FRANCISCO ISIDRO MASSETTO

Hybrid MPI – Uma Implementação MPI para Ambientes Distribuídos Híbridos

Tese apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção de Título de Doutor em Engenharia Elétrica.

Orientadora: Profa. Dra. Líria Matsumoto Sato

São Paulo

2007

FRANCISCO ISIDRO MASSETTO

Hybrid MPI – Uma Implementação MPI para Ambientes Distribuídos Híbridos

Tese apresentada à Escola Politécnica da Universidade de São Paulo para a obtenção de Título de Doutor em Engenharia Elétrica.

Área de Concentração:
Sistemas Digitais

Orientadora: Profa. Dra. Líria Matsumoto Sato

São Paulo
2007

“Confia no Senhor e faze o bem; assim habitarás na Terra, e te alimentarás em segurança.

Deleita-se também no Senhor, e ele te concederá o que deseja o teu coração

Entrega teu caminho ao Senhor; confia Nele, e Ele tudo o fará”

Salmo 37 (3-5)

Agradecimentos

Primeiramente gostaria de agradecer a Deus pelo dom da vida, pela força que me dá a cada dia quando acordo e pela família maravilhosa que me reservou.

Agradeço aos meus maravilhosos familiares, meus anjos da guarda: meu pai, Ivair Massetto, minha mãe Izildinha Sueli Corrêa Massetto, meu irmão Ivair Massetto Junior, minha cunhada Laura Beatriz, e todos os meus familiares (Gu, Re, Déia, Karina, Tia Iraci, Vó Cida) que compartilharam, durante estes cinco anos, toda luta, angústias, vitórias e celebrações. Não teria conseguido sem vocês.

Agradeço à minha orientadora, Profa. Dra. Líria Matsumoto Sato, que além de todo apoio técnico, muitas vezes assumiu o papel de "mãe" de um filho cheio de problemas e confusões. Com certeza a senhora é um anjo que Deus colocou em meu caminho. E a todos os amigos da USP (Kakugawa, Rivera, Nilton, Mathias, Charles, Léo, Rogério, Jean, Waldemar, e todos cujos nomes não mencionei).

Agradeço também a dois irmãos que fiz nesta caminhada: Augusto Mendes Gomes Junior e Calebe de Paula Bianchini. A força, as risadas, o futebol às sextas-feiras, a parceria no trabalho, estudo e na vida em geral serão sempre lembradas com carinho e com certeza continuaremos com essa amizade.

A todos os meus amigos que também estiveram sempre ao meu lado: Andréia Machion, Sandro Ferraz. Obrigado de coração pela força.

À toda galera de Jaú, minha irmãzinha Raquel Padoveze, Naila Meschini, Fábio Titato, Thiago Zago, Françoise Guelfi, que enxugaram as muitas lágrimas derramadas e sempre emprestaram seu ombro mais que amigo para me consolar.

Aos parceiros da Universidade Anhembi Morumbi (Elenice, Bonfá, Dália, Dani Marney e Tia Sonia que me levavam almoço na fase de testes) e do Centro Universitário UNIFIEO (Mario Leston, Bernadete, Dra. Adair, Tânia, Marcelo Rollemberg, Dirce, Dete) pelo suporte, compreensão, apoio e companheirismo que recebi.

Enfim, a todos os que infelizmente não mencionei aqui neste trabalho por falta de espaço, mas que sabem que moram em meu coração. Meu mais sincero MUITO OBRIGADO.

RESUMO

O crescente desenvolvimento de aplicações de alto desempenho é uma realidade presente nos dias atuais. Entretanto, a diversidade de arquiteturas de máquinas, incluindo *monoprocessadores* e *multiprocessadores*, *clusters* com ou sem máquina *front-end*, variedade de sistemas operacionais e implementações da biblioteca MPI tem aumentado cada dia mais. Tendo em vista este cenário, bibliotecas que proporcionem a integração de diversas implementações MPI, sistemas operacionais e arquiteturas de máquinas são necessárias. Esta tese apresenta o HyMPI, uma implementação da biblioteca MPI voltada para integração, em um mesmo ambiente distribuído de alto desempenho, nós com diferentes arquiteturas, *clusters* com ou sem máquina *front-end*, sistemas operacionais e implementações MPI. HyMPI oferece um conjunto de primitivas compatíveis com a especificação MPI, incluindo comunicação ponto a ponto, operações coletivas, inicio e termino, além de outras primitivas utilitárias.

ABSTRACT

The increasing develpment of high performance applications is a reality on current days. However, the diversity of computer architectures, including mono and multiprocessor machines, clusters with or without front-end node, the variety of operating systems and MPI implementations has growth increasingly. Focused on this scenario, programming libraries that allows integration of several MPI implementations, operating systems and computer architectures are needed. This thesis introduces HyMPI, a MPI implementation aiming integratino, on a distributed high performance system nodes with different architectures, clusters with or without front-end machine, operating systems and MPI implementations. HyMPI offers a set of primitives based on MPI specification, including point-to-point communication, collective operations, startup and finalization and some other utility functions.

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Atualmente, a necessidade de alto desempenho tem se mostrado cada vez mais presente, tanto em estudos acadêmicos quanto em aplicações industriais. Entretanto, aplicações que demandam alto poder computacional necessitam de infra-estruturas de *hardware* e *software* específicas, o que pode ser, em muitos casos, de custo elevado. Tendo em vista essa necessidade, o desenvolvimento de aplicações que executem paralelamente em diversas máquinas, distribuindo os dados, tem se tornado uma alternativa viável, principalmente considerando fatores como o custo do número de processadores que os compõem.

Esse tipo de infra-estrutura caracteriza um ambiente distribuído de processamento, que pode ser implementado de diversas formas, seja através de *clusters* (aglomerados de máquinas), processadores maciçamente paralelos (MPP), ou redes de estações de trabalho (*Network of Workstations* – NOW).

Segundo Buyya [1], o que caracteriza tanto os MPPs, clusters e NOWs é a existência de um conjunto de processadores, com sua própria memória local, conectados através de um canal de comunicação, atuando cooperativamente na execução de uma aplicação, comportando-se como uma única máquina, na visão do usuário. A diferença entre um MPP e os demais está na presença de um barramento conectando diretamente o processador ao adaptador de rede, aumentando o desempenho da comunicação entre

esses dois elementos, além do canal de comunicação ser, em geral, proprietário, de alta velocidade e baixa latência[2].

Tanto os *clusters* como as *NOWs* caracterizam-se por utilizarem redes de comunicação menos eficientes e mais econômicas se comparadas aos MPPs, além de equipamentos *commodities* e processadores ditos “de prateleira” (*of-the-shelf*). Tais tipos de computadores possuem um barramento hierárquico entre processador e adaptador de rede, comprometendo, em parte, o desempenho global.

A diferença básica entre NOWs e *clusters* é quanto à utilização e otimizações de software realizadas em cada um. *Clusters* possuem otimizações no sistema operacional, eliminando serviços que não são estritamente necessários para atingir desempenhos maiores, além de utilizar máquinas dedicadas no processamento, eliminando a necessidade de monitor, teclado e mouse, diferenciando-as das máquinas em uma NOW. Além disso, para a criação de *clusters* as máquinas são organizadas em um *domínio*, onde há usuários cadastrados que possuem acesso a todos os recursos de todos os nós de processamento.

Entretanto, devido à limitação nos endereçamentos de máquinas e também para um melhor gerenciamento, opta-se por utilizar, na implementação de MPP, NOWs ou *clusters*, máquinas com endereçamento privado. Neste caso, existe uma máquina, denominada *front-end*, com dois adaptadores de rede, que permite acesso externo a partir da rede pública (Internet) para a rede privada (nós de processamento). Este modelo é mais comumente visto em implementações de *clusters* e é referenciado, nesta tese, como *cluster* com máquina *front-end*. Para aquelas implementações, onde todas as

máquinas possuem endereçamento público, a denominação *clusters* sem máquina *front-end* será utilizada.

Em função da existência de diversas máquinas conectadas entre si, a comunicação entre os diversos processadores não é feita através de memória, e sim, através da troca de mensagens pela rede. Dessa forma, bibliotecas que auxiliem o desenvolvimento de aplicações que façam uso dessa infra-estrutura são imprescindíveis. Aplicações para computação em ambientes distribuídos, principalmente voltadas para *clusters* e MPPs, são desenvolvidas utilizando-se bibliotecas específicas, tais como MPI [3,4] (*Message Passing Interface* – Interface de Passagem de Mensagens) e PVM [5] (*Parallel Virtual Machine* – Máquina Paralela Virtual), otimizando a comunicação, garantindo assim uma transmissão de mensagens eficiente.

Além da relação "custo x desempenho" que os ambientes distribuídos possuem, comparados aos ambientes centralizados, a possibilidade de expansão (ou escalabilidade) é outro fator atraente. Para obter maior desempenho no sistema, acrescentam-se novos processadores (novos nós mono ou multiprocessados), realizando configurações para que estes processadores façam parte do sistema. A inclusão de novos nós de processamento com diferentes organizações ou arquiteturas, em um ambiente distribuído, torna-o heterogêneo com relação a *hardware* e a inclusão de novos nós com sistemas operacionais ou bibliotecas distintas, heterogêneo com relação a *software*. Dessa forma, havendo heterogeneidade em relação a *hardware*, *software* e à própria organização do ambiente distribuído (MPP, cluster, NOW, entre outras), tem-se um *ambiente distribuído híbrido*. A grande dificuldade, nestes casos, é a compatibilidade de software entre os diversos componentes do sistema, isto é, a presença de nós com

Sistemas Operacionais ou bibliotecas de diferentes versões torna as máquinas incompatíveis, inviabilizando a integração e, principalmente a visão pelo usuário de um sistema único.

Neste sentido, para superar essas dificuldades, são necessárias bibliotecas e camadas de software que garantam a interoperabilidade entre esses nós. Além disso, devido a muitas infra-estruturas de *clusters* com máquina *front-end* existentes, a comunicação entre um nó interno ao cluster e o ambiente externo, deve ser realizada passando pelo nó *front-end*, que funciona como um *gateway* entre os nós internos e externos.

Motivação e Justificativa

MPI é uma biblioteca amplamente difundida e utilizada por programadores que desenvolvem sistemas para ambientes distribuídos. Por ser uma especificação aberta, diversas implementações encontram-se disponíveis, tais como LAM-MPI[6], MPICH[7], WMPI[8,9], MPI-Pro[10], NT-MPICH[11,12], MP-MPI[13], entre outras. Entretanto, tais implementações não são interoperáveis, tanto por serem implementações proprietárias, pois a biblioteca MPI define um padrão e não uma forma de implementação, quanto por terem versões para sistemas operacionais (SO) distintos. Desse modo, não é possível, por exemplo, que uma aplicação desenvolvida com LAM-MPI comunique-se e sincronize-se com outra aplicação desenvolvida com MPICH. Sendo assim, uma interface que realize a integração entre as diversas implementações MPI e diversos sistemas operacionais é de grande relevância, pois permite a inclusão de nós com SO distintos, e até mesmo o desenvolvimento de ambientes distribuídos em

larga escala, formados por outros *clusters*, cada um executando seu próprio sistema operacional e sua implementação MPI. Diversas iniciativas foram propostas e desenvolvidas e serão discutidas ao longo deste trabalho.

Objetivos Gerais

Este trabalho apresenta uma estratégia para integração de ambientes de alto desempenho híbridos. Considera-se o termo “híbrido” conforme descrito na seção anterior, porém respeitando a mesma representação numérica em todos os nós do sistema. Este sistema é composto de aplicações baseadas em passagem de mensagens (mais especificamente MPI), de modo que diferentes ambientes, executando diferentes Sistemas Operacionais e implmementações MPI podem comunicar-se entre si de forma transparente para a aplicação, através de uma estratégia de comunicação para operações ponto a ponto, tais como envio e recebimento, e coletivas, como, por exemplo, dispersão e redução de dados.

Desse modo, um protótipo, denominado HyMPI foi desenvolvido, aplicando a estratégia proposta, visando comprovar a viabilidade da integração entre diversas implementações MPI, Sistemas Operacionais e organizações de ambientes distribuídos. HyMPI faz uso de abordagens distintas para viabilizar a comunicação entre nós heterogêneos, sendo eles membros de um mesmo *cluster* ou de diferentes organizações (nós internos e externos a um cluster).

Objetivos Específicos

Os seguintes objetivos específicos deste trabalho podem ser destacados:

- Propor uma arquitetura de comunicação para a criação de um sistema híbrido, contemplando *clusters* heterogêneos, com relação a nós executando sistemas operacionais e implementações MPI distintas, e também "*clusters* de *cluster*", ou seja, um *cluster* heterogêneo ou homogêneo pode ser considerado um nó do sistema.
- Prover uma implementação baseada na interface MPI, integrando diversos sistemas operacionais e implementações MPI, sem a necessidade de reescrita de código, apenas recompilação do código;
- Desenvolver um protótipo para aplicar a solução proposta utilizando um ambiente heterogêneo, voltado para Windows e Linux, porém expansível para quaisquer sistemas operacionais, executando em um ambiente de máquinas heterogêneas e clusters heterogêneos com ou sem nó *front-end*.

Proposta e Contribuições

A proposta deste trabalho é oferecer uma estratégia de integração de sistemas híbridos, através de uma arquitetura de comunicação, baseada nas primitivas MPI. Dessa forma, é possível ao projetista ou desenvolvedor criar ou portar uma aplicação para executar em um ambiente distribuído híbrido. O desenvolvedor faz uso da biblioteca HyMPI, que implementa a estratégia proposta, da mesma maneira que MPI, onde a comunicação pode ser tanto através do uso das bibliotecas nativas MPI já instaladas, envio direto via TCP/IP ou encaminhamento por um processo *gateway* para

permitir que nós internos a um *cluster* com nó *front-end* possam enviar e receber mensagens de nós externos ao *cluster*.

A principal contribuição deste trabalho é a estratégia de comunicação, que viabiliza a integração, em um mesmo domínio, de diferentes implementações MPI (incluindo versões, implementações e sistemas operacionais distintos), formando um único sistema de alto desempenho. Além disso, apresentam-se formas de integrar, além de computadores únicos, organizações de clusters, composto de nós heterogêneos em software, ou seja, com nós distintos executando sistemas operacionais e/ou implementações MPI distintas, fazendo uso ou não de máquinas *front-end*, permitindo a comunicação ponto a ponto entre membros externos e internos ao *cluster*.

Essa integração é possível sem a necessidade de alteração da aplicação original que irá executar nesta infra-estrutura, bastando nova compilação do código. HyMPI é utilizada na forma de uma biblioteca que deve ser ligada ao código-fonte, compatível com as implementações MPI disponíveis na linguagem C, tornando a comunicação entre elementos heterogêneos transparente para o programador.

HyMPI implementa a estratégia de comunicação, oferecendo um subconjunto de primitivas MPI, englobando envio e recebimento padrão ponto a ponto, primitivas de comunicação coletiva, início, finalização e algumas funcionalidades utilitárias, como, por exemplo, captura do tempo atual, identificação de processos, entre outras. As funcionalidades são baseadas na especificação MPI e utilizam as funcionalidades nativas de cada diferente componente do ambiente distribuído. Entretanto sua infra-estrutura permite suporte para expansão das funcionalidades,

proporcionando a continuidade do trabalho e desenvolvimento das demais funcionalidades que podem torná-lo totalmente compatível com a especificação MPI.

Organização do Trabalho

Este trabalho está organizado da seguinte maneira: o Capítulo 2 apresenta os trabalhos referentes às diversas implementações do MPI. No Capítulo 3 será apresentado o HyMPI, sua arquitetura, modelos de comunicação, protocolos e estrutura como também uma proposta de implementação do MPI, que oferece algumas facilidades de operação com relação às anteriores. O Capítulo 4 apresenta alguns resultados preliminares de testes realizados em diversos ambientes, comparados com ambientes homogêneos. Finalmente o Capítulo 5 apresenta as conclusões e propostas para continuidade no desenvolvimento do HyMPI.

Capítulo 2

Trabalhos Relacionados

Neste capítulo, serão apresentadas uma breve descrição sobre o padrão MPI e algumas propostas de integração entre diferentes implementações MPI existentes para sistemas de alto desempenho, dentre elas o, IMPI [14, 15, 16], PACX-MPI [17, 18, 19], MPICH-G2 [20, 21] e PVMPI [22, 23].

2.1 Message-Passing Interface (MPI)

MPI é um padrão especificado pelo MPI-Forum [24] para o desenvolvimento de uma interface de passagem de mensagens entre sistemas multiprocessadores e multicomputadores. A passagem de mensagens engloba tanto a transmissão da mensagem (via algum canal de comunicação), quanto as sincronização e ordenação das mensagens. A especificação do emissor e do receptor no envio de uma mensagem é necessária para garantir a sincronização no envio e recebimento.

Em MPI existe o conceito de *Grupos de Processos* determinados por um comunicador. Todo o conjunto de processos participantes de uma aplicação faz parte de um comunicador global denominado MPI_COMM_WORLD e cada processo participante é identificado por um número inteiro maior ou igual a zero, denominado *rank*. Assim, se uma aplicação tiver *n* processos participantes do comunicador MPI_COMM_WORLD, seus respectivos *ranks* irão variar de *0* a *n-1*. O comunicador

global é visível a todos os processos. Entretanto, é possível criar outros comunicadores, agrupando processos em subconjuntos.

Um ou mais processos podem ser executados em um mesmo *host*, denominado *nodo* ou *nó*. Caso dois ou mais processos em um mesmo nó necessitem trocar mensagens, implementações usam a comunicação através da memória, tornando-a mais eficiente.

O padrão MPI define algumas operações que podem ser realizadas por processos, não se limitando apenas à comunicação. Dentre elas, podem-se destacar:

- Envio e recebimento ponto a ponto padrão: *MPI_Send* e *MPI_Recv*, onde o emissor pode ficar em estado bloqueado enquanto a mensagem é enviada e o receptor fica também em estado bloqueado enquanto não houver recebido nenhuma mensagem;
- Envio e recebimento ponto a ponto não bloqueante: *MPI_Isend* e *MPI_Irecv*, transferindo o envio e recebimento da mensagem para o sistema operacional, permitindo ao emissor ou receptor manipularem o buffer de mensagem enquanto uma cópia é enviada ou recebida;
- Envio ponto a ponto *bufferizado*: *MPI_Bsend*, permitindo ao emissor enviar uma mensagem sem que o receptor esteja apto a recebê-la, armazenando a mensagem temporariamente em um buffer registrado pela aplicação;
- Operação coletiva de difusão: *MPI_Bcast*, enviando o mesmo conjunto de dados para todos os processos pertencentes ao mesmo comunicador;

- Operação coletiva de redução: *MPI_Reduce*, onde os dados são coletados de todos os participantes, é aplicada uma operação lógica, aritmética ou relacional e o resultado final é recebido por um processo denominado *raiz*;
- Operação coletiva de difusão com divisão: *MPI_Scatter*, onde uma diferente fração de um conjunto de dados é enviada a cada participante a partir do processo *raiz*;
- Operação coletiva de agrupamento: *MPI_Gather*, onde cada participante envia sua fração de dados e o resultado (todos as frações reunidas) é concentrado no processo *raiz*;
- Operação coletiva de sincronização: *MPI_Barrier*, bloqueando todos os processos em um determinado ponto
- Controle de envio e recebimento: *MPI_Test* e *MPI_Probe*, para verificação do envio ou recebimento completo da mensagem. Seu uso é associado com as operações de envio e recebimento assíncrono (*MPI_Isend* e *MPI_Irecv*);
- Inicialização e término: *MPI_Init* e *MPI_Finalize*.
- Criação e de comunicadores: *MPI_Comm_create*, permitindo agrupar processos em novos grupos, em função da semântica da aplicação;
- Utilitários: *MPI_Wtime*, para obtenção do instante de tempo vigente e *MPI_Get_processor_name*, que retorna o nome do *host* em que o processo está executando;

As operações disponíveis, entretanto, não se limitam às anteriores, havendo variações das operações citadas e também outras para propósitos específicos, tais como E/S. O disparo de uma aplicação MPI em várias máquinas utiliza protocolos de *shell* remoto como o *Remote Shell* ou *Secure Shell*. O aplicativo *mpirun* é responsável por iniciar os processos a partir de uma lista de *hosts* pré-definida pelo usuário, ou por uma lista de *hosts* disponíveis, onde o próprio aplicativo realiza a seleção de onde os processos serão disparados.

2.2 IMPI – Interoperable MPI

IMPI é um padrão que define regras e aspectos de interoperabilidade entre diferentes implementações MPI. Ele também define algumas terminologias para a compreensão do modelo, dentre elas:

- *Server*: ponto central para o início das tarefas.

- *Client*: cada implementação MPI diferente é considerada um *Client* e deve conectar-se ao *Server* para transmitir e receber comandos de início e término.

- *Host*: máquina que abriga um ou mais processos MPI.

- *Proc*: identificador do processo dentro do comunicador MPI_COMM_WORLD.

Em IMPI, existe uma numeração global para os processos. Isso significa que, se houver ***n*** processos de uma implementação “A” (processos com *ranks* entre *0* e *n-1* em seu comunicador MPI_COMM_WORLD) e houver ***m*** processos de uma implementação “B” (*ranks* entre *0* e *m-1* analogamente), a numeração de todos os

processos será entre *0* e *m+n-1* para o comunicador global MPI_COMM_WORLD. A Figura 1 ilustra esta numeração.

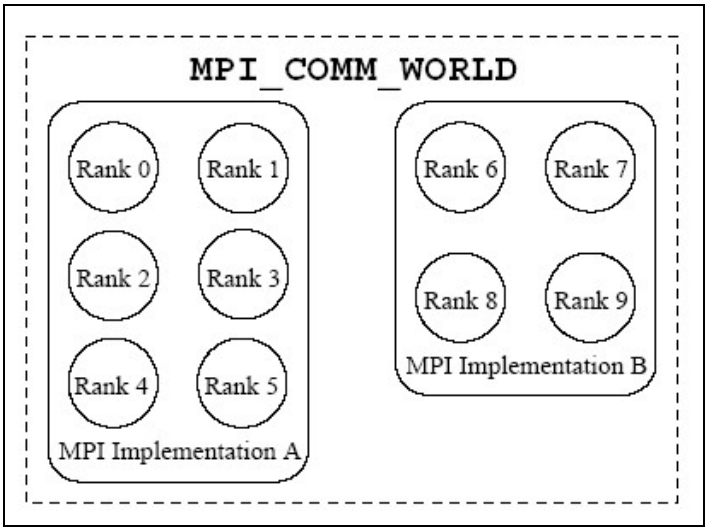


Figura 1 - Numeração de Processos em IMPI [14]

Nesta figura, pode-se observar dois *clients*, para os quais são identificadas duas diferentes implementações MPI (A e B), as quais possuem em seus comunicadores MPI_COMM_WORLD, respectivamente, 6 e 4 processos. Desse modo, unindo as implementações, tem-se, ao todo 10 *procs* com *ranks* de 0 a 9.

O processo *server*, responsável por toda a negociação de início e fim das aplicações MPI, é implementado para ser o mais simples possível. Funciona em um modelo *rendezvous*[1], onde recebe informações sobre os processos participantes de cada implementação e propaga tais informações para os demais processos, como ilustra a Figura 2. A partir de então, uma malha de comunicação entre os diversos *hosts* é criada, de modo que cada *host* possua pelo menos uma rota para alcançar os demais e seja capaz de transmitir mensagens.

[1] Uma maneira de comunicação e sincronização entre tarefas, onde uma das tarefas executa uma chamada para outra tarefa e esta segunda realiza uma instrução de aceitação (*accept*) desta chamada.

Para a comunicação ponto a ponto entre dois processos MPI, existem duas possibilidades. Se os processos pertencerem a um mesmo *client*, a comunicação é feita pela própria implementação MPI deste cliente. Entretanto, se a mensagem enviada envolve *clients* diferentes, ela deve ser encaminhada a partir do *host* local para o *host* remoto, ou seja, o padrão IMPI se preocupa apenas com o protocolo de envio e recebimento entre *hosts*.

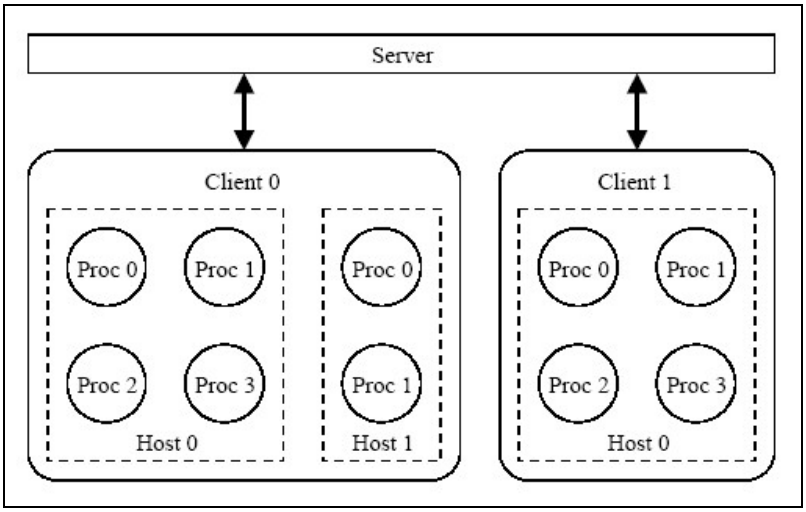


Figura 2 - Processo SERVER para início e finalização da comunicação [14]

Operações coletivas são definidas por um subconjunto de rotinas. Não só as primitivas de difusão (MPI_Bcast), barreira (MPI_Barrier) e espalhamento (MPI_Scatter) possuem especificações definidas, mas também a criação e remoção de comunicadores (MPI_COMM_CREATE e MPI_COMM_FREE, respectivamente), uma vez que os comunicadores devem ser únicos, mesmo que envolvam todas as implementações.

Também para as operações coletivas, existe um protocolo em duas fases. Uma fase local, entre todos os processos de uma mesma implementação MPI e uma fase global, coordenada entre as diversas implementações MPI. Durante a fase local utilizam-

se os algoritmos próprios de cada implementação e existem pseudocódigos específicos para coordenar as ações da fase global em cada uma das primitivas. Por exemplo, a operação de barreira define alguns processos chamados de *mestres locais* (*local master*). Na fase local, existe a sincronização entre os nós de cada implementação, ou seja, cada um dos processos realiza uma sincronização com seu mestre local, para que, após essa sincronização, todos os mestres locais possam realizar a sincronização entre si durante a fase global, como ilustra a Figura 3.

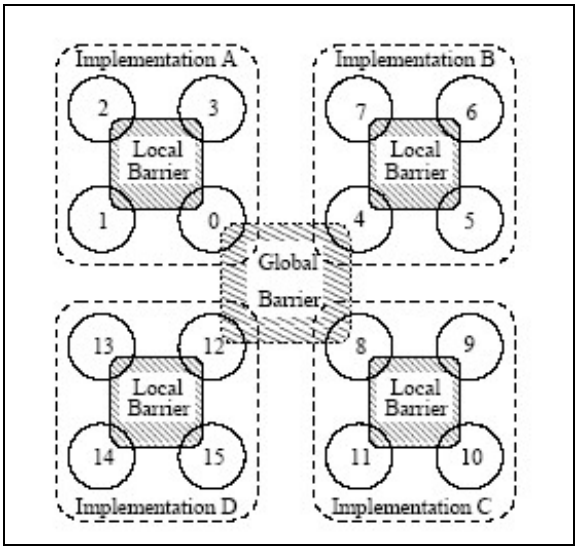


Figura 3 - Protocolo em duas fases para a operação MPI_Barrier em IMPI [14]

2.3 PACX-MPI

A extensão PACX-MPI (*PArallel Computer eXtensions*) foi concebida como um conjunto de bibliotecas ligadas ao padrão MPI para obter interoperabilidade entre dois MPPs[25] (*Massive Parallel Processors*) diferentes.

PACX-MPI apresenta um sistema de numeração global, no qual todos os processos das diferentes implementações pertencem a um único comunicador MPI_COMM_WORLD.

Essa extensão foi concebida como uma camada situada entre a aplicação do usuário e a biblioteca local MPI, dessa forma, a aplicação pode realizar chamadas MPI sem a necessidade de ter seu código reescrito, somente recompilado. Apesar de não adotar formalmente o padrão IMPI, PACX-MPI possui várias características definidas em IMPI, como, por exemplo, a forma de identificação dos processos. PACX determina se existe a necessidade de contatar um segundo MPP, dependendo da numeração global do emissor e do receptor. Caso, emissor e receptor estejam no mesmo MPP, a própria implementação local MPI é utilizada, caso contrário, a comunicação é feita através de sockets TCP/IP, como ilustra a Figura 4.

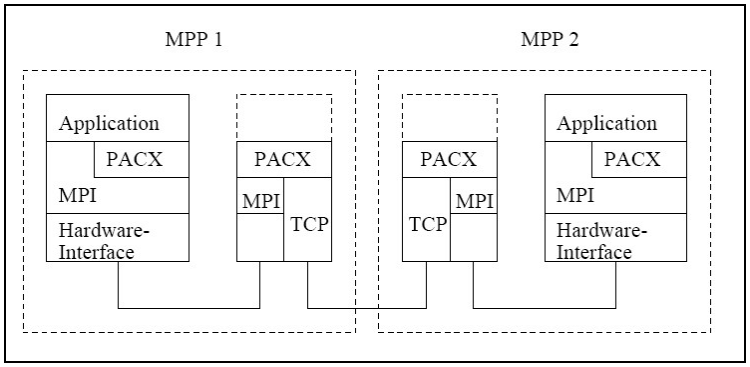


Figura 4 - Arquitetura do PACX-MPI [19]

Os recursos oferecidos pela biblioteca são:

- Iniciação e controle do ambiente.
- Comunicação ponto a ponto.
- Operações Coletivas: *MPI_Barrier*, *MPI_Bcast*, *MPI_Reduce* e *MPI_Allreduce*.
- Comunicação Não-Bloqueante.

A comunicação entre dois MPPs ocorre utilizando-se pseudo-processos MPI, sendo que cada um deles possui dois processos adicionais, responsáveis,

respectivamente pelo encaminhamento e recebimento das mensagens. A Figura 5 ilustra uma comunicação ponto a ponto entre dois MPPs.

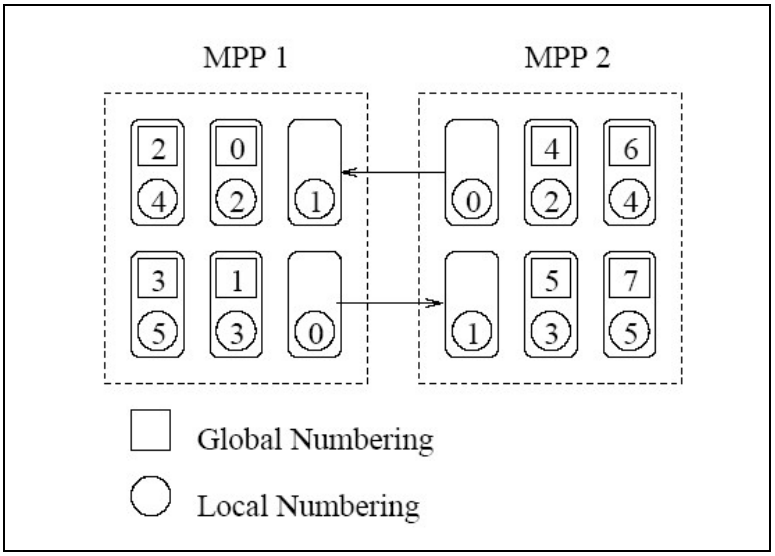


Figura 5 - Numeração de processos em PACX-MPI [19]

Conforme ilustrado na Figura 5, existem, em cada MPP, dois processos (0 e 1). O processo 0 é responsável pelo encaminhamento da mensagem e o processo 1 pelo recebimento, ou seja, eles não participam do processamento, são responsáveis apenas pela troca de mensagens portanto, a numeração global na extensão PACX-MPI não leva em consideração esses processos. Com isso, se o processo de *rank* global *3* (localizado no MPP 1, cujo *rank* local é *5*) necessita enviar uma mensagem para o processo de *rank* global *5* (localizado no MPP 2, cujo *rank* local é *3*), ele deve enviar uma mensagem para seu processo local *0* (através da biblioteca local MPI) que a encaminha para o processo *1* do MPP 2 via TCP/IP que, por sua vez, a encaminha para o processo *5*, utilizando a biblioteca MPI local. Em resumo, os processos 0 e 1 de cada implementação executam como *gateways* entre cada um dos MPP, podendo estar localizados na máquina *front-end* de um MPP com endereçamento privado.

A comunicação coletiva nessa extensão utiliza um protocolo em duas fases: considerando que o processo global de *rank 6* deseja realizar um broadcast de uma mensagem, ele envia inicialmente uma mensagem ao processo *0* (local) responsável pelo encaminhamento, identificando a mensagem como um *broadcast*. O processo *0* envia a mensagem ao processo 1 do outro MPP, enquanto que o processo *6* realiza seu *broadcast* local (em um comunicador denominado PACX_Comm_1, que inclui todos os processos locais exceto os de encaminhamento de mensagens). O processo *1* do segundo MPP realiza, então, a difusão da mensagem a todos os processos do PACX_Comm_2, que inclui todos os processos, exceto o processo *0* local. Um esquema da comunicação pode ser vista na Figura 6.

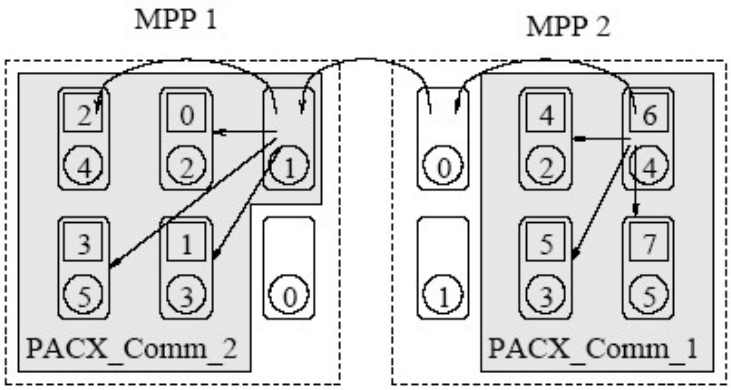


Figura 6 - Comunicação Coletiva (Broadcast) em PACX-MPI [19]

PACX-MPI é implementado para ser compatível com MPPs executando sistemas Linux ou Unix, não considerando, por exemplo, o sistema operacional Windows. Além disso, se dois MPPs diferentes que queiram se integrar não possuírem nós *front-end* em suas organizações, os processos 0 e 1 deverão executar em alguma máquina dedicada a isso ou concorrendo por memória e processamento com outros processos MPI.

2.4 MPICH-G2

MPICH-G2 é uma implementação MPI para o ambientes distribuídos, baseado no padrão MPI-1 [26], utilizando os serviços do *Globus Toolkit* [27] para estender a implementação MPICH, desenvolvida no Argonne Lab. Atualmente, essa implementação engloba várias funcionalidades do padrão MPI-2 [28].

A implementação MPICH-G2 oculta os aspectos de heterogeneidade utilizando os serviços do Globus para propósitos específicos de cada máquina ou conjunto de máquinas, tais como autenticação, autorização, criação, monitoramento, execução e controle de processos, comunicação, redirecionamento de entrada e saída e acesso remoto a arquivos. Como resultado disso, é possível executar programas MPI em diferentes localidades da mesma maneira e utilizando os mesmos comandos que seriam utilizados em um computador paralelo ou em um *cluster*

A iniciação de uma aplicação em MPICH-G2 é ilustrada na Figura 7 e descrita a seguir.

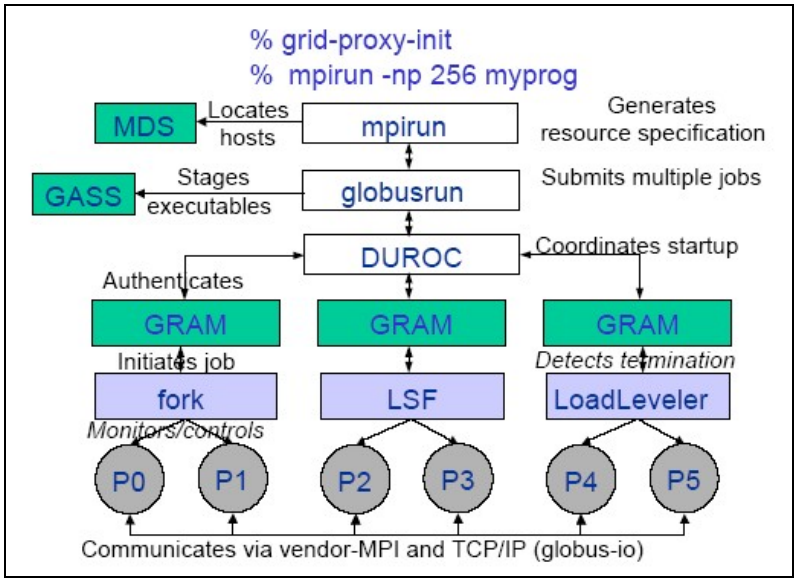


Figura 7 - Arquitetura de disparo de uma aplicação MPICH-G2 [20]

O usuário utiliza o comando *mpirun* para colocar em execução sua aplicação indicando o número de processadores. Este comando invoca o *Monitoring and Discovering Service* (MDS) [29] para resolver os endereços e alocar as máquinas descritas em um *script* previamente lido. Neste *script*, definido e fornecido pelo usuário, estão especificados parâmetros, tais como, os endereços das máquinas que participarão do processamento, o número de processadores que serão utilizados em cada máquina, caminho dos arquivos binários, parâmetros de linha de comando e arquivos de redirecionamento, caso necessário. Em seguida é executado o utilitário *globusrun* (pertencente ao pacote de softwares do Globus Toolkit) que ativa o *Dynnamically-Updated Request Online Coallocator* (DUROC) [30], responsável por escalonar e disparar as aplicações dentre os vários computadores/sites especificados. DUROC interage com o *Grid Resource Allocator and Management* (GRAM) [31] para disparar e gerenciar os processos em cada site específico e também com o *Global Access to Secondary Storage* (GASS), caso haja a necessidade da aplicação redirecionar a entrada/saída utilizando arquivos específicos. A partir daí, os escalonadores dos sistemas operacionais específicos são utilizados para criação, escalonamento e finalização de processos.

A Figura 8 ilustra um exemplo de *script* que dispara quatro processos em uma máquina A (*clstr.site1.org*) e oito em outra (*clstr.site2.org*).

```
(&(resourceManagerContact="clstr.site1.org")
  (count=4)
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 0)
    (GLOBUS_LAN_ID Univ_of_Stanford)
    (LD_LIBRARY_PATH "/usr/globus/lib"))
  (executable="/h/user/my_MPI_app_i386")
  (directory="/h/user/"))
(&(resourceManagerContact="clstr.site2.org")
  (count=8)
  (environment=(GLOBUS_DUROC_SUBJOB_INDEX 1)
    (GLOBUS_LAN_ID Univ_of_Stanford)
    (LD_LIBRARY_PATH "/ap/globus2.4.3/lib"))
  (executable="/h/john/MPI_appl_sparc"))
```

Figura 8 - Script de configuração e disparo das aplicações em MPICH-G2 [21]

A comunicação entre os processos nas diversas localidades pode ser feita através da própria implementação MPI (caso os processos estejam em um mesmo computador paralelo) ou TCP, caso a comunicação seja entre processos de localidades diferentes. Cada implementação MPI possui seu próprio comunicador MPI_COMM_WORLD, assim como nos demais modelos. MPICH-G2 também cria um comunicador MPI_COMM_WORLD global, que engloba todos os processos envolvidos. A Figura 9 ilustra dois sites: o Site A contém uma máquina IBM SP (*ranks* do MPI_COMM_WORLD de 0 a 3) com quatro processos, e o site B, dois clusters Linux (com *ranks* 4 a 7 e 8 a 12, respectivamente). Vale observar que o termo *cluster* em MPICH-G2 não contempla *clusters* com nós *front-end*, ou seja, é necessário que todos os nós deste cluster pertençam a um mesmo domínio, porém com endereçamento público visível a partir de qualquer outro nó.

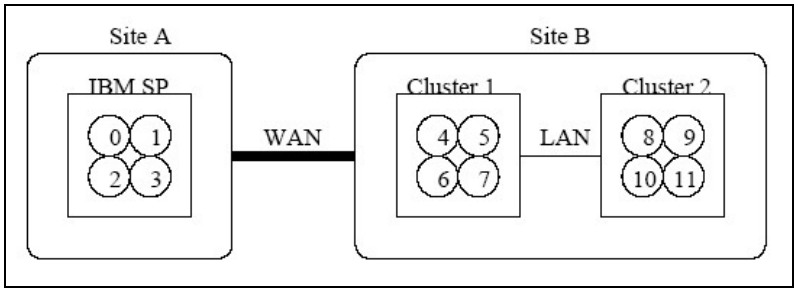


Figura 9 - Grid computacional composto de 2 sites e 12 processos [20]

Se um processo necessitar comunicar-se com outro em uma mesma máquina, a comunicação é feita pela própria implementação MPI. Caso contrário, canais de sockets TCP/IP são utilizados para realizar a troca de mensagens. A comunicação via sockets TCP/IP é notoriamente ineficiente entre nós em *clusters* distintos, se comparada com MPI para nós dentro de um mesmo *cluster*, por não utilizar recursos de *tunneling* e otimizações de pacotes. Entretanto, para aliviar esta ineficiência, MPICH-G2 implementa canais bidirecionais TCP/IP para tornar a comunicação mais independente possível e não haver conflitos de envio e recebimento de mensagens.

O recebimento de mensagens pode ser definido com o emissor identificado ou não. MPICH-G2 possui três tipos de recebimento para a comunicação ponto a ponto: *Specified*, *Specified-pending* e *Multimethod*. O recebimento do tipo *Specified* define que um processo deve especificar um emissor dentro da própria máquina e não há necessidade de recebimento assíncrono. Assim, é executada diretamente uma chamada à primitiva MPI_Recv, indicando o emissor. O tipo *Specified-pending* pode ser identificado em casos de *loops* de recebimentos. O emissor é especificado, porém pode haver várias requisições por parte de vários emissores fora de ordem (se o processo *0* necessitasse receber dos processos *1*, *2* e *3* e as mensagens chegassem na ordem *3*, *1* e *2*). Neste caso, MPICH-G2 realiza um *pooling* dentre os vários emissores para realizar

cada um dos recebimentos, isto é, uma fila é usada para armazenar cada mensagem recebida. A eficiência deste processo é menor, se comparada com o modo *Specified*. Finalmente, o tipo *Multimethod* é utilizado quando o receptor indica como emissor a constante MPI_ANY_SOURCE. Neste caso, pode haver recebimentos tanto via TCP/IP quanto via MPI. MPICH-G2 monitora continuamente ambos os canais de comunicação, porém com um custo computacional maior, tornando este modo o menos eficiente.

As operações coletivas são implementadas, considerando que alguns sites que compõem o Grid estão conectados por um canal de WAN (*Wide-Area Network*) menos veloz do que uma rede local. Desse modo, os protocolos mais lentos são utilizados primeiro e, em seguida são utilizados os protocolos mais ágeis. Cada *cluster* possui um processo denominado *master process*, considerado o processo representativo do *cluster*. Este processo é responsável pela disseminação da mensagem (em uma operação MPI_Bcast, por exemplo) ou concentração dos resultados (em uma operação MPI_Reduce, por exemplo) em cada cluster. A mensagem, então, é propagada entre os diversos *master processes*, levando em consideração os canais de comunicação mais lentos como prioritário. Isto significa que, se uma mensagem precisa ser enviada para *clusters* diferentes, primeiro utiliza-se o canal mais lento (como TCP/IP) e, em seguida, utiliza-se o canal mais eficiente (MPI nativo). Cada implementação MPI utiliza seu próprio algoritmo de difusão para propagar a mensagem dentro do *cluster*.

MPICH-G2 também não leva em consideração ambientes com sistema operacional Windows, não é projetado para *clusters* com nó *front-end*, além de garantir interoperabilidade entre diferentes implementações MPICH, e não qualquer implementação MPI.

2.5 PVM-MPI

PVMPI é uma interface de integração que utiliza PVM como uma “ponte” entre diferentes implementações MPI, unindo os dois universos distintos. A distinção entre eles se dá pelo fato de PVM possuir um comportamento dinâmico, comparado à MPI, principalmente com relação à organização e gerenciamento dos grupos de processos. Em PVM é possível criar a abstração de máquina virtual, onde processos podem aderir a um grupo, abandoná-lo ou até mesmo falhar sem afetar o restante da máquina virtual. Em contrapartida, uma vez que um processo é membro de um comunicador, ele não pode mais abandoná-lo ou falhar, pois todo o processamento pode ser comprometido. Em PVM é possível criar (*spawn*) um processo MPI, facilitando o gerenciamento global das aplicações paralelas.

Com isso, PVMPI oferece alguns recursos, tais como:

- Mapeamento e gerenciamento de identificadores (*ranks*) PVM e MPI.
- Passagem de mensagens MPI de modo transparente.
- Recursos de iniciação e gerenciamento de processos.

PVMPI oferece novas primitivas para o gerenciamento de grupos e processos, permitindo que grupos de processos MPI possam aderir ou abandonar grupos (através das primitivas *PVMPI_Register* e *PVMPI_Leave*). O *PVM Group Server* (PVMGS) gerencia a criação e disparo das aplicações MPI, gerando um comunicador global visível a todos os processos, como pode ser visto na Figura 10.

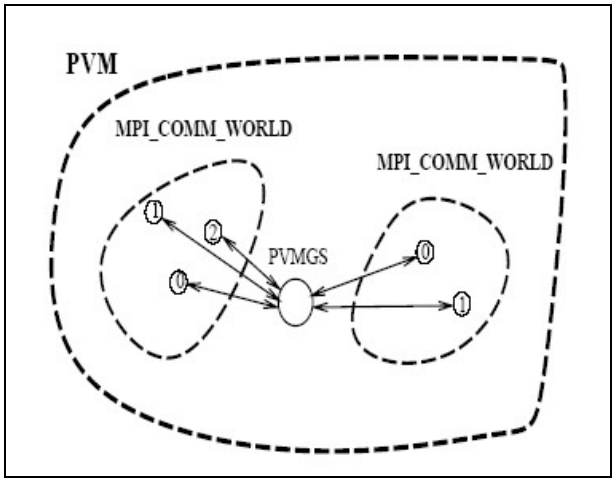


Figura 10 - Comunicadores MPI gerenciados pelo PVMGS [23]

Nesse caso, dois comunicadores MPI_COMM_WORLD com 3 e 2 processos, respectivamente, comunicam-se com o processo PVMGS, gerando uma única máquina virtual PVM com 2 comunicadores e 5 processos.

A transparência na comunicação é gerenciada por uma biblioteca que fornece toda a interface MPI para os processos, sem a necessidade de recodificação ou adequação de tipos de dados. A Figura 11 ilustra esta forma de comunicação.

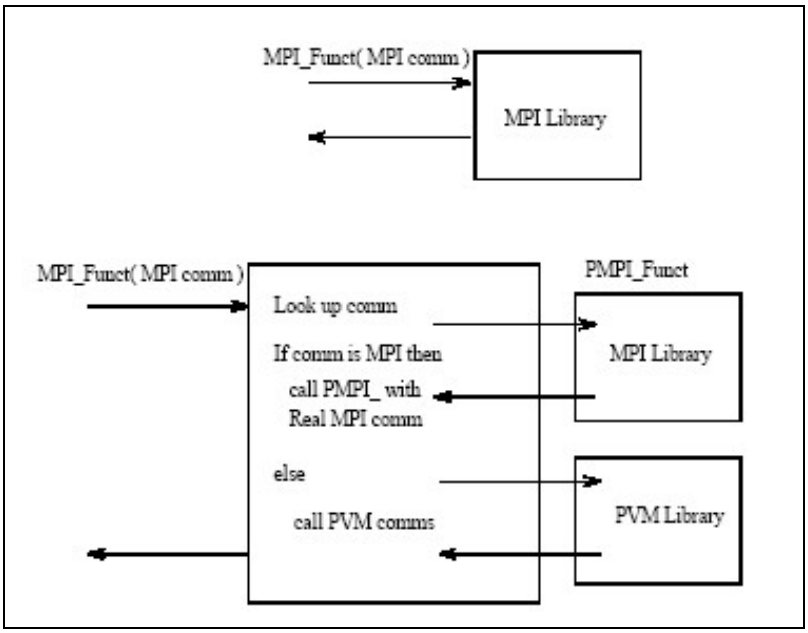


Figura 11 - Arquitetura da Interface de Comunicação PVMPI [23]

A comunicação, obrigatoriamente, é gerenciada pela interface PVMPI e caso um processo necessite comunicar-se com outro em um mesmo comunicador, é chamada a biblioteca nativa MPI. Caso contrário, PVMPI utiliza os recursos PVM para entrar em contato com o outro comunicador (que neste caso fazem parte da máquina virtual PVMPI), encaminhando a mensagem. Esta abordagem centraliza a comunicação entre diversos comunicadores. Entretanto, não há nenhuma especificação sobre como é a semântica das operações coletivas em PVMPI, quando há inclusão dinâmica de processos.

2.6 Considerações finais sobre as diferentes propostas

Uma análise qualitativa e comparativa entre PVMPI, MPICH-G2 e PACX-MPI sugere algumas reflexões. Tanto PVMPI quanto MPICH-G2 oferecem algumas estruturas de dados e outras primitivas, permitindo ao desenvolvedor escrever código, utilizando funções específicas (no caso de gerenciamento e *spawnning* de processos em PVMPI e interface com o *Globus* em MPICH-G2), comprometendo a portabilidade da aplicação. Caso não necessite dessas funções específicas, apenas a recompilação do código torna-se necessária para ligar as novas bibliotecas. A grande limitação do PVM-MPI está na falta de continuidade da biblioteca PVM, limitando seu uso e popularização.

Em relação à PACX-MPI, não há a necessidade de reescrita de código, apenas nova compilação. Entretanto cada MPP necessita de 2 processos MPI a mais, responsáveis exclusivamente pelo encaminhamento de mensagens, o que pode gerar uma sobrecarga no *host* que concentra toda a comunicação para dentro e fora do MPP. Porém pode ser uma alternativa viável para implementações de *clusters* ou MPPs com nó *front-*

end, o que não é permitido em MPICH-G2, que apenas pode contar com máquinas com endereçamento público, não permitindo o encaminhamento de mensagens por uma máquina *front-end*.

Além disso, as implementações das propostas apresentadas contemplam apenas ambientes distribuídos baseados em sistemas Unix ou Linux, não contemplando outros tipos de Sistemas Operacionais como, por exemplo, Windows.

Desse modo, na escolha de uma interface de integração é necessário que o usuário faça uma análise criteriosa (em termos qualitativos) do quão custoso pode ser a adaptação da aplicação em função da interoperabilidade, analisando cada uma das vantagens e desvantagens das alternativas. Entre os itens de análise, temos custo e esforço para configuração do ambiente, necessidade de reescrita e/ou recompilação de código, compatibilidade entre diferentes arquiteturas e sistemas operacionais, necessidade de disparo automático das aplicações, autenticação transparente, entre outras.

Capítulo 3

HyMPI – Hybrid MPI

Este capítulo apresenta a proposta desta tese, o HyMPI, como uma alternativa para a integração de diversos nós heterogêneos com relação aos sistemas operacionais e a diferentes implementações MPI. Essa alternativa, oferece, ainda, recursos para comunicação entre nós internos e externos a um *cluster*.

Hybrid MPI (HyMPI) [32, 33] é uma implementação MPI que visa integrar outras implementações MPI, bem como sistemas operacionais, compondo um ambiente distribuído híbrido. Com o objetivo de integrar quaisquer sistemas operacionais em um único sistema de alto desempenho, que pode incluir os seguintes tipos de nós:

- Máquinas homogêneas em um mesmo domínio, ou seja, todos os nós executando a mesma versão do sistema operacional e implementação MPI;
- Máquinas heterogêneas em um mesmo domínio, ou seja, nós com diferentes sistemas operacionais ou implementações MPI;
- *Clusters* sem máquinas *front-end*, composto de nós homogêneos ou heterogêneos;
- *Clusters* com máquinas *front-end*, composto de nós homogêneos ou heterogêneos;

- Ambientes com máquinas heterogêneas e clusters heterogêneos com ou sem máquina *front-end*, ou seja, sistemas compostos de elementos dos itens anteriores;

O subconjunto de primitivas MPI implementado abrange requisitos básicos para o desenvolvimento de uma aplicação paralela, tais como:

- Iniciação e Término: *MPI_Init* e *MPI_Finalize*.
- Controle de processos: *MPI_Comm_rank* e *MPI_Comm_size*.
- Comunicação Ponto a Ponto Bloqueante: *MPI_Send* e *MPI_Recv*.
- Comunicação Ponto a Ponto Não-Bloqueante: *MPI_Isend* e *MPI_Irecv*.
- Operações Coletivas: *MPI_Bcast, MPI_Reduce, MPI_Scatter, MPI_Gather, MPI_Barrier*.
- Teste e Verificação de envio/recebimento: *MPI_Probe* e *MPI_Test*.
- Utilitários: *MPI_Get_processor_name, MPI_Wtime*.

HyMPI implementa apenas o comunicador HYMPI_COMM_WORLD, não abordando aspectos de gerenciamento dinâmico de grupos, criação de novos processos (*spawnning*). Primitivas de entrada e saída, já encontradas no padrão MPI, também não são contempladas, além de outras primitivas de comunicação coletiva e envio e recebimento com armazenamento temporário. Tais características são propostas como continuidade deste trabalho.

3.1 Modelo de Sistema

A implementação HyMPI foi concebida para a interligação de diversos computadores mono ou multiprocessados e *clusters* de computadores, homogêneos ou heterogêneos, com ou sem nó *front-end*, sendo que nesses nós pode haver componentes, sistemas operacionais e/ou implementações MPI distintas, como ilustra a Figura 12.

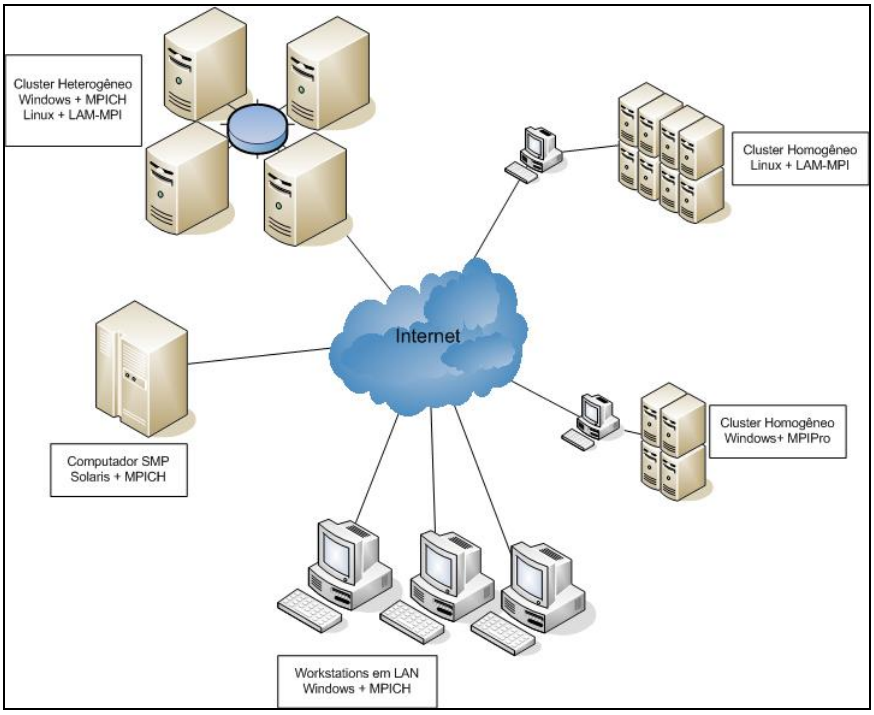


Figura 12 - Visão geral de um Modelo de Sistema para HyMPI

Nesse sistema pode haver vários ambientes de aglomerados de computadores (*clusters*) ou estações de trabalho. Por exemplo, podem ser encontrados *clusters* heterogêneos formado por nós Windows e biblioteca MPICH e Linux com LAM-MPI; *clusters* homogêneos com nós executando Windows e MPI-Pro; máquinas SMP com Solaris e MPICH; *workstations* em uma LAN com Windows e MPICH e, finalmente, *clusters* homogêneos com Linux e LAM-MPI.

O desafio principal é integrar em um único sistema os processos que serão executados em todos os nós, gerando um comunicador global HyMPI_COMM_WORLD, como ilustra a Figura 13. Nesta figura, estão representados três sistemas distintos. O Sistema A (*System A*) denota uma máquina SMP de quatro processadores com Windows e MPICH; o Sistema B (*System B*) denota uma máquina SMP Linux, também de 4 processadores, com LAM-MPI; e, finalmente, o Sistema C (System C) representa um cluster heterogêneo, composto de 16 nós, nos quais 8 executam Windows com NT-MPICH e os outros 8 executam Linux com LAM-MPI. Assim como as propostas apresentadas no Capítulo 2, HyMPI também possui uma numeração global (*rank_global*) e uma local (*rank_local*) de processos, utilizados, respectivamente, no sistema como um todo e para cada implementação MPI.

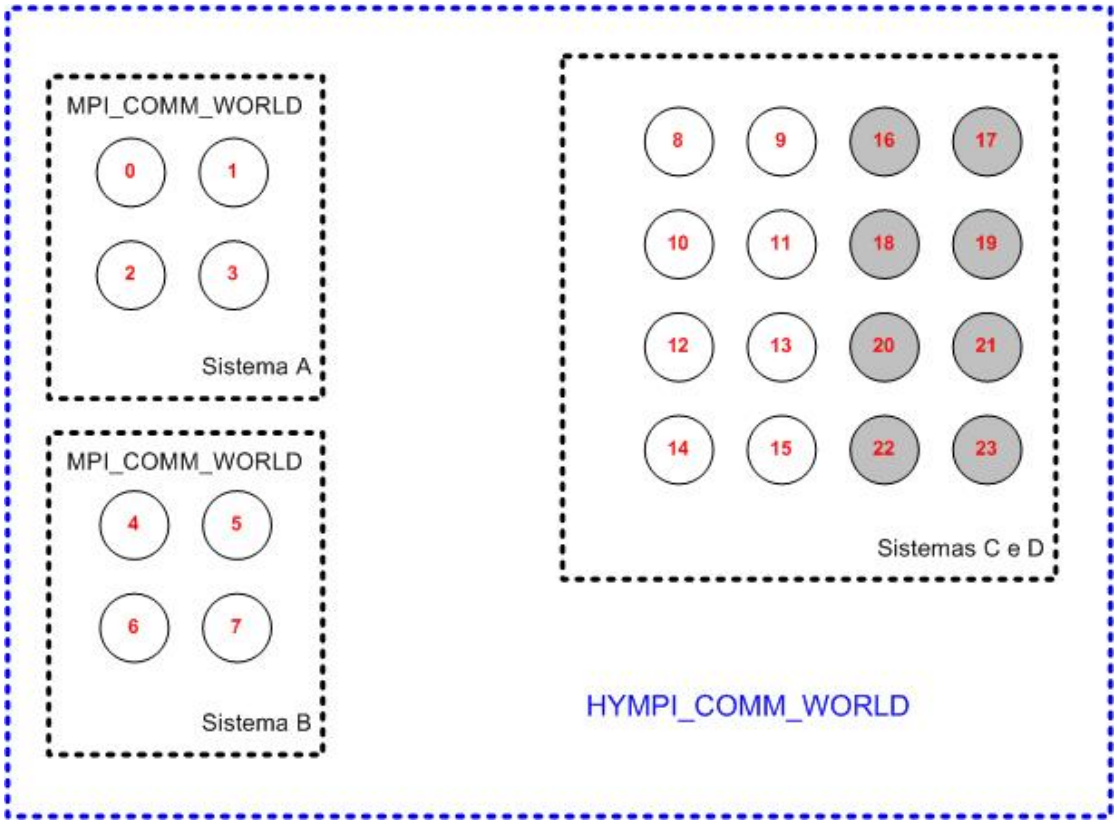


Figura 13 - Numeração Global de Processos HyMPI_COMM_WORLD

3.2 Arquitetura de uma aplicação HyMPI

HyMPI é implementada como uma biblioteca localizada entre a aplicação e a interface MPI, como mostra a Figura 14.

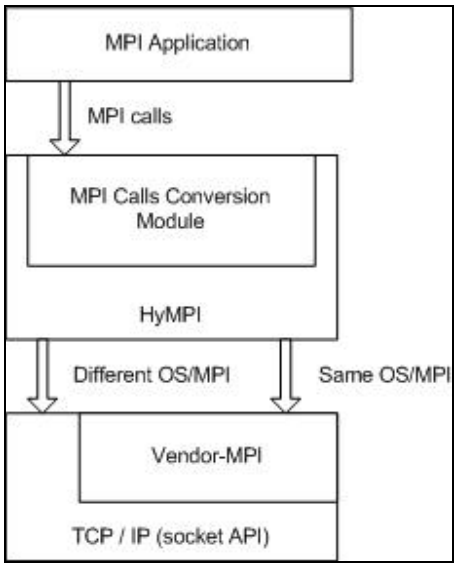


Figura 14 - Arquitetura de uma aplicação HyMPI

HyMPI foi desenvolvida para obter o maior grau de compatibilidade possível com MPI, sem necessidade de reescrita de código da aplicação. Toda a compatibilidade com as primitivas, tipos de dados e constantes definidas na especificação MPI foi mantida, entretanto, existe a necessidade de recompilação do código para realizar a ligação da biblioteca HyMPI.

As aplicações executam normalmente chamadas MPI, que são convertidas através de diretrizes de compilação para chamadas HyMPI. Neste caso, se um processo necessitar comunicar-se com outro, algumas considerações devem ser feitas:

- Se emissor e receptor pertencem à mesma implementação MPI, utiliza-se a interface nativa. É feita uma chamada à primitiva MPI

correspondente, atribuindo-se os ranks locais do emissor e receptor desta implementação;

- Se emissor e receptor pertencem a diferentes sistemas operacionais ou implementações MPI, porém estão dentro de um mesmo domínio, a comunicação é feita diretamente através da interface de *Sockets* TCP/IP. Um protocolo de envio e recebimento é utilizado para sincronizar emissor e receptor;
- Se emissor e receptor estiverem em domínios distintos, porém com nós com endereçamento público, também é utilizada a comunicação via Sockets TCP/IP, assim como no item anterior;
- Finamente se emissor e receptor estiverem em domínios diferentes, por exemplo, um externo a um *cluster* e outro interno a esse *cluster*, as mensagens devem ser encaminhadas através de um processo denominado *Gateway*, como ilustra a Figura 15;

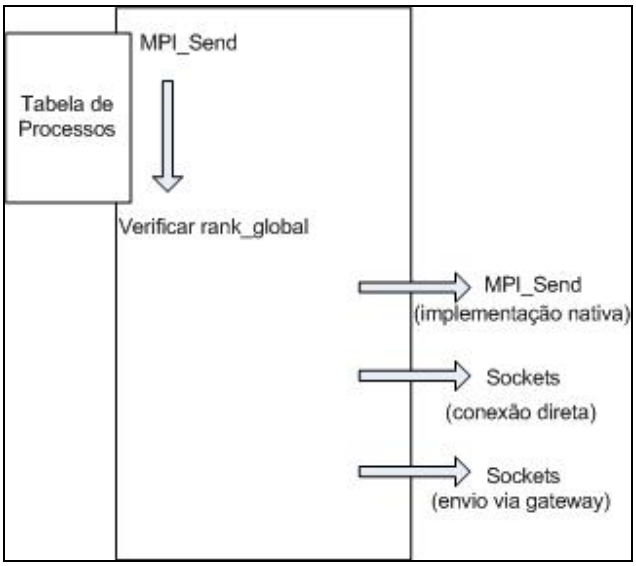


Figura 15 - Esquema de envio de mensagem em HyMPI

3.3 Processo Gateway

O processo *Gateway* tem por objetivo ser um processo encaminhador de mensagens e foi desenvolvido para ser o mais simples possível, para não sobrecarregar o processamento do nó em que estiver executando, garantindo o máximo de desempenho possível. Em função da própria arquitetura de um *cluster*, existe a necessidade de um terminal de acesso, chamado de nó *front-end*. Este nó faz a interface entre os nós pertencentes ao *cluster* e os nós externos. É exatamente no nó *front-end* que o processo Gateway deve ser instalado, como ilustra a Figura 16. Caso existam vários clusters, deve-se ter um processo *Gateway* em cada nó *front-end*.

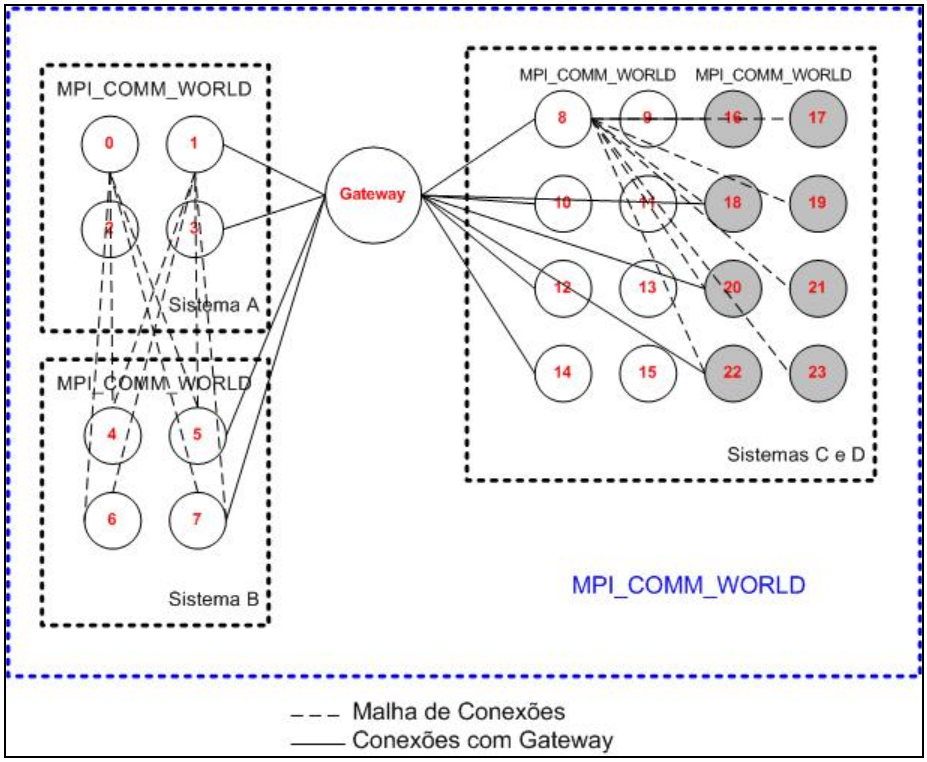


Figura 16 - Sistema com o processo Gateway

No modelo do sistema ilustrado na Figura 16, todos os processos pertencentes a uma mesma rede local, com implementações MPI ou Sistemas

Operacionais distintos, comunicam-se através de conexões diretas TCP/IP (malhas de conexões – *mesh of connections*), indicadas pelas linhas tracejadas. Processos internos e externos a um *cluster* devem comunicar-se através do Gateway, conforme indicam as linhas contínuas.

O funcionamento do processo *Gateway* consiste basicamente em um laço infinito, dedicado a receber pedidos de conexões e encaminhar mensagens para seu destino. Toda mensagem encaminhada pelo processo *Gateway* é antes, armazenada temporariamente em memória (*bufferizada*), e, em seguida, encaminhada, como pode ser visto no algoritmo ilustrado na Figura 17.

```
begin
   while (true)
   {
     receive(sender, header);
     buffer = malloc(header.size);
     receive(sender, buffer);
     connect(receiver)
     send(receiver, buffer);
     free(buffer);
  }
end.
```

Figura 17 - Algoritmo do processo *Gateway*

A decisão de armazenar temporariamente a mensagem em um *buffer* possui vantagens, tanto no envio bloqueante, quanto no envio não-bloqueante. Em ambos os casos, o processo emissor pode enviar a mensagem em sua totalidade, sendo, em seguida, liberado para continuar seu processamento. A responsabilidade do envio é, então, transferida para o processo *Gateway*. Isto significa que pode-se ter um melhor desempenho com a liberação do emissor, para que ele possa realizar outras tarefas enquanto o *Gateway* encaminha a mensagem.

Outra característica do processo *Gateway* é que ele encaminha apenas uma mensagem por vez. Quando vários emissores necessitam comunicar-se através do Gateway, os pedidos de conexão são agrupados em uma fila (gerenciada pelo próprio Sistema Operacional) e tratados um a um. Desse modo, a escalabilidade desse processo pode ser um ponto que comprometa o desempenho geral da aplicação, o que pode ser contornado tendo um processo *multithreaded*, onde cada uma das *threads* fica responsável por tratar uma requisição de cada emissor. Neste caso, mesmo com várias *threads*, a presença de uma única interface e rede limita o desempenho. Outra abordagem é a arquitetura *multithreaded* levando em conta emissor e receptor. Enquanto uma *thread* fica responsável por encaminhar uma mensagem através de uma interface de rede, outra *thread* pode realizar o recebimento de outra mensagem pela segunda interface de rede. Entretanto, tais características não estão atualmente implementadas, porém são previstas como continuidade no desenvolvimento deste trabalho.

3.4 Protocolos de comunicação

Os protocolos de comunicação desenvolvidos para HyMPI englobam dois tipos de operações principais: ponto a ponto e coletivas. Em operações ponto a ponto, a sincronização entre emissor e receptor é fundamental para que haja a ordenação correta das mensagens, tanto para envio bloqueante, quanto para envio não-bloqueante. Nas operações coletivas, utiliza-se o mesmo modelo definido nas propostas apresentadas no Capítulo 2, que consiste em um protocolo em duas fases, o qual será detalhado na seção 3.4.2.

3.4.1 Comunicação Ponto-a-Ponto

Em linhas gerais, HyMPI utiliza as informações relativas aos *ranks* locais e globais para decidir qual o tipo de envio ou recebimento, ou seja, se é através da implementação MPI nativa ou através de *Sockets*, com conexão direta ou pelo processo *Gateway*. A Figura 18 ilustra o algoritmo de envio padrão bloqueante (MPI_Send), enquanto que a Figura 19 ilustra o algoritmo de recebimento padrão bloqueante (MPI_Recv).

```
   MPI_Send(message,destino)
   Inicio
         Verificar rank_global do processo destino
         caso 1:
             rank_destino = rank_local
             MPI_Send(message,rank_destino);
         caso 2:
             connect(destino);
             send(message,destino)  /* socket */
         caso 3:
             connect(gateway);
             send(message,gateway); /* socket */
   Fim.

/* caso 1: destino pertence à mesma implementação MPI, mesmo
           Sistema Operacional e mesmo domínio(cluster)
   caso 2: implementação MPI ou sistema operacional distintos
           e mesmo domínio
   caso 3: implementação MPI ou Sistema Operacional distintos
           e domínios distintos
          */
```

Figura 18 - Algoritmo de envio padrão bloqueante (MPI_Send)

```
   MPI_Recv(message, origem)
   Inicio
         Verificar rank_global do processo origem
         caso 1:
             rank_origem = rank_local
             MPI_Recv(message,rank_local);
         caso 2:
             accept(origem);
             recv(message,origem);   /* socket */
         caso 3:
             accept(gateway);
             recv(message,gateway);  /* socket */
   Fim.

/* caso 1: destino pertence à mesma implementação MPI, mesmo
           Sistema Operacional e mesma sub-rede (cluster)
   caso 2: implementação MPI ou sistema operacional distintos
           e mesmo domínio
   caso 3: implementação MPI ou Sistema Operacional distintos
           e sub-rede distinta.
          */
```

Figura 19 - Algoritmo do recebimento padrão bloqueanto (MPI_Recv)

O conceito de Passagem de Mensagens (*Message-Passing*) [34] inclui tanto a transmissão de dados quanto a sincronização entre emissor e receptor. Tendo em vista essa necessidade, foi desenvolvido um **protocolo ponto a ponto** a fim de garantir que um receptor sempre consiga receber a mensagem vinda de um emissor especificado, assegurando a correta ordenação das mensagens.

Para tanto, um protocolo de comunicação foi definido em HyMPI para garantir tanto o envio correto quanto a sincronização entre emissor e receptor. Toda mensagem ponto a ponto enviada em HyMPI é iniciada com um cabeçalho (*envelope*), contendo informações sobre o número do processo origem, o número do processo destino, a quantidade de dados, o tamanho de cada dado, o modo de transmissão (ponto a ponto ou coletivo) e um *flag* indicando o tipo de mensagem, como ilustra a Figura 20.

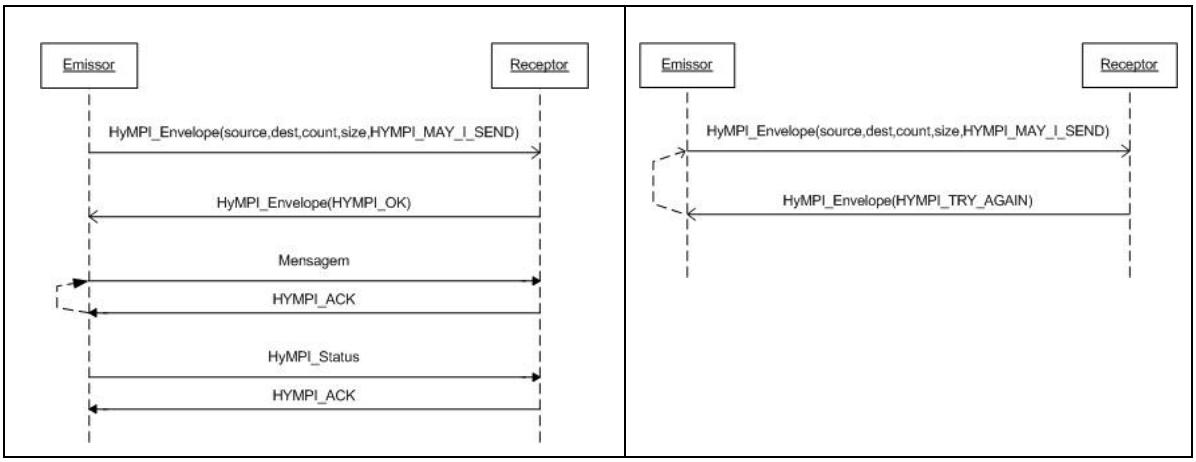
```
struct HyMPI_Envelope {
  int source;
  int destination;
  int data_size;
  int data_count;
  int mode;
  int HyMPI_Message_type;
};
```

Figura 20 - Estrutura do Cabeçalho HyMPI

As constantes definidas para descrever o protocolo de comunicação ponto a ponto HyMPI, são:

- HYMPI_MAY_I_SEND: indica uma solicitação de requisição, na qual o processo emissor envia uma mensagem ao receptor para verificar se a mensagem que é esperada pelo receptor é realmente a mensagem que será enviada;
- HYMPI_OK: indica que o receptor espera a mensagem definida e pode ser dado início ao processo de envio da mensagem;
- HYMPI_TRY_AGAIN: indica um conflito entre a mensagem que o receptor espera e a mensagem que o emissor tenta enviar. Neste caso, o emissor tenta novamente, até que o receptor esteja apto a receber a mensagem. A atual implementação do HyMPI prevê um laço infinito de requisições, finalizado somente quando o receptor responde com a mensagem HYMPI_OK. A simplicidade desta implementação tem como desvantagem um consumo desnecessário de rede, que pode ser facilmente resolvido com a utilização de uma fila de requisições por parte do receptor;

A etapa que antecede a transmissão efetiva da mensagem é denominada *negociação*. Caso o receptor receba um grande número de pedidos de envio de mensagens, ele coloca esses pedidos em uma fila gerenciada pelo Sistema Operacional e trata tais requisições uma a uma.



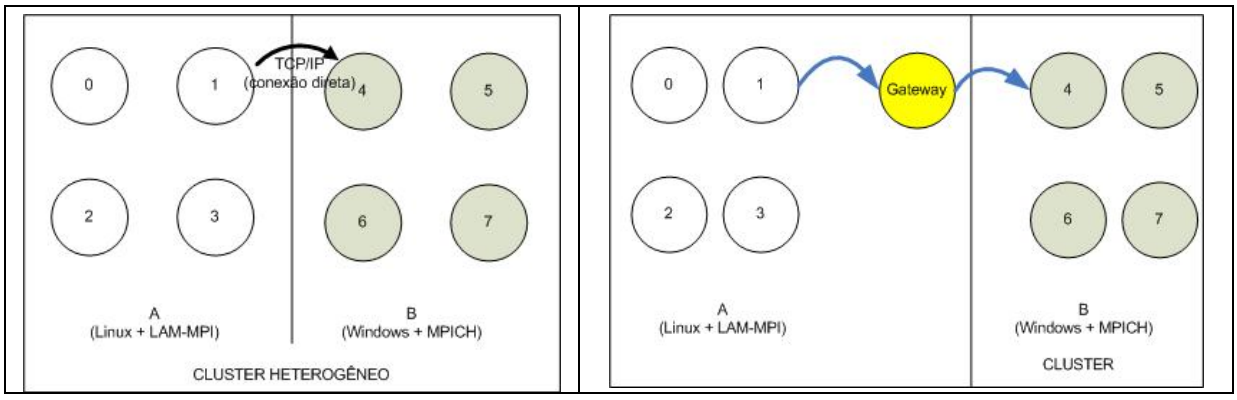
(a) (b)

Figura 21 - Seqüência das mensagens no protocolo ponto a ponto (a) corretamente (b) caso de conflito

A Figura 21 ilustra os casos de comunicação que podem ocorrer: No caso (a), o processo emissor envia uma mensagem com o envelope de cabeçalho com os dados específicos que o receptor deseja receber. O receptor, por sua vez, responde com uma mensagem do tipo HYMPI_OK e a mensagem é enviada. Ao final, a estrutura *HyMPI_Status* é enviada, concluindo a transmissão. Em (b) tem-se o caso no qual o emissor tenta enviar uma mensagem ao receptor que está esperando por outro emissor. Nessa situação, o receptor responde ao emissor para que tente mais tarde, pois não é o momento do envio da mensagem. O emissor, ao receber esse pacote, entra em um laço solicitando o envio ao receptor.

A comunicação ponto a ponto pode ser feita de duas maneiras: bloqueante (*blocking*) e não-bloqueante (*nonblocking*). De acordo com a especificação MPI, a diferença básica entre esses dois tipos de transmissão de mensagens é a possibilidade de o emissor poder manipular, no caso do envio não bloqueante, o buffer da mensagem enquanto ela é enviada.

O envio da mensagem em ambos os modos pode ser feito diretamente, caso os processos estejam dentro de um mesmo *cluster*. Para isso, o emissor deve solicitar uma conexão com o receptor e iniciar o envio normalmente. Caso o emissor seja um processo externo ao *cluster* e o receptor interno (ou vice-versa), a comunicação é intermediada pelo processo *Gateway*, que recebe a mensagem, faz seu armazenamento e encaminhamento para o receptor. A Figura 22 ilustra ambas as situações.



(a) (b)

Figura 22 - Comunicação ponto a ponto (a) direta e (b) através do processo gateway

Para o envio bloqueante, tanto a abordagem mostrada na Figura 22-a, quanto a mostrada na Figura 22-b utilizam chamadas do sistema operacional (*send* e *recv*) para envio de mensagens em modo bloqueante, não alterando, portanto, a semântica da operação.

No envio não-bloqueante, chamadas do tipo *ioctl* [35, 36] ao Sistema Operacional são utilizadas para que o *buffer* de mensagem seja liberado enquanto a mensagem é enviada. Caso o processo faça um envio, conforme a Figura 22-a, apenas a chamada ao sistema e o posterior envio são necessários. No caso do envio através do *Gateway* (Figura 22-b), para que a semântica não seja alterada, o processo emissor envia a mensagem ao *Gateway* de maneira não bloqueante, que a *bufferiza*, liberando o emissor. Em seguida, o *Gateway* encaminha a mensagem em modo bloqueante ao processo receptor. Essa última forma de envio não altera o desenvolvimento do processo *Gateway* e mantém a semântica da operação, uma vez que o envio não bloqueante permite que o emissor manipule a mensagem enquanto ela é enviada.

3.4.1.1 Recebimento indeterminado

O recebimento indeterminado ocorre quando o receptor utiliza a constante HYMPI_ANY_SOURCE e é considerada a operação mais custosa a ser realizada. Isso ocorre devido ao receptor ser capaz de receber mensagens provenientes tanto de processos de mesma implementação MPI quanto de processos de outras implementações/Sistemas Operacionais ou do processo *Gateway*.

Desse modo, é necessário monitorar recebimentos MPI e pedidos de conexões *sockets* simultaneamente. Duas *threads* são criadas e disparadas para monitorar tais recebimentos. Quando ocorre um recebimento por um dos canais, obrigatoriamente, o monitoramento do outro deve ser encerrado, para que o processo não fique indefinidamente bloqueado, mesmo recebendo mensagens por um dos canais. Essa forma de implementação exige uma carga de processamento maior do nó onde está

o processo receptor, comprometendo o desempenho geral da aplicação em função do custo computacional para criação e gerenciamento das *threads*.

3.4.2 Comunicação Coletiva

A comunicação coletiva é baseada nos modelos apresentado no Capítulo 2, onde é executado um protocolo em duas fases: *Fase Global* e *Fase Local*. A Fase Global envolve a comunicação entre processos de diferentes implementações MPI e a Fase Local envolve comunicação entre grupos de processos pertencentes a uma mesma implementação.

Durante as fases Global e Local, elege-se, para cada diferente implementação MPI, um processo chamado *raiz local*. Esse processo é o de número zero em cada comunicador MPI_COMM_WORLD e tem como função principal auxiliar na coordenação da comunicação. A fixação do processo de rank *0* em cada implementação facilita o desenvolvimento do HyMPI evitando um desperdício de tráfego de mensagens, caso seja utilizado um algoritmo de eleição.

Nas operações coletivas, a funcionalidade e a ocorrência das diferentes fases são distintas, em função da natureza da operação. Nas operações de dispersão (MPI_Bcast e MPI_Scatter) e barreira (MPI_Barrier) a Fase Global é executada antes da Fase Local. Isto significa que o processo raiz da mensagem envia a todos os processos *raiz local* de cada implementação diferente da sua. Em seguida, é executada a Fase Local, onde cada processo de cada implementação, já de posse da mensagem, encaminha-a, segundo seu próprio algoritmo nativo.

Nas operações de redução (MPI_Reduce e MPI_Gather), a Fase Local é executada antes da Fase Global. Isto significa que, primeiramente são recolhidos os resultados e concentrados no processo *raiz local* de cada implementação. Em seguida, cada um desses processos *raiz local* envia seus dados ao processo raiz da mensagem, gerando o resultado final.

O detalhamento de cada uma das operações pode ser visto nas próximas seções.

3.4.2.1 MPI_Bcast

Na operação MPI_Bcast, ilustrada na Figura 23, pode-se notar a ocorrência das duas fases descritas anteriormente.

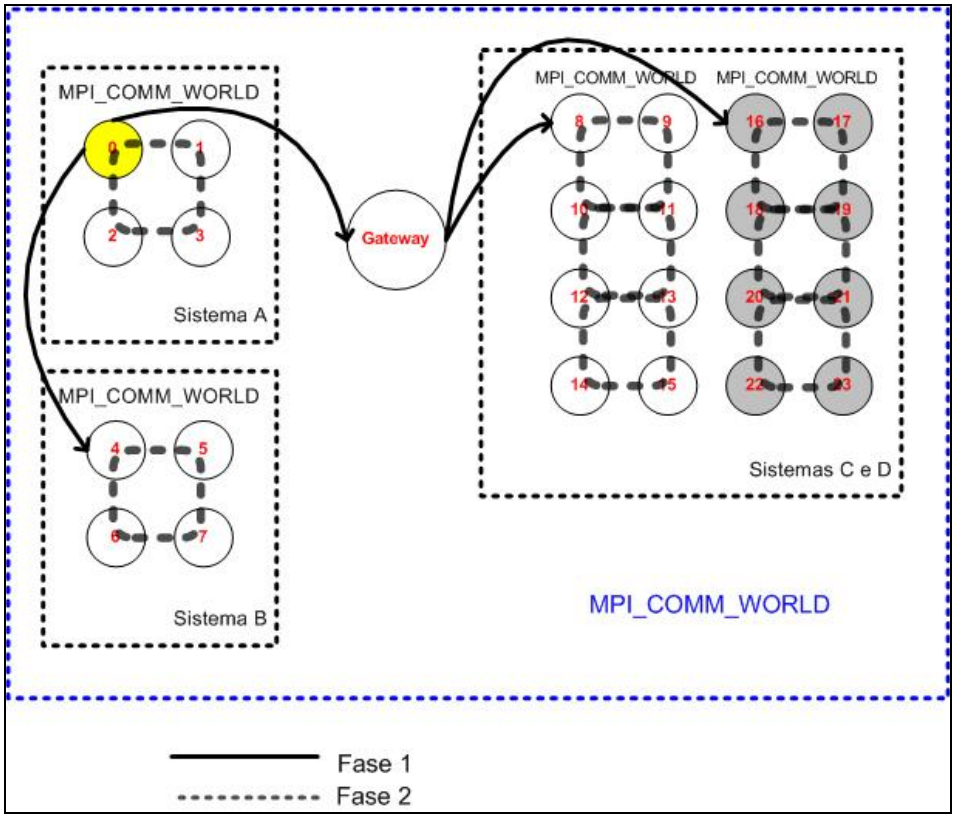


Figura 23 - Protocolo em duas fases para a operação Broadcast (MPI_Bcast)

Na Figura 23, pode-se notar que o processo denominado *raiz* é o de número (global) zero. Na Fase Global, o processo 0 envia a mensagem para todos os processos *raiz local* de cada implementação (neste caso, os processos de *rank* global igual a 4, 8 e 16). Como pode ser observado, o envio da mensagem para cada raiz local pode ser feito diretamente via *sockets* ou através do processo *Gateway*, indicado na figura pelo circulo ao centro. Com o recebimento da mensagem pela raiz local, utiliza-se o próprio algoritmo *Broadcast*, disponível na implementação, para a dispersão dos dados entre os demais processos, tendo como processo raiz, o processo raiz local da implementação MPI.

A Figura 24 ilustra um pseudocódigo relativo ao algoritmo.

```
   MPI_Bcast(message,root)
   Inicio
         Verificar rank do processo
         caso 1
             para cada root_local faça
                send(message,root_local);   /* fase global */
             fim-para
             MPI_Bcast(message,root);       /* fase local  */
         caso 2
             recv(message,root);            /* fase global */
             MPI_Bcast(message,root_local); /* fase local  */
         caso 3
             MPI_Bcast(message,0);          /* fase local */
   Fim.

/* caso 1: o processo é a raiz global da dispersão e deve
           enviar a mensagem para todos os processos raiz
           local, além de realizar o Broadcast entre os
           processos de seu comunicador MPI_COMM_WORLD
   caso 2: o processo não é o raiz global, mas é uma raiz
           local. Desse modo, deve receber a mensagem do
           processo raiz global e dispersa-lo entre seus
           pares.
   caso 3: o processo não é raiz global, nem raiz local.
           desse modo, ele deve aguardar o Broadcast de seu
           raiz local.*/
```

Figura 24 - Algoritmo da operação Broadcast (MPI_Bcast)

3.4.2.2 MPI_Reduce

Na operação (MPI_Reduce), é executada primeiramente a Fase Local e em seguida a Fase Global. A função lógico-aritmética definida é realizada localmente (Fase Local) em cada implementação MPI e o resultado gerado é propagado ao nó *raiz global* (Fase Global), que, por sua vez, aplica a função, gerando o resultado final, como ilustra a Figura 25.

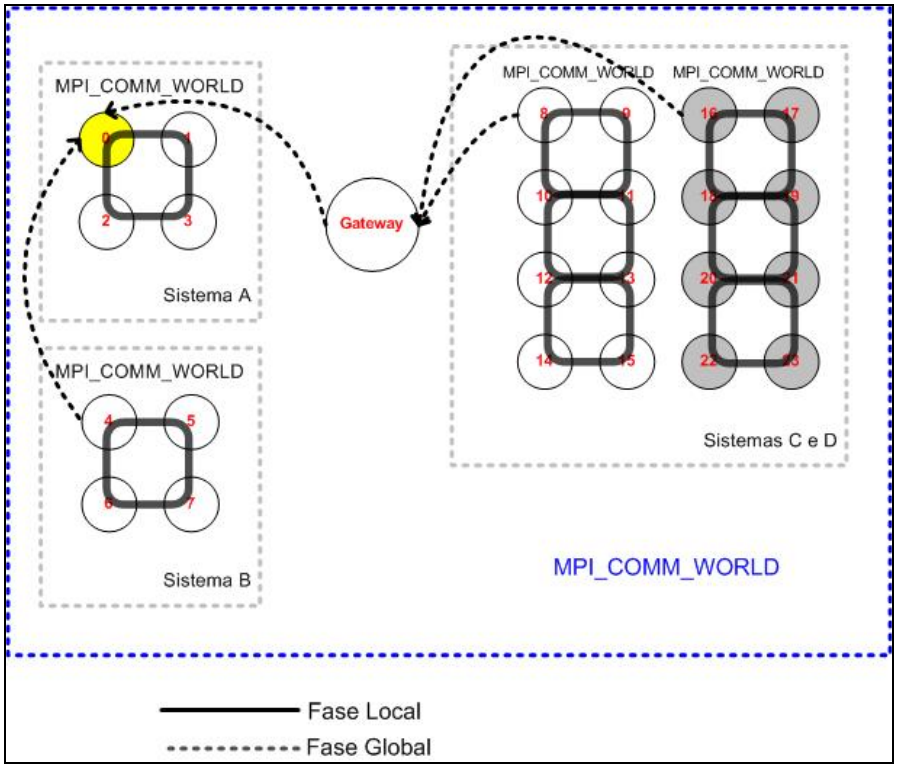


Figura 25 - Esquema da operação de Redução (MPI_Reduce)

Na figura anterior, a Fase Local é representada pela linha contínua, nela os processos *raiz local* realizam a função definida, concentrando os resultados parciais obtidos pelos demais processos, enviando-os para o processo *raiz* (global), na Fase Global (indicada pela linha tracejada). Da mesma forma que no algoritmo de Broadcast,

os processos *raiz local* são determinados como sendo os processos de *rank* 0 em cada implementação. O algoritmo MPI_Reduce pode ser visto na Figura 26.

```
   MPI_Reduce(message,root,função)
   Inicio
         Verificar rank do processo
         caso 1:
            MPI_Reduce(message, root_local, função) /*local*/
            para cada root_local, faça
               recv(root_local,message);        /*global*/
               execute(message,função)
            fim-para
         caso 2:
            MPI_Reduce(message, root_local, função) /*local*/
            send(root,message);                    /*global*/
         caso 3:
            MPI_Reduce(message, root_local, função) /*local*/
   Fim.

/* caso 1: processo é raiz global e deve realizar a operação
           entre seus pares, receber as mensagens dos
           processos raiz local e aplicar a função lógico-
           aritmética
   caso 2: processo é raiz local e deve aplicar a operação de
           redução entre seus pares e propagar o resultado
           para o nó raiz global.
   caso 3: processo não é nem raiz global, nem raiz local e
           deve, somente realizar a operação de redução
*/
```

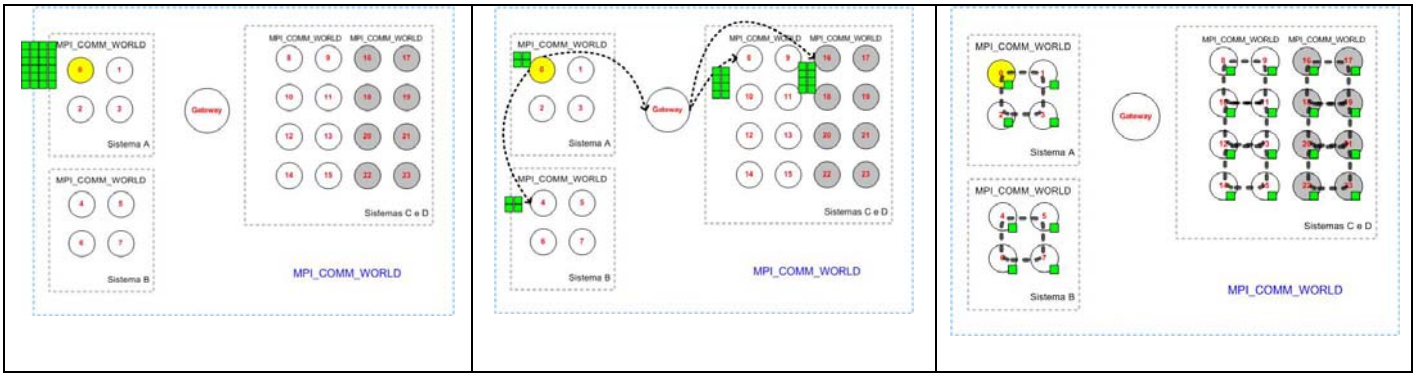
Figura 26 - Algoritmo da operação de Redução (MPI_Reduce)

3.4.2.3 MPI_Gather e MPI_Scatter

As operações MPI_Scatter e MPI_Gather obedecem o mesmo protocolo das primitivas MPI_Bcast e MPI_Reduce, respectivamente. Entretanto existe uma diferença básica entre os algoritmos: a divisão e reagrupamento dos dados. Em MPI_Scatter há uma divisão proporcional ao número de processos de cada implementação.

A Figura 27 ilustra um exemplo baseado no sistema visto na Seção 3.1, no qual o processo de *rank* 0 deseja distribuir um conjunto de 2400 elementos entre todos os processos, em uma operação MPI_Scatter (Figura 27-a).

Na primeira fase, o nó raiz local divide proporcionalmente o conjunto de dados entre os comunicadores MPI_COMM_WORLD de cada implementação e envia as frações do conjunto a todos os nós *raiz local*. Neste caso, os processos *raiz local* dos Sistemas A e B receberiam, cada um, 4/24 do conjunto, totalizando 400 elementos. Os processos *raiz local* dos sistemas C e D receberiam, cada um 8/24 do conjunto (Figura 27-b), totalizando 800 elementos. Por sua vez, cada *raiz local* divide este subconjunto entre seus processos pares, totalizando, para cada processo 1/24 do conjunto, ou seja, 100 elementos (Figura 27-c).



(a) (b) (c)

Figura 27 - Esquema de envio na operação MPI_Scatter

A operação MPI_Gather tem a mesma função do MPI_Reduce, porém faz o recolhimento dos dados da mesma forma que o MPI_Scatter.

3.4.3 Considerações sobre a implementação

HyMPI é desenvolvido para ser uma interface compatível com a especificação MPI e os protótipos disponíveis têm como foco principal a integração de sistemas Linux e Windows. Para as versões Windows, utiliza-se a biblioteca Winsockets 2.0 e para sistema Linux, utiliza-se a biblioteca Berkeley Sockets.

Para o envio de mensagens, HyMPI fragmenta os dados em pacotes de 1000 bytes, de acordo com o tamanho da janela *ethernet*. O envio e recebimento exigem mensagens de confirmação (HYMPI_ACK) para cada pacote enviado, garantindo o sincronismo e ordenação dos pacotes. A existência dos pacotes de confirmação gera uma sobrecarga na comunicação tolerável dentro do desempenho desejado.

HyMPI utiliza arquivos de configuração para construir a tabela de processos de seu comunicador HYMPI_COMM_WORLD. São necessários dois arquivos, um para especificar as configurações globais (*conf.global*), onde são informados:

- os nós que irão conter os processos;

- os sistemas operacionais de cada nó;

- bibliotecas MPI utilizadas;

- se cada nó faz parte ou não de um cluster;

- e qual o nó onde se encontra o processo *Gateway*.

O segundo arquivo (*conf.local*) especifica o sistema operacional e biblioteca MPI de cada nó. As estruturas dos arquivos de configurações são mostradas na Figura 28.

Arquivo *conf.global*

```
{hostname LINUX|WINDOWS|OTHER  LAM|MPICH|...|OTHER}+
{[CLUSTER GATEWAY hostname]
 {hostname LINUX|WINDOWS|OTHER  LAM|MPICH|...|OTHER}+}*
```

Arquivo *conf.local*

```
[CLUSTER GATEWAY hostname]
LINUX|WINDOWS|OTHER          LAM|MPICH|...|OTHER
```

Figura 28 - Estrutura dos arquivos de configuração HyMPI

No arquivo *conf.global*, o primeiro parâmetro define o *host* onde o processo será executado, seguido do tipo de sistema operacional (utilizando as macros

WINDOWS, LINUX ou OTHER) e, por último a biblioteca MPI (sendo suportadas as macros LAM-MPI, MPICH, MPIPRO ou OTHER). Caso haja uma configuração com nós dentro de um cluster, o processo *Gateway* é especificado na linha contendo os *tokens* CLUSTER GATEWAY, seguido do nome do host que o executa. Em seguida, uma lista de várias linhas contendo a descrição dos nós é informada. No caso do arquivo *conf.local*, é informado apenas se o nó faz parte ou não de um *gateway* (primeira linha opcional) e, em seguida, quais são o sistema operacional e biblioteca MPI utilizados.

Durante a execução da primitiva HyMPI_Init(), os dois arquivos são lidos e uma tabela de processos é criada em memória, disponível durante todo o tempo de vida do processo. Seja a seguinte configuração do arquivo *conf.global*.

```
node01     LINUX      LAM
node02     LINUX      LAM
node03     LINUX      LAM
node04     LINUX      LAM
node05     WINDOWS    MPICH
node06     WINDOWS    MPICH
CLUSTER GATEWAY frontendmachine
clusternode01   LINUX      MPICH
clusternode02   LINUX      MPICH
clusternode03   WINDOWS    MPIPRO
clusternode04   WINDOWS    MPIPRO
clusternode05   WINDOWS    MPIPRO
clusternode06   WINDOWS    MPIPRO
```

É gerada, pelo sistema, a tabela de processos a seguir.

Hostname	Sistema Operacional	Biblioteca MPI	Rank Local	Rank Global
node01	Linux	LAM	0	0
node02	Linux	LAM	1	1
node03	Linux	LAM	2	2
node04	Linux	LAM	3	3
node05	Windows	MPICH	0	4
node06	Windows	MPICH	1	5
clusternode01	Linux	MPICH	0	6
clusternode02	Linux	MPICH	1	7
clusternode01	Windows	MPIPRO	0	8
clusternode02	Windows	MPIPRO	1	9
clusternode01	Windows	MPIPRO	2	10
clusternode02	Windows	MPIPRO	3	11

Uma limitação atual do HyMPI é a existência apenas do comunicador global, ou seja, apenas o comunicador HYMPI_COMM_WORLD está disponível. Desse modo, todos os processos participantes farão parte deste único comunicador.

3.5 Disparo e Execução de uma aplicação HyMPI

No atual estágio de desenvolvimento do projeto, o disparo das aplicações HyMPI é feito de modo manual. Entretanto um módulo de disparo *HyMPIrun*, em desenvolvimento, encarrega-se de disparar automaticamente as aplicações.

O módulo de execução *HyMPIrun* opera utilizando o protocolo *Secure Socket Layer* (SSL) [37] e utiliza a aplicação *Secure Shell* (SSH) [38] para disparar a execução remota. HyMPI utiliza uma credencial definida como um par (login,senha), enviada para cada sistema distinto, habilitando a execução da aplicação. HyMPI monta uma *string* contendo o comando especificado para cada um dos sistemas, de acordo com o sistema operacional e implementação MPI correspondentes, considerando as particularidades de cada um.

Caso seja necessário disparar processos internos a um cluster a partir de um nó externo, a mensagem com a credencial deve passar, obrigatoriamente, pelo processo *Gateway*, que a interpreta, constrói o comando necessário para cada disparo e executa a aplicação, conforme ilustrado na Figura 29.

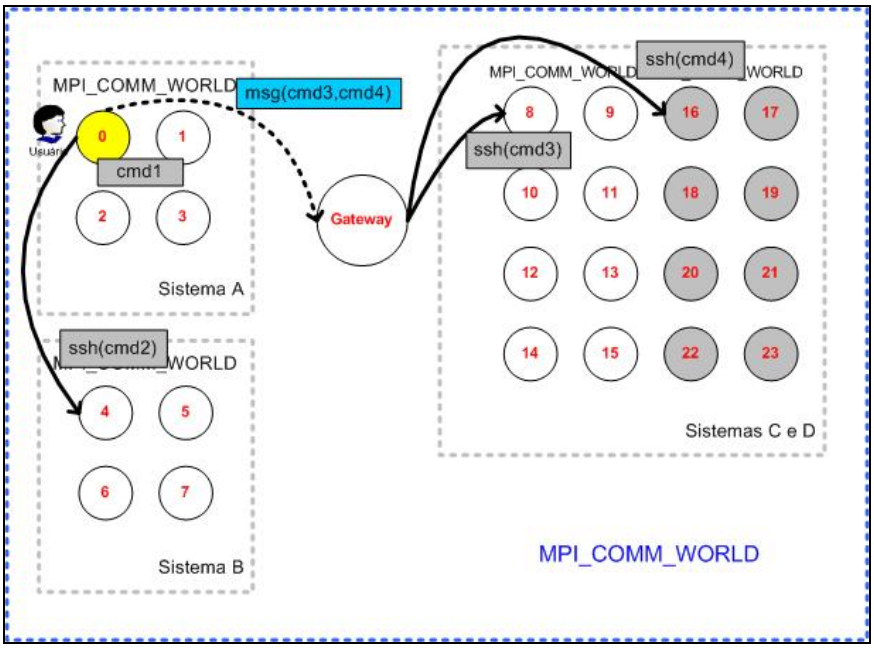


Figura 29 - Disparo automático de uma aplicação HyMPI

Como pode ser observado, o usuário pode estar conectado ao nó SMP (Sistema A). A partir desse nó, é executado o comando HyMPIrun, que interpreta o arquivo de configuração *conf.global* e inicia o disparo da aplicação. Os pares (login, senha) são armazenados em um arquivo binário e criptografado. Esse arquivo é lido durante o disparo para que as senhas sejam enviadas aos respectivos sistemas. Em seguida, o comando HyMPIrun constrói 4 comandos distintos (cmd1, cmd2, cmd3, cmd4), responsáveis pelo disparo das aplicações nos sistemas A, B, C e D respectivamente. O comando *cmd1* é executado localmente e o comando *cmd2* é executado via *ssh* (ambos indicados por linhas contínuas) no nó correspondente ao

Sistema B. Os comandos *cmd3* e *cmd4* são enviados ao *Gateway* (representado por uma linha tracejada), que os interpreta e realiza os disparos (via *ssh*) nos sistemas C e D. A única restrição é que os binários estejam previamente localizados em pastas visíveis a todos os nós que compõem os sistemas. Além disso, é necessário que cada diferente sistema possua um serviço de *Secure Shell* instalado em cada nó *front-end* (nos casos em que o *cluster* é organizado desta maneira) ou em cada nó independente (caso a organização não conte com nós *front-end*).

3.6 Considerações Finais

HyMPI oferece um conjunto de primitivas MPI, englobando suas principais funcionalidades, tanto para comunicação ponto a ponto, quanto comunicação coletiva. Entretanto no atual estágio de desenvolvimento, algumas funcionalidades não estão disponíveis, tais como:

- Criação e manipulação de comunicadores: MPI_COMM_Create e MPI_COMM_Free;
- Envio com Bufferização: MPI_Bsend;
- Envio Síncrono: MPI_Ssend
- Teste de Envio ou Recebimento: MPI_Test e MPI_Probe;
- Espera para completar envio ou recebimento: MPI_Wait;
- Outras primitivas coletivas: MPI_Allgather, MPI_Alltoall, MPI_Allreduce, entre outras;

Em função da forma que HyMPI foi desenvolvido, muitas características necessárias para a realização destas operações já encontram-se disponíveis. Isto significa

que, apesar de não estarem presentes atualmente, HyMPI suporta a expansão do número de funcionalidades, englobando estas e muitas outras primitivas presentes no padrão MPI. O desenvolvimento de mais funcionalidades motiva a continuidade deste projeto, oferecendo, gradativamente mais opções e recursos para o desenvolvimento de aplicações em ambientes distribuídos híbridos.

Capítulo 4

Testes e Discussões

Este capítulo mostra testes realizados em ambientes heterogêneos comparados com ambientes homogêneos, dentro de uma mesma sub-rede (cluster heterogêneo), e também em ambientes “clusters de clusters” (onde um cluster é considerado um nó do sistema). Tais testes visam medir tanto os tempos de execução global de uma aplicação (incluindo transmissão de mensagens e execução da lógica da aplicação), quanto comunicação no envio e recebimento de mensagens, tanto via MPI, quanto TCP/IP.

Os ambientes escolhidos envolvem, principalmente, sistemas Windows e Linux, visando uma maior heterogeneidade entre sistemas operacionais para um mesmo cluster e também para clusters de clusters.

Diversas versões de algoritmos foram desenvolvidas, visando testar o comportamento de HyMPI na comunicação ponto a ponto, quanto em comunicação coletiva, comparando seu desempenho com ambientes homogêneos somente com MPI.

4.1 Aplicações Utilizadas

Para realizar os testes, algumas aplicações foram escolhidas, em função de suas características de execução e comunicação. Dentre elas, o multiplicação de matrizes com comunicação ponto a ponto, multiplicação de matrizes com comunicação coletiva e o algoritmo Heat2D[39].

Os testes envolvendo a multiplicação de matrizes permitem, pela própria característica da aplicação, medir tempos de transmissão e execução, tanto utilizando as primitivas MPI_Send e MPI_Recv, quanto utilizando as primitivas de dispersão e agrupamento, MPI_Scatter, MPI_Bcast e MPI_Gather. Como as dimensões das matrizes variam entre 800, 1600 e 2400 é possível mensurar com detalhes o tempo de envio e recebimento das mensagens, além do tempo de execução.

Tanto o algoritmo ponto a ponto quanto o algoritmo coletivo operam no modelo mestre-escravo, onde o processo mestre (geralmente de rank 0) envia parte da matriz A e toda matriz B para cada escravo, efetua a multiplicação de sua submatriz e recolhe os resultados obtidos pelos escravos.

Já o algoritmo Heat2D [39] foi escolhido pelo fato de utilizar, em seu funcionamento interno, uma combinação de primitivas ponto a ponto (MPI_Send e MPI_Recv) com primitivas coletivas (MPI_Barrier), principalmente para a sincronização entre os processos participantes. Além disso, a comunicação entre escravos acontece, o que torna interessante medir tempos de transmissão entre ambientes homogêneos e heterogêneos, uma vez que pode haver escravos executando em sistemas operacionais diferentes.

Este algoritmo foi desenvolvido para efetuar o cálculo de dispersão de calor em uma lâmina metálica. Inicialmente assume-se que a temperatura da lâmina é alta no meio e de valor zero nas bordas. A lâmina (representada pela matriz *U*) é dividida em frações de mesma área pelo processo mestre e a temperatura de cada uma dessas frações é calculada por um processo escravo. Entretanto, para que o escravo tenha condições de calcular a temperatura de sua lâmina, ele necessita de informações de temperatura das

bordas das lâminas calculadas pelos escravos “vizinhos”. Neste caso, a sincronização e troca de mensagens entre os diversos escravos são fundamentais. A cada iteração (representando um novo instante de tempo), o cálculo da temperatura da fração da lâmina baseia-se no valor da iteração anterior do próprio escravo, além das informações obtidas dos escravos “vizinhos”.

4.2 Testes em Clusters Homogêneos e Heterogêneos

Nesta etapa dos testes foi utilizada uma infra-estrutura com 9 nós Pentium III 933 MHz, 256Mb de Memória, 10Gb HD, rede Fast Ethernet. Um dos nós foi considerado servidor de arquivos (Linux com Samba, NIS e NFS instalados) e máquina front-end do cluster. Os demais tiveram instalados, em modo Dual Boot, os Sistemas Operacionais Windows 2003 .net Server e Fedora Core 2. No sistema Windows foi utilizada a biblioteca NT-MPICH 1.25 e no sistema Fedora a biblioteca MPICH-1.2.5, como ilustra o esquema da Figura 30.

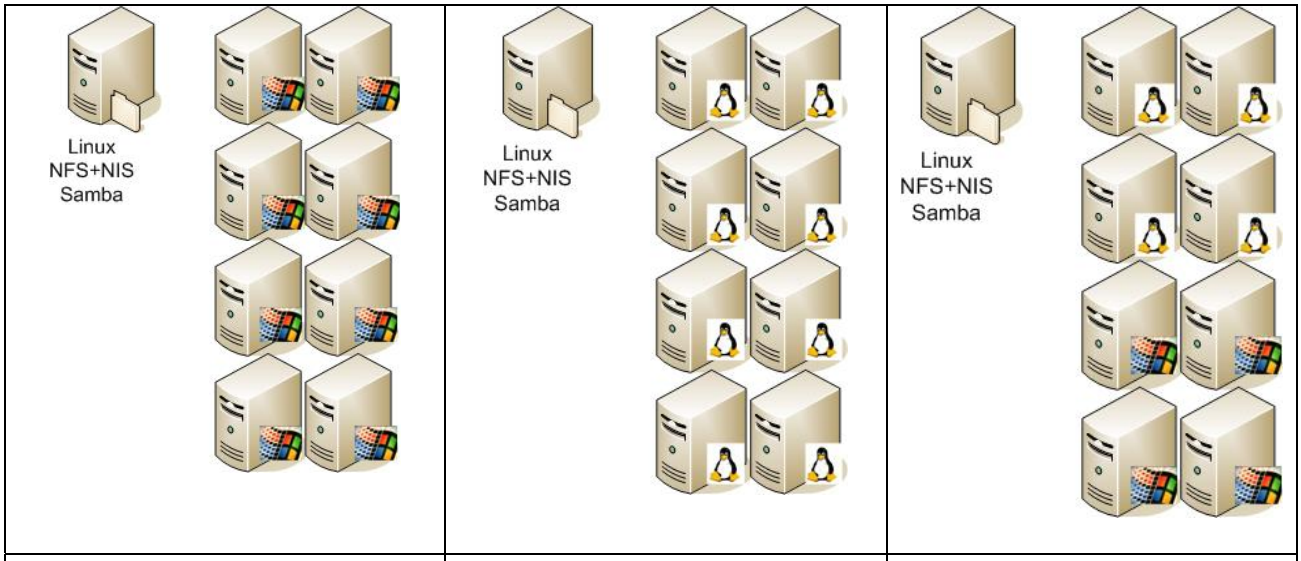


Figura 30 - Ambientes de Testes (a) Homogêneo – Windows, (b) Homogêneo – Linux e (c) Heterogêneo

4.2.1 Multiplicação de Matrizes com Comunicação Ponto a Ponto

O primeiro algoritmo testado foi o de Multiplicação de Matrizes Paralelo e Distribuído, considerando 4 e 8 processadores em modo homogêneo (somente Linux com MPICH e/ou somente Windows com NT-MPICH) e em modo heterogêneo (no caso de 4 processadores, com 2 Windows e 2 Linux e, para 8 processadores, 4 Windows e 4 Linux). Os tempos de execução para 4 processadores podem ser visualizados, respectivamente na Tabela 1 e na Tabela 2. A coluna DV indica o Desvio Padrão das medições.

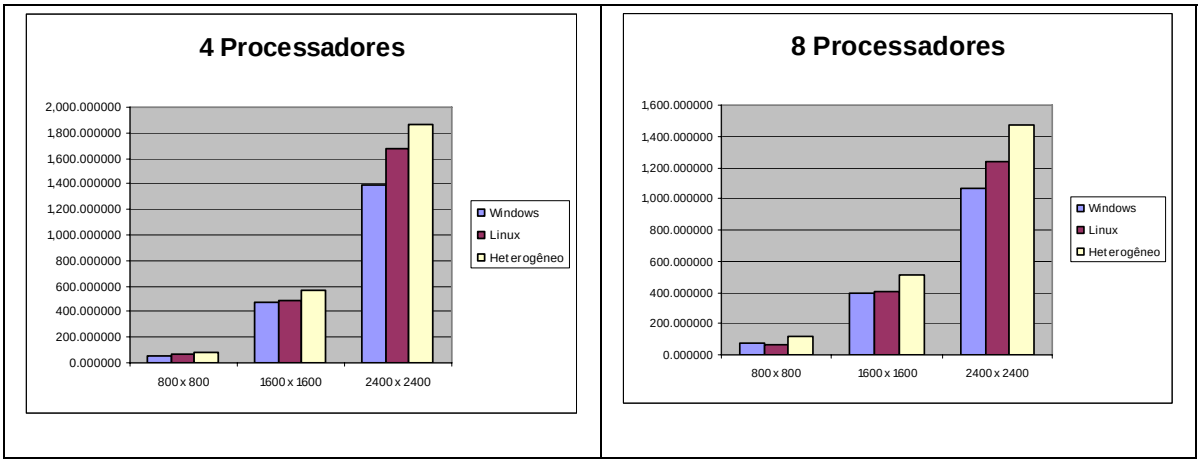
Dimensão	Windows	DV	Linux	DV	Heterogêneo	DV
800 x 800	60,735	0,565	61,505	0,821	80,365771	1,197
1600 x 1600	467,371	2,014	490,216	1,962	568,998611	2,361
2400 x 2400	1.391,332	9,998	1.678,707	10,728	1.858,188793	10,173

Tabela 1 - Tempos de execução do Algoritmo de Multiplicação de matrizes para 4 processadores

Dimensão	Windows	DV	Linux	DV	Heterogêneo	DV
800 x 800	71.955	0,826	68,285	0,751	112,035	1,002
1600 x 1600	399,302	2,175	404,665	3,109	513,891	4,011
2400 x 2400	1.063,902	8,291	1.240,616	10,830	1.472,836	11,129

Tabela 2 - Tempos de execução do Algoritmo de Multiplicação de matrizes para 8 processadores

O Gráfico da ilustra os tempos de execução para 4 e 8 processadores de acordo com os valores acima mencionados.



(a) (b)

Figura 31 - Gráficos de Tempos de Execução: (a) 4 processadores (b) 8 processadores

Através da Figura 31, pode-se notar que os tempos de execução entre Windows e Linux estão bem próximos, com ligeira vantagem para o sistema Windows nos testes com dimensões 1600 e 2400 e ligeira vantagem do sistema Linux nos testes com a dimensão 800. Isso é devido às medições de tempo de comunicação e processamento separados. O sistema Linux mostrou-se mais rápido na fase de transmissão de mensagens enquanto que o Windows mostrou-se mais rápido na fase de processamento das submatrizes.

Em termos gerais, o sistema heterogêneo teve tempos de execução maiores se comparados aos homogêneos, em virtude da comunicação via TCP/IP influenciar os tempos. Notadamente a transmissão TCP/IP é mais lenta que a transmissão MPI,

resultando um desempenho inferior. Além disso, o sistema heterogêneo mostrou-se, em média 24% mais lento nos testes com 4 processadores e 35% mais lento, comparado aos testes homogêneos com 8 processadores. Esse atraso é justificado em função dos tempos de transmissão das mensagens, principalmente pelo fato da existência de mais escravos no sistema com 8 processadores, levando o processo mestre a ter que enviar mais mensagens (e consequentemente mais vezes realizar a tarefa de negociação com o escravo a receber a mensagem).

Os tempos de comunicação para 4 processadores pode ser visualizado de modo detalhado na Tabela 3.

	Matriz A			Matriz B			Matriz C		
Dimensão 800x800	Windows	Linux	Heterog	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	1,522401	1,444000	1,362893	6,488627	5,969366	5,837294	1,480199	1,530753	1,482274
Escravo 2	1,603168	1,446062	2,805430	6,218690	5,897396	11,113075	1,471039	1,446023	3,052926
Escravo 3	1,599858	1,492997	2,812044	6,239316	5,889891	11,198101	1,462631	1,420618	2,811528

Dimensão 1600x1600	Windows	Linux	Heterog	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	6,474477	5,729809	5,997914	24,772355	23,381740	23,373465	5,952250	5,781251	5,739573
Escravo 2	6,265339	5,729007	11,232781	25,096023	23,174827	45,297231	6,158651	5,755183	11,006372
Escravo 3	6,402349	5,727299	11,269214	25,147605	23,439997	44,796404	6,044491	5,874292	11,187460

Dimensão 2400x2400	Windows	Linux	Heterog	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	14,304142	13,013776	13,438812	56,673676	52,617720	52,536276	13,668155	13,221110	13,151353
Escravo 2	14,464513	13,019690	26,588446	56,465700	52,189122	108,006290	13,881171	13,143728	25,308573
Escravo 3	14,519505	13,212326	25,250005	57,122842	52,647270	101,436985	13,955202	13,139347	25,164090

Tabela 3 - Tempos de Envio das Matrizes nos ambientes com 4 processadores

Os tempos de comunicação para 8 processadores podem ser vistos com detalhes na Tabela 4.

No ambiente heterogêneo, os escravos de 1 a 3 pertencem à mesma implementação e Sistema Operacional do processo mestre (no caso Linux com MPICH). Os escravos de 4 a 7 pertencem à outra implementação (no caso Windows com NT-

MPICH). Nota-se que os tempos de comunicação em ambientes Homogêneos Windows e Linux são equivalentes, com uma pequena vantagem para comunicação em ambiente Linux. Na versão heterogênea, a comunicação entre o processo mestre e os processos escravos de 4 a 7 é em torno de 90% mais lento, se comparado ao tempo de comunicação entre o processo mestre e os escravos de 1 a 3. Isso deve-se ao fato das próprias implementações MPI utilizarem recursos específicos de cada sistema operacional (*pipes* no caso de Linux e *DCOM* no caso de Windows), o que é um fator que impossibilita a interoperabilidade entre implementações MPI. Além disso, o protocolo TCP já possui um *overhead* nativo de cabeçalhos de controle, o que torna o envio efetivo de dados menos eficiente, tornando a comunicação do sistema mais lenta.

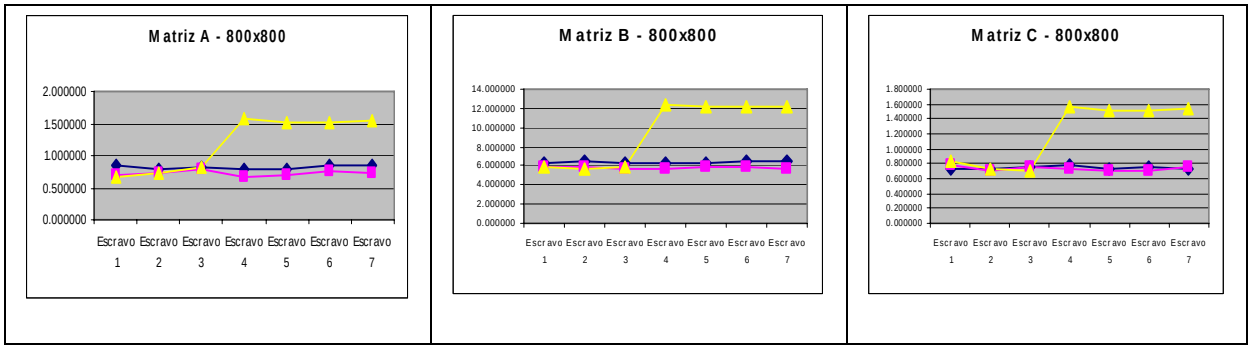
Entretanto, um fator a ser considerado é que este comportamento de comunicação manteve-se médio em todos os casos, como podem mostrar os gráficos das Figura 32, 32 e 33. Apesar de mais lento, as medidas realizadas mostram que essa demora mostrou-se constante e com a mesma média de 90% em todos os casos. A legenda utilizada é interpretada da seguinte maneira: a linha com triângulos indica os tempos de comunicação no ambiente Heterogêneo; a linha com quadrados indica o ambiente Linux e a linha com losangos indica o ambiente Windows.

	Matriz A			Matriz B			Matriz C		
Dimensão 800x800	Windows	Linux	Heterog	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	0,858962	0,691081	0,669265	6,378839	5,879585	5,954231	0,731432	0,780176	0,831188
Escravo 2	0,799111	0,721828	0,716279	6,405163	5,798918	5,778758	0,731432	0,710725	0,742994
Escravo 3	0,810118	0,775858	0,810423	6,296985	5,773448	5,862795	0,745725	0,743665	0,693445
Escravo 4	0,802860	0,673501	1,589564	6,354023	5,769376	12,420471	0,789785	0,737860	1,567902
Escravo 5	0,778069	0,695437	1,525867	6,254638	5,797637	12,261366	0,728965	0,710665	1,524408
Escravo 6	0,850290	0,746799	1,521591	6,489831	5,930762	12,181553	0,755818	0,701011	1,516890
Escravo 7	0,862953	0,715072	1,543186	6,430459	5,758008	12,275361	0,738947	0,748052	1,529165

Dimensão 1600x1600	Windows	Linux	Heterog	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	3,282348	2,882027	2,954814	25,853749	23,255175	23,300833	3,009822	2,903077	3,000261
Escravo 2	3,225860	2,864329	2,861453	25,370414	22,928160	23,035956	3,185151	2,970027	2,881879
Escravo 3	3,114020	3,005496	2,892788	25,262090	23,375469	23,203795	2,991066	2,954759	2,942667
Escravo 4	3,086713	2,798878	5,583053	24,826438	23,059875	44,464952	3,266844	2,873943	5,503186
Escravo 5	3,139265	2,862287	5,645222	25,228440	23,342870	44,786071	3,002389	2,980604	5,621221
Escravo 6	3,157476	2,971645	5,650265	25,221168	23,349738	44,828631	3,229815	2,930160	5,628135
Escravo 7	3,174133	2,997242	5,647303	25,121077	23,164529	44,880856	2,996585	3,003939	5,618897

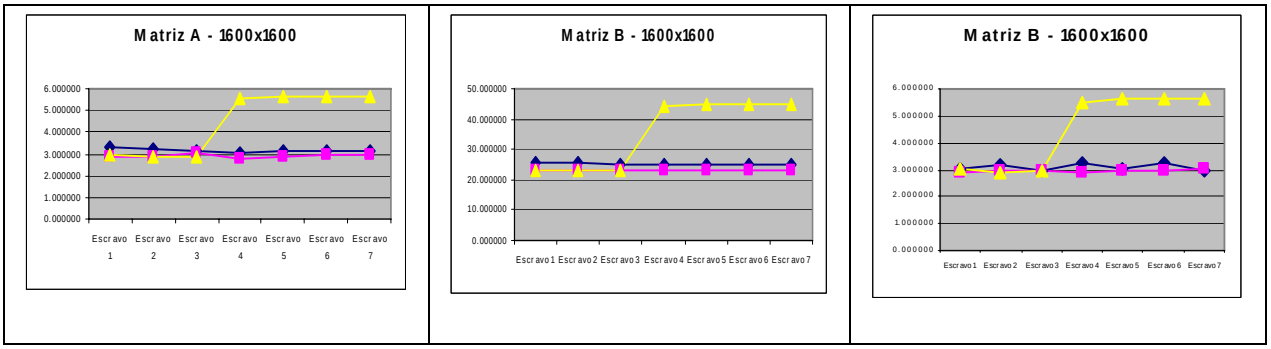
Dimensão 2400x2400	Windows	Linux	Heterog	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	6,964854	6,629554	6,560159	56,867972	52,785250	52,022790	6,808700	6,604105	6,596627
Escravo 2	7,191119	6,502303	6,491250	56,516294	51,886522	52,695707	6,904055	6,727593	6,640966
Escravo 3	7,290532	6,557919	6,608279	57,002667	53,028257	52,046382	6,877539	6,628139	6,695614
Escravo 4	7,409724	6,558020	12,779672	55,237724	52,347559	101,768229	7,100982	6,742627	13,245733
Escravo 5	7,546544	6,493711	12,655703	57,213317	52,880664	101,364219	6,935601	6,614137	13,173791
Escravo 6	7,099093	6,656057	12,698861	56,340182	52,905936	101,274910	6,901224	6,851037	12,640123
Escravo 7	7,339948	6,677571	12,616888	57,248839	52,435523	101,479730	6,934691	6,694554	12,722751

Tabela 4 - Tempos de Envio das Matrizes nos ambientes de 8 processadores



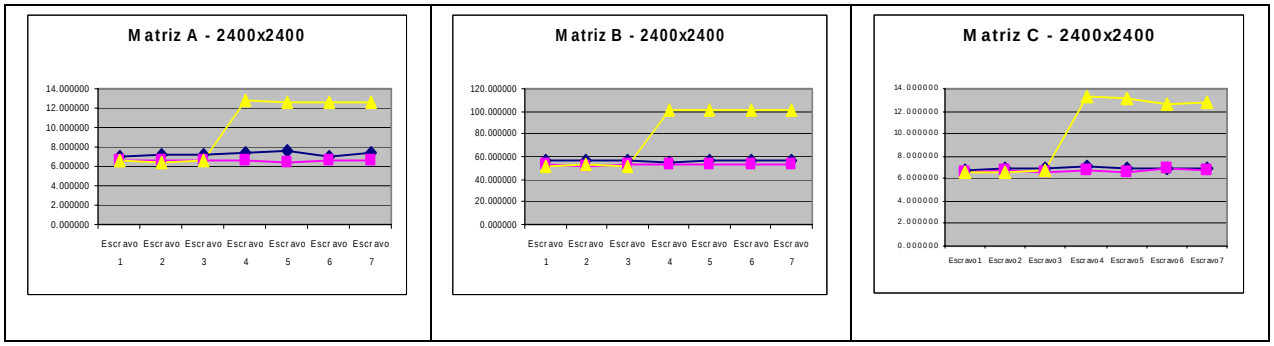
(a) (b) (c)

Figura 32 - Tempos de Comunicação para 8 processadores - Dimensão 800



(a) (b) (c)

Figura 33 - Tempos de Comunicação para 8 processadores - Dimensão 1600



(a) (b) (c)

Figura 34 - Tempos de Comunicação para 8 processadores - Dimensão 2400

4.2.2 Multiplicação de Matrizes com Comunicação Coletiva

Foram realizados testes com o mesmo ambiente de hardware, utilizando configurações homogêneas e heterogêneas de 4 e 8 processadores. As medições realizadas de execução incluem tanto o envio das Matrizes A e B, a multiplicação efetiva e a coleta dos resultados. As medições realizadas incluem as operações de Scatter, Broadcast e Gather.

Os tempos de execução podem ser visualizados a seguir.

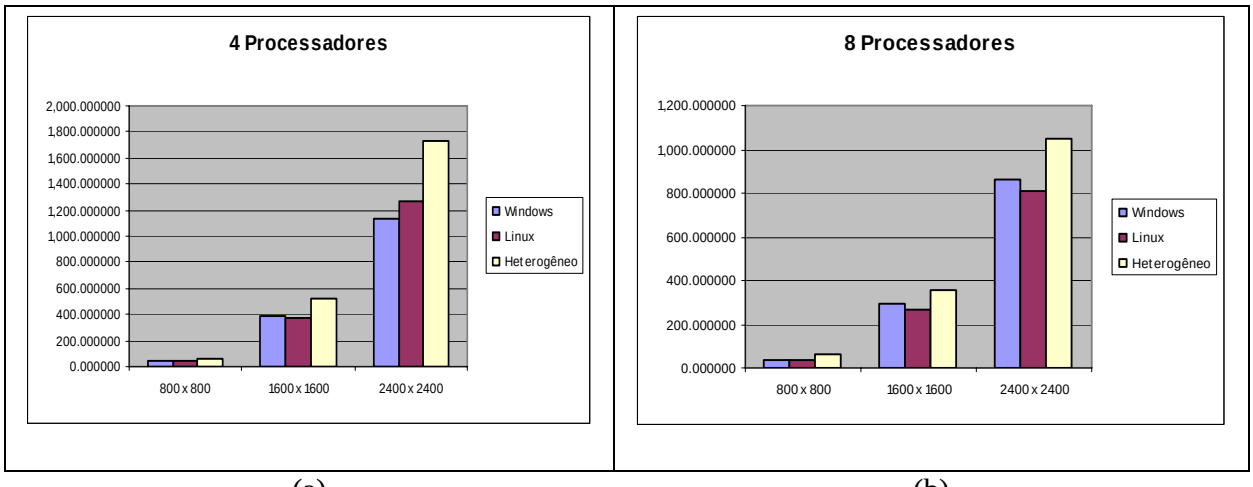
Dimensão	Windows	Linux	Heterogêneo
800 x 800	40,023072	41,220120	56,646706
1600 x 1600	389,932059	380,256739	526,904381
2400 x 2400	1.133,746818	1.261,844049	1.733,288096

Tabela 5 - Tempos de Execução para o algoritmo Coletivo para 4 processadores

Dimensão	Windows	Linux	Heterogêneo
800 x 800	39,139587	39,335273	65,546352
1600 x 1600	289,804412	268,047880	354,053017
2400 x 2400	858,926235	805,875715	1.048,584625

Tabela 6 - Tempos de Execução para o algoritmo Coletivo para 8 processadores

Em ambas as configurações, os tempos em Linux e Windows também comportaram-se de maneira equivalente, ora com pequena vantagem nas configurações Linux, ora com pequena vantagem nas configurações Windows. Entretanto, comparando-se as execuções heterogêneas, tanto com Windows quanto com Linux, obteve-se uma média de tempos de execução em torno 40% maiores (tanto em 4 quanto em 8 processadores), variando-se as dimensões, como pode ser visto pelos gráficos da Figura 35. Novamente, essa demora deve-se à fase global da comunicação, onde a transmissão do processo mestre (*raiz global*) para os processos raízes locais é feita via TCP/IP, atrasando a execução do algoritmo como um todo.



(a) (b)

Figura 35 - Tempos de Execução do Algoritmo de Multiplicação Coletiva

Os tempos de comunicação medidos com 4 e 8 processadores seguiram o mesmo comportamento dos testes da seção 4.1.1. A Tabela 7 ilustra estes tempos para as primitivas Scatter, Broadcast e Gather para 4 processadores.

	Scatter Matriz A			Broadcast Matriz B			Gather Matriz C		
Dimensão 800x800	Windows	Linux	Heterog	Windows	Linux	Heterog.	Windows	Linux	Heterog
Escravo 1	4,988737	4,667706	8,839253	12,320023	11,738404	19,807887	5,053591	4,712236	9,728464
Escravo 2	4,910652	4,721700	8,287240	11,688766	11,451468	19,149684	4,974490	4,766745	9,120920
Escravo 3	4,908721	4,730781	8,301418	11,717410	11,536468	19,243983	4,972534	4,775913	9,136524

Dimensão 1600x1600	Windows	Linux	Heterog	Windows	Linux	Heterog,	Windows	Linux	Heterog,
Escravo 1	19,095886	17,908698	35,157009	45,707953	46,846419	83,128311	19,344133	18,079547	38,693734
Escravo 2	19,145910	18,191955	32,460596	42,211754	46,337300	81,640376	19,394807	18,365506	35,726067
Escravo 3	19,505048	18,203988	32,465427	42,133115	46,676711	82,010479	19,758614	18,377654	35,731384

Dimensão 2400x2400	Windows	Linux	Heterog	Windows	Linux	Heterog,	Windows	Linux	Heterog,
Escravo 1	41,732404	40,674204	74,032381	108,214963	105,490836	175,669423	42,274925	41,062236	81,479890
Escravo 2	41,585752	40,709425	73,105290	106,891504	104,849407	172,233035	42,126367	41,097793	80,459536
Escravo 3	41,400401	40,732223	73,105290	107,105410	105,588385	173,032858	41,938606	41,120808	80,459536

Tabela 7 - Tempos de Comunicação Scatter, Bcast e Gather para 4 processadores

Como nos testes ponto a ponto, os Escravos de 1 a 3 indicam processos executando com a mesma Implementação MPI e Sistema Operacional do processo Mestre (Linux). Os Escravos de 4 a 7 indicam processos executando em uma implementação MPI e Sistema Operacional diferente do processo Mestre (Windows). Os tempos de comunicação para 8 processadores pode ser visto na Tabela 8.

	Scatter Matriz A			Broadcast Matriz B			Gather Matriz C		
Dimensão 800x800	Windows	Linux	Heterog	Windows	Linux	Heterog.	Windows	Linux	Heterog.
Escravo 1	5,647547	5,229181	9,869539	18,837094	18,106022	25,618141	5,720965	5,279067	10,862395
Escravo 2	5,577331	5,274464	9,746831	17,765105	17,288470	24,807954	5,649836	5,324782	10,727342
Escravo 3	5,594253	5,280812	9,776403	17,781217	17,304765	24,835754	5,666978	5,331191	10,759890
Escravo 4	5,640181	5,340362	9,856666	17,303742	17,972956	24,699158	5,713503	5,391309	10,848227
Escravo 5	5,637217	5,342549	9,851486	17,332446	17,987841	24,791595	5,710501	5,393517	10,842526
Escravo 6	5,702769	5,370494	9,966044	17,287105	17,882747	24,688987	5,776905	5,421729	10,968608
Escravo 7	5,700438	5,377332	9,961970	17,297640	17,881977	24,784915	5,774544	5,428632	10,964124

Dimensão 1600x1600	Windows	Linux	Heterog	Windows	Linux	Heterog,	Windows	Linux	Heterog,
Escravo 1	22,778949	21,017505	39,808030	72,354783	69,907650	100,120408	23,075075	21,218012	43,812638
Escravo 2	22,440700	21,046122	39,216913	68,250309	69,723754	99,558572	22,732429	21,246902	43,162056
Escravo 3	22,417246	21,063949	39,175925	68,291198	72,530257	99,593528	22,708670	21,264899	43,116945
Escravo 4	21,600901	21,160940	37,749297	66,451085	72,188949	99,806634	21,881713	21,362815	41,546801
Escravo 5	21,566976	21,167008	37,690010	66,517096	72,243264	100,147910	21,847347	21,368941	41,481550
Escravo 6	21,669466	21,197758	37,869120	66,716850	72,219734	99,835344	21,951169	21,399985	41,678677
Escravo 7	21,629454	21,217477	37,799195	66,775044	72,299072	100,170162	21,910637	21,419892	41,601719

Dimensão 2400x2400	Windows	Linux	Heterog	Windows	Linux	Heterog,	Windows	Linux	Heterog,
Escravo 1	51,032690	47,144634	82,389007	158,457604	150,101265	230,840080	51,696115	47,594394	90,677176
Escravo 2	50,295905	47,281613	81,591228	162,719125	150,519445	225,510889	50,949752	47,732680	89,799142
Escravo 3	50,209484	47,288478	81,505908	158,343547	151,925641	225,510889	50,862207	47,739610	89,705239
Escravo 4	49,227034	47,459168	79,293384	158,270913	153,292446	223,336507	49,866985	47,911928	87,270140
Escravo 5	49,120894	47,456022	79,291527	158,398799	155,794007	224,123054	49,759466	47,908752	87,268096
Escravo 6	48,729857	47,676182	79,160947	158,008564	155,982943	223,882886	49,363345	48,131013	87,124380
Escravo 7	48,622053	47,681490	79,176092	158,124624	155,992864	224,643661	49,254140	48,136371	87,141049

Tabela 8 - Tempos de Comunicação Scatter, Bcast e Gather para 8 processadores

Nos resultados obtidos com a operação de Scatter da Matriz A, tanto para 4 quanto para 8 processadores, houve um desempenho heterogêneo superior ao homogêneo em torno de 75%. Isso se deve ao envio, através de TCP de porções da Matriz A pelo processo mestre ao outro processo raiz local, caracterizando a fase global. Após isso, o algoritmo Scatter de cada implementação é invocado, caracterizando a fase local. Deste modo, o envio via TCP é significativo, porém em todos os escravos, percebem-se tempos muito próximos, em função de haver apenas duas implementações

MPI (MPICH e NT-MPICH) executando, respectivamente, em dois sistemas operacionais distintos (Windows e Fedora).

A operação de Broadcast da Matriz B também se mostrou equivalente à operação de Scatter, mesmo porque houve a necessidade da transmissão de toda a Matriz B do processo Mestre para o processo raiz local da outra implementação MPI. Neste caso os tempos de comunicação no teste heterogêneo foram superiores em torno de 50%, bastante próximo da operação de Scatter anteriormente realizada. Entretanto esta redução do tempo de envio pode ser atribuída ao tamanho da matriz B, uma vez que a operação Broadcast é relativamente custosa para os nós de mesma implementação MPI, o que resulta em um menor impacto da transmissão TCP comparado à matrizes de dimensões inferiores.

O Scatter da Matriz C nos testes realizados obteve resultados um pouco mais lentos na versão heterogênea (cerca de 90%), comparado ao seu desempenho em configurações homogêneas. Neste caso, a transmissão TCP teve um impacto mais significativo no cômputo geral das medições de tempo.

4.1.3 – Algoritmo Heat2D

Nos testes realizados, tanto os tempos totais de execução do mestre e dos escravos, quanto o tempo de comunicação para troca de mensagens mestre-escravo e escravo-escravo foram coletados. Desse modo, é possível analisar de modo preciso o comportamento das diferentes versões (homogêneas com Windows e Linux e heterogêneas).

Foram realizados testes com 4 e 8 processadores, em ambientes homogêneos (somente com Linux executando MPICH-1.2.5 e somente com Windows executando NT-MPICH) e heterogêneo (mesclando ambos os sistemas operacionais). A Tabela 9 ilustra os tempos de execução de cada versão do algoritmo, para 4 processadores, indicando os tempos de execução do processo mestre e dos processos escravos em ambos os ambientes. A Tabela 10 ilustra os tempos de execução para 8 processadores.

	Windows	Linux	Heterogêneo
Mestre	577,974652	623,133745	637,829664
Escravo 1	187,038660	99,919131	239,834069
Escravo 2	177,016478	90,406824	218,589304
Escravo 3	166,572844	80,954156	198,691228

Tabela 9 - Tempos de Execução (Mestre e Escravo) do Algoritmo Heat2D para 4 processadores

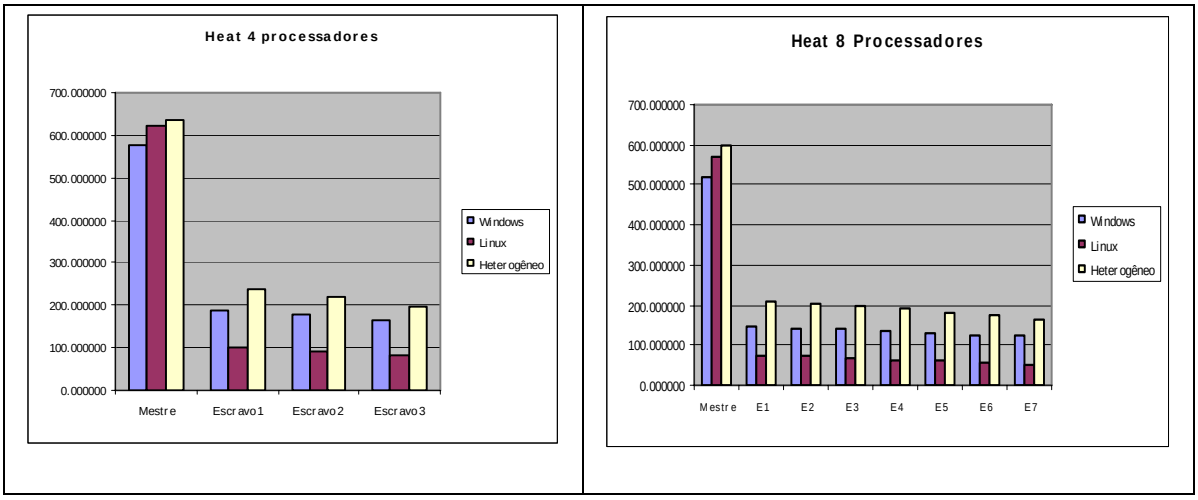
	Windows	Linux	Heterogêneo
Tempo Mestre	519,083170	572,664121	597,946125
Escravo 1	148,259054	75,954992	208,260333
Escravo 2	143,840408	71,945285	203,716423
Escravo 3	139,486384	68,135423	199,241201
Escravo 4	134,978218	64,135745	190,085438
Escravo 5	130,370587	60,189514	181,536219
Escravo 6	126,138908	56,071139	173,058776
Escravo 7	121,692391	51,868603	164,579362

Tabela 10 - Tempos de Execução (Mestre e Escravo) do Algoritmo Heat2D para 8 processadores

Podem-se notar em ambos os testes, que os tempos de execução do processo mestre no ambiente Windows são inferiores ao ambiente Linux, entretanto seu comportamento é alterado nos escravos, onde a execução Linux mostrou-se mais rápida que a execução em Windows, mesmo utilizando algoritmos com graus de otimização idênticos (O2).

Os tempos no ambiente heterogêneo mostraram-se superiores, muito em função da comunicação mestre-escravo e escravo-escravo via TCP, que a torna mais

lenta, se comparada com MPI. A Figura 36 ilustra os gráficos destes tempos de execução.



(a) (b)

Figura 36 - Tempos de execução para o Algoritmo Heat em 4 (a) e 8 (b) processadores

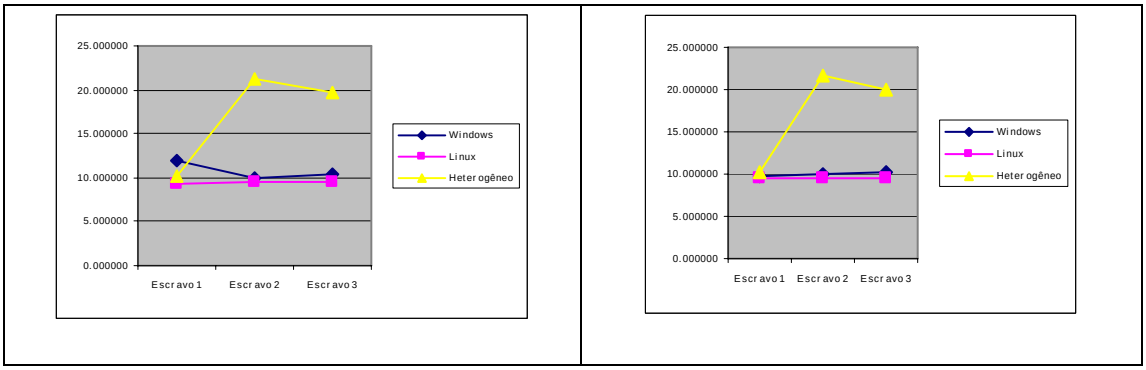
Os tempos de comunicação também foram analisados e medidos, para comprovar se a média de tempo de transmissão via TCP em relação a MPI mantinha-se, como nos testes envolvendo o algoritmo de Multiplicação de Matrizes na versão ponto a ponto.

As medições dos tempos de comunicação levaram em conta a transmissão da Matriz U do processo mestre para os processos escravos, por ser a parte mais significativa dos dados (e consequentemente levaria mais tempo para transmitir), permitindo uma medição mais precisa. A Tabela 11 ilustra os tempos de transmissão para 4 processadores.

	Envio			Recebimento		
Matriz U	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	11,981636	9,257357	10,179764	9,760485	9,552648	10,216189
Escravo 2	10,022175	9,513610	21,309703	9,971527	9,530456	21,622500
Escravo 3	10,389076	9,452536	19,774984	10,255133	9,617501	20,091594

Tabela 11 - Tempos de Transmissão da Matriz U do processo Mestre para os Escravos

Nestes testes, o processo Escravo 1 executa em uma máquina de mesmo Sistema Operacional e biblioteca MPI do processo Mestre (neste caso Linux com MPICH) e os processos 2 e 3 executam em máquinas com SO e bibliotecas MPI diferentes do processo mestre (Windows e NT-MPICH). Os tempos de transmissão nas configurações heterogêneas mantiveram uma média de 100% mais lenta em relação à transmissão homogênea, tanto no envio, quanto no recebimento, como pode ser visto nos gráficos da Figura 37 para Envio (a) e Recebimento (b). Tanto o envio, quanto o recebimento da Matriz U é feito através de primitivas ponto a ponto, estando, o que é confirmado pela demora na transmissão dos escravos 2 e 3, onde a comunicação é através de TCP/IP.



(a) (b)

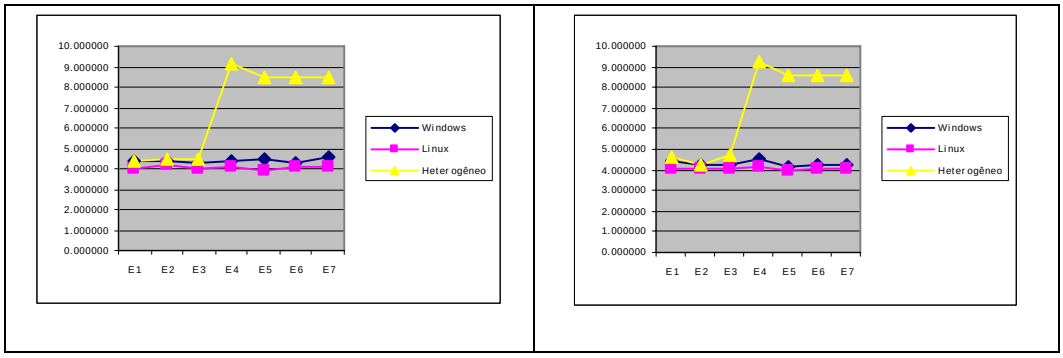
Figura 37 - Tempos de Transmissão da Matriz U para 4 processadores: (a) Envio e (b) Recebimento

A comunicação, agora com 8 processadores, também mostra a mesma proporção de tempo dos ambientes heterogêneos com relação ao ambiente homogêneo, como indicam os dados da Tabela 12. A configuração do ambiente seguiu o mesmo padrão do teste com 4 processadores, onde os processos de *rank* 1 a 3 executam na mesma configuração do processo mestre e os processos 4 a 7 em uma configuração diferente (tanto em relação ao Sistema Operacional, quanto implementação MPI).

	Envio			Recebimento		
Matriz U	Windows	Linux	Heterogêneo	Windows	Linux	Heterogêneo
Escravo 1	4,423566	4,045046	4,401323	4,476147	4,059288	4,627627
Escravo 2	4,395491	4,190375	4,508945	4,261181	4,051368	4,228608
Escravo 3	4,306755	3,975244	4,447255	4,279882	4,061581	4,693946
Escravo 4	4,351096	4,050197	9,171990	4,570572	4,149188	9,268202
Escravo 5	4,507598	3,906763	8,450423	4,183786	3,985490	8,602887
Escravo 6	4,320752	4,063920	8,442326	4,230568	4,103382	8,580965
Escravo 7	4,542750	4,138513	8,511027	4,251287	4,060468	8,575611

Tabela 12 - Tempos de Transmissão da Matriz U para 8 processadores

Tanto o envio quanto o recebimento da matriz *U* por parte do processo mestre para os escravos sofreu um acréscimo de aproximadamente 100% do tempo, se comparadas as configurações heterogêneas em relação as configurações homogêneas. Novamente a transmissão TCP/IP determina o tempo de transmissão mais lento, influenciando a execução do algoritmo como um todo. Os gráficos das Figuras a seguir ilustram estes tempos de transmissão.



(a) (b)

Figura 38 - Tempos de Transmissão da Matriz U para 8 processadores: (a) envio e (b) recebimento

4.2 Testes em Clusters de Clusters

Nesta seção serão apresentados testes em ambientes envolvendo clusters de computadores interligados entre si, de modo que um cluster seja considerado um nó do

sistema. Uma configuração física de 4 *workstations* somada a um cluster composto de 4 outros nós foi utilizada, conforme ilustra a Figura 39.

Ambiente A é composto de 4 nós de configuração AMD Duron 1.8 GHz, 256 Mb de RAM, 40Gb de HD e Interface de rede Fast Ethernet. Há um nó principal (*node01*), instalado com Linux e MPICH-1.2.5, também responsável por realizar as funções de servidor de arquivos e usuários; os demais nós, com os sistemas Operacionais Linux (com MPICH 1.2.5) e Windows 2003 .net Server (com NT-MPICH) em modo *dual-boot*, dedicados exclusivamente para processamento. Especificamente o *node02* possui 2 interfaces de rede, uma vez que este nó realiza a função de *gateway* entre as duas redes distintas.

O Ambiente B, por sua vez é composto de 4 nós de configuração Intel Celeron D 2.8GHz, com 256Mb de RAM, 40 Gb de HD e interface Fast Ethernet, onde também existe um nó responsável pelas tarefas de servidor de arquivos e usuários (*machine1*) possui o Sistema Operacional Linux e biblioteca MPICH-1.2.5. Os demais nós, assim como o Ambiente A, são dedicados para processamento e possuem instalados os sistemas Windows XP (com NT-MPICH) e Linux (com MPICH-1.2.5) em modo *dual-boot*.

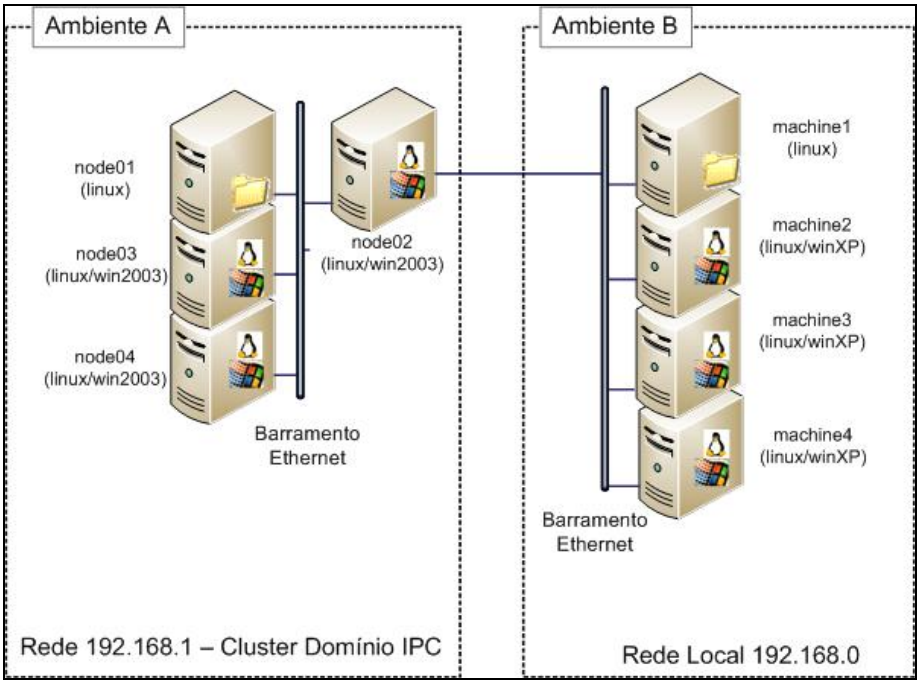


Figura 39 - Ambiente heterogêneo composto de um Cluster e uma Rede Local

Duas diferentes configurações foram testadas de modo a mensurar os tempos de transmissão de mensagens entre os diversos processos, descritas da seguinte forma:

a) A “Configuração 1” conta com os Ambientes A e B executando de modo homogêneo, ou seja, todos os nós executando o mesmo sistema operacional (neste caso Linux), com o nó servidor também participando do processamento, entretanto, em redes distintas (obrigando a comunicação pelo processo Gateway).

b) A “Configuração 2” conta com os ambientes A e B executando em modo heterogêneo, ou seja, dos 4 nós existentes em cada 2 deles executam o Sistema Operacional Windows e os outros 2, o sistema Linux.

As configurações lógicas, juntamente com as respectivas nomeações dos processos participantes das diferentes implementações MPI pode ser visto na Figura 40.

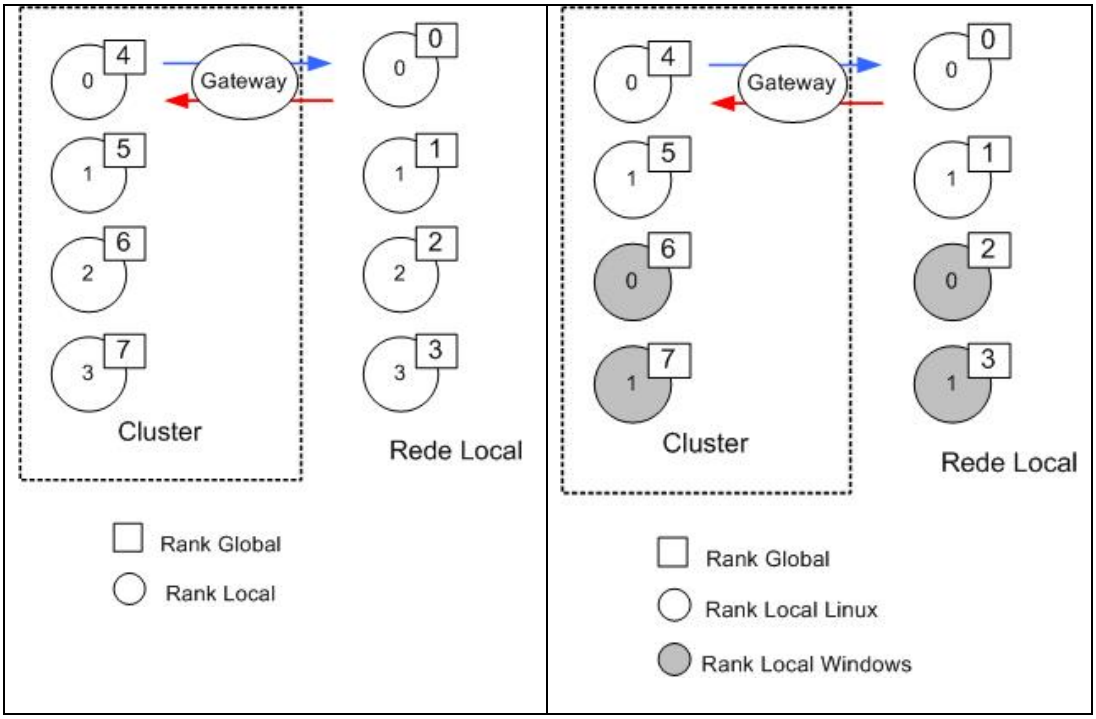


Figura 40 - Representação das configurações lógicas

Duas diferentes versões do algoritmo de multiplicação de matrizes (ponto a ponto e coletiva) foram utilizadas para este teste, cada uma executando em versões diferentes do ambiente (A e B), totalizando quatro diferentes configurações de teste, como seguem:

- Teste A (MM-LGL): Multiplicação Ponto a Ponto, utilizando a Configuração 1.
- Teste B (MM-HGH): Multiplicação Ponto a Ponto, utilizando a Configuração 2.
- Teste C (MB-LGL): Multiplicação Coletiva, utilizando a Configuração 1.

- Teste D (MB-HGH): Multiplicação Coletiva, utilizando a Configuração 2.

Para cada teste, sete diferentes execuções foram feitas e foram obtidos os tempos médios de execução e de transmissão das matrizes do processo mestre para os processos escravos. Em ambos os tipos de medições 3 parâmetros de análise foram considerados: a média simples dos valores (coluna **Média**), o desvio padrão (coluna **Desv. P**) e a média calculada (coluna **Media C**), que consiste em eliminar o maior e o menor valor obtidos, concentrando-se apenas nos valores intermediários.

Além disso, a característica do processo *Gateway* de ser apenas um encaminhador de mensagens pode levar o sistema a um *deadlock*, dependendo da natureza e *granularidade* da aplicação. Por exemplo, no caso da multiplicação ponto a ponto, o processo mestre envia uma fração da Matriz A para todos os escravos. Como a comunicação TCP medida é consideravelmente superior ao tempo de processamento da sub-matriz, enquanto o processo 0 (mestre) realiza o envio, alguns escravos já terminaram suas execuções, apenas aguardando o envio do resultado.

Dessa forma foi observado que, enquanto o processo 0 ainda tenta enviar porções da Matriz A para os processos 5, 6 e 7 (internos ao cluster e, portanto, via TCP utilizando o *gateway*), o processo 4 já havia terminado sua multiplicação e tenta fazer o envio do resultado. Com isso, o processo 4 ocupava o *gateway* para o envio de sua porção da Matriz C, enquanto que o mestre ainda não havia terminado o envio para todos os escravos e, portanto, não estava preparado para começar a receber resultados.

Portanto, o processo 4 não consegue enviar seu resultado (pois o *gateway* tenta sem sucesso uma conexão com o mestre) e o processo mestre não consegue enviar

os dados das matrizes aos escravos, pois o *gateway* está ocupado. Para resolver esse problema foi a separação das funcionalidades do processo *gateway*, tornando-o *multithreaded*. Uma *thread* é responsável pelo encaminhamento de mensagens de dentro para fora do cluster (*way out*) e a outra *thread* é responsável pelo encaminhamento de fora do cluster para dentro (*way in*).

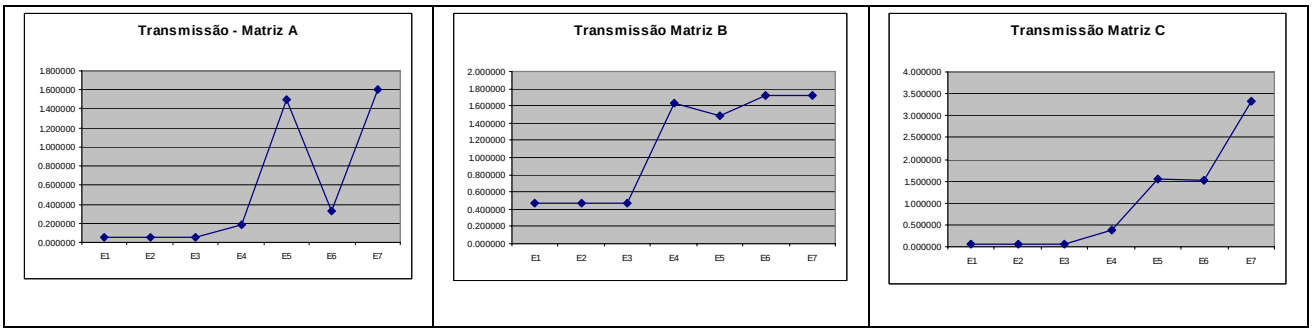
4.2.1 Teste A

Os tempos de execução do Teste A podem ser vistos a seguir na Tabela 13.

		800 x 800			1600 x 1600			2400 x 2400		
Matriz	**Escravo**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**	**Média Calc**	**Média**	**Desv. P**
A	E1	0,057126	0,057132	0,000028	0,230190	0,230218	0,000234	0,521365	0,535110	0.037108
	E2	0,058687	0,058693	0,000134	0,231670	0,231735	0,000400	0,521586	0,521553	0.000397
	E3	0,058754	0,058783	0,000384	0,231878	0,231898	0,000461	0,521425	0,521438	0.000132
	E4	0,185172	0,185283	0,000918	0,733500	0,662016	0,189733	1,653175	1,659483	0.019653
	E5	1,492403	1,492661	0,002463	5,977530	5,978149	0,007483	13,517915	13,553432	0.218141
	E6	0,334381	0,333428	0,003389	1,322621	1,322914	0,017455	3,056696	3,057140	0.039766
	E7	1,607176	1,607164	0,006185	6,236148	6,234971	0,009180	14,406640	14,451047	0.137828
B	E1	0,464894	0,464897	0,000219	1,861163	1,861094	0,000315	4,186582	4,186608	0.000877
	E2	0,465017	0,465059	0,000272	1,860969	1,860972	0,000312	4,186880	4,186997	0.000653
	E3	0,465063	0,464966	0,000423	1,861264	1,861239	0,000532	4,186767	4,188644	0.005218
	E4	1,626858	1,626876	0,001847	6,512041	6,511892	0,008005	14,666299	14,709403	0.135130
	E5	1,479890	1,479731	0,004036	5,930957	5,930034	0,011054	13,360620	13,389085	0.092321
	E6	1,725914	1,726181	0,002144	6,729786	6,730241	0,005283	15,535608	15,536573	0.022918
	E7	1,725147	1,725995	0,003562	6,726283	6,726196	0,003267	15,536896	15,535493	0.021407
C	E1	0,060321	0,060318	0,000014	0,238952	0,238975	0,000109	0,560329	0,582484	0.079065
	E2	0,060183	0,060185	0,000015	0,238497	0,238501	0,000039	0,550565	0,576248	0.075618
	E3	0,060138	0,060136	0,000165	0,238560	0,238565	0,000041	0,543111	0,564627	0.060180
	E4	0,377551	0,377623	0,000976	1,417241	1,417526	0,002089	4,341208	4,947500	2.317852
	E5	1,539352	1,526264	0,182818	0,835299	0,835756	0,003009	38,431089	38,335424	1.172005
	E6	1,528656	1,418767	0,372713	1,509244	1,547764	0,219005	5,428798	8,634378	9.512545
	E7	3,334428	3,394813	0,167149	7,894772	8,334181	3,231021	25,664282	23,379740	11.365478
Tempo de Exec.		22.568712	22,431091	0,376244	124,803859	125,244293	3,373526	422,257215	428,287871	52,334208

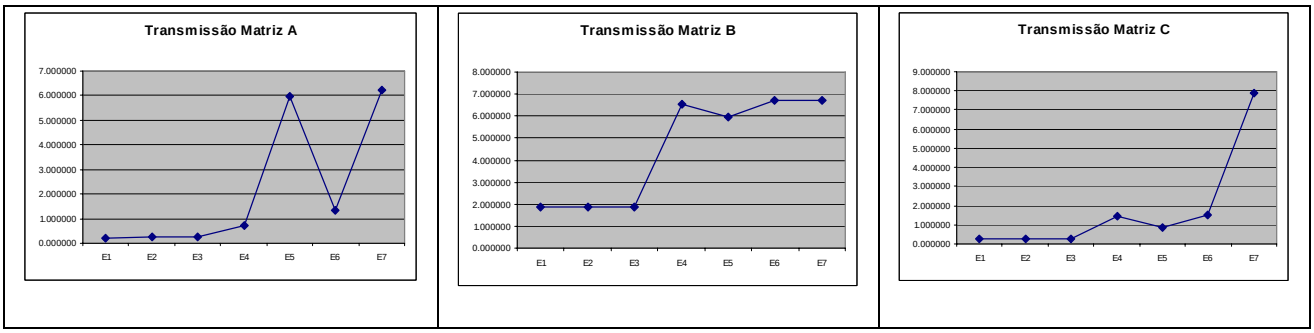
Tabela 13 - Tempos Médios de Execução e Transmissão para o Teste A

Os gráficos das figuras a seguir mostram o comportamento da transmissão da Matriz A (através da operação de *Scatter*), Matriz B (através da operação de *Broadcast*) e da Matriz C (através da operação de *Gather*).



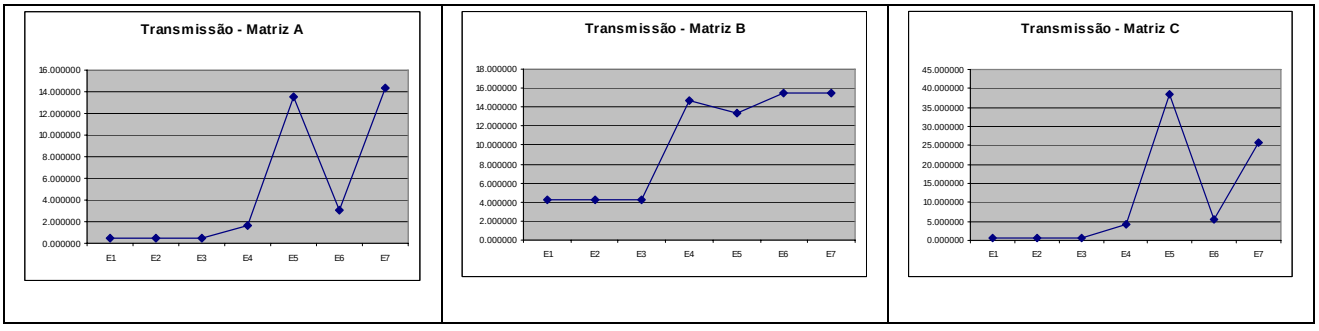
(a) (b) (c)

Figura 41 - Tempos de Transmissão das Matrizes A, B e C para a dimensão 800 no Teste A



(a) (b) (c)

Figura 42 - Tempos de Transmissão das Matrizes A, B e C para dimensão 1600 no Teste A



(a) (b) (c)

Figura 43 - Tempos de Transmissão das Matrizes A, B e C para dimensão 2400 no Teste A

Pode-se observar que a transmissão da Matriz A nos testes realizados com as três diferentes dimensões comportou-se da mesma maneira. É interessante salientar a diminuição no tempo de transmissão entre os Escravos 5 e 6, uma vez que ambos fazem parte do cluster e obrigatoriamente a mensagem deve ser transmitida pelo processo *Gateway* (*way in*).

A transmissão de qualquer mensagem pelo processo *Gateway* é feita em dois passos: o primeiro é o envio do emissor até o *Gateway* (incluindo a *bufferização* da mensagem); o segundo é o envio da mensagem armazenada no *Gateway* até o processo receptor e, conseqüente liberação do *buffer* de memória. A medição do tempo de envio nos testes envolvendo clusters de clusters leva em consideração o transporte da mensagem até o processo *Gateway*. Em função disso, a transmissão do processo *Gateway* até o seu receptor só é considerada nos envios subseqüentes.

Fisicamente o Escravo 5 está localizado na mesma máquina onde o processo *Gateway* executa. Desse modo, a transmissão do processo *Gateway* até o Escravo 5 é feita diretamente pela memória (reduzindo significativamente o tempo). Entretanto este tempo só será percebido na transmissão da mensagem para o Escravo 6, uma vez que a segunda etapa da transmissão da mensagem anterior ocorreu de forma muito mais eficiente. Essa diferença é bastante significativa uma vez que a dimensão da matriz é menor e o tempo de transmissão é afetado de forma mais brusca

A transmissão da Matriz B não sofre o mesmo fenômeno por, provavelmente, ter um tempo de transmissão maior. Porém, pode-se comprovar curvas semelhantes na transmissão da matriz B nas três diferentes dimensões.

Por fim, a transmissão da Matriz C sofre muitas alterações, principalmente pelo fato do cálculo do tempo ser levado em consideração a partir do recebimento pelo processo mestre. O tempo de execução da multiplicação nos diversos escravos tornou-se determinante para o cálculo do tempo de transmissão, pois escravos executavam em máquinas heterogêneas com diferentes capacidades (processador e memória), fazendo com que algumas delas tivessem que ficar esperando para transmitir, por não obedecerem a ordem de envio. Isto é, muitas vezes o processo mestre está pronto para receber mensagens de um processo que ainda está executando. Os demais estão prontos para enviar, porém devem esperar a ordem designada pelo mestre para iniciarem a transmissão.

4.2.2 Teste B

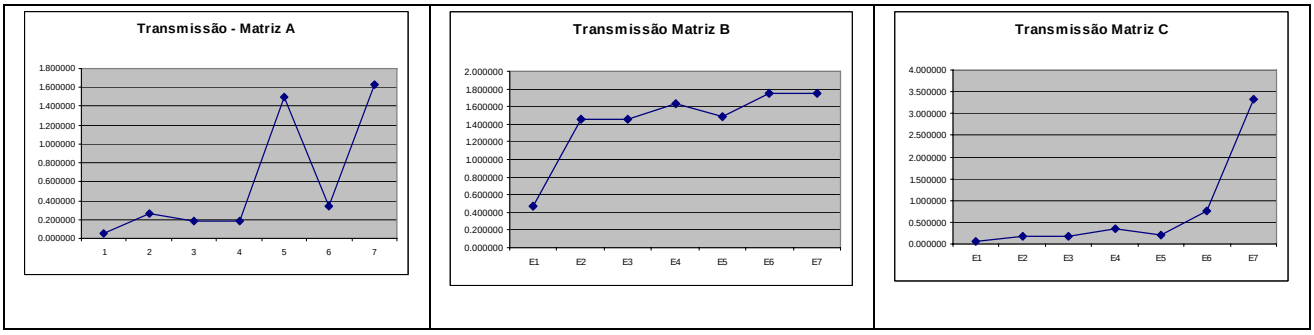
Os tempos de execução e transmissão do Teste B podem ser visualizados a seguir na Tabela 14.

Os tempos de transmissão das Matrizes A, B e C mostraram-se muito semelhantes aos tempos obtidos no Teste A, exceto nos casos de processos heterogêneos em um mesmo cluster. Ou seja, neste ambiente, os processos 0 e 1 são homogêneos (portanto comunicação via MPI) e os processos 2 e 3 são heterogêneos, porém fazem parte do mesmo cluster (comunicação via TCP/IP). No caso dos processos de 4 a 7, toda a comunicação é feita através de TCP/IP, por serem internos ao cluster, obtendo valores bastante próximos dos obtidos no teste anterior.

		800 x 800			1600 x 1600			2400 x 2400		
Matriz	**Escravo**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**
A	E1	0,057399	0,057399	0,000015	0,231174	0,231161	0,000204	0,520267	0,534562	0,038213
	E2	0,268400	0,688217	1,161033	0,717983	0,761058	0,116635	1,625796	1,647597	0,066081
	E3	0,182414	0,182376	0,001076	0,726593	0,726461	0,003560	1,628672	1,629681	0,023314
	E4	0,185187	0,184956	0,001449	0,734920	0,735089	0,000912	1,654084	1,687754	0,090980
	E5	1,500915	1,500692	0,001650	5,987975	5,989913	0,011652	13,518755	13,752413	0,690541
	E6	0,344029	0,508541	0,437553	1,334453	1,334951	0,010840	3,058597	3,189174	0,378972
	E7	1,626006	1,627448	0,005658	6,290981	6,293635	0,011734	14,552546	14,551976	0,028196
B	E1	0,466718	0,466737	0,000386	1,868250	1,868216	0,000454	4,185478	4,310845	0,332338
	E2	1,461743	1,461777	0,002909	5,753637	5,754689	0,013250	12,980458	12,988843	0,141806
	E3	1,456958	1,456063	0,007807	5,800576	5,797376	0,025166	12,926198	12,935476	0,107731
	E4	1,635444	1,635367	0,001690	6,537572	6,541359	0,011742	14,721540	14,852279	0,415730
	E5	1,489583	1,489565	0,001388	5,948674	5,949494	0,004311	13,338185	13,364826	0,086050
	E6	1,744066	1,745700	0,006619	6,776039	6,775101	0,007926	15,543423	15,541859	0,011180
	E7	1,743230	1,744248	0,004479	6,777346	6,780118	0,015539	15,548188	15,546628	0,023651
C	E1	0,060375	0,060386	0,000049	0,239124	0,239122	0,000042	0,576374	0,688559	0,323605
	E2	0,179862	0,179816	0,000417	5,289133	5,289368	0,003660	13,282673	13,824789	3,073818
	E3	0,183613	0,183557	0,000695	0,730175	0,730067	0,002830	1,631230	1,631369	0,012463
	E4	0,356894	0,357156	0,001438	1,424445	1,424687	0,001726	3,399487	3,407878	0,036232
	E5	0,212672	0,213071	0,001465	0,841490	0,841521	0,003443	37,316539	35,598092	7,471644
	E6	0,766496	0,772569	0,084390	1,488372	1,488081	0,016183	118,548563	118,015519	1,715479
	E7	3,339295	3,336278	0,019248	5,211517	5,214020	0,019140	29,423100	29,439347	0,388808
Tempo de Exec.		24.992090	25,412014	1,175038	137,482347	137,705274	1,271983	543,591543	544,124672	1,846634

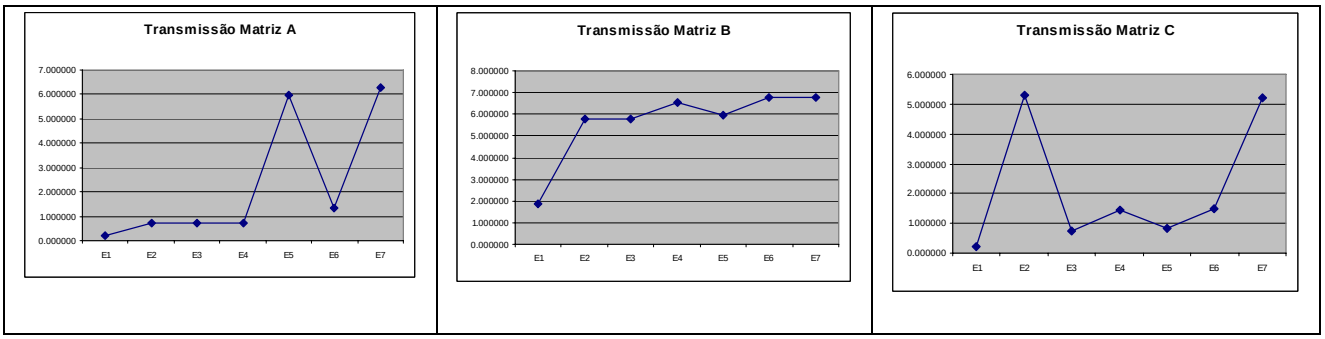
Tabela 14 - Tempos Médios de Execução e Transmissão para o Teste B

Dessa forma, comparando-se o Teste A ao Teste B, há um aumento no tempo de transmissão das mensagens para os processos 2 e 3. Os demais tempos permanecem semelhantes. Os gráficos das Figuras a seguir ilustram esse comportamento.



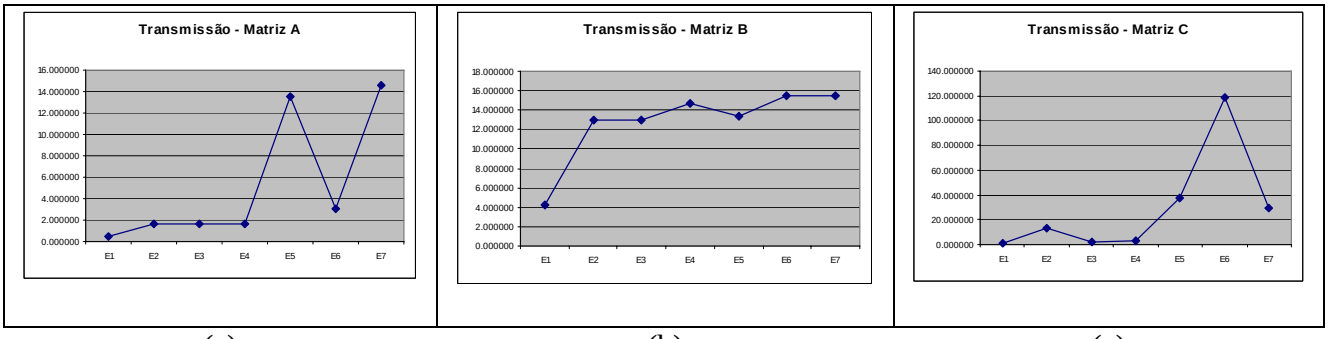
(a) (b) (c)

Figura 44 - Tempos de Transmissão das Matrizes A, B e C para a dimensão 800 no Teste B



(a) (b) (c)

Figura 45 - Tempos de Transmissão das Matrizes A, B e C para a dimensão 1600 no Teste B



(a) (b) (c)

Figura 46 - Tempos de Transmissão das Matrizes A, B e C para a dimensão 2400 no Teste B

Comparando-se os tempos de execução dos Testes A e B, pode-se perceber que existe um acréscimo no tempo total de execução (incluindo comunicação), cujo fator

principal de influência é o envio das Matrizes A, B e C entre os nós heterogêneos fora do cluster, como pode ser visualizado na Tabela 15 e no gráfico da Figura 47.

Dimensão	Teste A	Teste B
800x800	22,568712	24,992090
1600x1600	124,803859	137,482347
2400x2400	422,257215	543,591543

Tabela 15 - Tempos de Execução dos Testes A e B

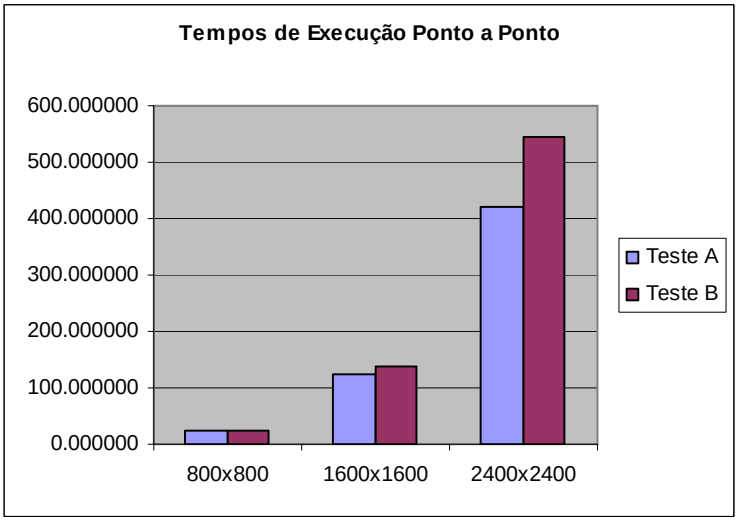


Figura 47 - Tempos de Execução dos Testes A e B

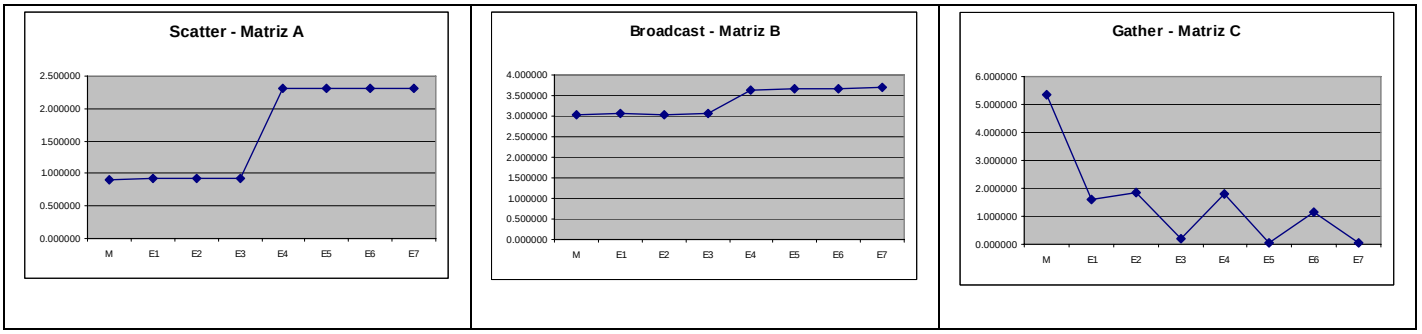
4.2.3 Teste C

Este teste contempla a comunicação entre implementações MPI e sistemas operacionais idênticos, porém, por estarem em clusters distintos, necessitam do processo *gateway* para a comunicação entre processos internos e externos ao cluster, suprindo a limitação encontrada nas diversas implementações MPI existentes. Os tempos de transmissão e execução podem ser visualizados a seguir na Tabela 16.

		800 x 800			1600 x 1600			2400 x 2400		
Matriz	**Proc**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**
A	M	0,911668	0,911613	0,000555	3,638613	3,724530	0,227887	8,187479	8,202886	0,042497
	E1	0,927590	0,927440	0,008639	3,742968	3,828805	0,233065	8,443755	8,446396	0,025034
	E2	0,933821	0,933192	0,010198	3,753533	3,839430	0,232941	8,474069	8,500013	0,083078
	E3	0,933890	0,949378	0,043886	3,753963	3,840944	0,233063	8,471929	8,497100	0,085300
	E4	2,303340	2,327245	0,195092	6,978743	6,891299	0,311502	15,171005	15,301081	0,578370
	E5	2,302593	2,326540	0,195353	6,977432	6,890260	0,310438	15,167385	15,297487	0,578340
	E6	2,307587	2,331999	0,196103	7,002443	6,914885	0,310157	15,228152	15,357848	0,578138
	E7	2,308250	2,332614	0,196161	7,002985	6,915455	0,310061	15,227809	15,357855	0,578591
B	M	3,041798	3,041728	0,001374	12,177936	12,178007	0,006614	27,415225	27,473065	0,159289
	E1	3,056007	3,055943	0,001502	12,230813	12,231437	0,004674	27,530065	27,711754	0,533058
	E2	3,048815	3,048683	0,001502	12,209619	12,210135	0,006643	27,483317	27,565666	0,224227
	E3	3,061860	3,048273	0,036830	12,262014	12,263969	0,010086	27,599447	27,731970	0,356635
	E4	3,645756	3,646112	0,002402	14,606180	14,605573	0,012579	32,873690	32,873905	0,012895
	E5	3,670791	3,669683	0,005448	14,711053	14,712727	0,013167	33,107053	33,111115	0,024542
	E6	3,662304	3,663267	0,003765	14,682562	14,701096	0,054650	33,048072	33,053212	0,031831
	E7	3,684184	3,683895	0,002270	14,780031	14,805523	0,074319	33,302501	33,357695	0,177779
C	M	5,336488	5,439589	2,150049	18,935180	18,763738	1,013470	87,327265	88,356736	3,420686
	E1	1,580995	1,677478	1,956008	0,805913	0,973443	1,046142	1,165941	1,686409	1,770196
	E2	1,874362	1,931481	1,846771	1,099083	1,156988	0,323141	1,789360	2,384688	1,669023
	E3	0,202959	0,220757	0,175672	0,229932	0,229951	0,000076	0,532080	0,581844	0,138891
	E4	1,808758	1,682105	0,351102	3,562726	4,565031	2,684906	10,567071	12,499127	5,735684
	E5	0,053940	0,053949	0,000142	0,216944	0,224796	0,020958	0,489746	0,638141	0,393018
	E6	1,145878	1,015851	0,357667	1,785879	2,482029	2,690355	4,259382	3,884168	2,321552
	E7	0,053853	0,053910	0,000398	0,254623	0,521618	0,726672	0,494953	2,894676	6,351121
Tempo de Exec		15.607012	15,608053	0,018583	99,573729	99,687441	0,341522	342,081036	343,912991	5,479242

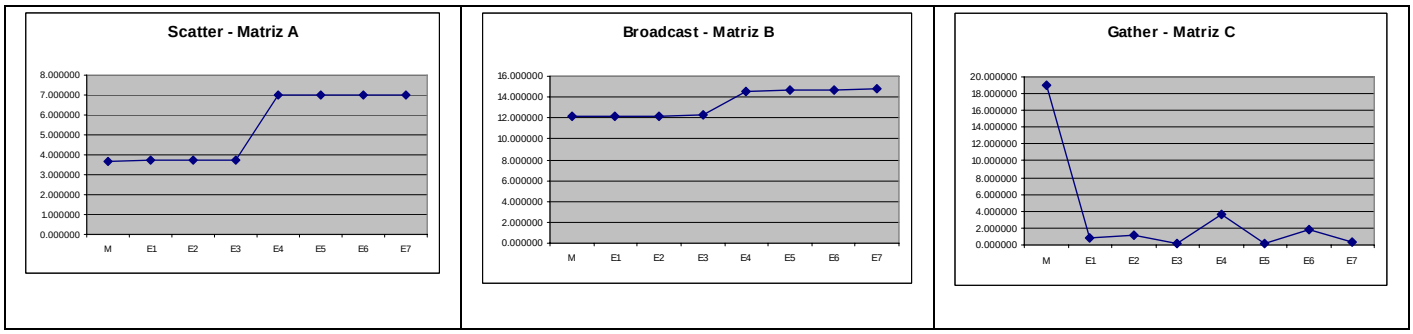
Tabela 16 - Tempos Médios de Transmissão e Execução para o Teste C

Os gráficos das Figuras a seguir ilustram os tempos de comunicação para as operações de Scatter, Broadcast e Gather.



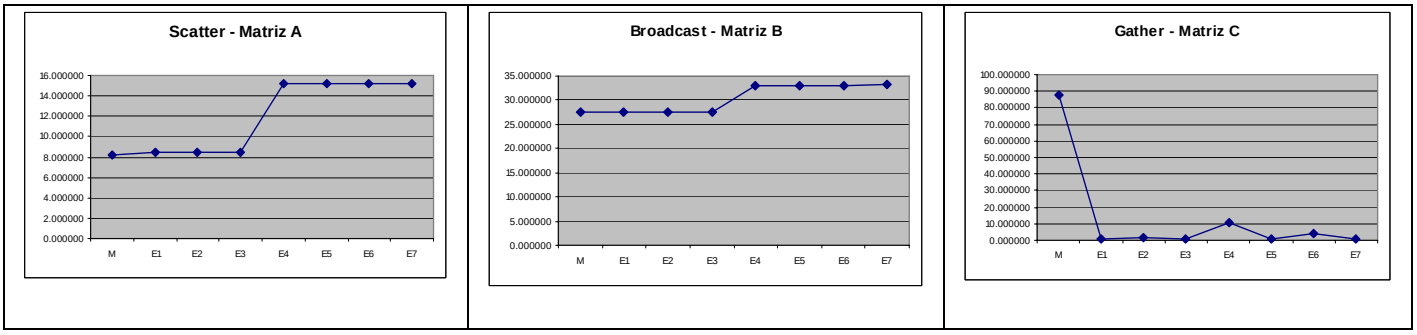
(a) (b) (c)

Figura 48 - Tempos de Transmissão para as matrizes A, B e C para a dimensão 800 no Teste C



(a) (b) (c)

Figura 49 - Tempos de Transmissão para as matrizes A, B e C para a dimensão 1600 no Teste C



(a) (b) (c)

Figura 50 - Tempos de Transmissão para as matrizes A, B e C para a dimensão 2400 no Teste C

Observando os valores e os gráficos obtidos das operações Scatter e Broadcast, pode-se notar um aumento nos tempos de transmissão a partir do Escravo 4. Este escravo é o escravo denominado *raiz local* da implementação MPI interna ao

cluster. Tanto na operação Scatter, quanto na Operação Broadcast, o processo *raiz global* (0) deve enviar, em sua fase global, a mensagem para o processo *raiz local*. Em seguida, os processos raízes realizam a operação entre os processos de mesma implementação MPI e SO.

Apesar de comportamentos semelhantes em ambas as operações, a operação de Scatter tem uma diferença de tempo maior entre os processos internos ao cluster, com relação aos processos externos. Esta diferença pode ser atribuída à transmissão TCP/IP ter um impacto maior na transmissão por tratarem com dimensões de matrizes menores, se comparada à operação Broadcast.

No caso da operação de Gather, observa-se um pico nos tempos de transmissão nos processos Mestre e Escravo 4, melhor notados nas dimensões 1600 e 2400. Novamente, por sua característica de raízes locais, eles são concentradores de dados. O Escravo 4 recebe a matriz parcial C dos processos internos ao cluster, enquanto que o processo Mestre recebe dos processos externos ao cluster, caracterizando a fase local. Em seguida, o Escravo 4 envia sua matriz C ao Mestre, finalizando, a comunicação.

Esta transmissão da mensagem do Escravo 4 para o Mestre é feita via TCP/IP através do processo *Gateway*, aumentando significativamente o tempo de recebimento.

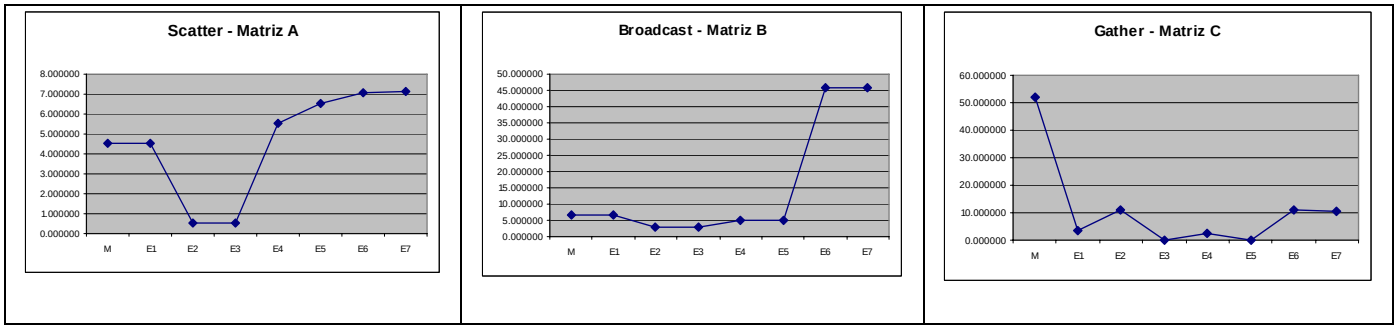
4.2.4 Teste D

Os tempos de transmissão e execução do Teste D podem ser vistas na Tabela 17 a seguir:

		800 x 800			1600 x 1600			2400 x 2400		
Matriz	**Proc**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**	**Média C**	**Média**	**Desv. P**
A	M	4,527319	4,328294	2,128488	6,286728	6,393554	0,679347	13,653392	13,763693	0,625352
	E1	4,540034	4,342747	2,132618	6,396178	6,499351	0,675765	13,934392	14,034905	0,614556
	E2	0,549177	0,717257	0,524132	2,033540	2,310096	0,942084	4,134180	4,167846	0,465612
	E3	0,553243	0,721245	0,523045	2,032954	2,309528	0,940631	4,140159	4,169810	0,457454
	E4	5,502911	5,128275	3,076996	7,144121	7,007024	1,110395	13,685844	13,729896	0,772689
	E5	6,541228	6,293571	2,486554	7,145578	7,014884	1,099414	13,691577	13,735181	0,773751
	E6	7,099015	6,496543	3,055946	10,332398	10,277063	0,871399	19,357558	19,069684	2,249804
	E7	7,102721	6,498896	3,057959	10,336231	10,282541	0,863297	19,341540	19,057195	2,254166
B	M	6,598682	6,610432	0,054012	25,254149	25,250184	0,047648	56,426476	56,418646	0,102582
	E1	6,613103	6,624845	0,053796	25,312403	25,307210	0,045869	56,548072	56,631017	0,295638
	E2	2,967727	3,274717	0,816348	11,905944	11,908807	0,017721	27,110856	27,108511	0,050970
	E3	2,969995	3,277133	0,816474	11,913850	11,916667	0,017740	27,126649	27,124367	0,050993
	E4	5,008452	5,314371	0,812682	20,081990	20,103695	0,067554	45,204770	45,195506	0,055231
	E5	5,037880	5,342407	0,809944	20,218368	20,238860	0,066075	45,510246	45,511667	0,073732
	E6	45,713578	45,613271	1,052283	154,640781	152,656426	8,310487	337,387222	322,113645	48,327292
	E7	45,715859	45,615577	1,052331	154,652285	152,667909	8,310535	337,405399	322,135199	48,322575
C	M	52,160975	51,933592	1,120767	184,139038	182,774098	9,818852	425,597297	408,842662	51,508410
	E1	3,337989	2,926062	1,272362	1,000082	0,963252	0,376409	0,524883	1,098689	1,520506
	E2	10,753228	10,315615	1,303567	18,772135	18,939829	0,557021	107,303914	107,754755	1,522546
	E3	0,066182	0,066112	0,004257	0,617338	0,611850	0,043023	0,538773	0,538994	0,001137
	E4	2,300235	2,373246	0,456521	4,056440	4,253941	0,594178	13,922563	13,972893	0,254822
	E5	0,053881	0,053886	0,000065	0,217377	0,217356	0,000619	0,490442	0,490838	0,002582
	E6	10,855358	10,665267	1,119614	41,470756	41,874298	2,238677	79,399863	80,571624	4,608381
	E7	10,508987	10,295607	1,147793	40,710621	41,342503	2,660192	77,881870	79,136999	4,690078
										,
Tempo de Exec		69.440099	67,605942	7,748304	280,803713	279,648011	9,696010	710,938928	695,377637	48,973189

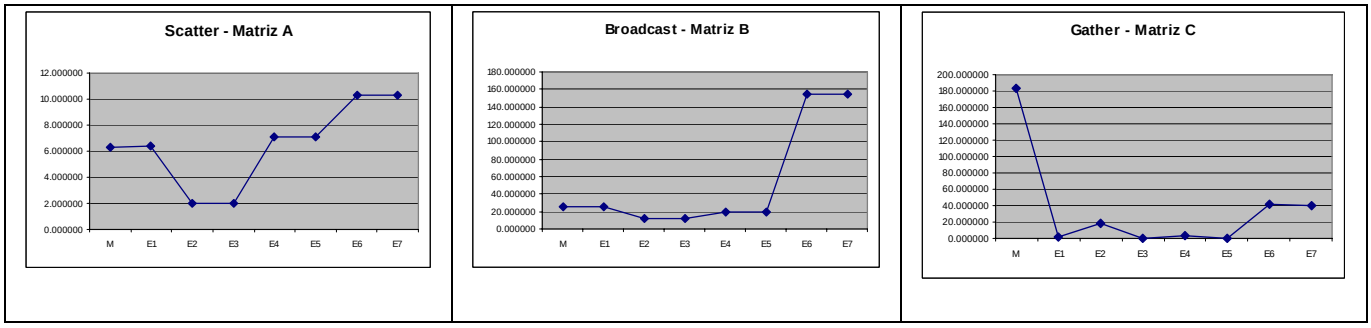
Tabela 17 - Tempos Médios de Transmissão e Execução para o Teste D

Através da obtenção destes dados, pode-se gerar o conjunto de gráficos de transmissão, conforme as Figuras a seguir.



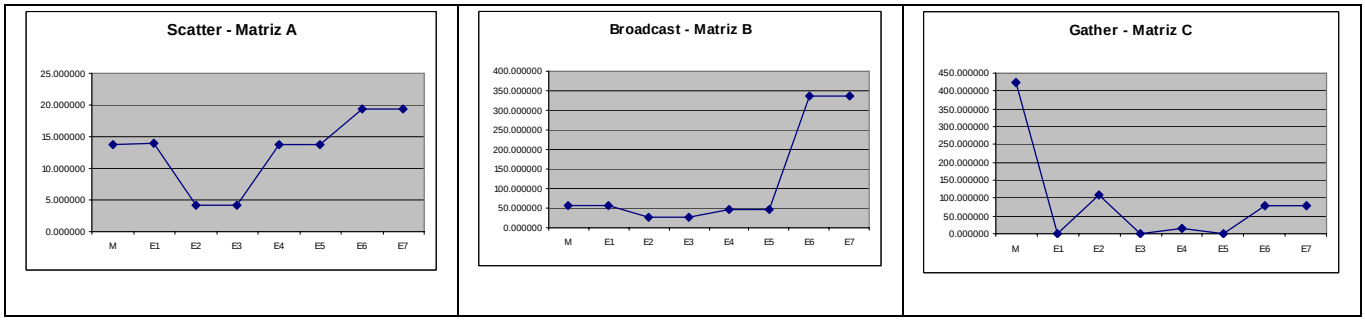
(a) (b) (c)

Figura 51 - Tempos de Transmissão para as matrizes A, B e C para a dimensão 800 no Teste D



(a) (b) (c)

Figura 52 - Tempos de Transmissão para as matrizes A, B e C para a dimensão 1600 no Teste D



(a) (b) (c)

Figura 53 - Tempos de Transmissão para as matrizes A, B e C para a dimensão 2400 no Teste D

Novamente pode-se observar um comportamento bastante semelhante das operações na transmissão das matrizes de diferentes dimensões. No teste da operação *Scatter*, os comportamentos dos diferentes escravos são bastante próximos quando fazem parte da mesma implementação (a transmissão é feita por MPI), havendo

diferenças de transmissão entre escravos de diferentes implementações. A diferença de tempo de transmissão dos escravos E2 e E3 em relação aos demais é devido ao instante em que a fase local ocorre. Ou seja, assim que o processo raiz global envia a mensagem para o escravo E2 (neste caso, E2 é um processo raiz local), logo em seguida, E2 é capaz de realizar a fase local, enquanto o nó raiz global envia a mensagem para os demais processos raízes locais.

Durante a operação de *Broadcast*, o comportamento também é relativamente próximo, com maiores diferenças nos escravos E6 e E7, devidos, tanto ao tamanho da mensagem ser maior, se comparado à operação Scatter, e também ao atraso gerado pelo processo *gateway*.

Na operação *Gather* pode-se novamente identificar os picos de tempo de transmissão em todos os processos raízes locais (processos M, E2, E4 e E6). Os processos raízes locais concentram a mensagem para encaminhamento ao processo raiz global, comprovando o mesmo comportamento identificado no Teste C.

Comparando-se os tempos de execução dos Testes C e D, observa-se um acréscimo significativo no Teste D, devido, principalmente à transmissão da mensagem. Como se trata de integrar diferentes implementações MPI, muita comunicação TCP/IP dentro do mesmo cluster e através do processo *gateway* fez-se necessária. Esta comunicação, obviamente, é mais custosa se comparada à comunicação MPI, além da transmissão nas primitivas coletivas ser codificada de forma linear. A Tabela 18 ilustra os tempos de execução e a Figura 54 mostra uma representação gráfica destes tempos.

Dimensão	LGL	HGH
800x800	15,607012	67,605942
1600x1600	99,573729	279,648011
2400x2400	342,081036	695,377637

Tabela 18 - Tempos de Execução dos Testes C e D

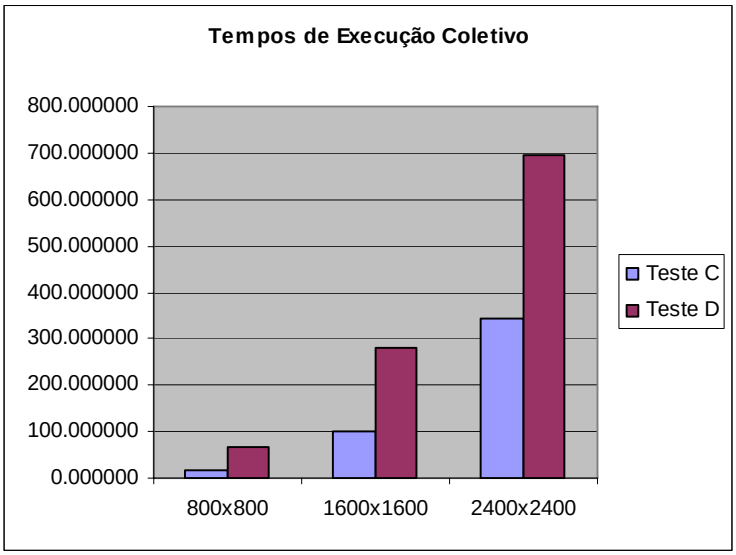


Figura 54 - Tempos de Execução dos Testes C e D

Como se pode notar, os tempos médios do Teste D foram em torno de 300% maiores em relação ao Teste C. A comunicação neste caso teve forte influência, principalmente na transmissão das matrizes para as duas diferentes implementações MPI que estavam executando dentro do cluster. Para cada implementação fez-se uso intensivo do processo *gateway*, aumentando o gargalo de comunicação.

4.3 Comparativo HyMPI e MPICH-G2

Esta seção apresenta um comparativo qualitativo das características do HyMPI e MPICH-G2 em relação à compilação da aplicação, configuração do ambiente, formas de execução, autenticação de usuários, entre outras. A análise comparativa com MPICH-G2 está sendo apresentada por ser esta uma biblioteca que possui muitas

características comuns ao HyMPI, além de amplamente utilizada em sistemas de *Grid Computing*.

HyMPI tem por objetivo criar um ambiente portável compatível com as principais implementações MPI existentes, independente de Sistema Operacional. As características particulares de cada implementação MPI são preservadas, buscando a maior interoperabilidade possível com outras implementações, atuando como uma interface entre a aplicação e a biblioteca MPI nativa.

MPICH-G2, por sua vez, também utiliza recursos para fazer uso das implementações MPI existentes em um sistema já instalado. MPICH-G2 provê suporte a vários protocolos atuando, da mesma forma que HyMPI, como o intermediário entre aplicação e a biblioteca MPI. Porém MPICH-G2 foi concebido, conforme informa seu *website* oficial, para quaisquer implementações do tipo MPICH. Isto é, inicialmente MPICH-G (a versão antecessora do MPICH-G2) suportava SGI MPI (para sistemas Origin2000) ou IBM MPI (para sistemas IBM-SP), porém não utilizava MPICH-GM (para sistemas Linux equipados com rede Myrinet). MPICH-G2 supre essa carência, caracterizando cada uma dessas implementações como *vendor-MPI*.

HyMPI foi concebido primeiramente para viabilizar a compatibilidade das bibliotecas MPI, tendo como foco a integração de Windows e Linux; por sua vez, MPICH-G2 necessita da infra-estrutura do Globus Toolkit para gerenciamento da iniciação de seus processos. Entretanto o Globus não é disponível em sua totalidade para Windows. Sua distribuição conta apenas com um pacote que consiste em um conjunto de WebServices somente para acesso a recursos do Sistema Operacional, em muitos casos, acessíveis somente se for utilizado o aplicativo CygWin [40]. As funcionalidades

principais de GridFTP, DUROC e GRAM não encontram-se disponíveis.. Essa fundamental característica de integração Linux e Windows é um dos principais atrativos do HyMPI em comparação com MPICH-G2.

Com relação à iniciação dos processos nas diversas máquinas, MPICH-G2 utiliza o aplicativo *mpirun* combinado com o serviço *globusrun*, parte integrante do Globus. Existe a necessidade de um *script* de configuração inicial, indicando quais máquinas/clusters irão compor o sistema e quantos processos por *site*[2]. O serviço *globusrun*, por sua vez entra em contato, via o DUROC, para autenticação, e o GRAM, com o escalonador nativo de cada sistema (podendo ser *rsh*, *ssh*, *fork*, *LoadLever* ou *LSF*).

Já no *HyMPIrun*, o disparo das aplicações utiliza o arquivo *conf.global* para identificar os diferentes sistemas existentes, além dos arquivos de configuração nativos de cada implementação MPI. No caso do disparo em Linux na mesma rede local onde o usuário está conectado, comandos *ssh* são utilizados diretamente para a execução. Para Windows, um pequeno *daemon* é instalado em uma das máquinas para mapear o diretório onde o binário do aplicativo encontra-se (geralmente em uma pasta compartilhada) e iniciar a aplicação MPI. O disparo em máquinas dentro de um cluster faz uso do próprio processo *Gateway*, no qual recebe uma mensagem indicando um comando de *startup* (como ilustra a Figura 29) e replica estes comandos através do protocolo *ssh* para as máquinas que irão abrigar os processos raízes locais de cada implementação e/ou Sistema Operacional.

[2] Neste caso, em MPICH-G2, entende-se por “site” um local que pode conter tanto um cluster quanto uma máquina multiprocessada

Para que a comunicação entre processos internos a um cluster através de MPICH-G2, todos os nós devem ter endereçamento IP público [41,42], ou seja, as arquiteturas de clusters que utilizam nós com endereçamento privado não podem fazer uso da infra-estrutura Globus para a comunicação ponto a ponto de processos internos ao cluster para processos externos ao cluster, uma vez que o MPICH-G2 não oferece recursos de encaminhamento de mensagens. A Tabela 19 resume algumas das diferentes características qualitativas e funcionais entre HyMPI e MPICH-G2.

Além disso, testes utilizando quatro processos, onde dois deles executam em máquinas Dual Xeon 2.0 GHz, 512Mb de RAM, 20GB de HD, com e os outros dois em Dual Xeon 2.3 GHz, 512 Mb de RAM, 20Gb de HD, todos conectados por rede Fast Ethernet. As aplicações utilizadas foram as de Multiplicação de Matrizes Ponto a Ponto e Coletiva, a fim de obter uma comparação quantitativa entre as duas diferentes bibliotecas.

Característica	HyMPI	MPICH-G2
Aplicação de Iniciação	HyMPIrun	mpirun + globusrun
Disparo no Ambiente nativo	Processo Gateway + ssh	DUROC + ssh/fork
Suporte à diversas Implementações MPI	Sim	Sim (desde que MPICH-G2 e unix-like)
Suporte a Sistema Operacional Windows	Sim, independente de implementação MPI	Somente usando CigWin (apenas Globus)
Comunicação com processos em clusters	Processo Gateway	Apenas se os nós tiverem endereçamento público
Reescrita de Código	Não	Opcional
Recompilação de Código	Sim	Sim
Scripts de Configuração	Arquivos conf.global e conf.local	Scritps RSL
Complexidade de definição do script	1 linha por processador	8 linhas por site
Infra-Estrutura Adicional	Nenhuma	Globus Toolkit
Complexidade de Configuração da Infra-Estrutura	Não Existe	Requer usuários experientes
Suporte para Linguagem C em ambiente Windows	Sim	Não (apenas usando Cygwin)

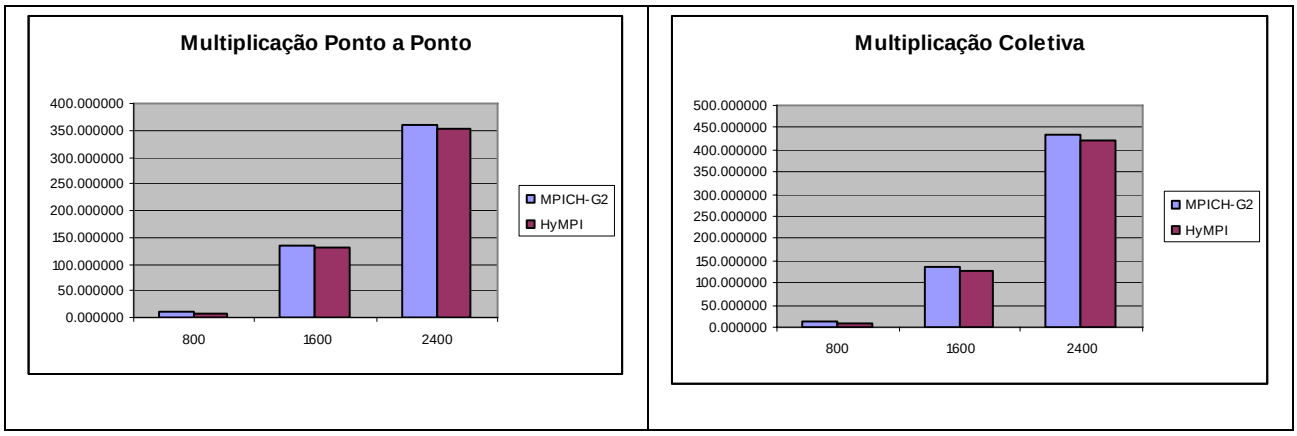
Tabela 19 - Comparação das Características Gerais entre HyMPI e MPICH-G2

Nesses testes foram levados em consideração não somente os tempos de execução e comunicação medidos no código das aplicações, como também o tempo de disparo das aplicações a partir da linha de comando. A Tabela 20 ilustra os tempos médios de execução de ambos os algoritmos.

	Ponto a Ponto			Coletiva		
Dimensão	800	1600	2400	800	1600	2400
MPICH-G2	12,512667	134,819667	360,535333	12,463333	134,507667	432,230333
HyMPI	7,153000	129,872333	354,064000	6,927667	127,963667	420,673667

Tabela 20 - Tempos de Execução entre MPICH-G2 e HyMPI

Como pode ser visto, os mesmos algoritmos tiveram um acréscimo no tempo de execução usando MPICH-G2 se comparado ao HyMPI. Isto é devido ao MPICH-G2 utilizar um mecanismo mais complexo de disparo das aplicações, onde há a necessidade de trocas de certificados digitais entre os diversos *hosts*. HyMPI nestes casos utiliza um mecanismo mais simplificado, fazendo uso do próprio aplicativo *mpirun* nativo do MPI, consumindo menos tempo. Os gráficos da Figura a seguir ilustram essa diferença de tempos de execução.



(a) (b)

Figura 55 - Tempos de Execução dos algoritmos de Multiplicação de Matrizes (a) Ponto a ponto e (b) Coletiva

Além disso, foram medidos os tempos de transmissão das mensagens em cada um dos algoritmos e o tempo de execução do próprio algoritmo (não considerando o tempo a partir do disparo da aplicação pela linha de comando). A Tabela 21 ilustra estes tempos de transmissão e execução para o algoritmo em sua versão ponto a ponto.

Dimensão		800		1600		2400	
		MPICH-G2	HyMPI	MPICH-G2	HyMPI	MPICH-G2	HyMPI
A	Escravo1	0,094676	0,107599	0,422786	0,434028	0,982279	0,978412
	Escravo2	0,109586	0,109148	0,433399	0,435542	0,981850	0,979735
	Escravo3	0,109383	0,109063	0,436752	0,435509	0,981138	0,979598
B	Escravo1	0,435959	0,439229	1,774225	1,757257	3,936413	3,962454
	Escravo2	0,435860	0,439139	1,740869	1,755026	3,939048	3,948360
	Escravo3	0,435824	0,439216	1,740579	1,755389	3,938770	3,949103
C	Escravo1	0,247593	0,112743	0,449843	0,518330	13,167180	6,983344
	Escravo2	0,007282	0,112912	15,189723	14,974308	4,644873	1,593773
	Escravo3	0,353121	0,115073	0,016304	2,649167	8,029019	4,472352
Tempo de Execução		6.065618	6,285833	128,701516	128,871607	351,439323	352,435318

Tabela 21 - Tempos de transmissão e execução do algoritmo de Multiplicação ponto a ponto

Pode-se notar que os tempos de transmissão das matrizes A e B são bastante próximos, utilizando qualquer uma das bibliotecas. O tempo de transmissão da matriz C é variável, em função das diferentes configurações de máquinas existentes. Esta diferença, em alguns casos, força um dos processos a esperar o receptor tornar-se apto para o recebimento.

Além disso, os tempos de execução (que não considera o disparo pela linha de comando) mostraram-se muito próximos, confirmando que o disparo da aplicação em MPICH-G2 é de fato mais custoso do que em HyMPI.

Na versão Coletiva, pode-se observar um comportamento semelhante. Os tempos de transmissão e execução utilizando HyMPI mostraram-se inferiores, se comparados ao MPICH-G2, além dos tempos de execução, tanto medidos diretamente

no algoritmo, quanto medidos a partir do disparo de linha de comando. A Tabela 22 apresenta os tempos obtidos

Dimensão		800		1600		2400	
		MPICH-G2	HyMPI	MPICH-G2	HyMPI	MPICH-G2	HyMPI
A	Mestre	0,344338	0,329122	1,388121	1,319687	3,130355	2,968267
	Escravo1	0,471188	0,364540	0,952076	1,458170	1,699878	3,307480
	Escravo2	0,482524	0,371988	1,298815	1,488374	2,709707	3,377763
	Escravo3	0,492794	0,372733	1,645541	1,492282	3,566774	3,384738
B	Mestre	1,290270	0,878892	5,209063	3,519246	11,736456	7,909433
	Escravo1	0,666314	0,886719	2,680283	3,558133	11,026070	7,956893
	Escravo2	0,987074	0,889817	3,968567	3,551556	8,944287	8,071723
	Escravo3	1,307985	0,896970	5,260048	3,587745	11,852174	8,366685
C	Mestre	1,124690	1,382436	24,315774	17,481018	67,177378	64,172606
	Escravo1	1,025218	0,108710	0,470123	0,433403	1,086792	0,978007
	Escravo2	0,109511	0,332840	0,464029	1,674285	1,074409	5,379016
	Escravo3	0,096405	0,270958	0,462843	0,560223	1,070002	1,272524
Tempo de Execução		6.304681	6,040558	127,745563	126,859695	413,723695	416,468733

Tabela 22 - Tempos de transmissão e execução do algoritmo de Multiplicação coletiva

Nesta Tabela, os tempos de execução da aplicação usando HyMPI nas matrizes de dimensão 800 e 1600 é inferior, se comparado à versão MPICH-G2, entretanto, nas matrizes de dimensão 2400, MPICH-G2 mostrou-se mais rápido, entretanto no cálculo a partir do disparo da linha de comando, MPICH-G2 obteve um desempenho menor, confirmando, novamente o custo do disparo das aplicações em MPICH-G2.

5. Conclusões

A atual diversidade de ambientes computacionais, incluindo *hardware*, sistemas operacionais e bibliotecas para a construção de ambientes distribuídos de alto desempenho tem tornado a necessidade de integração cada vez mais imprescindível. Diversas implementações de bibliotecas MPI, dentre elas MPICH, LAM-MPI, entre outras, além da variedade de sistemas operacionais, tais como os baseados em Unix e também o sistema Windows, tem proporcionado aos projetistas e desenvolvedores alternativas para a construção de aplicações distribuídas.

Entretanto a integração dessas diversas infra-estruturas é foco de várias pesquisas, onde bibliotecas são apresentadas, tornando essa integração o mais transparente possível para a aplicação, muitas vezes requerendo, somente a recompilação de código. Algumas apresentam limitações com relação à organização do ambiente computacional, como por exemplo, *cluster*, MPP, NOW, entre outros, e também limitações com relação à implementação das bibliotecas compatíveis ou sistema operacional que executa nas máquinas.

Neste sentido, a estratégia proposta proporciona uma integração entre diferentes ambientes distribuídos, fazendo uso da infra-estrutura já existente, englobando clusters heterogêneos de mesmo domínio ou domínios distintos, com a inclusão de sistemas operacionais alternativos, ao contrário das propostas investigadas no Capítulo 2. As limitações das propostas investigadas ocorrem principalmente com relação ao tipo de sistema distribuído suportado, tais como *clusters*, MPPs ou NOWs, com

endereçamento público ou privado, apresentando ou não máquina *front-end* em sua organização, além de contemplarem apenas sistemas operacionais baseados em Unix.

No Capítulo 4, os testes realizados com HyMPI comparam seu desempenho de execução e comunicação com ambientes homogêneos, utilizando aplicações com características distintas, fazendo uso de comunicação ponto a ponto e comunicação coletiva. Além disso, é realizada uma análise qualitativa e quantitativa com a biblioteca MPICH-G2, comparando as características principais, tempos de execução e comunicação. As análises comprovam a estabilidade do uso de HyMPI, apesar da comunicação apresentar-se mais lenta, se comparado aos testes em ambientes homogêneos. Porém, comparando-se com MPICH-G2, o tempo de disparo da aplicação mostrou-se inferior, por utilizar um protocolo mais simples, além de tempos de execução compatíveis. Dessa forma, o uso de HyMPI apresenta-se bastante promissor, principalmente por permitir a integração em larga escala de nós heterogêneos e *clusters* com ou sem máquina *front-end*, proporcionando tempos de comunicação aceitáveis.

5.1 Contribuições da Tese

No desenvolvimento desta Tese, algumas importantes contribuições foram indentificadas e são apresentadas a seguir:

- Propor uma estratégia que integre ambientes distribuídos com diferentes organizações. Isto significa que é possível construir ambientes distribuídos composto de *clusters*, MPPs ou NOWs, contendo ou não máquinas *front-end*, proporcionando um mecanismo transparente de comunicação, além de abranger, ao contrário das

demais implementações, ambientes distribuídos cujos nós executam outros sistemas operacionais, tais como o Windows. Adicionalmente, a estratégia proposta proporciona flexibilidade na implementação, onde, ao contrário sistemas como PACX-MPI, não há a necessidade de processos adicionais para encaminhamento de mensagens nos casos em que os nós sejam *clusters* sem máquina *front-end* ou computadores únicos. A inclusão do processo *gateway* em HyMPI faz-se necessária somente nos casos em que haja integração de *clusters* com máquina *front-end*. O oferecimento deste conjunto de características e as estratégias de integração adotadas podem ser consideradas as principais contribuições deste trabalho;

- Implementação de um protótipo denominado HyMPI, baseado na especificação MPI, incluindo tipos de dados, operações, funções de comunicação ponto a ponto e coletivas, permitindo construir aplicações que executem em ambientes distribuídos composto por nós que executam diferentes sistemas operacionais e implementações MPI. Foi implementado um subconjunto de primitivas existentes na especificação MPI;
- Eliminar a necessidade de reescrever código, sendo necessária, apenas, nova compilação para ligar os arquivos da aplicação distribuída à biblioteca HyMPI;

5.2 Propostas de Trabalhos Futuros

A continuidade deste projeto cria oportunidades para o desenvolvimento de diversos outros trabalhos, ampliando a capacidade, uso e funcionalidades do HyMPI. Dentre as propostas de continuidades identificadas, podem ser destacadas as seguintes:

- Desenvolver outras primitivas MPI, ampliando o seu poder e uso, tais como: MPI_Allgather, MPI_Allreduce, MPI_Bsend, MPI_Ssend, MPI_Alltoall, primitivas de manipulação de comunicadores e de E/S, primitivas de empacotamento de dados (MPI_Pack), que tornarão o HyMPI mais completo e com uma abrangência maior, podendo ser utilizado em uma variedade maior de aplicações de *Benchmarks*;
- Continuidade e ampliação das funcionalidades do processo *Gateway*, buscando características para otimizar a comunicação, através de conjuntos de *threads* emissoras e receptoras. Apesar de muitas vezes a comunicação tornar-se restrita em função do canal existente, uma melhor utilização deste canal pode aumentar o desempenho da comunicação TCP/IP, diminuindo o tempo de envio e recebimento das mensagens.
- Outra proposta é a construção de uma interface mais amigável de disparo e execução que gerencie as credenciais do usuário nos diversos sistemas onde a aplicação será executada, tornando transparente a autenticação, aumentando, assim sua segurança.
- Por fim, uma ferramenta de FTP, assim como o GridFTP pode ser incorporada de forma a tornar o transporte dos arquivos binários da

aplicação automático, minimizando a necessidade de interferência do usuário.

Referências Bibliográficas

[1] Buyya, R. “High Performance Cluster Computing – Volume 1: Architectures and Systems”, Prentice-Hall, 1999.

[2] Culler, D. E., Bal, S. J., Gupta, A. “Parallel Computer Architecture: A Hardware/Software approach”. Morgan-Kaufmann Publishers, 1999.

[3] Snir M., Gropp W., “MPI the Complete Reference”. The MIT Press (1998).

[4] Gropp, W. et al. (1996). A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Parallel Computing, 22 (6): pages 789-828.

[5] Geist, A. et al. (1994). PVM: Parallel Virtual Machine. A User’s Guide and Tutorial for Networked Parallel Computing. The MIT Press, 1994.

[6] LAM Team. LAM/MPI Parallel Computing, MPI General Information. Disponível em http://www.lam-mpi.org/mpi/. Acesso em 25/05/2006

[7] Mpich, (2006). A Portable MPI Implementation. http://www.unix.mcs.anl.gov/mpi/mpich/. Acesso em 30/10/2006.

[8] Marinho, J., Silva, J. G. WMPI - Message Passing Interface for Win32 Clusters. Lecture Notes In Computer Science; Vol. 1497 archive Proceedings of the 5th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Páginas 113- 120.

[9] WMPI, (2006). HPC, High Performance Computing Home. Disponível em: http://www.criticalsoftware.com/hpc/. Acesso em 18/05/2004.

[10] MPIPro, (2006). MPI Software Technology. http://www.mpi-softech.com/. Acesso em 30/03/2004.

[11] NT-MPICH (2006). NT-MPICH Guide. Disponível em http://www.windowsclusters.org/nt-mpich_guide.htm. Acesso em 30/03/2004.

[12] NT-MPICH, (2006). Message-Passing Interface Implementations. Disponível em: http://www-unix.mcs.anl.gov/mpi/implementations.html. Acesso em 30/03/2004.

[13] MP-MPICH, (2006). MP-MPICH - MPI for heterogeneous Clusters, disponível em: http://www.lfbs.rwth-aachen.de/content/mp-mpich. Acesso em 05/04/2004.

[14] Squyres, J. et al. The Interoperable Message Passing Interface (IMPI) Extensions to LAM/MPI. MPI Developer's Conference, Ithica, NY, 2000.

[15] George, W. et al. IMPI: Making MPI interoperable. Journal of Research of the National Institute of Standards and Technology, 105(3), 2000.

[16] Skjellum, A., McMahon, T., Interoperability of Message-Passing Interface (MPI) Implementations. Journal of Development - Message Passing, 1997.

[17] Gabriel E. et al. Distributed Computing in a Heterogeneous Computing Environment. In PVM/MPI, pages 180--187, 1998

[18] Gabriel E. et al. Implementing MPI with optimized algorithms for metacomputing, 1999.

[19] Gabriel E. et al. An extension to MPI for distributed computing on MPPs. Lecture Notes in Computer Science, 1332:75-82, 1997. 19

[20] Karonis, N. et al. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Disponível em www.globlus.org. Acesso em 20/11/2006.

[21] Lacour, S. MPICH-G2 Collective Operations: Performance Evaluation, Optimizations. Master's thesis, Rapport de stage MIM2, Magistère d'informatique et modélisation (MIM), ENS Lyon, Mathematics and Computer Science Division, Argonne National Laboratory, USA, September 2001.

[22] Fagg, G., Dongarra, J. PVMPI: An integration of the PVM and MPI systems. Technical report, Department of Computer Science, UniversityofTennessee, April 1996. Disponivel em: www.cs.utk.edu/~library/TechReports.html. Acesso em 13/01/2006.

[23] Fagg, G., Dongarra, J., Geist, A. Heterogeneous MPI Application Interoperation and Process Management under PVMPI. In Recent Advances in Parallel Virtual Machine and Message Passing Interface, LNCS, No. 1332. Springer, 1997.

[24] MPIForum (2006). Message-Passing Interface (MPI) Forum Home Page. Disponível em www.mpi-forum.org. Acesso em 30/03/2006.

[25] Buyya, R. (editor). High Performance Cluster Computing. Prentice Hall PTR, NJ, USA, 1999.

[26] MPI (2006), MPI- A Message-Passing Interface Standard. International Journal of Supercomputer Applications 8(3/4): 165-414, 1994.

[27] Globus (2006). The Globus Toolkit. Disponível em www.globus.org. Acesso em 20/11/2006

[28] MPI2, MPI-2 : Extensions to the Message-Passing Interface. University of Tennessee, 2003.

[29] Fitzgerald. S. et al. A directory service for configuring high-performance distributed computations. 6th IEEE Symp. on High Performance Distributed Computing, 365-375, 1997.

[30] Czajkowski, K. et. al. Co-allocation Services for Computational Grids. 8th IEEE Symp. on High Performance Distributed Computing, 1999.

[31] Czajkowski, K. et. al. A resource management architecture for metacomputing systems. 4th Workshop on Job Scheduling Strategies for Parallel Processing, 1998.

[32] Massetto, F. I., et al. HMPI - Hybrid MPI". 14th IEEE Symposium on High Performance and Distributed Computing. 2005.

[33] Massetto, F. I., Gomes Jr. A. M., Sato, L. M., HyMPI - A MPI Implementation for Heterogeneous High-Performance Systems. Lecture Notes in Computer Science LNCS 3497. Proceedings of First International Conference GPC 2006. Pags. 314-323, 2006.

[34] Stallings, W. Operating Systems. 4th Edition. Prentice-Hall, 2004.

[35] MSDN, 2006. Microsoft Development Network. Disponível em: www.msdn.com. Acesso em 12/02/2006.

[36] IOCTL, 2006. Disponível em: http://linux.about.com/library/cmd/blcmdl2_ioctl.htm. Acesso em 12/02/2006.

[37] SSL, 2006. SSL Request for Comments. Disponível em: http://www.ietf.org/rfc/rfc2246.txt. Acesso em 20/02/2006

[38] SSH, 2006. OpenSSH. Disponível em http://www.openssh.com/. Acesso em 20/02/2006

[39] BARNEY, B. Lecture Notes. Disponível em: http://carbon.cudenver.edu/csprojects/csc5809F99/mpi_examples/2d_heat_equation.htm. Acesso em 10/03/2005

[40] CygWin Information and Installation. Disponível em http://www.cygwin.com. Acesso em 20/02/2006

[41] Globus Mailing List – MPICH-G2 for an inter-cluster application. Disponível em:http://www-unix.globus.org/mail_archive/mpich-g/2003/09/msg00033.html. Acesso em 10/06/2007.

[42] NGPP, “Grid-Enabling Execution of MPI-based Programs”. Disponível em www.ngp.org.sg/ngpp/document/Infosheet%20-%20RSIP%20v2.pdf. Acesso em 08/06/2007.