resultados quando não havia concorrência entre os processos. Já o MPICH alcançava melhores resultados quando ocorria concorrência entre os processos.

O MPICH-G2 tem melhor desempenho que o MPICH quando não submetido à situação de concorrência entre os processos porque seu *device* de comunicação **globus2**, aproveita de forma mais eficiente à comunicação TCP/IP do que o *device* de comunicação **ch_p4** que é utilizado pelo MPICH.

Para o tipo de aplicação testada no LFC, os melhores desempenhos são alcançados conforme o tamanho da matriz aumenta, e o melhor desempenho foi alcançado pelo MPICH-G2 quando a matriz foi distribuída em 8 processos nos nós de 64 bits.

Com isso, pode-se concluir que apesar do MPICH-G2 não ser exclusivamente construído para um ambiente de *cluster*, ele apresenta bons resultados para o tipo de aplicação utilizada pelo LFC, e que em computadores com arquitetura de 64 bits ele tem o desempenho quase 2 vezes melhor que o MPICH.

7.1 TRABALHOS FUTUROS

Os próximos passos deste trabalho serão:

Aumentar o número de nós de processamento disponíveis podendo assim distribuir as matrizes em mais processos, como por exemplo, 16 e 25 processos.

Realizar os mesmos testes com as outras máquinas do LFC que não estão conectadas diretamente ao *switch* do *cluster* LFC, assim será possível determinar o desempenho do MPICH-G2 e do MPICH quando não estão dentro de um ambiente com a rede controlada e com os recursos de processamento disponíveis em 100% do tempo, pois são computadores utilizados pelos alunos do laboratório.

Além disto, estender os testes realizados utilizando computadores distribuídos pela internet. Com isso, conseguiremos avaliar o desempenho do MPICH-G2 em um ambiente para o qual ele foi realmente desenvolvido, ou seja, redes sujeitas a intermitências causadas por qualquer motivo, como indisponibilidade de recursos e capacidade de processamento dos recursos.

Referências

[1] FOSTER, I.; KESSELMAN, C. **The Grid 2: blueprint for a new computing infrastructure**. 2003.

[2] FOSTER, I.; KESSELMAN, C. **The Grid 2: Blueprint for a New Computing Infrastructure**. 2003.

[3] BARKER, M.; BUYYA, R.; LAFORENZA, D. **The Grid: international efforts in global computing**. 2006

[4] KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Software-Practice & Experience**, v. 32, n. 2, p. 135-164, 2002..

[5] SOARES, L. F. G.; LEMOS, G.; COLCHER, S. (eds.). **Redes de computadores:** das LANs, MANs e WANs às redes ATM. 1995.

[6] TAURION, C. **Grid computing: um novo paradigma computacional**. Rio de Janeiro: Brasport, 2004.

[7] FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. **International Journal of High Performance Computing Applications**, v. 15, n. 3, p. 200-222, 2001.

[8] MURHAMMER, M.W. **TCP/IP Tutorial e Técnico**. 2000.

[9] GridCafé. How does it work? - Grid architecture. Disponível em:< http://gridcafe.web.cern.ch/gridcafe/gridatwork/architecture.html>. Acesso em: 17-7-2007.

[10] The Globus Alliance. Disponível em:< http://www.globus.org>. Acesso em: 10-12-2007.

[11] O cientista da "grelha". Disponível em: <http://janelanaweb.com/digitais/defanti.html>. Acesso em: 6-8-2007.

[12] FOSTER, I. et al. **Multimethod Communication for High-Performance Metacomputing Applications**. 1996. p. 41.

[13] About the Globus Toolkit. Disponível em:< http://www.globus.org/toolkit/about.html> Acesso em: 10-12-2007.

[14] **Projects with Globus Alliance Participants**. Disponível em:< http://www.globus.org/alliance/projects.php>. Acesso em: 17-7-2007.

[15] Overview of the Grid Security Infrastrucuture. Disponível em: < http://www.globus.org/security/overview.html>. Acesso em: 1-12-2007.

[16] PITANGA, Marcos. **Computação em cluster - O Estado da Arte em Computação**. Rio de Janeiro: Editora Brasport. 2003.

[17] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. Disponível em: < http://www.mpi-forum.org/docs/mpi-11-html/mpi-report.html>. Acesso em: 21-1-2008.

[18] Message Passing Interface Forum. MPI-2: Extensions to the Message-Passing Interface. Disponível em: <http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.htm>. Acesso em: 21-1-2008.

[19] MPICH Home Page. Disponível em:<http://www-unix.mcs.anl.gov/mpi/mpich1/ >. Acesso em: 16-1-2008.

[20] Saphir W. The ch_p4 device. Disponível em: <http://www.netlib.org/utk/mpi-review/node15.html >. Acesso em: 9-1-2008.

[21] MPICH-G2.Disponível em:<http://www3.niu.edu/mpi/>. Acesso em: 16-1-2008.

[22] KARONIS, N. T.; TOONEN, B.; FOSTER, I. MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. **Journal of Parallel and Distributed Computing**, v. 63, n. 5, p. 551-563, 2003.

[23] FERGUSON, J. W.; TOWNS, J. The alliance grid. **Advances in Engineering Software**, v. 32, n. 5, p. 417-422, 2001.

[24] JUAN D.; YIJUN Z.; SHOUZHENG Z. **Parallelization of the FDTD Algorithm over MPICH-G2**. Disponível em: <http://ieeexplore.ieee.org/iel5/10688/33747/01606758.pdf?arnumber=1606758>. Acesso em 18/01/2008.

[25] BLACKFORD, L. S.; DONGARRA, J.; A.PETITET. **ScaLAPACK Users' Guide**. 1997.

[26] DONGARRA, J.; GEIJN R.V.D. Two dimensional basic linear algebra communication subprograms. ComputerScience Dept. Technical Report, University of Tennessee, LAPACK Working Note No.37, p. 91-138, 1991.

[27] MPI Dubugging and Profiling. Disponível em:< http://www.pgroup.com/resources/mpich/mpich126_pgi60.htm>. Acesso em: 10-10-2007.

[28] Jones, R. Disponível em:<http://www.netperf.org/netperf/> . Acesso em:15-1-2008.

[29] **MPICH-G2**. http://www3.niu.edu/mpi/ . Acesso em: 15-12-2005. 16-1-2008.

[30] TOONEN, B. **MPICH-G2 x MPICH - performance behavior**. (mensagem eletrônica). Envio em: 11-2-2008.

[31] CULLER, D. E.(ed.). **Parallel Computer Architeture: A Hardware/Software Approach**. 1998.

Apêndice A - Lista de rotinas do ScaLAPACK

Disponível em http://www.mathkeisan.com/UsersGuide/E/scalapack.html

Acessado em 28/12/2007

ScaLAPACK Routine List

Simple Driver and Divide and Conquer Driver Routines

† `?` indicates prefix which must be filled with a combination of:
`S = REAL(kind=4), D = REAL(kind=8), C = COMPLEX(kind=4), Z = COMPLEX(kind=8)`

Name†	Prefixes	Description
`P?DBSV`	`S D C Z`	Solves a general banded system of linear equations *AX=B* with no pivoting.
`P?DTSV`	`S D C Z`	Solves a general tridiagonal system of linear equations *AX=B* with no pivoting.
`P?GBSV`	`S D C Z`	Solves a general banded system of linear equations *AX=B*.
`P?GELS`	`S D C Z`	Solves over-determined or under-determined linear systems involving a matrix of full rank.
`P?GESV`	`S D C Z`	Solves a general system of linear equations *AX=B*.
`P?GESVD`	`S D`	Computes the singular value decomposition of a general matrix, optionally computing the left and/or right singular vectors.
`P?PBSV`	`S D C Z`	Solves a symmetric/Hermitian positive definite banded system of linear equations *AX=B*.
`P?POSV`	`S D C Z`	Solves a symmetric/Hermitian positive definite system of linear equations *AX=B*.
`P?PTSV`	`S D C Z`	Solves a symmetric/Hermitian positive definite tridiagonal system of linear equations *AX=B*.
`P?SYEV`	`S D`	Computes selected eigenvalues and eigenvectors of a symmetric matrix.
`P?SYEVD`	`S D`	Computes all eigenvalues, and optionally, eigenvectors of a real symmetric matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.
`P?HEEV`	`C Z`	Computes all eigenvalues and, optionally, eigenvectors of a Hermitian matrix.
`P?HEEVD`	`C Z`	Computes all eigenvalues and, optionally, eigenvectors of a Hermitian matrix. If eigenvectors are desired, it uses a divide and conquer algorithm.

Expert Driver Routines

† ? indicates prefix which must be filled with a combination of:
`S = REAL(kind=4), D = REAL(kind=8), C = COMPLEX(kind=4), Z = COMPLEX(kind=8)`

Name	Prefixes†	Description
P?GESVX	S D C Z	Solves a general system of linear equations $AX=B$.
P?POSVX	S D C Z	Solves a symmetric/Hermitian positive definite system of linear equations $AX=B$.
P?SYEVX	S D	Computes selected eigenvalues and eigenvectors of a symmetric matrix.
P?SYGVX	S D	Computes selected eigenvalues and eigenvectors of a real generalized symmetric-definite eigenproblem.
P?HEEVX	C Z	Computes selected eigenvalues and eigenvectors of a Hermitian matrix.
P?HEGVX	C Z	Computes selected eigenvalues and eigenvectors of a generalized Hermitian-definite eigenproblem.

Computational Routines

† ? indicates prefix which must be filled with a combination of:
`S = REAL(kind=4), D = REAL(kind=8), C = COMPLEX(kind=4), Z = COMPLEX(kind=8)`

Name	Prefixes†	Description
P?DBTRF	S D C Z	Computes an *LU* factorization of a general band matrix with no pivoting.
P?DBTRS	S D C Z	Solves a general banded system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$, using the *LU* factorization computed by P?DBTRF.
P?DBTRSV	S D C Z	Solves a banded triangular system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$, using the *LU* factorization computed by P?DBTRF.
P?DTTRF	S D C Z	Computes an *LU* factorization of a general tridiagonal matrix with no pivoting.
P?DTTRS	S D C Z	Solves a general tridiagonal system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$, using the *LU* factorization computed by P?DTTRF.
P?DTTRSV	S D C Z	Solves a tridiagonal triangular system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$, using the *LU* factorization computed by P?DTTRF.
P?GBTRF	S D C Z	Computes an *LU* factorization of a general band matrix, using partial pivoting with row interchanges.
P?GBTRS	S D C Z	Solves a general banded system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$, using the *LU* factorization computed by P?GBTRF.
P?GEBRD	S D C Z	Reduces a general rectangular matrix to real bidiagonal form by an orthogonal/unitary transformation.
P?GECON	S D C Z	Estimates the reciprocal of the condition number of a general matrix.
P?GEEQU	S D C Z	Computes row and column scalings to equilibrate a general rectangular matrix and reduce its condition number.
P?GEHRD	S D C Z	Reduces a general matrix to upper Hessenberg form by an orthogonal/unitary similarity transformation.
P?GELQF	S D C Z	Computes an *LQ* factorization of a general rectangular matrix.

P?GEQLF	S D C Z	Computes a *QL* factorization of a general rectangular matrix.
P?GEQPF	S D C Z	Computes a *QR* factorization with column pivoting of a general rectangular matrix.
P?GEQRF	S D C Z	Computes a *QR* factorization of a general rectangular matrix.
P?GERFS	S D C Z	Improves the computed solution to a system of linear equations and provides error bounds and backward error estimates for the solutions.
P?GERQF	S D C Z	Computes an *RQ* factorization of a general rectangular matrix.
P?GETRF	S D C Z	Computes an *LU* factorization of a general matrix, using partial pivoting with row interchanges.
P?GETRI	S D C Z	Computes the inverse of a general matrix, using the *LU* factorization computed by P?GETRF.
P?GETRS	S D C Z	Solves a general system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$, using the *LU* factorization computed by P?GETRF.
P?GGQRF	S D C Z	Computes a generalized *QR* factorization.
P?GGRQF	S D C Z	Computes a generalized *RQ* factorization.
P?LAHQR	S D	Computes the Schur decomposition and/or eigenvalues of a matrix already in Hessenberg form.
P?ORGLQ	S D	Generates all or part of the orthogonal matrix *Q* from an *LQ* factorization determined by PSGELQF.
P?ORGQL	S D	Generates all or part of the orthogonal matrix *Q* from a *QL* factorization determined by PSGEQLF.
P?ORGQR	S D	Generates all or part of the orthogonal matrix *Q* from a *QR* factorization determined by PSGEQRF.
P?ORGRQ	S D	Generates all or part of the orthogonal matrix *Q* from an *RQ* factorization determined by PSGERQF.
P?ORMBR	S D	Multiplies a general matrix by one of the orthogonal transformation matrices from a reduction to bidiagonal form determined by PSGEBRD.
P?ORMHR	S D	Multiplies a general matrix by the orthogonal transformation matrix from a reduction to Hessenberg form determined by PSGEHRD.
P?ORMLQ	S D	Multiplies a general matrix by the orthogonal matrix from an *LQ* factorization determined by PSGELQF.
P?ORMQL	S D	Multiplies a general matrix by the orthogonal matrix from a *QL* factorization determined by PSGEQLF.
P?ORMQR	S D	Multiplies a general matrix by the orthogonal matrix from a *QR* factorization determined by PSGEQRF.
P?ORMRQ	S D	Multiplies a general matrix by the orthogonal matrix from an *RQ* factorization determined by PSGERQF.
P?ORMRZ	S D	Multiplies a general matrix by the orthogonal transformation matrix from a reduction to upper triangular form determined by PSTZRZF.
P?ORMTR	S D	Multiplies a general matrix by the orthogonal transformation matrix from a reduction to tridiagonal form determined by PSSYTRD.
P?PBTRF	S D C Z	Computes the Cholesky factorization of a symmetric/Hermitian positive definite banded matrix.

P?PBTRS	S D C Z	Solves a symmetric/Hermitian positive definite banded system of linear equations *AX=B*, using the Cholesky factorization computed by P?PBTRF.
P?PBTRSV	S D C Z	Solves a banded triangular system of linear equations *AX=B*, using the Cholesky factorization computed by P?PBTRF.
P?POCON	S D C Z	Estimates the reciprocal of the condition number of a symmetric/Hermitian positive definite distributed matrix.
P?POEQU	S D C Z	Computes row and column scalings to equilibrate a symmetric/Hermitian positive definite matrix and reduce its condition number.
P?PORFS	S D C Z	Improves the computed solution to a symmetric/Hermitian positive definite system of linear equations *AX=B*, and provides forward and backward error bounds for the solution.
P?POTRF	S D C Z	Computes the Cholesky factorization of a symmetric/Hermitian positive definite matrix.
P?POTRI	S D C Z	Computes the inverse of a symmetric/Hermitian positive definite matrix, using the Cholesky factorization computed by P?POTRF.
P?POTRS	S D C Z	Solves a symmetric/Hermitian positive definite system of linear equations *AX=B*, using the Cholesky factorization computed by P?POTRF.
P?PTTRF	S D C Z	Computes the Cholesky factorization of a symmetric/Hermitian positive definite tridiagonal matrix.
P?PTTRS	S D C Z	Solves a symmetric/Hermitian positive definite tridiagonal system of linear equations *AX=B*, using the Cholesky factorization computed by P?PTTRF.
P?PTTRSV	S D C Z	Solves a tridiagonal triangular system of linear equations *AX=B*, using the Cholesky factorization computed by P?PTTRF.
P?STEBZ	S D C Z	Computes the eigenvalues of a symmetric/Hermitian tridiagonal matrix by bisection.
P?STEDC	S D	Computes all eigenvalues and, optionally, eigenvectors of a symmetric tridiagonal matrix using the divide and conquer algorithm.
P?STEIN	S D C Z	Computes the eigenvectors of a symmetric/Hermitian tridiagonal matrix using inverse iteration.
P?SYGST	S D	Reduces a symmetric-definite generalized eigenproblem to standard form.
P?SYTRD	S D	Reduces a symmetric matrix to real symmetric tridiagonal form by an orthogonal similarity transformation.
P?TRCON	S D C Z	Estimates the reciprocal of the condition number of a triangular matrix.
P?TRRFS	S D C Z	Provides error bounds and backward error estimates for the solution to a system of linear equations with a triangular coefficient matrix.
P?TRTRI	S D C Z	Computes the inverse of a triangular matrix.
P?TRTRS	S D C Z	Solves a triangular system of linear equations $AX=B$, $A^TX=B$ or $A^HX=B$.
P?TZRZF	S D C Z	Reduces an upper trapezoidal matrix to upper triangular form by means of orthogonal transformations.

P?HEGST	C Z	Reduces a Hermitian-definite generalized eigenproblem to standard form.
P?HETRD	C Z	Reduces a Hermitian matrix to Hermitian tridiagonal form by a unitary similarity transformation.
P?UNGLQ	C Z	Generates all or part of the unitary matrix *Q* from an *LQ* factorization determined by PCGELQF.
P?UNGQL	C Z	Generates all or part of the unitary matrix *Q* from a *QL* factorization determined by PCGEQLF.
P?UNGQR	C Z	Generates all or part of the unitary matrix *Q* from a *QR* factorization determined by PCGEQRF.
P?UNGRQ	C Z	Generates all or part of the unitary matrix *Q* from an *RQ* factorization determined by PCGERQF.
P?UNMBR	C Z	Multiplies a general matrix by one of the unitary transformation matrices from a reduction to bidiagonal form determined by PCGEBRD.
P?UNMHR	C Z	Multiplies a general matrix by the unitary transformation matrix from a reduction to Hessenberg form determined by PCGEHRD.
P?UNMLQ	C Z	Multiplies a general matrix by the unitary matrix from an *LQ* factorization determined by PCGELQF.
P?UNMQL	C Z	Multiplies a general matrix by the unitary matrix from a *QL* factorization determined by PCGEQLF.
P?UNMQR	C Z	Multiplies a general matrix by the unitary matrix from a *QR* factorization determined by PCGEQRF.
P?UNMRQ	C Z	Multiplies a general matrix by the unitary matrix from an *RQ* factorization determined by PCGERQF.
P?UNMRZ	C Z	Multiplies a general matrix by the unitary transformation matrix from a reduction to upper triangular form determined by PCTZRZF.
P?UNMTR	C Z	Multiplies a general matrix by the unitary transformation matrix from a reduction to tridiagonal form determined by PCHETRD.

Apêndice B - Listagem do programa sample_pcheevx_call.f modificado para os teste do LFC

Observações:

As alterações realizadas no código foram:

1. Comentar algumas saídas que eram impressas nas tela para plotar gráficos no MatLAB;
2. A variável MAXN variou segundo os valores:1000, 2000, 4000, 6000, 8000, 10000 e 11000;
3. Os valores dos parâmetros LWORK, LIWORK e LRWORK foram modificados para 10000000, 70000 e 32000000;
4. A variável N foi igualada a MAXN;
5. A variável NB foi igualada a 8;
6. As variáveis NPCOL e NPROW, variaram de acordo com os números de processos que foram testados;
7. Foi inserida uma chamada MPI_INIT e MPI_FINALIZE;

```
*
*
      PROGRAM SAMPLE_PCHEEVX_CALL
*
*
*  -- ScaLAPACK routine (version 1.2) --
*     University of Tennessee, Knoxville, Oak Ridge National Laboratory,
*     and University of California, Berkeley.
*     May 10, 1996
*
*     This routine contains a sample call to PCHEEVX.
*     When compiled and run, it produces output which can be
*     pasted directly into matlab.
*
*
*     .. Parameters ..
      INTEGER            LWORK, MAXN, LIWORK
      REAL               ZERO
      PARAMETER          (LWORK = 10000000, MAXN=11000, LIWORK = 70000,
     $                   ZERO = 0.0E+0 )
      INTEGER            LDA
      REAL               MONE
      INTEGER            MAXPROCS
      PARAMETER          ( LDA = MAXN, MONE = -1.0E+0, MAXPROCS = 512 )
      INTEGER            LRWORK
      PARAMETER          ( LRWORK = 32000000 )
```

```
*     ..
*     .. Local Scalars ..
      INTEGER            CONTEXT, I, IAM, INFO, M, MYCOL, MYROW, N, NB,
     $                   NPCOL, NPROCS, NPROW, NZ
*     ..
*     .. Local Arrays ..
      INTEGER            DESCA( 50 ), DESCz( 50 ),
     $                   ICLUSTR( MAXPROCS*2 ), IFAIL( MAXN ),
     $                   IWORK( LIWORK )
      REAL               GAP( MAXPROCS ), RWORK( LRWORK ), W( MAXN )
      COMPLEX            A( LDA, LDA ), WORK( LWORK ), Z( LDA, LDA )
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLACS_EXIT, BLACS_GET, BLACS_GRIDEXIT,
     $                   BLACS_GRIDINFO, BLACS_GRIDINIT, BLACS_PINFO,
     $                   BLACS_SETUP, DESCINIT, PCHEEVX, PCLAMODHILB,
     $                   PCLAPRNT

      INTEGER IERR
*     ..
*     .. Executable Statements ..
*
*
      CALL MPI_INIT(IERR)
*     Set up the problem
*
      N = MAXN
      NB = 8
      NPROW = 3
      NPCOL = 3
*
*
*     Initialize the BLACS
*
      CALL BLACS_PINFO( IAM, NPROCS )
      IF( ( NPROCS.LT.1 ) ) THEN
         CALL BLACS_SETUP( IAM, NPROW*NPCOL )
      END IF
*
*
*     Initialize a singlE BLACS context
*
      CALL BLACS_GET( -1, 0, CONTEXT )
      CALL BLACS_GRIDINIT( CONTEXT, 'R', NPROW, NPCOL )
      CALL BLACS_GRIDINFO( CONTEXT, NPROW, NPCOL, MYROW, MYCOL )
*
*     Bail out if this process is not a part of this context.
*
      IF( MYROW.EQ.-1 )
     $   GO TO 20
*
*
*     These are basic array descriptors
*
      CALL DESCINIT( DESCA, N, N, NB, NB, 0, 0, CONTEXT, LDA, INFO )
      CALL DESCINIT( DESCZ, N, N, NB, NB, 0, 0, CONTEXT, LDA, INFO )
*
*     Build a matrix that you can create with
*     a one line matlab command:  hilb(n) + diag([1:-1/n:1/n])
*
      CALL PCLAMODHILB( N, A, 1, 1, DESCA, INFO )
```

```
*
*
*     Uncomment this line to see the matrix printed out.
*
*      CALL PCLAPRNT( N, N, A, 1, 1, DESCA, 0, 0, 'A', 6, WORK )
*
*
*     Ask PCHEEVX to compute the entire eigendecomposition
*
      CALL PCHEEVX( 'V', 'A', 'U', N, A, 1, 1, DESCA, ZERO, ZERO, 13,
     $              -13, MONE, M, NZ, W, MONE, Z, 1, 1, DESCZ, WORK,
     $              LWORK, RWORK, LRWORK, IWORK, LIWORK, IFAIL, ICLUSTR,
     $              GAP, INFO )
*
*
*     Print out the eigenvectors
*
*      CALL PCLAPRNT( N, N, Z, 1, 1, DESCZ, 0, 0, 'Z', 6, WORK )
*
*
*     Print out matlab code which will check the residual
*
      IF( MYROW.EQ.0 .AND. MYCOL.EQ.0 ) THEN
         PRINT *, ' N =', N
         PRINT *, ' A =  hilb(N) + toeplitz( [ 1 (1:(N-1))*i ] )'
*         DO 10 I = 1, N
*            PRINT *, ' W(', I, ')=', W( I ), ';'
*   10    CONTINUE
         PRINT *, ' backerror = A - Z * diag(W) * Z'' '
         PRINT *, ' resid = A * Z - Z * diag(W)'
         PRINT *, ' ortho = Z'' * Z - eye(N)'
         PRINT *, ' norm(backerror)'
         PRINT *, ' norm(resid)'
         PRINT *, ' norm(ortho)'
      END IF
*
      CALL BLACS_GRIDEXIT( CONTEXT )
*
   20 CONTINUE
*
      CALL BLACS_EXIT( 0 )
*
*
*     Uncomment this line on SUN systems to avoid the useless print out
*
*      CALL IEEE_FLAGS( 'clear', 'exception', 'underflow', '')
*
*
 9999 FORMAT( 'W=diag([', 4D16.12, ']);' )
*
*      CALL MPI_FINALIZE(IERR)
*      STOP
      END
*
      SUBROUTINE PCLAMODHILB( N, A, IA, JA, DESCA, INFO )
*
*  -- ScaLAPACK routine (version 1.2) --
*     University of Tennessee, Knoxville, Oak Ridge National Laboratory,
*     and University of California, Berkeley.
*     May 10, 1996
*
```

```
*
*
*
*     .. Parameters ..
      INTEGER            BLOCK_CYCLIC_2D, DLEN_, DT_, CTXT_, M_, N_,
     $                   MB_, NB_, RSRC_, CSRC_, LLD_
      PARAMETER          ( BLOCK_CYCLIC_2D = 1, DLEN_ = 9, DT_ = 1,
     $                   CTXT_ = 2, M_ = 3, N_ = 4, MB_ = 5, NB_ = 6,
     $                   RSRC_ = 7, CSRC_ = 8, LLD_ = 9 )
      REAL               ONE
      PARAMETER          ( ONE = 1.0E+0 )
*     ..
*     .. Scalar Arguments ..
      INTEGER            IA, INFO, JA, N
*     ..
*     .. Array Arguments ..
      INTEGER            DESCA( * )
      COMPLEX            A( * )
*     ..
*     .. Local Scalars ..
      INTEGER            I, J, MYCOL, MYROW, NPCOL, NPROW
*     ..
*     .. External Subroutines ..
      EXTERNAL           BLACS_GRIDINFO, PCELSET
*     ..
*     .. Intrinsic Functions ..
      INTRINSIC          CMPLX, REAL
*     ..
*     .. Executable Statements ..
*
*
*     The matlab code for a real matrix is:
*         hilb(n) + diag( [ 1:-1/n:1/n ] )
*     The matlab code for a complex matrix is:
*         hilb(N) + toeplitz( [ 1 (1:(N-1))*i ] )
*
*       This is just to keep ftnchek happy
      IF( BLOCK_CYCLIC_2D*CSRC_*CTXT_*DLEN_*DT_*LLD_*MB_*M_*NB_*N_*
     $    RSRC_.LT.0 )RETURN
*
      INFO = 0
*
      CALL BLACS_GRIDINFO( DESCA( CTXT_ ), NPROW, NPCOL, MYROW, MYCOL )
*
*
      IF( IA.NE.1 ) THEN
         INFO = -3
      ELSE IF( JA.NE.1 ) THEN
         INFO = -4
      END IF
*
      DO 20 J = 1, N
         DO 10 I = 1, N
            IF( I.EQ.J ) THEN
               CALL PCELSET( A, I, J, DESCA,
     $                       CMPLX( ONE / ( REAL( I+J )-ONE ) )+
     $                       CMPLX( ONE ) )
            ELSE
               CALL PCELSET( A, I, J, DESCA,
     $                       CMPLX( ONE / ( REAL( I+J )-ONE ),
     $                       REAL( J-I ) ) )
```

```
            END IF
   10    CONTINUE
   20 CONTINUE
*
*
      RETURN
*
*     End of PCLAMODHLIB

*
      END
```

Apêndice C - Instalação e configuração do *Globus Toolkit* realizada no cluster do LFC

1 ANTES DE INSTALAR

É necessário ter conhecimento de 5 componentes que serão configurados durante a instalação do Globus Toolkit, alguns desses componentes foram explicados na Capítulo 3 mas são relembrados a seguir:

1. *Simple Certificate Authority* ou simplesmente *SimpleCA*: tem a função de autoridade certificadora do *grid*, ou seja, é este serviço que autoriza usuários e *hosts* a fazer parte de um *grid*. Está certificação é baseado no esquema de chaves de criptografia pública e privada do protocolo SSL. A *SimpleCA* é a forma mais utilizada para criar uma autoridade certificadora, pois a criação de chaves de criptografia é efetuada pelo próprio usuário que está instalando o globus utilizando o *OpenSSL* e portanto não tem nenhum custo. A geração dos certificados, no entanto, também pode ser feita através de alguma empresa de certificação como por exemplo a *Certisign*;
2. *HostCertificate*: os recursos de um *grid* podem estar geograficamente distantes e os dados que serão computados entre esses recursos podem navegar por vários roteadores na internet antes de chegar ao seu destino. Devido a isso, é necessário que cada componente do *grid* envie seus dados de forma criptografada. Cada recurso de processamento que irá compor o *grid* deve requisitar um *HostCertificate* (certificado do *host*) à *SimpleCA*;
3. *UserCertificate*: de forma análoga ao *HostCertificate*, cada usuário que utilizará os recursos do *grid* deve requisitar um certificado de usuário à *SimpleCA* para poder fazer parte do *grid* e enviar processos a ele. Da mesma forma, esses dados serão enviados aos recursos do *grid* de forma criptografada;
4. **GridFTP**: responsável pelo transporte de arquivos no *grid*;

5. *Gatekeeper*: pode ser considerado uma interface entre o usuário e o GRAM, pois é responsável pela autenticação do usuário e sua associação com uma conta no computador local.

2 INSTALAÇÃO E CONFIGURAÇÃO DO GLOBUS TOOLKIT

O *Globus Toolkit* é o *midleware* responsável por habilitar um *grid*. Sua instalação é trabalhosa pois existem vários passos e várias configurações que devem ser seguidos para tê-lo completamente funcional.

O *Globus Toolkit* deve ser instalado em todas as máquinas que irão compor o *grid*. No *cluster* do LFC ele foi instalado em todos os 8 *hosts*.

Existem dois tipos de distribuição para a instalação do *Globus Toolkit*:

- **Binária**: são distribuições pré-compiladas para vários tipos de SO´s e arquiteturas. Sua vantagem é a rapidez na instalação. No entanto, para a posterior instalação do MPICH-G2 ela não preenche os requisitos de dependências das bibliotecas exigidas para a ligação (*flavor*) do MPICH com o globus;
- ***Source***: Ideal para instalações onde não existe a distribuição binária e para a configuração do MPICH com as bibliotecas de comunicação do globus. Em todas as máquinas do LFC o globus foi instalado através desta distribuição;

A última versão estável do *Globus Toolkit* é a 4.0.6, mas a versão instalada no *cluster* do LFC é a 4.0.4. O *Toolkit* pode ser obtido por *download* através da URL http://www.globus.org/toolkit/downloads/4.0.6/.

Neste apêndice são apresentadas a instalação e a configuração de forma resumida, apenas com os serviços que foram configurados no *cluster* do LFC e explicando os pontos principais. A instalação completa com todos os passos e todas as configurações está descrita

no site do *Globus Toolkit* na URL http://www.globus.org/toolkit/docs/4.0/admin/docbook/quickstart.html . A seguir, é apresentado um passa a passo da instalação e configuração dos componentes mais importantes que foram configurados no cluster do LFC. Este passo a passo está baseado na distribuição *source* do *Globus Toolkit* utilizando o *shell bash* em um SO Debian 4r0.

3 INSTALANDO

Os primeiros passos são: criar um usuário que será responsável pelo processo de instalação e configuração do globus, criar um diretório onde o globus será instalado e setar as permissões de leitura, escrita e execução para o usuário criado. No Quadro 1 segue a seqüência de comandos necessários.

```
root@corto1:~$ adduser globus
root@corto1:~$ mkdir /usr/local/globus-4.0.4/
root@corto1:~$ chown globus:globus /usr/local/globus-4.0.4/
```

Quadro 1 - Criando usuário e área para instalação

Em seguida, com o usuário globus que foi criado deve-se descompactar os arquivos de instalação e compilar a distribuição, como mostra a seqüência de comandos do Quadro 2. O tempo total do processo de compilação é demorado. No *host* **corto1** que é uma máquina de 64 bits leva em torno de 1,5 hora.

```
globus@corto1:~$ tar xzf gt4.0.4-all-source- installer.tar.gz
globus@corto1:~$ cd gt4.0.4-all-source-installer
globus@corto1:~$ ./configure --prefix=/usr/local/globus-4.0.4
globus@corto1:~$ make
globus@corto1:~$ make install
```

Quadro 2 - Compilação do *Globus Toolkit*

Após o término da instalação devem ser configurados a *SimpleCA*, o *HostCertificate*, o *UserCertificate*, o **GridFTP** e o *Gatekeeper*.

3.1 CONFIGURANDO UMA ENTIDADE CERTIFICADORA (SimpleCA)

Como usuário globus, deve-se setar a variável de ambiente GLOBUS_LOCATION, apontando esta para o diretório onde o globus foi instalado e em seguida carregar o arquivo **globus-user-env.sh** com o comando **source**. Este arquivo carregará todas as variáveis de ambiente necessárias para a execução dos comandos do globus. Esta seqüência de comandos está descrita no Quadro 3.

```
globus@corto1:~$ export GLOBUS_LOCATION=/usr/local/globus-4.0.4
globus@corto1:~$ source $GLOBUS_LOCATION/etc/globus-user-env.sh
```

Quadro 3 - Setar variáveis de ambiente para o usuário globus

Com o comando **setup-simple-ca** serão criadas as chaves públicas e privadas que serão utilizadas para certificar *hosts* e usuários que farão parte do *grid*. Este comando solicita um nome para a CA, o número de dias em que este certificado será válido (o *default* são 1825 dias, que correspondem a 5 anos) e uma senha que será utilizada, pelo administrador da CA, para a autenticação dos certificados. No Quadro 4 são apresentados os comandos e o conteúdo do arquivo criado no diretório do usuário globus que representa o certificado desta CA.

```
globus@corto1:~$ $GLOBUS_LOCATION/setup/globus/setup-simple-ca
WARNING: GPT_LOCATION not set, assuming:
         GPT_LOCATION=/usr/local/globus-4.0.4
    C e r t i f i c a t e   A u t h o r i t y   S e t u p
This script will setup a Certificate Authority for signing Globus
users certificates.  It will also generate a simple CA package that
can be distributed to the users of the CA.
The CA information about the certificates it distributes will
be kept in:
/home/globus/.globus/simpleCA/
/usr/local/globus-4.0.4/setup/globus/setup-simple-ca: line 250:
test: res: integer expression expected
The unique subject name for this CA is:

cn=Globus Simple CA, ou=simpleCA-corto1.cluster, ou=LFC, o=Grid

Do you want to keep this as the CA subject (y/n) [y]: y
```

```
Enter the email of the CA (this is the email where certificate
requests will be sent to be signed by the CA): joioso@ifsc.usp.br
The CA certificate has an expiration date. Keep in mind that
once the CA certificate has expired, all the certificates
signed by that CA become invalid.  A CA should regenerate
the CA certificate and start re-issuing ca-setup packages
before the actual CA certificate expires.  This can be done
by re-running this setup script.  Enter the number of DAYS
the CA certificate should last before it expires.
[default: 5 years (1825 days)]:RETURN
Enter PEM pass phrase:******
Verifying - Enter PEM pass phrase:******
creating CA config package...
A self-signed certificate has been generated
for the Certificate Authority with the subject:

/O=Grid/OU=LFC/OU=simpleCA-corto1.cluster/CN=Globus Simple CA
If this is invalid, rerun this script
/usr/local/globus-4.0.4/setup/globus/setup-simple-ca
and enter the appropriate fields.
-------------------------------------------------------------
The private key of the CA is stored in
/home/globus/.globus/simpleCA/private/cakey.pem
The public CA certificate is stored in
/home/globus/.globus/simpleCA/cacert.pem

The distribution package built for this CA is stored in

/home/globus/.globus/simpleCA/ globus_simple_ca_ff170b2e_setup-
0.19.tar.gz

This file must be distributed to any host wishing to request
certificates from this CA.

CA setup complete.

The following commands will now be run to setup the security
configuration files for this CA:
$GLOBUS_LOCATION/sbin/gpt-build \
/home/globus/.globus/simpleCA /globus_simple_ca_ff170b2e_setup-
0.19.tar.gz
$GLOBUS_LOCATION/sbin/gpt-postinstall
-------------------------------------------------------------
setup-ssl-utils: Configuring ssl-utils package
```

```
Running setup-ssl-utils-sh-scripts...
************************************************************
Note: To complete setup of the GSI software you need to run the
following script as root to configure your security configuration
directory:
/usr/local/globus-4.0.4/setup/globus_simple_ca_ff170b2e_setup-
0.19/setup-gsi
For further information on using the setup-gsi script, use the –help
option.  The -default option sets this security configuration to be
the default, and -nonroot can be used on systems where root access
is not available.

************************************************************
setup-ssl-utils: Complete
```

Quadro 4 - Configuração da *SimpleCA*

O Quadro 5 apresenta a listagem do conteúdo do diretório criado após a execução do comando **setup-simple-ca**. É importante destacar o arquivo **globus_simple_ca_ff170b2e_setup-0.19.tar.gz** que tem as configurações das chaves pública e privada e deve ser instalado em todos os *host* que forem requisitar um certificado a essa CA e portanto fazer parte do *grid* do LFC.

```
globus@corto1:~$ ls -l .globus/simpleCA/
total 264
-rw-r--r-- 1 globus globus    924 2007-11-07 17:11 cacert.pem
drwx------ 2 globus globus   4096 2007-11-07 17:11 certs
drwx------ 2 globus globus   4096 2007-11-07 17:11 crl
-rw-r--r-- 1 globus globus 214913 2007-11-07 17:11
globus_simple_ca_ff170b2e_setup-0.19.tar.gz
-rw-r--r-- 1 globus globus   2859 2007-11-07 17:11 grid-ca-ssl.conf
-rw-r--r-- 1 globus globus    928 2007-11-10 19:02 index.txt
drwx------ 2 globus globus   4096 2007-11-10 19:02 newcerts
drwx------ 2 globus globus   4096 2007-11-07 17:11 private
-rw-r--r-- 1 globus globus      3 2007-11-10 19:02 serial
```

Quadro 5 - Conteúdo do diretório onde são criados as chaves de autenticação da *SimpleCA*

Em seguida o certificado da CA deve ser copiado para o diretório **/etc**, para isso é necessário estar logado como superusuário, setar a variável de ambiente GLOBUS_LOCATION e executar o comando **setup-gsi**. Este comando descompactará o arquivo **globus_simple_ca_ff170b2e_setup-0.19.tar.gz** no diretório **/etc/grid-security** que armazena os certificados de autenticação do *host* no *grid* como mostra o Quadro 6.

```
root@corto1:~$ export GLOBUS_LOCATION=/usr/local/globus-4.0.4
root@corto1:~#
$GLOBUS_LOCATION/setup/globus_simple_ca_ff170b2e_setup-0.19/setup-
gsi -default
setup-gsi: Configuring GSI security
Making /etc/grid-security...
mkdir /etc/grid-security
Making trusted certs directory: /etc/grid-security/certificates/
mkdir /etc/grid-security/certificates/
Installing /etc/grid-security/certificates/grid-
security.conf.ff170b2e...
Running grid-security-config...
Installing Globus CA certificate into trusted CA certificate
directory...
Installing Globus CA signing policy into trusted CA certificate
directory...
setup-gsi: Complete
root@corto1:~# ls /etc/grid-security/
certificates  globus-host-ssl.conf  globus-user-ssl.conf  grid-
security.conf
root@corto1:~# ls /etc/grid-security/certificates/
ff170b2e.0                      globus-user-ssl.conf.ff170b2e
ff170b2e.signing_policy         grid-security.conf.ff170b2e
globus-host-ssl.conf.ff170b2e
```

Quadro 6 - Comando setup-gsi e listagem dos certificados do *host*

3.2 CONFIGURANDO UM HOSTCERTIFICATE

Todo *host* que for participar de um *grid* deve fazer a requisição de um certificado (*HostCertificate*) à CA daquele *grid*. No caso do grid do LFC, a CA foi configurada para ser a máquina **corto1** todos os outros *hosts* devem, portanto, fazer a requisição de um certificado

e enviar para a entidade certificadora (**corto1**) autenticar este certificado.

No Quadro 7 temos um exemplo de solicitação de *HostCertificate* para o *host* **corto1** e a listagem do diretório **/etc/grid-security**. É importante observar a criação de 3 novos arquivos:

- **hostkey.pem:** Chave privada do *host* **corto1**;
- **hostcert_request.pem:** Deve ser enviado para a *SimpleCA*, para que a entidade certificadora assine o certificado e o devolva para o *host*. O envio pode ser feito através do comando **scp** no caso de um *cluster* ou por e-mail em casos de *hosts* que não compartilham a mesma rede;
- **hostcert.pem:** *HostCertificate*, que é o arquivo de certificado do *host* devolvido pela CA após ter sido autenticado.

A solicitação do certificado é efetuada através do comando **grid-cert-request –host <nome_do_host>**. Um detalhe importante é que o nome do *host* deve satisfazer os requisitos de **FQDN** (*Fully Qualified Domain Name*), ou seja, o *host* deve ter um nome bem conhecido na rede. No caso de recursos espalhados através da internet este nome deve estar cadastrado em registros de **DNS** (*Domain Name Service*), em caso de um *cluster* basta que esse nome esteja cadastrado no arquivo **/etc/hosts**.

```
root@corto1:~# grid-cert-request -host corto1
Generating a 1024 bit RSA private key
..++++++
.....................................................++++++
writing new private key to '/etc/grid-security/hostkey.pem'
...
Your certificate will be mailed to you within two working days.
If you receive no response, contact LFC Simple CA at
joioso@ifsc.usp.br
root@corto1:# ls /etc/grid-security/
certificates       globus-user-ssl.conf  grid-security.conf
hostcert_request.pem      globus-host-ssl.conf
hostcert.pem        hostkey.pem
```

Quadro 7 -Exemplo de requisição do *HostCertificate*

Após a requisição do certificado, o arquivo **hostcert_request.pem** deve ser enviado à CA que deve assinar o certificado e devolver o arquivo assinado para o *host* solicitante, que deve ser copiado para o arquivo **/etc/grid-security/hostcert.pem**.

No caso do *grid* LFC, o administrador da CA é o próprio usuário globus e a validação do *HostCertificate* é realizada através do comando **grid-ca-sign**, como é apresentado pelo Quadro 8. Os parâmetros deste comando são o arquivo de requisição do certificado (**hostcert_request.pem**) e o nome do arquivo assinado (**hostcert.pem**) que em seguida deve ser enviado para o *host* requisitante (**corto1**) e copiado para o diretório **/etc/gri-security**.

```
globus@corto1:~$ grid-ca-sign -in hostcert_request.pem –out
hostcert.pem
To sign the request
please enter the password for the CA key:******

The new signed certificate is at:
/home/globus/.globus/simpleCA/newcerts/01.pem
```

Quadro 8 - Assinatura do *HostCertificate*

5.3.4 CONFIGURANDO UM USERCERTIFICATE

Da mesma maneira que o **host** o usuário também deve requisitar um certificado ao *grid* que irá participar. Essa requisição é chamada de *UserCertificate* é realizada pelo usuário através do comando **grid-cert-request**.

Primeiramente, o usuário deve setar a variável de ambiente GLOBUS_LOCATION. Em seguida, ele deve carregar o arquivo **globus-user-env.sh** que configura todo o ambiente para o usuário executar comandos do globus, e por último, executar o comando **grid-cert-request**, que solicitará ao usuário uma senha. Essa senha será utilizada pelo usuário para submeter processo ao *grid*. O Quadro 9 apresenta essa seqüência de comandos.

```
joioso@corto1:~$ export GLOBUS_LOCATION=/usr/local/globus-4.0.4
joioso@corto1:~$ source /usr/local/globus-4.0.4/etc/globus-user-
env.sh
joioso@corto1:~$ grid-cert-request
A certificate request and private key is being created.
You will be asked to enter a PEM pass phrase.
This pass phrase is akin to your account password,
and is used to protect your key file.
If you forget your pass phrase, you will need to
obtain a new certificate.

Generating a 1024 bit RSA private key
.............++++++
........++++++
writing new private key to '/home/joioso/.globus/userkey.pem'
Enter PEM pass phrase: *****
Verifying - Enter PEM pass phrase: *****
-----
You are about to be asked to enter information that will be
incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or
a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Level 0 Organization [Grid]:Level 0 Organizational Unit
```

```
[GlobusTest]:Level 1 Organizational Unit [simpleCA-
corto1.cluster]:Level 2 Organizational Unit [cluster]:Name (e.g.,
John M. Smith) []:
A private key and a certificate request has been generated with the
subject:
/O=Grid/OU=GlobusTest/OU=simpleCA-
corto1.cluster/OU=cluster/CN=Aparecido Luciano B Joioso

If the CN=Aparecido Luciano B Joioso is not appropriate, rerun this
script with the -force -cn "Common Name" options.
Your private key is stored in /home/joioso/.globus/userkey.pem
Your request is stored in /home/joioso/.globus/usercert_request.pem

Please e-mail the request to the Globus Simple CA joioso@ifsc.usp.br
You may use a command similar to the following:

  cat /home/joioso/.globus/usercert_request.pem | mail
joioso@ifsc.usp.br

Only use the above if this machine can send AND receive e-mail. if
not, please mail using some other method.
Your certificate will be mailed to you within two working days.
If you receive no response, contact Globus Simple CA at
joioso@ifsc.usp.br
```

Quadro 9 - Requisição de um UserCetificate

Após a requisição do certificado, é criado na área do usuário o diretório .globus que contém os seguintes arquivos:

- **userkey.pem:** Chave privada do usuário;
- **usercert_request.pem:** Deve ser enviado para a *SimpleCA*, para que a entidade certificadora assine o certificado e o devolva para o usuário. O envio pode ser feito através do comando **scp** no caso de um *cluster* ou por e-mail em casos de *hosts* que não compartilham a mesma rede;
- **usercert.pem:** *UserCertificate*, que é o arquivo de certificado do usuário devolvido pela CA após ter sido autenticado.

O Quadro 10 mostra a listagem do diretório **.globus** na área local do usuário.

```
joioso@corto1:~$ ls -l .globus
-rw-r--r-- joioso joioso    0 2008-01-15 15:17 usercert.pem
-rw-r--r-- joioso joioso 1426 2008-01-15 15:17 usercert_request.pem
-r-------- joioso joioso  963 2008-01-15 15:17 userkey.pem
```

Quadro 10 - Listagem dos arquivos criados na área do usuário

Em seguida, o usuário deve enviar o arquivo **usercert_request.pem** ao administrador da CA, que irá validar esse certificado e devolver para o usuário copiá-lo no diretório **.globus/usercert.pem**. Esses passos são os mesmos realizados na validação *HostCertificate*. No entanto o arquivo de entrada deve ser o **usercert_request.pem** e o de saída o **usercert.pem**.

3.5 CONFIGURANDO O GRIDFTP

Como já foi explicado, o **GridFTP** é o serviço responsável pela transferência de arquivos no *grid*. Para configurá-lo, deve ser criado um arquivo de inicialização de serviço através do *daemon* do sistema operacional **xinetd.d**. Em seguida, o *daemon* **xinetd** deve ser reinicializado. Um exemplo do arquivo de inicialização do **GridFTP** e a reinicialização do **xinetd** pode ser acompanhado no Quadro 11.

```
corto1:~# cat /etc/xinetd.d/gridftp
service gridftp
{
        instances = 100
        socket_type = stream
        wait = no
        user = root
        env += GLOBUS_LOCATION=/usr/local/globus-4.0.4
        env += LD_LIBRARY_PATH=/usr/local/globus-4.0.4/lib
        server = /usr/local/globus-4.0.4/sbin/globus-gridftp-server
        server_args = -i
        log_on_success += DURATION
        nice = 10
        disable = no
}
corto1:~# /etc/init.d/xinetd reload
```

Quadro 11 - Arquivo de inicialização do GridFTP e reinicialização do *daemon* xinetd

5.3.6 CONFIGURANDO O GATEKEEPER

A configuração do *Gatekeeper* é similar à do **GridFTP**, pois é necessária a criação de um arquivo de inicialização do serviço *Gatekeeper* através do *daemon* **xinetd** e reinicalizar o **xinetd**. O arquivo de incialização do *gatekeeper* e a reinicialização do *daemon* **xinetd** são apresentados pelo Quadro 12.

```
corto1:~# cat /etc/xinetd.d/globus-gatekeeper
service gatekeeper
{
        socket_type = stream
        protocol = tcp
        wait = no
        user = root
        env = LD_LIBRARY_PATH=/usr/local/globus-4.0.4/lib
        server = /usr/local/globus-4.0.4/sbin/globus-gatekeeper
        server_args = -conf /usr/local/globus-4.0.4/etc/globus-
gatekeeper.conf
        disable = no
}
corto1:~# /etc/init.d/xinetd reload
```

Quadro 12 - Arquivo de inicialização do Gatekeeper e reinicialização do daemon xinetd

Esses são os passos necessários para colocar o *Globus Toolkit* em funcionamento em uma máquina. Repita esses passos em quantas máquinas forem compor seu *grid*, lembrando que os passos de criação de um *HostCertificate* e *UserCertificate* devem ser realizados solicitando a assinatura das chaves públicas e privadas para o nó que foi definido como a *SimpleCA.*