Universidade do Estado da Bahia – UNEB

Departamento de Ciências Exatas e da Terra

Colegiado de Sistemas de Informação

Felippe Vieira Zacarias

Uma ferramenta de *benchmark* para auxiliar na análise de desempenho de *cluster* de computadores

Salvador

2014

Felippe Vieira Zacarias

Uma ferramenta de *benchmark* para auxiliar na análise de desempenho de *cluster* de computadores

Monografia apresentada à Banca Examinadora como exigência parcial para obtenção do título de bacharel em Sistemas de Informação pela Universidade do Estado da Bahia.

Orientador: Josemar Rodrigues de Souza, Ph.D

Salvador

2014

Felippe Vieira Zacarias

Uma ferramenta de *benchmark* para auxiliar na análise de desempenho de *cluster* de computadores/ Felippe Vieira Zacarias. – Salvador, 2014-

64 p. : il. (algumas color.) ; 30 cm.

Orientador: Josemar Rodrigues de Souza, Ph.D

Monografia (Graduação) – Universidade do Estado da Bahia – UNEB
Departamento de Ciências Exatas e da Terra
Colegiado de Sistemas de Informação, 2014.

1. Indicadores de Desempenho. 2. *Benchmark* Computacional. 3. Computação Paralela. I. Josemar Rodrigues de Souza. II. Universidade do Estado da Bahia. III. Departamento de Ciências Exatas e da Terra. IV. Uma ferramenta de *benchmark* para auxiliar na análise de desempenho de *cluster* de computadores

Felippe Vieira Zacarias

Uma ferramenta de *benchmark* para auxiliar na análise de desempenho de *cluster* de computadores

Monografia apresentada à Banca Examinadora como exigência parcial para obtenção do título de bacharel em Sistemas de Informação pela Universidade do Estado da Bahia.

Trabalho aprovado. Salvador, 29 de Julho de 2014:

Josemar Rodrigues de Souza, Ph.D
Ph.D em Informática
Universidade do Estado da Bahia

Oberdan Rocha Pinheiro
Mestre em Modelagem Computacional
Faculdade SENAI CIMATEC

Marcos Ennes Barreto
Doutor em Ciência da Computação
Universidade Federal da Bahia

Murilo do Carmo Boratto
Doutorando em Informática
Universidade do Estado da Bahia

Salvador
2014

Agradecimentos

Em primeiro lugar a Deus por cada dia e noite de vida, pela minha existência e das pessoas maravilhosas que me cercam e por sua inesgotável fonte de amor e perdão.

- A toda minha família principalmente minha avó Nair, minha mãe Elisângela, minha madrinha Cristina, meu primo Rafael e minha cunhada Edinete, pelos momentos de suporte durante esses anos de graduação.
- Ao meu orientador, o professor Josemar, por me orientar na conclusão deste trabalho e por suas contribuições sempre pertinentes que colaboraram na construção deste trabalho.
- A equipe águia/resistência: Thomas Magnum, Leonardo Campos, Kal Lenon, Daniel Amaral, Marcelo Tonete, Raylan, Rafael Guimarães e Marcus Alexandre.
- Aos colegas que desistiram mas que sempre serão lembrados: Alexandre (maratá), Tiago (swat), barata.
- Aos colegas Rafael Paim e João Paulo pela leitura no trabalho.
- A todos os meus professores, que nesses anos de graduação contribuiram para a minha formação não só como profissional mas também como pessoa.

"Ó Mestre, Fazei que eu procure mais
Consolar, que ser consolado;
compreender, que ser compreendido;
amar, que ser amado.
Pois, é dando que se recebe.
É perdoando que se é perdoado
e é morrendo que se vive para a vida eterna."
(Oração de São Francisco, Anônimo)

Resumo

Na computação, um *benchmark* é um programa usado para efetuar testes de desempenho no sistema computacional, visando avaliar a influência da arquitetura no desempenho. Para o usuário é importante selecionar a configuração do *cluster* de acordo com o conjunto de aplicações que serão executados por ele, como *a priori* esta informação é desconhecida, *benchmarks* tem sido desenvolvidos para permitir uma avaliação de desempenho do sistema de maneira padronizada. Porém, o estabelecimento de modelos ou mecanismos que permitam a análise dos dados obtidos a partir das execuções desses *benchmarks*, torna-se uma tarefa não trivial. Desta maneira, o objetivo deste trabalho é propor e validar uma ferramenta de *benchmark* que forneça indicadores para avaliar o desempenho de *clusters* de computadores, compreendendo os seguintes fatores: largura de banda e taxa de transferência da interconexão de rede, operações em ponto flutuante e tempo de processamento. Esta ferramenta é composta de aplicações já existentes, além de possuir uma interface web para o *upload* de códigos paralelos do usuário e que inicia os *scritps* de configuração e execução do *benchmark* na arquitetura. Ao final, no *testbed* utilizado para execução da ferramenta, foi possível constatar que com o *benchmark* HPL, em sua versão multi*thread*, pôde-se alcançar um desempenho 1,6 vezes melhor que a versão com processos distribuídos sem *thread* do *benchmark*. Com a aplicação de processamento de imagem, os resultados das versões híbridas foram semelhantes com a versão de processos distribuídos. Já com a aplicação de multiplicação de matriz, sua versão híbrida obteve os melhores desempenhos em relação as outras abordagens executadas.

Palavras-chaves: *Benchmarks*, Computação Paralela, Indicadores de Desempenho, Memória Distribuída.

Abstract

Benchmarks are programs used to execute performance tests on computational systems aiming evaluate the leverage of the architecture on its performance. For user is important select the cluster's configuration according the set of applications will be executed, but in advance it is a unknown information, so benchmarks have been developed to allow standard evaluation of system's performance. But set up models or mechanisms that allow analysis of data from its executions isn't a ordinary task. Thus, the main goal of this research is set up and validate a benchmark which give performance indicators to evaluate distributed memory computers performance. This tool is made up of applications which already exist and it has a Web interface to upload parallel codes and launch the execution of benchmark. It also comprise indicators such bandwidth and transfer rate of network, floating point operations and wallclock. At the end, on the testbed used to execute the benchmark, running the multithread benchmark HPL, the average performance was 1,6x better than its distributed processes execution. Running the image processing algorithm, the result of its hybrid execution was similar to its distributed processes execution. But with matrix multiplication algorithm, its hybrid execution was better than the others executions.

Key-words: Benchmarks, Parallel Computing, Performance Indicators, Distributed Memory.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ANSI	American National Standards Institute
API	Application Programming Interface
ATLAS	Automatically Tuned Linear Algebra Software
BLAS	Basic Linear Algebra Subprograms
CPU	Central Processing Unit
CSMA/CD	Carrier Sense Multiple Access with Collision Detection
CSS	Cascading Style Sheets
D-ITG	Distributed Internet Traffic Generator
Flops	Floating-point Operations Per Second
HP	Hewlett-Packard
HPC	High Performance Computing
HPL	High Performance Linpack
HTML	HyperText Markup Language
IEEE	Institute of Electric and Electronic Engineers
IP	Internet Protocol
IPM	Integrated Performance Monitoring
MIMD	Multiple Instruction Multiple Data
MIPS	Milhões de Instruções por Segundo
MISD	Multiple Instruction Single Data
MPI	Message Passing Interface
NPB	NAS Parallel Benchmarks
OpenMP	Open Multi-Processing
PHP	Hypertext Preprocessor

SAN	System Area Network
SIMD	Single Instruction Multiple Data
SISD	Single Instruction Single Data
SPEC	System Performance Evaluation Cooperation
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol

Sumário

1 Introdução

Uma das principais justificativas para a utilização de computadores paralelos é que o paralelismo é uma forma de evitar o gargalo atingido com os computadores *single core*. A computação de alto desempenho ou *High Performance Computing* (HPC) utiliza super computadores e *clusters* de computadores para resolver problemas que demandam alta capacidade de rescursos computacionais. Neste contexto, segundo Boratto, Coelho e Barreto (2012), a utilização da computação de alto desempenho para a disponibilização de recursos computacionais a custos acessíveis tem sido um tema de relevância mundial.

Um *cluster* se caracteriza por ser um conjunto de computadores interconectados através de uma rede para a troca de informação e de recursos, operando sobre o controle de *softwares* específicos. Esse *software* faz com que o conjunto opere como se fosse um único equipamento com capacidade e escalabilidade mais elevada e custo proporcional à capacidade computacional.

Na computação, um *benchmark* é um programa usado para efetuar testes de desempenho do sistema computacional, objetivando avaliar a influência da arquitetura no desempenho. Por isso, uma das razões mais importantes para se medir e avaliar o desempenho dos computadores paralelos é saber se o desempenho atual pode ser melhorado (ZACARIAS et al., 2013). Contudo, com a evolução das arquiteturas heterogêneas e *multicore*, torna-se difícil comparar o desempenho desses sistemas especialmente quando análises detalhadas e projeções são desejadas (KOGGE; DYSART, 2011).

Para o usuário é importante selecionar a configuração de um *cluster* de acordo com as características do conjunto de aplicações que serão executados por ele. Porém, é difícil saber isto a *priori*, já que podem ser diferentes aplicações por usuário, além de frequentemente terem tempos e frequência de execução desconhecido. Por isso, os *benchmarks* têm sido desenvolvidos para permitir uma avaliação do desempenho do sistema de maneira padronizada, baseando-se em características específicas que podem ser mensuradas (RAUBER; RÜNGER, 2010).

O objetivo deste trabalho é propor e validar uma ferramenta de *benchmark* que fornece indicadores para avaliar o desempenho de *clusters* de computadores, compreendendo os seguintes fatores: largura de banda e taxa de transferência da interconexão de rede, operações em ponto flutuante e tempo de processamento.

Além de diminuir o esforço do usuário na otimização das diversas aplicações disponíveis, a ferramenta proposta tenta apresentar ao usuário um resultado final mais detalhado em relação a comunicação e comportamento do que alguns *benchmarks* utilizados, como por exemplo o *High Performance Linpack* (HPL) (DONGARRA; WHALEY, 2008).

A construção desta ferramenta é orientada por uma análise de *benchmarks*, que executam em *cluster* de computadores, disponíveis de modo gratuito na internet e pela automação da execução dessas aplicações juntamente com a integração com uma camada Web para apresentação dos resultados ao usuário.

Um importante fato que se refere a configuração de um *cluster* é que seu desempenho final dependerá da escolha harmoniosa de seus itens de configuração, como por exemplo a interconexão de rede, pois sua eficácia depende das propriedades básicas do meio (velocidade, latência), da topologia, da implementação do protocolo de mensagens e do comportamento da aplicação. Se esta configuração não for cuidadosamente escolhida pode impactar negativamente no desempenho geral.

Na análise de desempenho de aplicações e arquiteturas paralelas, o estabelecimento de modelos ou mecanismos que permitam a análise dos dados obtidos a partir das execuções de *benchmarks* torna-se uma tarefa não trivial, considerando-se parâmetros e graus de liberdade envolvidos na análise, além de exigir do projetista um conhecimento acumulado em várias áreas da computação. Essas ferramentas de mensuração e análise de dados permitem ao usuário identificar pontos de gargalo e fontes de ineficiência em uma solução paralela.

Este trabalho apresenta 5 capítulos e está estruturado da seguinte forma:

Capítulo 1 - Introdução: Contextualiza o ambiente no qual a pesquisa está inserida. Apresenta a definição do problema, objetivos e justificativas da pesquisa e a estrutura do trabalho;

Capítulo 2 - Estado da Arte: Neste capítulo são apresentados conceitos básicos sobre arquitetura, modelos de programação e métricas em computação paralela, bem como as tecnologias de interconexões de rede utilizadas em *cluster* de computadores e ferramentas de *benchmark*, utilizadas para obter indicadores de desempenho, descrevendo principalmente o *benchmark* HPL;

Capítulo 3 - A Ferramenta de *Benchmark*: Neste capítulo é apresentado o projeto para o desenvolvimento da ferramenta de *benchmark*, também é apresentado a especificação da arquitetura da ferramenta e o seu fluxo de execução;

Capítulo 4 - Validação da Ferramenta de *Benchmark*: Neste capítulo são apresentados os resultados experimentais da ferramenta de *benchmark* proposta, além da validação da ferramenta utilizando a técnica de teste de unidade;

Capítulo 5 - Conclusões e Trabalhos Futuros: Neste capítulo, são apresentadas as conclusões, contribuições da pesquisa e algumas sugestões de pesquisa a serem desenvolvidas no futuro.

2 Estado da Arte

Segundo Sutter (2005), não se pode esperar que os algoritmos fiquem mais rápidos apenas com a atualização dos processadores. O que acontece muitas vezes é o aproveitamento por parte das aplicações dos ganhos regulares de desempenho quando há novas versões de processadores, disco ou memória. Apenas aumentar o número de transistores nos circuitos integrados esbarra nos limites físicos de dissipação de calor e do alto consumo de energia, além de problemas relativos ao custo de pesquisas para a redução de componentes sem quebrar a compatibilidade do modelo atual e perda informações.

De acordo com Boratto, Coelho e Barreto (2012) não restaram opções senão modificar radicalmente a organização dos computadores introduzindo o paralelismo, introduzido nas instruções, depois com o *hyperthreading*. Logo após, a tendência seguiu com a adição de mais núcleos nos *chips* dos processadores e com uso de *chips* gráficos, ao invés de se concentrarem no *clock*, porém os algoritmos sequencias não são beneficiados por esta abordagem (PACHECO, 2011). Neste cenário a computação de alto desempenho e computação paralela se apresentam como uma alternativa para solucionar essas demandas por recursos computacionais e ramos de negócios em que confiabilidade e velocidade de transações são fatores importantes para se manter no mercado.

2.1 Arquiteturas Paralelas

Uma das diversas formas de classificar uma arquitetura paralela é utilizando a Taxonomia de Flynn (FLYNN, 1972). Apesar de ser clássica, esta classificação serve de base até hoje para o agrupamento de máquinas paralelas, além disso outras formas de classificação mais detalhadas utilizam a taxonomia de Flynn como base. Sua classificação se baseia nos conceitos de fluxo de instruções e fluxos de dados, que são independentes entre si, gerando assim quatro categorias de máquinas: Single Instruction Single Data (SISD), Multiple Instruction Single Data (MISD), Single Instruction Multiple Data (SIMD) e Multiple Instruction Multiple Data (MIMD).

Os SISD são computadores seriais onde cada instrução é executada uma após a outra. Nela estão contidas as máquinas que seguem o modelo tradicional de Von Neumman, sem qualquer paralelismo (PITANGA, 2008). Já os SIMD são uma arquitetura desenhada para problemas específicos com alto padrão de regularidade de dados. Todas as unidades devem receber do processador principal a mesma instrução no mesmo instante de tempo de modo que possam executá-las de forma simultânea sobre um fluxo de dados diferente (NULL; LOBUR, 2010).

Existe também a arquitetura MISD que assume um múltiplo fluxo de instruções sobre um único fluxo de dados, ou seja, executam várias instruções ao mesmo tempo sobre um mesmo dado (PITANGA, 2008). Constituída por um *pipeline* de unidades de processamento que operam sobre o mesmo fluxo de dados enviando os resultados de uma unidade para a outra, sendo que cada unidade de processamento executa instruções diferentes a cada momento.

A arquitetura MIMD é muito usada atualmente. Cada unidade de processamento executa instruções diferentes a cada momento, e pode operar sobre um fluxo de dados diferentes. Diferentemente das máquinas SIMD, nos sistemas MIMD os elementos processadores trabalham de forma assíncrona (RAUBER; RÜNGER, 2010) sem um cíclo de instrução global, devendo estes serem instruídos por programas para fazerem a sincronização.

A taxonomia de Flynn apenas oferece uma classificação pouco aperfeiçoada, mas ela é útil ao dar uma visão geral do domínio dos computadores paralelos (RAUBER; RÜNGER, 2010). Um de seus problemas está sobre as máquinas MIMD, pois muitos computadores recaem sobre esta categoria sem levar em consideração como os processadores estão conectados ou o tipo de acesso a memória (NULL; LOBUR, 2010). Por isso esforços para refinar esta categoria subdividem-na em multiprocessadores ou multicomputadores que compartilham memória ou não (QUINN, 2004).

Na categoria MIMD os computadores com memória distribuida vêm tendo uma maior participação no cenário de máquinas paralelas, principalmente os *cluster* de computadores. Eles consistem de elementos processadores (chamados de nós) interconectados por uma rede que suporte a transferência de dados entre eles (RAUBER; RÜNGER, 2010). Cada nó é independente com sua própria memória, processadores, discos e periféricos, isto quer dizer que a memória é privada e só os processadores do nó podem acessar os dados alocados nela (RAUBER; RÜNGER, 2010). Por isso quando um nó necessita de uma informação da memória local de um outro nó, os dados devem ser passados via mensagem pela interconexão de rede entre os nós.

2.2 Modelos de Programação Paralela

Na computação convencional, para resolver determinado algoritmo, um único processador processa um fluxo sequencial de instruções e dados previamente alocados em memória. Apenas uma instrução é executada ou um dado é manipulado por ciclo de *clock*. Com o objetivo de diminuir o tempo total de processamento de um algoritmo e de obter um desempenho melhor em relação a uma versão sequencial (ORELLANA, 2011), os algoritmos paralelos consistem em executar simultaneamente partes de uma aplicação em um mesmo processador, em uma máquina multiprocessada ou em um *cluster*.

O programa paralelo pode conter o paralelismo de dados ou funcional. O paralelismo funcional divide o problema em tarefas independentes que são associadas a um processo ou *thread* e que podem operar sobre o mesmo conjunto de dados ou não. Muitos problemas podem ser resolvidos usando qualquer um dos tipos, mas o paralelismo de dados é o mais usado e o mais fácil de ser implementado, além de ser o modelo adotado nas aplicações utilizadas neste trabalho.

No paralelismo de dados, os dados são particionados em subconjuntos e associados a um processo (PACHECO, 2011) ou *thread* (PACHECO, 2011) que executam os mesmos comandos sobre seu subconjunto de dados. Depois de processados esses dados são combinados novamente para ser obtido o conjunto resposta. Essa técnica permite uma maior exploração do paralelismo em processadores *multicore*, utilizando toda a potencialidade dos *cores* que não seriam aproveitados pela aplicação tradicional. Programas deste tipo são facilmente escaláveis, pois podem ser utilizados para a resolução de problemas cada vez maiores apenas adicionando mais processos (ORELLANA, 2011).

Além do tipo de paralelismo que é usado para escrever códigos paralelos, outro importante fator é o modelo de programação a ser usado. O modelo de programação de uma aplicação em paralelo define a forma de troca de dados e de sincronização dos mesmos na execução, além dos modelos *Divide and Conquer* (BORATTO; ALONSO; VIDAL, 2008) e Pipeline (OLIVEIRA; SOUZA, 2008), pode-se destacar o modelo *Master/Slave* (PINHEIRO, 2013).

O modelo *Master/Slave* é um paradigma que consegue elevados desempenhos e uma boa escalabilidade, pois a comunicação só existe entre o computador denominado mestre (*master*) que é o responsável por inicializar a execução da aplicação, distribuir o trabalho para o restante dos processos denominados escravos (*slave*) e reunir o resultado. Quando o número de escravos é grande este controle centralizado torna-se um problema, neste caso a solução é aumentar o numero de mestres onde cada um gerenciaria um grupo de escravos. No balanceamento de carga desta abordagem as tarefas podem ser alocadas a *priori*, se o mestre pode estimar o tamanho das tarefas, ou fazer um mapeamento dinâmico, designando pequenos pedaços de trabalho para os trabalhadores de tempos em tempos (GRAMA et al., 2003).

2.3 Modelo de Comunicação

No sistema de comunicação de um ambiente paralelo existem, basicamente, dois paradigmas de comunicação: memória compartilhada (NULL; LOBUR, 2010) e troca de mensagens (PITANGA, 2008). O modelo de memória compartilhada é um ambiente onde vários processadores compartilham a mesma memória. Por causa disso eles têm acesso ao mesmo local de memória, podem interagir e sincronizar um com o outro através

de variáveis compartilhadas (QUINN, 2004). O paradigma padrão de um programa de memória compartilhada é o *fork/join*, onde o programa inicia como uma única *thread* mestre, então nos pontos onde operações paralelas são requeridas, o mestre cria novas *threads* (*fork*). No final do código paralelo o fluxo volta para a *thread* mestre (*join*).

Neste modelo a interface de programação mais usada é a *OpenMP* (OPENMP, 2013), um padrão que define como os compiladores devem gerar códigos paralelos através de diretivas e funções. É composto por três componentes básicos: diretivas de compilação, variáveis de ambiente e biblioteca de função. Seu paralelismo é explícito, pois cabe ao programador identificar as tarefas para a execução em paralelo e definir os pontos de sincronização utilizando as diretivas de programação no código, porém a criação, inicialização e finalização das *threads* são feitas pelo ambiente de execução. Algumas das diretivas do OpenMP são: *parallel, for, parallel for, section, sections, single, master, critical, flush, ordered, e atomic*. Essas diretivas especificam o compartilhamento de trabalho entre *threads* ou instruções de sincronização (FILHO, 2012).

O modelo de passagem de mensagem tem sido tradicionalmente empregado em sistemas fracamente acoplados, representados pelas arquiteturas baseadas em memória distribuída. É o método de comunicação baseado no envio e recebimento de mensagens através da rede seguindo regras de protocolo de comunicação entre os vários processadores que têm seu próprio bloco de memória (PITANGA, 2008). Um processador tem acesso direto apenas as instruções e dados guardados na memória local. Contudo, a rede de interconexão suporta passagem de mensagem entre os processadores (QUINN, 2004). No desenvolvimento de um programa paralelo por troca de mensagens, o programador deve distribuir explicitamente os dados entre os processsadores definindo o número de processos concorrentes no começo da execução.

O padrão para essa comunicação é o *Message Passing Interface* (MPI) que provê uma base poderosa para construir programas paralelos (MPI, 2013). Apesar de acessar sua memória sem interferência e com rapidez, nesse modelo existe um *overhead* que é o de comunicação entre os processos com o envio e recebimento de dados. O MPI assume que todos os processos são estáticos, sendo que nenhum novo processo é criado em tempo de execução. Cada processo é identificado através de um *Rank* que varia de 0 a P-1, onde P é o numero de processadores. O MPI possibilita a implementação de programação paralela em memória distribuída em qualquer ambiente (RIBEIRO, 2011), visto que provê a padronização de tipos, comunicação por grupos, sincronização, controle de erros e etc. O OpenMPI é a combinação de vários projetos MPI já existente, com o objetivo de disponibilizar uma única implementação MPI integrando funcionalidades destes projetos (PINHEIRO, 2013).

Defensores do modelo de passagem de mensagem apontam diversas vantagem em relação a outros modelos. Primeiro, a passagem de mensagem executa bem em uma larga

variedade de arquiteturas MIMD, e também pode ser executado em multiprocessadores usando variáveis compartilhadas como *buffer* de mensagem (QUINN, 2004). Porém, com a tendência da inclusão de mais núcleos nos chips dos processadores, nós computacionais em arquiteturas MIMD de memória distribuída possuem cada vez mais processadores com vários núcleos, por isso estudos como o de (COSTA; SOUZA, 2013) evidenciam a vantagem da união deses dois modelos para uma exploração mais eficiênte na utilização dos recursos computacionais.

2.4 Métricas de Desempenho

De acordo com (GRAMA et al., 2003), existem duas classes distintas de métricas de desempenho:

Métricas de Desempenho para Processadores: métricas que permitem avaliar a performance de um processador tendo por base a velocidade/número de operações que este consegue realizar num determinado espaço temporal. Nesta categoria encontra-se o *Floating-point Operations Per Second* (Flops) como a métrica mais utilizada atualmente.

Métricas de Desempenho para Aplicações Paralelas: métricas que permitem avaliar a performance de uma aplicação paralela tendo por base a comparação entre a execução com múltiplos processadores e a execução com um só processador. Dentre elas as mais utilizadas são o *speedup* e a eficência.

2.4.1 *Floating-point Operations*

No campo do cálculo científico, que faz uso de grandes quantidades de cálculos em pontos flutuantes, o Flops é usado para determinar o desempenho de um computador. A fórmula para MFlops (M de *Million*) é dada pela equação 2.1 (RAUBER; RÜNGER, 2010).

$$MFlops = \frac{\text{Contagem de Operações em Ponto Flutuante}}{\text{Tempo de execução} * 10^6} \tag{2.1}$$

Uma operação de ponto flutuante pode ser uma operação de adição, subtração, multiplicação ou divisão, que utiliza operandos de ponto flutuante com precisão simples ou dupla. A métrica Flops não se baseia no número de instruções executadas, como o Milhões de instruções por segundo (MIPS), mas no número de operações aritméticas em ponto flutuante das instruções executadas (RAUBER; RÜNGER, 2010). Isto quer dizer que instruções que não executam operações sobre valores em ponto flutuantes não têm efeito no Flops. Além disso, existem outros fatores relevantes na performance do

computador para medir a velocidade do cálculo de pontos flutuantes, como a performance de Entrada/Saída, coerência de *cache* e a hierarquia de memória.

Estas influências significam que supercomputadores, em geral, são apenas capazes de uma pequena fração do seu desempenho teórico. O desempenho teórico (*Rpeak*) é o pico de performance teórica que a máquina consegue atingir (SINDI, 2013), ele representa o limite superior da performance. O cálculo do *Rpeak* de uma máquina pode ser obtido a partir da Fórmula 2.2.

Rpeak : Número de nós x Número de *sockets* por nó x Número de *cores* por *socket* x Frequência do processador x Número de operações por ciclo (2.2)

O número de operações por ciclo é o número de operações que um processador faz em um ciclo de relógio, levando em consideração os processadores atuais e que as operações são geralmente feitas em precisão dupla, utilizam-se 4 operações por ciclo (2 adições e multiplicações) para os cálculos de *Rpeak*.

O uso do Flops apresenta um incoveniente quando se compara diferentes tipos de operações em ponto flutuante. Operações como divisão e raiz quadrada são mais custosas, porém são contadas da mesma maneira como operações de multiplicação e adição que são mais rápidas de executar (RAUBER; RÜNGER, 2010). Contudo, a métrica Flops é bastante adequada quando se utiliza versões que executam as mesmas operações em ponto flutuante.

2.4.2 SpeedUp e Eficiência

A complexa interação entre *hardware* e *software* determina como a aplicação vai explorar os recursos computacionais, influenciando no ganho de performance destes sistemas. As métricas de medição de desempenho de um ambiente paralelo ajudam a formular hipóteses sobre o comportamento do sistema, auxiliando em possíveis melhorias. As que se destacam são o *speedup* e a eficiência.

Ao usar o dobro de recursos computacionais, pode-se aceitavelmente esperar que o programa execute duas vezes mais rápido. Contudo, isso raramente acontece com programas paralelos, devido a variedade de sobrecargas associadas com o paralelismo (GRAMA et al., 2003). Sendo a quantificação exata dessas sobrecargas crucial para a compreensão do desempenho do programa paralelo. Quando se está interessado em saber o quanto de performance foi alcançado na paralelização de determinado problema em relação a sua versão sequencial, utiliza-se o *speedup*.

O *SpeedUp* é uma grandeza adimensional (por ser uma relação de grandezas de mesma dimensão) que representa qual o ganho que se tem ao utilizar um certo número de processadores para resolver determinado problema. Seu cálculo é obtido através da divisão entre o tempo serial e o tempo em paralelo, utilizando dois ou mais processadores, de uma aplicação.

$$S(n) = \frac{\text{tempo de execução serial}}{\text{tempo de execução paralela}} = \frac{t_s}{t_p} \tag{2.3}$$

A situação típica é que a aplicação paralela nem alcançe o *speedup* linear que é igual ao número de elementos processadores, já que existe uma sobrecarga adicional para o gerenciamento do pralelismo com as trocas de dados entre os elementos processadores (RAUBER; RÜNGER, 2010).

Uma outra métrica para avaliação de desempenho de programas paralelos é a eficiência. A eficiência indica a fração de tempo que o processador é utilmente utilizado (RAUBER; RÜNGER, 2010), ou seja indica o grau de aproveitamento dos recursos computacionais. É calculada pela razão entre o *speedup* e a quantidade de processadores utilizados no processamento em paralelo.

$$E = \frac{S(n)}{(n)} * 100 \tag{2.4}$$

Em um sistema paralelo ideal o *speedup* é igual ao número de elementos processadores e a eficiência é igual a 1 (100%). Contudo, na prática o *speedup* é menor que o número de processadores e a eficiência fica entre zero e um, dependendo da efetividade do uso dos processadores (GRAMA et al., 2003).

2.5 Redes de Interconexão

A configuração de um *cluster* pode variar em termos de opções arquiteturais utilizadas: nós computacionais, redes de interconexão e sistemas de armazenamento são as principais delas. Segundo Navaux, Rose e Pilla (2011), a rede de interconexão desempenha um papel muito importante influenciando diretamente na eficiência da troca de informações nos multicomputadores. Desta maneira esta comunicação é essencial para o processamento sincronizado e o compartilhamento de dados (NULL; LOBUR, 2010) principalmente em arquiteturas paralelas de memória distribuída.

Se não houver um bom balanceamento entre processamento e comunicação neste tipo de arquitetura, os nós computacionais poderão ficar ociosos devido à interconexão

desperdiçando desta maneira recursos enquanto espera por dados de outros nós. Por outro lado, a rede pode se tornar financeiramente inviável devido ao seu alto custo, tornando o custo final do *cluster* dispendioso.

2.5.1 Métricas de Redes

Para um melhor desenvolvimento da estrutura de rede é necessário a análise de alguns conceitos, principalmente os parâmetros de redes que afetam a velocidade com que as mensagens podem ser transferidas entre dois computadores interligados, são eles latência e taxa de transferência de dados ponto a ponto (COULOURIS; DOLLIMORE; KINDBERG, 2007).

Latência é o tempo decorrido após uma operação de envio ser executada e antes que os dados comecem a chegar ao seu destino (COULOURIS; DOLLIMORE; KINDBERG, 2007). A latência é composta por vários fatores, dentre eles o atraso no acesso a rede, tempo de processamento gasto pelos serviços de comunicação do sistema operacional no processos de envio e recepção e etc. Já a taxa de transferência é a velocidade com que os dados podem ser transferidos entre dois computadores de uma rede (COULOURIS; DOLLIMORE; KINDBERG, 2007). Normalmente medida em bits por segundo (bps) é determinada principalmente pelas características físicas da rede.

Para a construção de soluções paralelas distribuídas é desejável canais de comunicação com baixa latência e máxima largura de banda (COELHO, 2012), que por sua vez devem antender a requisitos como escalabilidade, facilidade de atualização, alta disponibilidade e custo/benefício.

2.5.2 Topologias de Redes

Geralmente a interconexão de rede pode ser classificada como estática ou dinâmica (EL-REWINI; ABD-EL-BARR, 2005). De acordo com Navaux, Rose e Pilla (2011), redes estáticas são usadas na maiora dos casos em multicomputadores. Uma rede é dita estática quando existe uma ligação direta e dedicada entre dois componentes, com o número de conexões variando de apenas um, em uma topologia tipo estrela (RAUBER; RÜNGER, 2010), até todos os nós completamente conectados.

Na rede dinâmica as conexões são estabelecidas conforme a necessidade (RAUBER; RÜNGER, 2010), exemplos deste tipo de rede são as baseadas em barramento ou baseadas em *switching*, que consiste em *switches* conectados por cabos (RAUBER; RÜNGER, 2010). Esse tipo de rede é usado tanto para sistemas com memória distribuída quanto para sistemas de memória compartilhada.

A topologia determina as características da rede (NAVAUX; ROSE; PILLA, 2011) e é um dos principais fatores que determinam a sobrecarga do custo de passagem de

mensagem (NULL; LOBUR, 2010). No caso ideal, a topologia corresponde exatamente ao padrão de comunicação da aplicação paralela que executa na máquina (NAVAUX; ROSE; PILLA, 2011) como a topologia em árvore binária que favorece a execução de algoritmos de divisão e conquista, ou malhas bidimensionais que favorecem a algoritmos que utilizem cálculos com matrizes.

Porém, não há como prever *a priori* os padrões de comunicação e cargas de todas as aplicações que irão executar no ambiente. Neste caso, para um melhor aproveitamento dos recursos, o número de mensagens requeridas e a distância que a mesma deve percorrer são itens que tendem a ser minimizados na proposta de redes para soluções paralelas de memória distribuída.

Redes baseadas na topologia em barramento são as mais simples e eficientes quando o custo é uma preocupação (NULL; LOBUR, 2010). Nelas os processadores compartilham o mesmo canal de comunicação (GRAMA et al., 2003). Apesar de ser a alternativa de menor custo, por se tratar de um canal compartilhado por todas as possíveis conexões, têm baixa tolerância a falhas e são altamente bloqueantes (NAVAUX; ROSE; PILLA, 2011). Quando um processador necessita comunicar-se com outro, ele aguarda até que o barramento esteja livre para propagar sua mensagem. Em uma comunicação simultânea, colisões são detectadas e os processadores voltam a tentar a comunicação após um período de tempo determinado aleatoriamente.

O uso de redes *switched* tornou-se mais comum em substituição ao barramento pois ela segmenta a rede para cada nó. Os *switchs* hoje chegam a suportar centenas de segmentos dedicados, desta forma na comunicação entre dois segmentos, o *switch* pega a mensagem e encaminha diretamente ao destinatário atuando como uma ponte. Isso permite muitas comunicações simultâneas na rede.

2.5.3 Tecnologias de Interconexão de Rede

A maioria dos *clusters* de alto desempenho utilizam soluções *System Area Network* (SAN) (WATANABE et al., 2007). SAN são redes de conexões de alta performance que podem conectar *cluster* de computadores com alta largura de banda e baixa latência. Por outro lado, de acordo com Coelho (2012), após a popularização dos *cluster beowulf*, a *Ethernet* em pequenas redes locais tornou-se bastante usada. Mas esforços na eliminação de gargalos de comunicação e a maximização da velocidade de transmissão estão presentes em tecnologias como Myrinet (DANTAS, 2005), Infiniband (COELHO, 2012), *Quadrics Network* (EL-REWINI; ABD-EL-BARR, 2005).

O padrão *Ethernet* é a tecnologia mais usada em redes locais e seu preço não é muito elevado. Especificada no padrão *Institute of Electric and Electronic Engineers* (IEEE) 802.3, provê velocidade de transmissão de 10 Mbit/s. Como todos os computa-

dores de uma rede *ethernet* estão conectados a uma mesma linha de transmissão, eles competem pelo acesso usando o protocolo *Carrier Sense Multiple Access with Collision Detection* (CSMA/CD) (EL-REWINI; ABD-EL-BARR, 2005). Com este protocolo qualquer máquina está autorizada a emitir sobre a linha a qualquer momento, porém cada máquina verifica antes se não existe nenhuma comunicação na linha antes de emitir. No caso de envio simultâneo as duas máquinas interrompem sua comunicação e esperam um prazo aleatório para o reenvio.

Com a necessidade do aumento de desempenho das redes locais surgiram os padrões *Fast Ethernet* que elevam o tráfego de dados a taxa nominal de 100 Mbit/s e o padrão *Gigabit Ethernet* cujos dados podem trafegar a taxa de 1 Gigabit/s mantendo a compatibilidade e características básicas do antigo padrão. Como atualmente as instalações *Ethernet* utilizam de *switchs* para criar um "canal dedicado" entre o emissor e o receptor, esta característica deixa o processo de comunicação sem colisões e o protocolo CSMA/CD, em alguns casos, é usado como controle de fluxo para a prevenção de estouro no uso dos *buffers* internos do *switch*.

O uso da rede *Switched Ethernet* promoveu um outro avanço na comunicação que foi a habilidade de enviar e receber dados ao mesmo tempo, também chamada de *Full-duplex*. Esta habilidade deve-se ao fato de redes deste tipo utilizarem cabos par trançados ou fibra óptica os quais utilizam condutores separados para envio e recebimento de dados.

2.6 *Benchmarks* para Análise de Desempenho

A performance do sistema computacional pode variar significativamente dependendo do programa utilizado. Portanto, o *benchmark* deve ser projetado para fornecer comparações justas e eficazes entre sistemas de computação de alto desempenho (EL-REWINI; ABD-EL-BARR, 2005). Para o *benchmark* ser significativo, ele deve avaliar o desempenho com fidelidade para o uso pretendido do sistema. Para tanto, diferentes *benchmarks* tem sido propostos e usados, incluindo as abordagens listadas a seguir.

Os *benchmarks* sintéticos são pequenos programas artificiais contendo uma mistura de instruções que são selecionados de tal modo a tornar-se representativos a uma grande classe de aplicações reais (RAUBER; RÜNGER, 2010). Geralmente *benchmarks* sintéticos não executam grandes operações em grandes conjuntos de dados. Sua desvantagem é que esse tipo de *benchmark* não é capaz de combinar o comportamento de grandes aplicações com suas complexas interações entre cálculos do processador e acessos a memória (RAUBER; RÜNGER, 2010).

Os *benchmarks* denominados kernel possuem pequenas partes relevantes de aplicações reais, o que tipicamente capturam grande parte do tempo de execução da aplicação. Sua vantagem em relação a programas reais é que são pequenos e fáceis de analisar (RAU-

BER; RÜNGER, 2010). Já *benchmarks* de aplicações reais compreendem programas inteiros que refletem uma carga de trabalho de um usuário padrão. Esse tipo de *benchmark* tem a vantagem de capturar todos os aspectos dos programas selecionados, produzindo resultados de desempenho significativos para o usuário (RAUBER; RÜNGER, 2010).

2.6.1 *Benchmarks* Paralelos

Visando a análise de desempenho de computadores paralelos, esse tipo de *benchmark* é aplicado em máquinas com múltiplos núcleos, processadores ou sistemas constituídos de várias máquinas. Geralmente são associados com as características de desempenho do *hardware* que avaliam, como por exemplo a capacidade de operações de ponto flutuante do processador.

O mais popular conjunto de *benchmarks* é o *System Performance Evaluation Cooperation* (SPEC) *benchmark suite* (RAUBER; RÜNGER, 2010). O SPEC é uma corporação formada para estabelecer, manter e apoiar um conjunto padronizado de *benchmarks* relevantes que podem ser aplicados as novas gerações de computadores de alto desempenho (EL-REWINI; ABD-EL-BARR, 2005). O SPEC já desenvolveu uma série de *benchmarks*, dos quais pode-se destacar os focados para computadores pessoais, sistemas de arquivos, servidores web e sistemas paralelos. O SPEC MPI2007 é o *benchmark* do SPEC para calcular o desempenho de uma gama de *clusters* em ponto flutuante utilizando MPI. Os programas deste *benchmark* são desenvolvidos a partir de aplicações MPI de usuário finais, diferentemente dos *benchmarks* sintéticos ou versões paralelizadas de *benchmark* sequenciais.

O *NAS Parallel Benchmarks* (NPB) é um conjunto de *benchmarks* para avaliação de desempenho de computadores paralelos. Seus problemas são baseados em aplicações e núcleos de aplicações paralelas de dinâmica de fluidos (NAS, 2013). A implementação original consistia de oito problemas focados em alguns aspectos da computação científica. O principal foco era na computação aerospacial, contudo muitos desses *benchmarks* tem uma relevância em várias áreas, sendo que muitas aparecem em várias aplicações do mundo real. As versões mais atuais incluem mais três *benchmarks* (NAS, 2013), entre as regras para a implementação de NAS *Parallel benchmarks* estão especificados que todas as operações de ponto flutuante devem ser executadas em precisão dupla e o *benchmark* deve ser implementado em linguagem C ou FORTRAN.

2.6.2 *Benchmarks* para Análise de Rede

De acordo com Botta, Pescape e Ventre (2005) pode-se dividir as ferramentas de análise de redes em: (1) ferramentas de estimativa de capacidade ponto-a-ponto, (2) ferramentas de estimativa de largura de banda e (3) ferramentas de medição da capacidade

da largura de banda e transferência de dados sobre *Transmission Control Protocol* (TCP). A seguir são apresentados algumas ferramentas de análise de redes que possuem seus projetos ativos na internet. Além de oferecerem diferentes características para uma análise da rede, essas ferramentas são *benchmarks* que usam grandes tranferências de dados sobre TCP e/ou *User Datagram Protocol* (UDP) para avaliar a taxa de transferência em um caminho ponto-a-ponto (VELÁSQUEZ; GAMESS, 2009).

O Netperf é um *software* de código aberto, originalmente desenvolvida pela *Hewlett Packard*, que fornece a largura de banda da rede através do teste entre duas máquinas em uma rede. Baseado no modelo cliente-servidor, seu foco é na transferência de uma massa de dados e na performance de solicitação/resposta utilizando os protocolos TCP ou UDP (VELÁSQUEZ; GAMESS, 2009). A saída de sua execução informa o tamanho do *socket* do *sender* e do *receiver* em bytes, o tamanho da mensagem enviada em bytes, a duração do teste e a largura de banda da rede (VELÁSQUEZ; GAMESS, 2009).

O *Distributed Internet Traffic Generator* (D-ITG) é uma plataforma de código aberto que segue o modelo cliente-servidor capaz de gerar tráfego a nível de pacote (VELÁSQUEZ; GAMESS, 2009). O D-ITG pode executar a medição da latência de ida e volta de um pacote, a latência de ida de um pacote, avaliação da taxa de perda de pacotes, medição da taxa de transferência e *jitter* da rede. É uma plataforma multi-componentes que pode trabalhar no modo simples, onde é permitido a geração de um fluxo simples de pacotes do cliente para o servidor, e no modo script que permite a criação de de vários fluxos simultâneos de um único cliente (VELÁSQUEZ; GAMESS, 2009).

2.6.3 *High Performance Linpack*

Principal ferramenta usada para classificar os computadores na lista do Top500, o HPL resolve um sistema de equações lineares com precisão dupla em computadores de memória distribuída. Além de fornecer o desempenho da máquina em Flops, sua saída serve tanto para quantificar a precisão da solução quanto o tempo gasto na computação. Esse sistema de equações lineares é do tipo $A.x = b$, onde A é uma matriz densa gerada aleatoriamente de dimensão N x N; e onde x e b são vetores de tamanho N. A matriz A é primeiramente fatorada como sendo o produto $A = L.U$, onde L e U representam as matrizes triangulares inferior e superior respectivamente (DONGARRA; WHALEY, 2008).

Esse *benchmark* permite ao usuário dimensionar o tamanho do problema e otimizar o *software* a fim de obter o melhor desempenho possível da máquina. Esse desempenho não reflete o desempenho "completo" de um determinado sistema, como nenhum número nunca pode. Ele, no entanto, reflete o desempenho de um sistema dedicado para resolver um sistema de equações lineares denso. Desde que o problema seja muito regular, o desempenho alcançado é bastante elevado, e o desempenho dá uma boa aproximação do

desempenho máximo (DONGARRA; WHALEY, 2008).

O algoritmo usado para resolver o sistema de equações deve cumprir a fatoração LU com pivotamento parcial. Em particular, o número de operações para o algoritmo deve ser $(2/3)n^3+O(n^2)$ em ponto flutuante e precisão dupla (DONGARRA; WHALEY, 2008), onde a fatoração requer $(2/3)n^3$ operações e as duas soluções triangulares totalizando n^2 operações cada, com a fase de fatoração dominando o tempo de computação a medida que N aumenta. O desempenho obtido via HPL é abaixo do desempenho teórica pois deve-se levar em conta aspectos como o algoritmo, o tamanho do problema, a linguagem de alto nível, a implementação, o nível de esforço humano usado para otimizar o programa, a capacidade do compilador para otimizar, a idade do compilador, o sistema operacional, a arquitetura do computador e as características de *hardware* (DONGARRA; WHALEY, 2008).

O HPL requer a instalação da biblioteca *Basic Linear Algebra Subprograms* (BLAS) que é uma coleção de rotinas que proveêm as bases para implementar programas que façam operações com matrizes e vetores. As funcionalidades da biblioteca são divididas em três níveis: no primeiro estão as funcionalidades que executam operações de escalares com escalares, escalares com vetores e vetores com vetores; no nível dois encontram-se as operações entre matrizes e vetores; e o nível três implementa operações de matrizes com matrizes (MILANI, 2010).

Por ser uma biblioteca eficiente, portável e amplamente utilizada, a BLAS costuma ser empregada no desenvolvimento de softwares de álgebra linear de alta qualidade. Além disso, por ser disponibilizada gratuitamente, existem versões como a biblioteca *Automatically Tuned Linear Algebra Software* (ATLAS) a qual permite gerar automaticamente uma versão otimizada da BLAS para a arquitetura escolhida (WHALEY; PETITET, 2005).

3 A Ferramenta de *Benchmark*

O presente trabalho propõe uma ferramenta para a execução de um *benchmark*, fornecendo indicadores para a avaliação de desempenho em *clusters* de computadores, compreendendo fatores como tempo de processamento, operações em ponto flutuante, métricas de aplicações paralelas, largura de banda e taxa de transferência da rede.

Apesar de ser uma ferramenta que utiliza uma interface Web, devido a necessidade de dados sobre o *hardware* das máquinas do *cluster*, esta ferramenta terá como servidor uma máquina constituinte do *cluster* no qual se deseja realizar o *benchmark*. Isso permitirá que a ferramenta possa configurar e distribuir os arquivos necessários à execução entre as máquinas.

3.1 Justificativa do Projeto

Como foi apresentado no Capítulo 2, existem algumas ferramentas que realizam carga no sistema para executar testes de desempenho e avaliar a influência desta arquitetura no desempenho. Porém, muitas vezes, resultados detalhados são requeridos para aferir relações entre indicadores, com a finalidade de descobrir oportunidades de melhoria. Em vista disso, a Tabela 1 apresenta alguns indicadores obtidos com essas ferramentas.

Tabela 1 – Quadro comparativo entre os *benchmarks*.

Benchmarks	Indicadores de Desempenho					
	Wallclock	Precisão	Flops	Flops Teórico	SpeedUp/ Eficiência	Latência e largura de banda
HPL	X	X	X			
Processamento de Imagem	X					
SPEC	X		X			
NAS Parallel Benchmarks	X		X			
Netperf						X
D-ITG						X
Ferramenta Proposta	X		X	X	X	X

Uma característica presente nas ferramentas citadas na Tabela 1, reside no fato de que para sua execução, o usuário deve realizar no ambiente uma série de configurações para compilar e executar o *benchmark*. Muitas vezes, desestimulando a prática de *benchmark* em usuários novos na área. Além disso, os indicadores obtidos com estas aplicações, por

si só não ajudam a responder outros tipos de questionamentos levantados com o teste de desempenho.

Desta forma, além de possuir uma interface web que isola o usuário da complexidade dos procedimentos de *benchmark*, dispor os resultados em gráficos para uma melhor visualização, a ferramenta proposta procura disponibilizar uma série de indicadores para que a maior parte dos questionamentos levantados no *benchmark* possam ser esclarecidos. Alguns destes indicadores apresentados pela ferramenta proposta, para serem obtidos utilizando as outras aplicações, o usuário teria que executar o *benchmark* e calcular manualmente os indicadores derivados para cada iteração ou cenário de teste.

Tomando como exemplo a aplicação de processamento de imagem, para a obtenção de outros indicadores além do tempo de processamento, o usuário tem que executar a aplicação serialmente. Depois, para cada cenário e variação de processo ou *thread*, o usuário deve calcular o *speedup* e eficiência. Mas, para obter também o tempo de comunicação, a aplicação de processamento de imagem tem que ser compilada com outra ferramenta que permita a obtenção dos indicadores de comunicação e o processo descrito acima deve ser repetido, desta vez extraindo o percentual de comunicação de cada iteração realizada no processo.

Isto quer dizer que, para obter outros indicadores com as aplicações listadas acima, o usuário deve além de calculá-los para cada cenário de teste que deseje, deve integrá-las para obter aqueles indicadores de que a aplicação sozinha não consegue apresentar. Esta circunstância torna o processo de *benchmark* uma tarefa extenuante, além de diminuir a quantidade de inferências que o usuário pode obter de sua arquitetura em uma avaliação de desempenho.

Além do ganho de tempo na automação dos procedimentos de *benchmark*, que concede ao usuário mais tempo em sua análise dos resultados do que na própria ação de obtê-los, a ferramenta proposta traz consigo um complemento em sua arquitetura para a execução de outros códigos paralelos. Esse complemento permite ao usuário executar algoritmos paralelos que deseje e obtenha os resultados da execução desses algoritmos em gráficos com a mesma metodologia utilizada nas aplicações anteriores.

3.2 Planejamento do Projeto

Como etapa inicial para o desenvolvimento da ferramenta de *benchmark*, apresenta-se aqui o planejamento do projeto que norteou a execução das etapas necessárias para o seu desenvolvimento. Esse planejamento objetivou exatamente definir o que seria feito neste trabalho e prover subsídios que serviram como guia para a construção das etapas seguidas na metodologia. Além disso, o planejamento visou delimitar o escopo do projeto, ou seja, o que seria e o que não seria compreendido pelo mesmo e pelas atividades seguintes.

Considerando o objetivo deste trabalho, seu produto é composto de outras aplicações que são usadas como carga no sistema para a coleta de indicadores de desempenho e que servem para prover um ponto de partida para a avaliação de desempenho da arquitetura do *cluster* por parte dos usuários. Essas aplicações usadas provêm indicadores compatíveis com os indicadores propostos no objetivo do trabalho. Por outro lado, as mesmas são de configuração flexível e natureza distinta para não orientar o usuário a uma análise de desempenho viciada.

Em consequência da elaboração da especificação da ferramenta, a mesma foi implementada através da seleção de tecnologias apropriadas cuja implementação concreta foi submetida a teste em uma arquitetura de *cluster*, visando a validação e conclusões sobre a mesma. No que diz respeito a abrangência da ferramenta, a partir do objetivo proposto, definiu-se como objetos da mesma o estudo de *benchmarks* similares, execução automatizada, interface Web com baixo grau de complexidade de uso, indicadores referentes a processamento e uso de rede, implementação de um algoritmo paralelo para executar em conjunto com as aplicações que fazem parte do produto deste trabalho e implementação de um complemento para execução das aplicações paralelas adicionadas pelo usuário.

Também foi definido que não se configura como objeto do trabalho itens como indicadores de desempenho referentes a outros aspectos que fazem parte da arquitetura de um *cluster*, como por exemplo a hierarquia de memória, configurações ou otimizações por parte do usuário e execução para outros tipos de conexão de rede como *infiniband* e *myrinet*.

Seguindo o planejamento proposto nesta seção, buscou-se definir as atividades focando em macro entregas que foram etapas funcionais da ferramenta deste trabalho. Com o objetivo de detalhar como as atividades foram conduzidas, a seção 3.3 aborda a metodologia adotada no desenvolvimento da ferramenta.

3.3 Metodologia para Desenvolvimento da Ferramenta

Para a elaboração da ferramenta de *benchmark*, será apresentado aqui a metodologia utilizada objetivando a definição e apresentação clara das atividades que foram realizadas para alcançar o desenvolvimento completo da ferramenta. A Figura 1, derivada do planejamento abordado na Seção 3.2, contempla a sequência de passos que foram necessários para alcançar o objetivo proposto no trabalho.

Para que a ferramenta fosse contemplada, inicialmente foi feita uma pesquisa de *benchmarks* similares de acordo com os indicadores de desempenho propostos como objetivo do trabalho. Esses resultados ajudaram na decisão da montagem da arquitetura da ferramenta proposta, a qual contém aplicações já existentes e validadas no meio acadêmico. As aplicações constituíntes da ferramenta são:

a) o *benchmark* HPL (DONGARRA; WHALEY, 2008),

b) um algoritmo de processamento de imagem,

c) o Ifstat (YONG; ABDULLAH; ABDULLAH, 2012) e

d) o *Integrated Performance Monitoring* (IPM) (SKINNER, 2005).

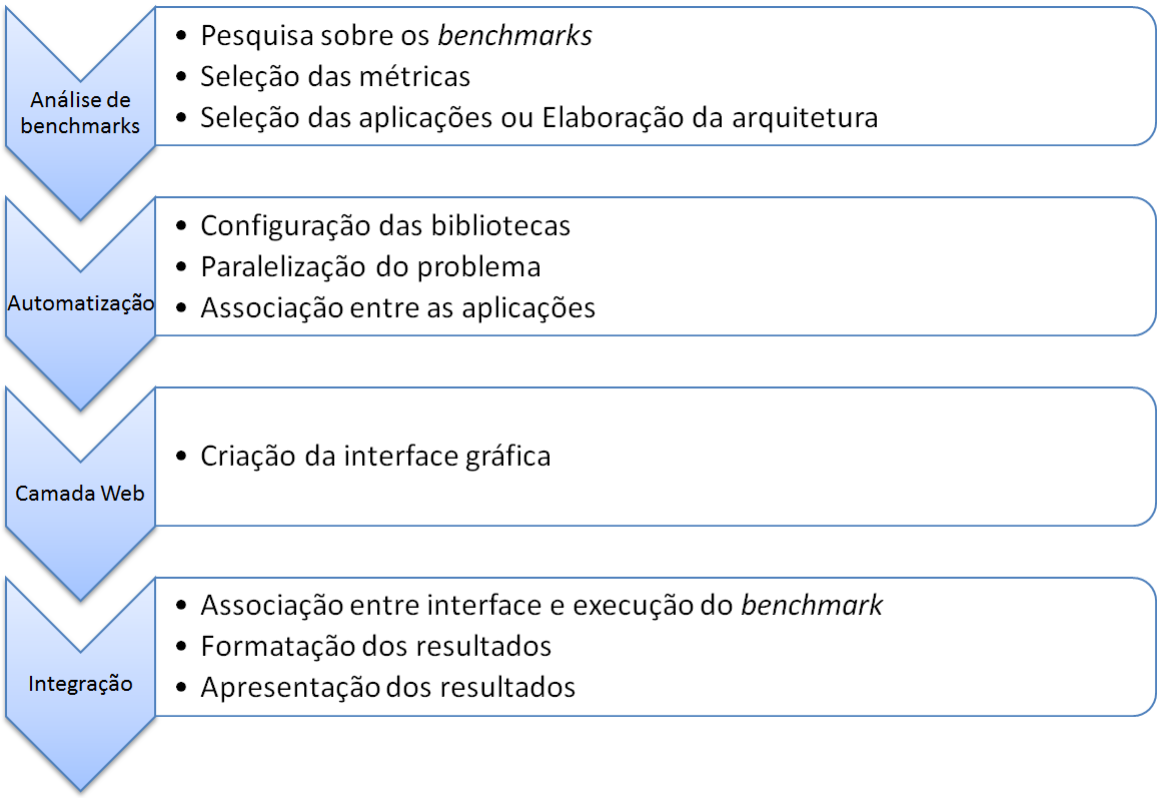


Figura 1 – Representação das etapas da metodologia utilizada para desenvolvimento da ferramenta.

Deste modo, para manter-se em sintonia com a premissa de que o *benchmark* deve ser de fácil execução e que proporcione ao usuário um resultado mais detalhado em relação as aplicações utilizadas, foi necessário uma maneira de tirar esta tarefa do usuário. Então, para prover uma melhor análise de desempenho da arquitetura, todo o processo de execução foi automatizado, cabendo a ferramenta a correta configuração e execução do *benchmark*.

Nesta etapa, foram criados os *scripts* responsáveis pela configuração de parâmetros referentes a blibliotecas, flags e outros requisitos necessários para a execução das aplicações internas. Foi nesta etapa também que o algoritmo de processamento de imagem é paralelizado, utilizando o paralelismo de dados e o modelo *Master/Slave* para a exploração das características da arquitetura. Os resultados mais detalhados são alcançados ao associar a execução do *benchmark* HPL e da aplicação de processamento de imagem com

as outras duas aplicações, permitindo assim a expansão do tipo de análise que pode ser feito pelo usuário em relação a execução das mesmas.

O Ifstat e o IPM dão suporte e trabalham no monitoramento e contagem dos parâmetros associados com os indicadores que se deseja obter. A primeira aplicação coleta informações do tráfego da rede, o que ajuda a compreender o comportamento de comunicação do algoritmo na arquitetura. Já a segunda, é uma ferramenta de *proffiling* para aplicações paralelas que provê pouca sobrecarga sobre a execução da aplicação enquanto coleta dados. Desta forma, em oposição a execução dessas ferramentas de maneira isolada, relacionar a execução e saída dessas aplicações proporciona um entendimento melhor e mais detalhado do resultado desses *benchmarks* na arquitetura.

Em seguida, foi criada a camada de interface Web que serve como uma ponte entre o usuário e a ferramenta que irá executar o *benchmark*. Optou-se pelo desenvolvimento de uma interface simples na qual o usuário estivesse apenas a um *click* de executar o *benchmark* e o resultado fosse apresentado de uma maneira interativa. A interface foi desenvolvida desta maneira para justamente liberar o usuário de boa parte das configurações adicionais, cabendo a ferramenta a automatização completa da execução do *benchmark* como número de nós total, carga e número de testes.

Depois de concluida a parte de automatização e interface gráfica, partiu-se para a integração entre estas duas etapas. Nela a execução do *benchmark* foi associada com a interface web e os resultados das execuções do *benchmark* foram formatados e organizados para serem apresentados ao usuário de forma gráfica, com uma melhor legibilidade do que a saída padrão proporcionada pelas aplicações de maneira isolada.

3.4 Arquitetura da Ferramenta de *Benchmark*

Para atingir seu objetivo, a ferramenta proposta neste trabalho contempla internamente quatro aplicações que são essenciais para alcançar os resultados desejados. Essas aplicações têm suas saídas exploradas e relacionadas, de modo a prover indicadores de desempenho que permitam um melhor entendimento do comportamento da arquitetura analisada.

Além disso, apesar de isentar o usuário de configurações adicionais, no que tange a execução das aplicações principais deste *benchmark*, foi disponibilizado também uma extensão na ferramenta. Esta extensão foi implementada objetivando permitir a execução de alguma aplicação paralela, com a qual o usuário tenha interesse em realizar o *benchmark*.

Observando a Figura 2, percebe-se que uma das aplicações utilizadas é o *benchmark* HPL. A opção por utilizar o HPL deve-se ao fato do mesmo ser um *benchmark* consolidado e mundialmente usado para ranquear os supercomputadores mais potentes do mundo.

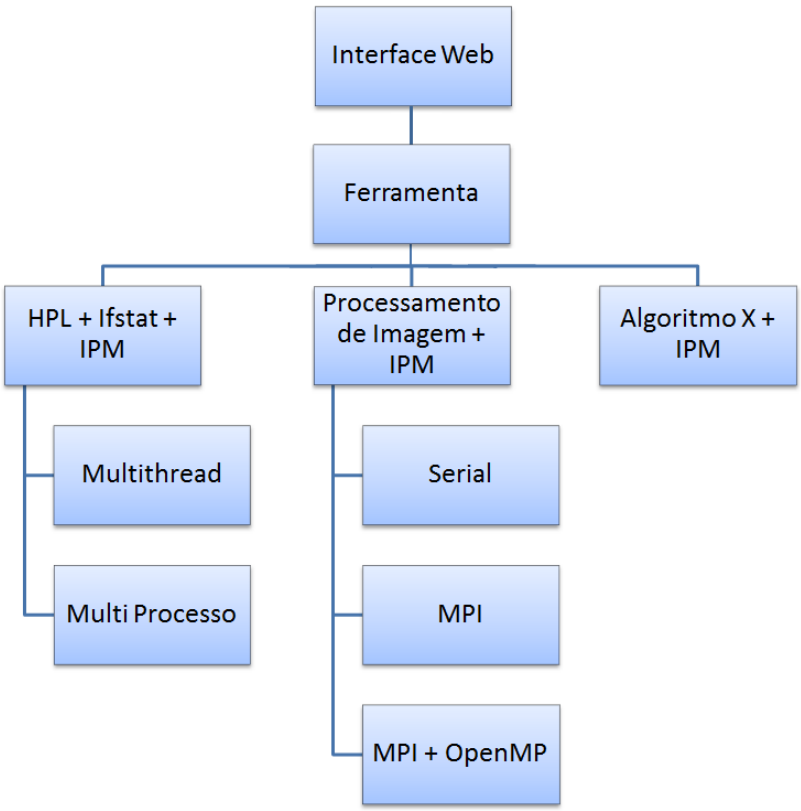


Figura 2 – Aplicações constituintes da ferramenta de *benchmark*.

Apesar de apresentar o desempenho em ponto flutuante, a precisão da solução e o tempo gasto no processamento, sua saída não apresenta nenhuma relação de sua execução com o tráfego da rede no nó mestre ou com as taxas de transferência, o que permitiria evidenciar gargalos de rede na arquitetura do *cluster*. Por isso que durante sua execução, faz-se necessário a utilização das ferramentas Ifstat e IPM.

Desta forma, para permitir uma melhor análise de desempenho com os indicadores apresentados pela aplicação, na hora da configuração das aplicações para execução, o *benchmark* HPL é compilado em duas versões (*Multithread* e Multi processo) conforme visto na Figura 2. Isso permite ao usuário comparar e perceber a vantagem em utilizar a abordagem com *threads* ou processos em algoritmos que executem na sua arquitetura.

Para ampliar a análise obtida com os resultados do *benchmark* HPL, outra aplicação utilizada no *benchmark*, para provê mais indicadores de desempenho, é a aplicação de processamento de imagem. Por ser uma aplicação utilizada em situações reais, que requer um grau de processamento considerável e possui uma problemática diferente da resolvida pelo *benchmark* HPL, esta aplicação proporciona saber o comportamento da arquitetura para problemas desta classe.

O algoritmo em questão é o filtro de mediana, esse algoritmo possibilita a redução de ruídos na imagem substituindo o valor de cada *pixel* da imagem pelo valor da mediana

da vizinhança ao redor do *pixel*. Na sua execução o primeiro passo é ordenar os valores da vizinhança que pode ser 3x3, 5x5, 7x7 ou até maiores se forem necessários, calcular a mediana deste conjunto e então substituir o valor do *pixel* pela mediana. Esse algoritmo é computacionalmente custoso, porque encontrar a mediana significa ordenar a vizinhança para cada *pixel* percorrido na imagem.

A partir da versão serial do problema, foi obtida a versão paralela utilizando MPI e uma versão híbrida utilizando MPI e OpenMP, seguindo a mesma abordagem utilizada com o *benchmark* HPL. Ambas as versões (MPI e híbrida) estão implementadas sobre o modelo *Master/Slave*, sem que o processo denominado *Master* participe da computação. No início da computação, o processo mestre é responsável pela divisão dos dados da imagem, sua distribuição para os trabalhadores e posterior recebimento dos dados para apresentar o resultado. Na execução, apenas é utilizada a ferramenta IPM para ajudar a prover resultados mais detalhados.

Os indicadores apresentados pelo *benchmark* são relevantes para usuários que executem aplicações no modelo de programação *Master/Slave* ou utilizem problemas com estruturas bidimensionais em suas arquiteturas, pois as aplicações utilizadas neste *benchmark* utilizam destes modelos em seus algoritmos. Por isso, questões como comunicação no nó mestre e tempo de comunicação, geralmente evidenciam problemas relacionados com gargalos entre o mestre e os outros trabalhadores. Esse tipo de problema não permite a máxima utilização dos nós computacionais, o que culmina em baixa eficiência e disperdícios de recursos.

A aplicação denominada “Algoritmo X” na Figura 2, representa o complemento implementado para que o usuário possa executar algum algoritmo paralelo adicional que deseje. Esse complemento visa eliminar a complexidade de executar algum algoritmo extra utilizando a ferramenta. Neste caso, não é necessário o usuário configurar a ferramenta para executar seu algoritmo, este complemento configura a execução desse algoritmo provendo o mesmo tipo de indicadores que foram extraídos da aplicação de processamento de imagem anteriormente executado.

Para que o usuário possa usufruir deste complemento, a ferramenta leva em consideração que o código fornecido pelo usuário é executável e escalável para o número máximo de processos configurável pela ferramenta (número de *cores* x número de máquinas). Além disso, deve-se seguir alguns cuidados como: utilizar algoritmos paralelos escritos em linguagem C e salvos na extensão *.c, utilizar MPI e caso esse algoritmo contenha diretivas OpenMP, o mesmo não deve possuir diretivas comentadas ou funções de setar número de *threads*.

3.5 Interface Web da Ferramenta

A interface web, que serve de primeiro contato entre o usuário e o *benchmark*, foi desenvolvida utilizando *HyperText Markup Language* (HTML) e *Cascading Style Sheets* (CSS) para organização e apresentação dos dados, *Hypertext Preprocessor* (PHP) para a programação no servidor (*server side*) e JavaScript para processamentos no cliente (*client side*). A página inicial está dividida em cinco seções, que são: *Home*, O *Benchmark*, Executar *Benchmark*, Visualizar Resultados e Autores. Além disso, são disponibilizados mais dois itens no menu para o acesso as áreas de *upload* e exclusão de arquivos que foram adicionados pelo usuário.

A interface oferece uma execução limpa, isolando o usuário da complexidade e do esforço de otimização de diversas aplicações disponíveis ou das aplicações que fazem parte do *benchmark*. Isto é alcançado graças ao fato da ferramenta aproveitar da configuração interna da máquina servidora, que ao mesmo tempo é um nó participante do *cluster*. Apesar da ferramenta possuir uma interface web, é recomendado que apenas um usuário por vez execute o *benchmark*, pois como os testes exigem o máximo de desempenho das máquinas, múltiplas execuções irão atrapalhar umas as outras baixando os números dos indicadores ou até inviabilizando a execução.

A camada web também dispõe de uma interface implementada para a submissão de códigos paralelos que serão executados junto ao *benchmark*. Esta seção disponibiliza ao usuário o uso do complemento implementado, visto na Figura 2, para a execução de códigos que o usuário tem interesse em executar no *benchmark*. Nele o usuário pode adicionar tanto o algoritmo paralelo, quanto algum arquivo de parâmetro que seja requisitado pelo mesmo. Então, na hora da execução, a ferramenta será responsável por configurar itens como o número de testes e os cenários de teste, variando o número de processos e *threads*.

Para fazer um *upload* de maneira correta, o usuário deve escolher se deseja enviar um arquivo fonte do código paralelo ou um arquivo de parâmetro, respeitando as regras requeridas pelo complemento para a execução de códigos adicionais. Caso seja um arquivo fonte, o mesmo deve ter seu caminho informado na área de arquivo e caso possua parâmetros para sua execução, seus nomes devem ser escritos na caixa de texto da área de parâmetro separados por vírgula. É de responsabilidade do usuário a correta informação da ordem dos parâmetros para a execução do algoritmo de maneira correta. Caso um dos parâmetros seja um arquivo, o mesmo pode ser enviado utilizando a área de arquivo, após seu caminho ter sido informando.

Na interface de exclusão de códigos enviados, o usuário pode retirar algum código adicionado ao *benchmark*. Nele são listados todos os códigos que foram enviados, neste caso o usuário pode escolher e excluir o código que desejar. Quando o código é selecionado e o

pedido de exclusão é requisitado, esta ação também afetará os parâmetros associados ao código que está sendo excluído. Isto que dizer que tanto o código quanto os parâmetros enviados para a execução do mesmo serão excluídos. Por isso, caso haja mais de um algoritmo enviado que utilize o mesmo arquivo de parâmetro, o usuário deve adicionar este arquivo de parâmetro novamente.

3.6 Fluxo de execução da Ferramenta de *Benchmark*

Além de diminuir o esforço do usuário na configuração das diversas aplicações disponíveis, a ferramenta proporciona ao usuário a visualização dos resultados dos indicadores de desempenho de um *benchmark* de forma mais comunicativa e detalhada. Então a Figura 3 mostra a interação entre o usuário e a ferramenta, através da sua interface web, demonstrando o fluxo desde o pedido de execução até a visualização dos resultados.

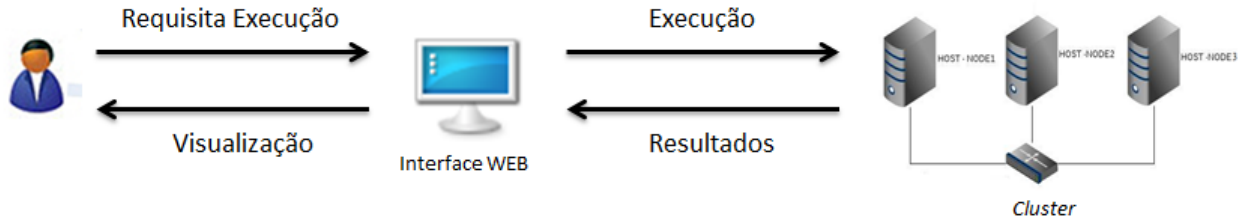


Figura 3 – Fluxo de utilização da ferramenta.

No processo de execução do *benchmark*, a ferramenta realiza alguns passos claramente definidos para a correta execução da mesma. A Figura 4 apresenta o fluxo das macro etapas realizadas pela ferramenta onde uma etapa é realizada logo após a outra. Na sequência, cada etapa abordada na Figura 4 será detalhada.

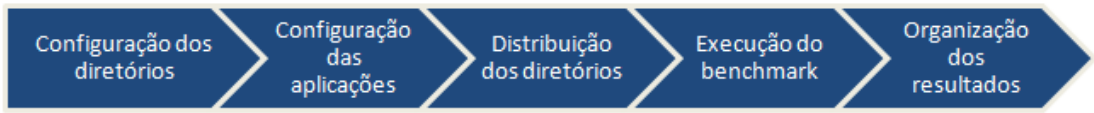


Figura 4 – Passos realizados na execução da ferramenta de *benchmark*.

1. **Configuração dos diretórios:** Essa é a primeira etapa, onde a ferramenta cria no nó mestre o diretório padrão que irá receber os arquivos necessários para a execução do *benchmark*. Este passo é importante pois, além de configurar os caminhos que serão usados nos passos seguintes, cria a raíz de trabalho para o *benchmark* na pasta do usuário usado na comunicação via *Secure Shell (SSH)* do *cluster* para que os arquivos de saída não se espalhem no sistema conflitando com outros arquivos.

2. **Configuração das aplicações:** Após configurado os diretórios que serão a raíz de trabalho da ferramenta, é feita a configuração dos arquivos essenciais utilizados. Como a ferramenta utiliza-se internamente de outras aplicações, detalhadas na seção 3.4, para a realização de seu *benchmark*, nesta etapa questões como configurações das versões a serem executadas, quantidade de execuções, cargas de trabalho, destino das saídas e número de máquinas participantes são definidas.

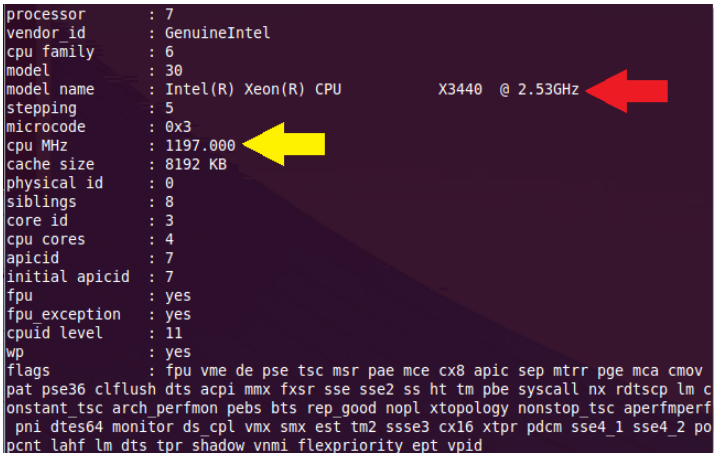
3. **Distribuição dos diretórios:** Com os diretórios completos e as aplicações prontas para serem executadas, os diretórios então são distribuídos para as máquinas constituíntes do *cluster* a partir dos arquivos de configurações gerados na etapa anterior.

4. **Execução do *benchmark*:** A ferramenta executa cada versão das aplicações, configuradas nas etapas anteriores, de acordo com a quantidade de amostras estabelecidas também nas etapas anteriores. Cada versão é executada uma após a outra e ao término de cada execução, os resultados são dispostos em arquivos para que posteriormente sejam formatados.

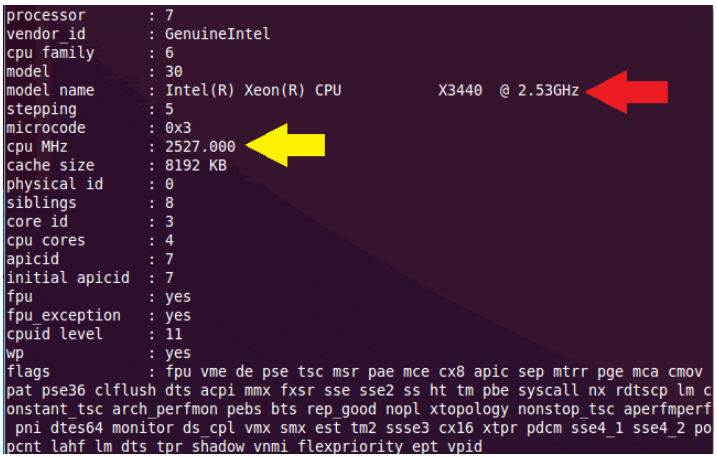
5. **Organização dos resultados:** Com os resultados em arquivos é possível formatá-los usando comandos de manipulação de arquivos nativos do sistema operacional linux, para que os dados possam ser usados na construção dos gráficos e tabelas visualizados pelo usuário. Ao final da execução também é criado um histórico para visualização de resultados prévios.

Por utilizar bastante de comandos e programas nativos do sistema operacional Linux, para a correta execução e obtenção de dados coerentes e consistentes, é necessário que alguns cuidados sejam tomados.

- O usuário do cluster deve estar devidamente configurado com acesso remoto sem senha (SSH) para as outras máquinas.
- O arquivo /etc/hosts deve estar mapeado com os *Internet Protocol*'s (IP) e nomes de todas as máquinas do *cluster* e que irão participar do *benchmark*.
- O MPI deve estar instalado em todas as máquinas do *cluster*.
- Deve-se desabilitar a funcionalidade do processador conhecida como Central Processing Unit *(CPU) throttling*. Essa funcionalidade é uma técnica que automaticamente ajusta a frequência de operação do processador para conservar energia ou reduzir o nível de calor gerado pelo chip. Isso porque a frequência máxima do processador é usada para os cálculos de desempenho teórico, ao usar os dados de frequência em escala reduzida, tanto o valor do desempenho teórico quanto os de eficiência não serão corretos.



(a) *CPU throttling* habilitado



(b) *CPU throttling* desabilitado

Figura 5 – Informações de um processador a partir de comandos nativos do Linux. A Figura 5a apresenta as informações obtidas com o técnica habilitada, enquanto que a Figura 5b apresenta as informações com a técnica desabilitada.

A Figura 5a mostra o processador de uma máquina com a frequência dinâmica, a seta vermelha indica a frequência máxima de operação do processador enquanto que a seta amarela indica a frequência reduzida. Fazendo as contas para calcular o desempenho teórico utilizando a equação 2.2, ao obter a frequência o resultado seria 19,152 GigaFlops (1 *Socket* x 1 Nó x 4 *Cores* x 1197 MHz de Frequência x 4 Operações por ciclo).

Na Figura 5b é possível perceber a frequência dinâmica desabilitada na mesma máquina, tanto a frequência máxima quanto a obtida pelo sistema são iguais. Desta forma utilizando a equação 2.2 para o novo cálculo do desempenho teórico é obtido 40,432 GigaFlops (1 *Socket* x 1 Nó x 4 *Cores* x 2527 MHz de Frequência x 4 Operações por ciclo). Isto implica que, ao utilizar a frequência dinâmica dada pelo sistema para os cálculos de desempenho teórico máximo e eficiência na execução do *benchmark* HPL, o resultado obtido não seria coerente com o resultado real, o que poderia causar confusão ao até uma má interpretação dos indicadores e da arquitetura estudada.

4 Validação da Ferramenta de *Benchmark*

Para analisar a eficácia da ferramenta, os testes foram realizados no *cluster multi-core* do laboratório de HPC do SENAI CIMATEC. O *cluster* é composto de 8 máquinas Hewlett-Packard (HP) ProLiant DL120 G6, com processador *Intel*®*Xeon*®*QuadCore* X3440 2.53 GHz com *HyperThreading*, 2 TB de armazenamento, placa de rede NetXtreme® BCM5723 *Gigabit Ethernet*, 8GB de memória RAM, sistema operacional Linux 3.2.0-32-generic X86-64 GNU/Linux Ubuntu 12.04LTS, conectadas através de uma rede *switched ethernet* com um *switch* GTS®modelo 73.1724S 10/100 Mbit/s utilizando cabos par trançado UTP categoria 5e. Na compilação foi utilizado o compilador mpicc do OpenMPI versão 1.7.2, que é uma implementação *open source* do MPI-2, além do OpenMP versão 3.1 contido no compilador GCC 4.8.1.

Por possuir uma interface web que auxilia o usuário na execução do *benchmark*, para conduzir o processo de validação da ferramenta, primeiramente foi proposto a utilização da técnica de teste funcional da engenharia de *software*. Também conhecida como teste de caixa preta, refere-se a testes que são conduzidos na interface do *software* (PRESSMAN, 2006), esses testes avaliam o comportamento externo do componente da aplicação sem considerar detalhes da implementação.

Para tanto, pretendia-se utilizar a ferramenta *open source* chamada selenium, que é um conjunto de ferramentas para automação de testes em aplicações do tipo web. Essa ferramenta simula a interação do usuário com o sistema, verificando o comportamento do sistema e de seus processos internos, através de suas interfaces e da análise de suas saídas ou resultados.

Contudo, a ferramenta proposta neste trabalho não possue formulários com muitas informações para interação com o usuário, o que não tornaria os resultados da utilização do selenium representativos para validar a ferramenta. Além disso, o componente mais importante desta ferramenta reside nos *scripts* de configuração e execução do *benchmark*. Por isso, definiu-se que para a validação, a estrutura interna da aplicação seria testada, permitindo a escolha de componentes específicos para os testes.

Os testes foram realizados ao nível de unidade. Segundo Pressman (2006), nos testes de unidade focaliza-se o esforço de verificação na menor unidade da aplicação, ou seja, subrotinas ou pequenos trechos de códigos constituem o alvo deste tipo de teste. Assim sendo, para a validação da ferramenta proposta, os testes foram divididos em três tipos: os testes para validar os retornos de funções, que alimentam as variáveis responsáveis pelas configurações das aplicações, os testes para os laços e os testes para os desvios condicionais.

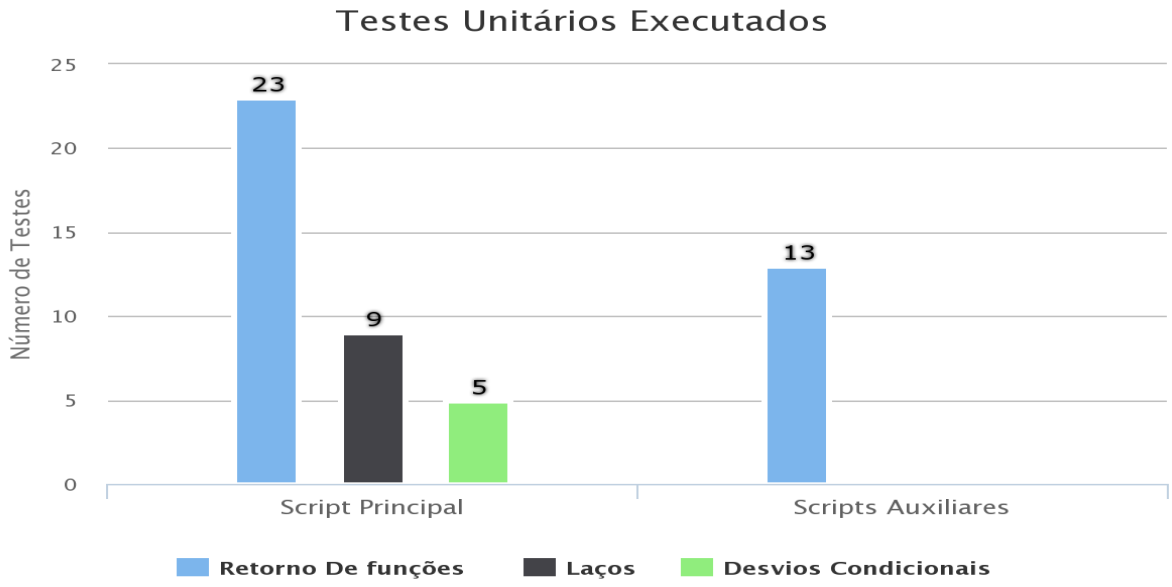


Figura 6 – Resultado do teste unitário nos *scripts* de execução.

Os testes que incidiram sobre o valor de retorno das funções foram realizados nos *scripts* principal e auxiliares. O *script* principal é responsável pela execução das aplicações no *benchmark*, ou seja, pela execução de cada iteração e das coletas dos dados dessas execuções, enquanto que os *scripts* auxiliares são responsáveis pela criação de arquivos de configurações extras, usados pelas aplicações. Os valores de retorno dos testes de validação foram comparados com valores previamente estabelecidos, denominados valores esperados, configurando-se falha quando os resultados fossem distintos e sucesso quando fossem iguais. No total foram realizados 50 testes sobre funções e configurações dos *scripts*, sendo 37 testes no *script* principal e 13 testes nos *scripts* auxiliares.

O gráfico representado pela Figura 6 demonstra a quantidade dos testes de validação executados. Na execução dos testes, houve apenas duas divergências, isso porque a função testada refere-se ao valor da frequência do processador. Nesse teste, o valor usado foi a frequência dada pelo fabricante do processador, já o valor de frequência obtido foi a frequência real de operação do processador e do barramento do sistema, conforme medida pelo retorno da função. Apesar de divergente, este resultado necessariamente não se configura como uma falha, já que a freqüência operacional pode ser ligeiramente maior ou menor que a frequência esperada para o processador. Diferenças de frequência em 1% são devido a ligeiras variações na fabricação de componentes que são considerados dentro das especificações.

Esta divergência consequentemente afeta o cálculo do desempenho teórico que utiliza essa frequência, cujo valor calculado com a frequência divergente é diferente do valor esperado. Nos testes realizados sobre os laços, foram testados as condições lógicas e os valores dos dados iterados imediatamente abaixo e acima dos laços do *script*. Todos os

testes foram realizados no *script* principal obtendo sucesso em sua totalidade, as condições lógicas de parada e verificação não possuiam erros, além dos valores iterados ao final do laço possuirem o valor esperado. Para as condições lógicas existentes em desvios condicionais, os testes demonstraram que tanto para a condição falsa quanto para a condição verdadeira, o comportamento do código segue o fluxo apropriado de execução quando os desvios são utilizados.

4.1 *Benchmark* HPL

Para o *testbed* descrito no começo do Capítulo, internamente a aplicação configurou para o *benchmark* HPL o seguinte quadro de testes.

Tabela 2 – Parâmetros do *benchmark* HPL.

HPL	Multi Processo	Multithread
Número de Testes	4	4
Número de Máquinas Utilizadas	8	8
Tamanho de N	82316	82316
Processos por Máquina	4	1
Threads por Processo	1	4

O valor de N, ordem do sistema linear utilizado pelo HPL, calculado para os testes deve ser limitado pela quantidade de memória total do sistema, contudo o objetivo é encontrar o maior tamanho do problema que caiba na memória. Geralmente 90% de memória configura-se como uma boa escolha de N, pois se o valor de N for muito baixo, não irá resultar em trabalho suficiente para os processadores, o que dará resultados ruins e baixa eficiência. Se o valor de N exceder a capacidade de memória ou não deixar memória suficiente para os outros processos do sistema operacional, ocorrerá *swap* e o desempenho irá cair sensivelmente. Para encontrar o N apresentado na Tabela 2 a aplicação utiliza a equação 4.1 que devolve um valor que utiliza cerca de 90% da memória (SINDI, 2013).

$$N = \sqrt{\frac{(\text{Memória em GBytes} * 1024 * 1024 * 1024 * \text{Número de Nós})}{8}} * 0.90 \qquad (4.1)$$

Ao executar o *benchmark* HPL, será obtida a saída correspondente a Figura 7, com informações referentes ao tamanho do problema, tempo gasto na resolução do problema, a performance em gigaflops, a precisão do problema e o número de testes realizados. A Figura 7a representa a saída para a execução do *benchmark* HPL Multi Processo enquanto que a Figura 7b representa a saída do *benchmark* HPL Multi*thread*.

```
================================================================================
HPLinpack 2.0  --  High-Performance Linpack benchmark  --   September 10, 2008
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   82316
NB     :     224
PMAP   : Row-major process mapping
P      :       2
Q      :      16
PFACT  :   Crout    Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout    Right
BCAST  :   1ring
DEPTH  :       0
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be            1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00C2C4       82316   224     2    16            9429.19              3.944e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0005420 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00C2R4       82316   224     2    16            9442.90              3.938e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0005376 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2C4       82316   224     2    16            9448.64              3.936e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0005420 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2R4       82316   224     2    16            9470.66              3.926e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0005376 ...... PASSED
================================================================================

Finished      4 tests with the following results:
              4 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

(a) HPL Multi Processo

```
================================================================================
HPLinpack 2.0  --  High-Performance Linpack benchmark  --   September 10, 2008
Written by A. Petitet and R. Clint Whaley,  Innovative Computing Laboratory, UTK
Modified by Piotr Luszczek, Innovative Computing Laboratory, UTK
Modified by Julien Langou, University of Colorado Denver
================================================================================

An explanation of the input/output parameters follows:
T/V    : Wall time / encoded variant.
N      : The order of the coefficient matrix A.
NB     : The partitioning blocking factor.
P      : The number of process rows.
Q      : The number of process columns.
Time   : Time in seconds to solve the linear system.
Gflops : Rate of execution for solving the linear system.

The following parameter values will be used:

N      :   82316
NB     :     224
PMAP   : Row-major process mapping
P      :       2
Q      :       4
PFACT  :   Crout    Right
NBMIN  :       4
NDIV   :       2
RFACT  :   Crout    Right
BCAST  :   1ring
DEPTH  :       0
SWAP   : Mix (threshold = 64)
L1     : transposed form
U      : transposed form
EQUIL  : yes
ALIGN  : 8 double precision words

--------------------------------------------------------------------------------

- The matrix A is randomly generated for each test.
- The following scaled residual check will be computed:
      ||Ax-b||_oo / ( eps * ( || x ||_oo * || A ||_oo + || b ||_oo ) * N )
- The relative machine precision (eps) is taken to be            1.110223e-16
- Computational tests pass if scaled residuals are less than                16.0

================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00C2C4       82316   224     2     4            5831.64              6.377e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0003546 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00C2R4       82316   224     2     4            5830.60              6.378e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0003620 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2C4       82316   224     2     4            5830.55              6.378e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0003546 ...... PASSED
================================================================================
T/V                N    NB     P     Q               Time                 Gflops
--------------------------------------------------------------------------------
WR00R2R4       82316   224     2     4            5830.62              6.378e+01
--------------------------------------------------------------------------------
||Ax-b||_oo/(eps*(||A||_oo*||x||_oo+||b||_oo)*N)=        0.0003620 ...... PASSED
================================================================================

Finished      4 tests with the following results:
              4 tests completed and passed residual checks,
              0 tests completed and failed residual checks,
              0 tests skipped because of illegal input values.
--------------------------------------------------------------------------------

End of Tests.
================================================================================
```

(b) HPL *Multithread*

Figura 7 – Saída padrão do *benchmark* HPL.

Diferentemente da saída padrão proporcionada pelo *benchmark* HPL, a ferramenta proposta apresenta ao usuário não só o desempenho obtido nos testes, mas relaciona o desempenho obtido com o desempenho teórico calculado para a execução. Desta relação se obtém a eficiência, apresentada na Tabela 3, que representa o quanto a arquitetura consegue atingir daquele limite máximo estabelecido pelo desempenho teórico, quanto maior a eficiência melhor é o aproveitamento dos recursos. Além disso, outros indicadores coletados na execução deste *benchmark* são apresentados ao usuário.

A Tabela 3, apresenta a média dos resultados obtidos a partir dos testes com a execução do *benchmark* HPL pela ferramenta proposta neste trabalho. Logo em seguida serão apresentados os gráficos gerados pela ferramenta.

Tabela 3 – Resultado médio da execução do *benchmark* HPL.

HPL	Multi Processo	Multithread
Número de Testes	4	4
Desempenho Teórico	323.456 Gigaflops	323.456 Gigaflops
Desempenho Médio	39.36 Gigaflops	63.77 Gigaflops
Eficiência	12.00%	19.00%
Tempo Médio de Execução	9447.85 Segundos	5830.85 Segundos
Tempo Total de Execução	37791.4 Segundos	23323.4 Segundos
Tempo Total de Comunicação	30233.12 Segundos	16326.38 Segundos
Percentual de Comunicação	80.72%	70.95%
Max Taxa de Transferência	94.8073 Mbits	94.3891 Mbits
Latência Média da Rede	0.271 milisegundos	0.271 milisegundos

Pode-se notar que o usuário não obtém apenas as informações oriundas da execução do *benchmark* HPL isoladamente para sua avaliação de desempenho. A ferramenta fornece também o tempo de resolução do problema, o tempo total de execução, o tempo médio de execução e a partir do percentual de comunicação, o tempo de comunicação na execução do problema é apresentado. A máxima taxa de transferência, a latência média da rede no momento da execução e o percentual da comunicação apresentados, ajudam a identificar possíveis gargalos proveniente da rede de interconexão na resolução do problema.

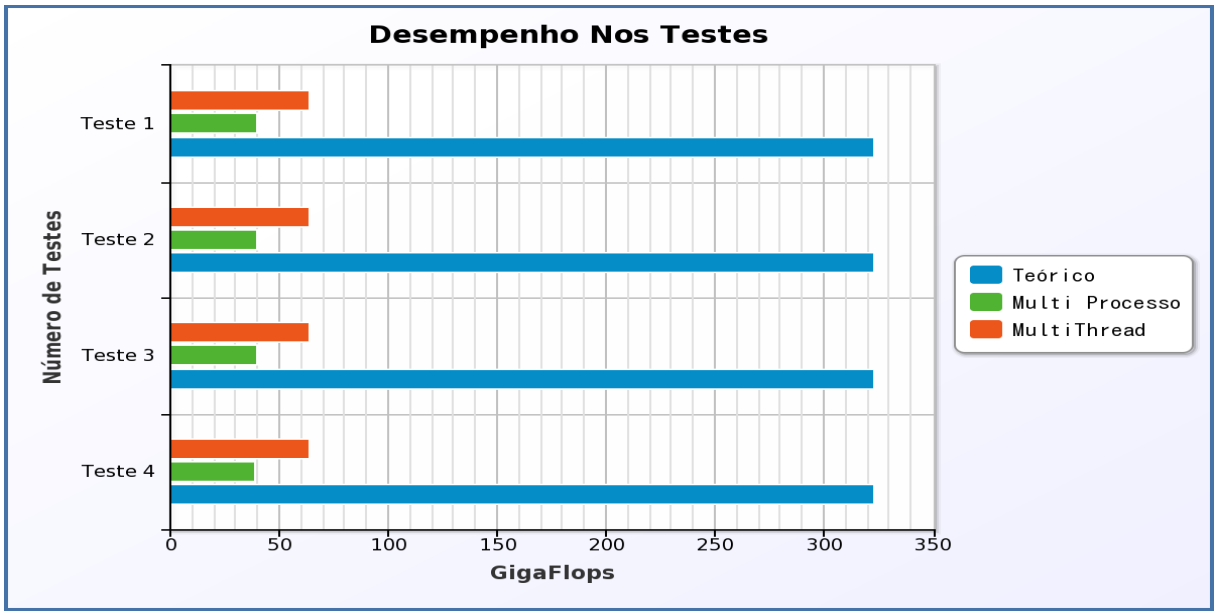


Figura 8 – Desempenho em Gigaflops do *testebed* com o *benchmark* HPL.

Fazendo uma análise do *testbed* utilizado no *benchmark*, de acordo com a Figura 8, tanto o desempenho obtido via execução multi processo quanto o obtido via multi*thread* não alcançaram nem 50% do desempenho teórico, porém o desempenho da versão multi*thread* com média de 63 gigaflops teve uma performance 1,6 vezes melhor que

desempenho obtido com a versão multi processos, que alcançou um desempenho médio de 39 gigaflops. Essa disparidade é refletida na eficiência apresentada na Figura 9, com média de 12% para os testes com multi processos e uma média de 19% para os testes com multi*thread*.

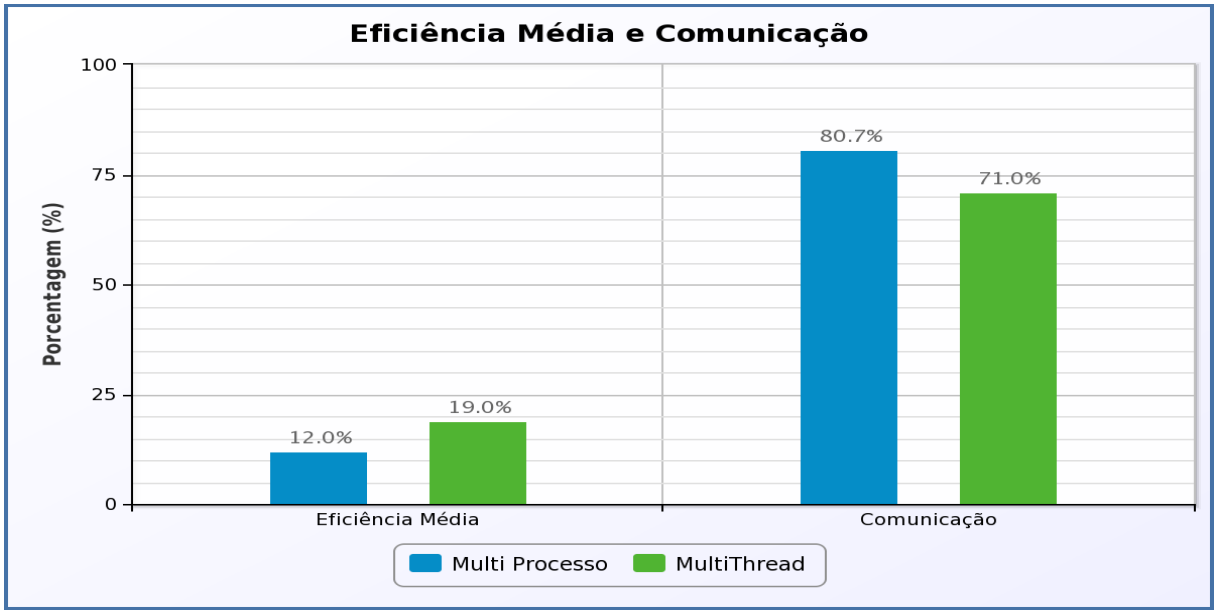


Figura 9 – Eficiência obtida e percentual de comunicação do *testebed* com o *benchmark* HPL.

Isso pode ser entendido observando ainda a Figura 9, pois além da abordagem multi*thread* ter números melhores nos quesitos desempenho e eficiência, seu percentual de comunicação é menor que a abordagem multi processos. Isto implica que os processos passam menos tempo esperando para enviar ou receber dados e seus esforços são mais concentrados em tempo de computação, pois além do trabalho ser dividido por um número menor de processos participantes, o que gera uma carga maior por processo, existem menos processos comunicantes que disputarão tanto o meio físico de transmissão quanto o nó mestre. Dada a natureza do problema, uma outra vantagem refere-se ao fato de se trabalhar com *threads* ao invés de processos, pois elas tiram proveito das especificidades internas da máquina com trocas de contexto menos custosas e o aproveitamento do compartilhamento de dados.

Considerando as vantagens citadas acima, a abordagem multi*thread* demora menos tempo para a execução do problema, conforme visto na Figura 10. Seu tempo médio chega a ser 1,6 vezes menor do que o tempo médio necessário para a resolução do problema utilizando a abordagem por multi processos. Tanto os resultados visto na Figura 9, quanto na Figura 10, podem ser relacionados com o padrão de comunicação do *benchmark* apresentado nas Figuras 11 e 12.

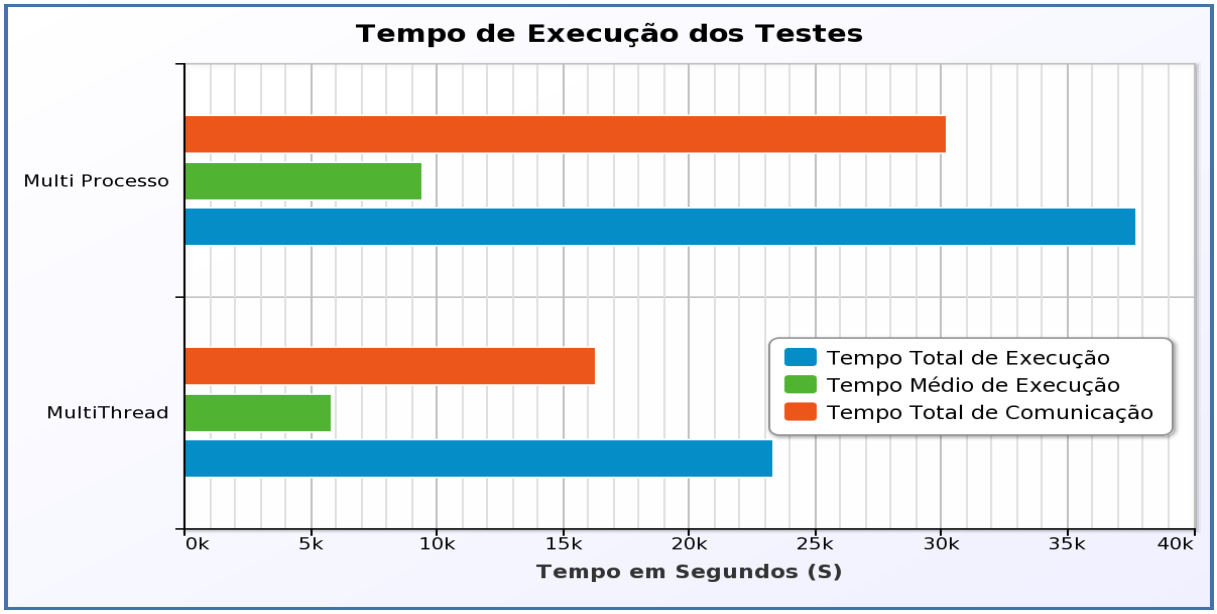


Figura 10 – Tempo de execução do *testebed* com o *benchmark* HPL.

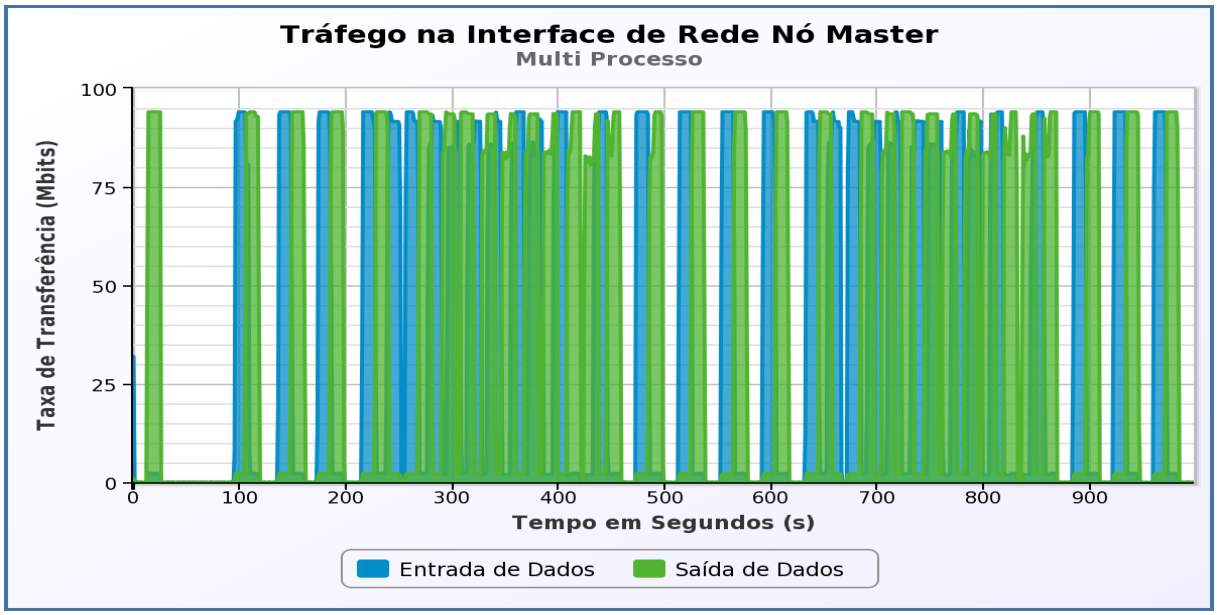


Figura 11 – Tráfego na interface de rede do nó mestre do *testebed* com o *benchmark* HPL versão Multi Processo.

Como pode ser visto nas Figuras 11 e 12, o *benchmark* encontra um limite no enlace de rede pertencente ao nó mestre saturando o mesmo em ambas abordagens. Ainda de acordo com a Tabela 3, a taxa de transferência máxima obtida pelas abordagens multi processo e multi*thread* no enlace são respectivamente 94.8073 Mbits e 94.3891 Mbits, por isso observando a Figura 11 pode-se perceber que existe um fluxo constante de entrada e saída na interface de rede do nó mestre o que pode gerar uma situação de comunicação que leva ao alto percentual de comunicação, baixa eficiência e alto tempo para resolução

do problema.

Na situação de comunicação gerada na abordagem multi processo o nó mestre não consegue distribuir dados para todos os processos antes de começar a receber resultados de outros processos. Apesar do fato da rede proporcionar comunicação bidirecional com a máxima taxa de transferência da rede para as duas máquinas que se comuniquem, não é garantia que no momento do processamento a mesma máquina esteja enviando e recebendo dados do nó mestre. Isso quer dizer que vai existir uma espera de envio de resultado de alguns processos que ficarão ociosos esperando uma "folga" do nó mestre para receber os seus dados, isto vai aumentar o percentual de comunicação do problema em relação ao tempo de resolução a medida que aumentarmos o número de processos, já que no momento a rede se apresenta como limitador do ganho de desempenho do sistema.

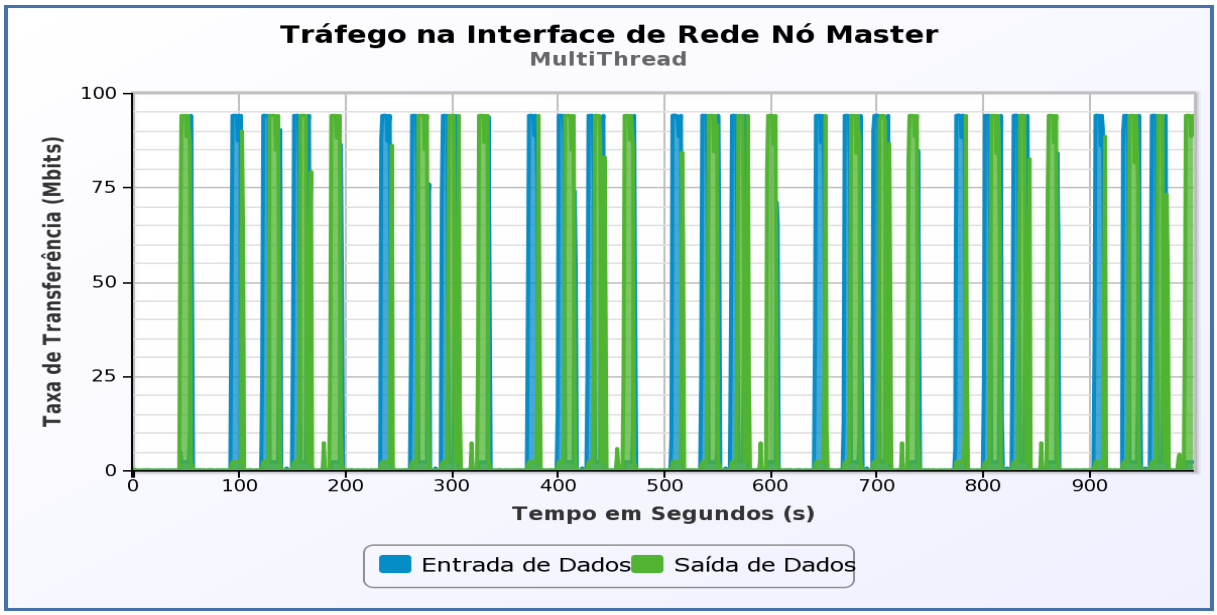


Figura 12 – Tráfego na interface de rede do nó mestre do *testebed* com o *benchmark* HPL versão *Multithread*.

Já na abordagem multi*thread*, representado na Figura 12, apesar do percentual de comunicação ser grande é visível a diferença produzida pelo número menor de processos comunicantes. A comunicação se concentra em picos de envio e recebimento com pouca sobreposição de entrada e saída de dados no enlace do nó mestre, em alguns momentos é possível observar até um ligeiro envio ou recebimento de dados com curta duração. Apesar de diminuir em cerca de 10% o tempo de comunicação em relação a abordagem multi processos, a abordagem multi*thread* ainda é afetada pelo limite do enlace a medida que em muitos pontos é visível o atraso na saída ou entrada de dados.

4.2 Processamento de Imagem

Para a aplicação de processamento de imagem, o *benchmark* utiliza a configuração detalhada na Tabela 4. A configuração de execução da aplicação de processamento de imagem na versão híbrida, explora o nível máximo de *threads* que podem ser obtidos no sistema. Como no *testbed* utilizado cada nó possui 1 *socket* e 1 processador *quadcore*, fica visível ao sistema a existência de 4 núcleos físicos no processador, então a aplicação configura o número de processos e *threads* até obter o número total de "processadores" visíveis ao sistema. Essa característica permite ao usuário visualizar algum tipo de vantagem em determinado número de processos/*threads* para a solução do problema na arquitetura.

Tabela 4 – Parâmetros da aplicação de processamento de imagem.

Processamento de Imagem	Serial	MPI	Híbrido	
Número de Testes	4	4	4	
Número de Máquinas Utilizadas	1	8	8	
Tamanho da Imagem (pixels)	30000x30000	30000x30000	30000x30000	
Processos por Máquina	1	4	1	2
Threads por Processo	1	1	4	2

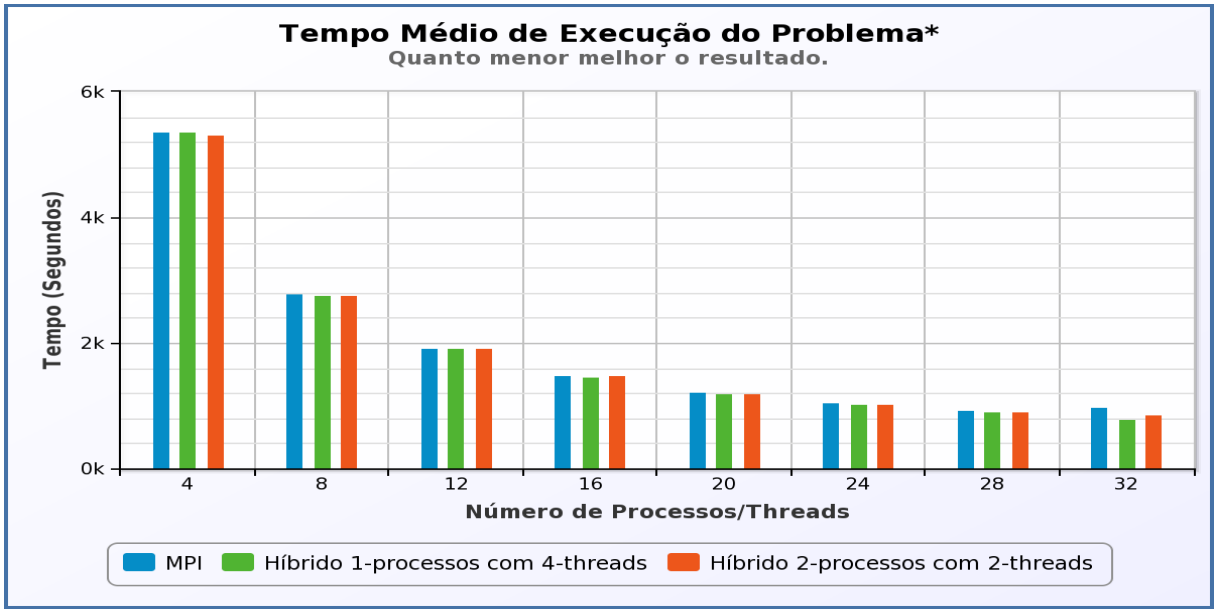


Figura 13 – Tempo de execução do *testbed* com a aplicação de processamento de imagem.

Um fato que vale ressaltar é que a aplicação de processamento de imagem em questão utiliza o paradigma *master/slave* em que existe um processo mestre que coordena a execução, sem participar da computação, e os processo trabalhadores que executam as tarefas. Por isso que na hora de executar, o processo mestre é alocado em um nó e os trabalhadores executam em outro nó, até que ao esgotar o número de nós restantes do

cluster, são alocados processos trabalhadores no nó que abriga o processo mestre. Além de forçar o uso da rede, isso evita que ocorram interferências do sistema operacional com perdas de posse do processador pelo processo mestre, devido ao número de processos requisitantes.

De acordo com Figura 13, pode-se observar que o tempo de processamento da aplicação diminuiu a medida que processos vão sendo adicionados, além disso o que acontece é que a diminuição no tempo ocorre de forma similar para cada abordagem utilizada. A redução no tempo, visto na Figura 13, reflete no *speedup* e eficiência obtidos com a aplicação utilizando o *testbed*, que podem ser vistos nas Figuras 14 e 15, com a Figura 14 mostrando o *speedup* alcançado pela aplicação na arquitetura.

Utilizando até 6 nós trabalhadores (24 processos/*threads*), o *speedup* alcançado pelas abordagens é idêntico para todas, crescendo de uma maneira constante em relação a adição de nós no processo. A partir da utilização de 7 nós trabalhadores (28 processos/*threads*), a aplicação começa a obter desempenhos distintos para as abordagens executadas.

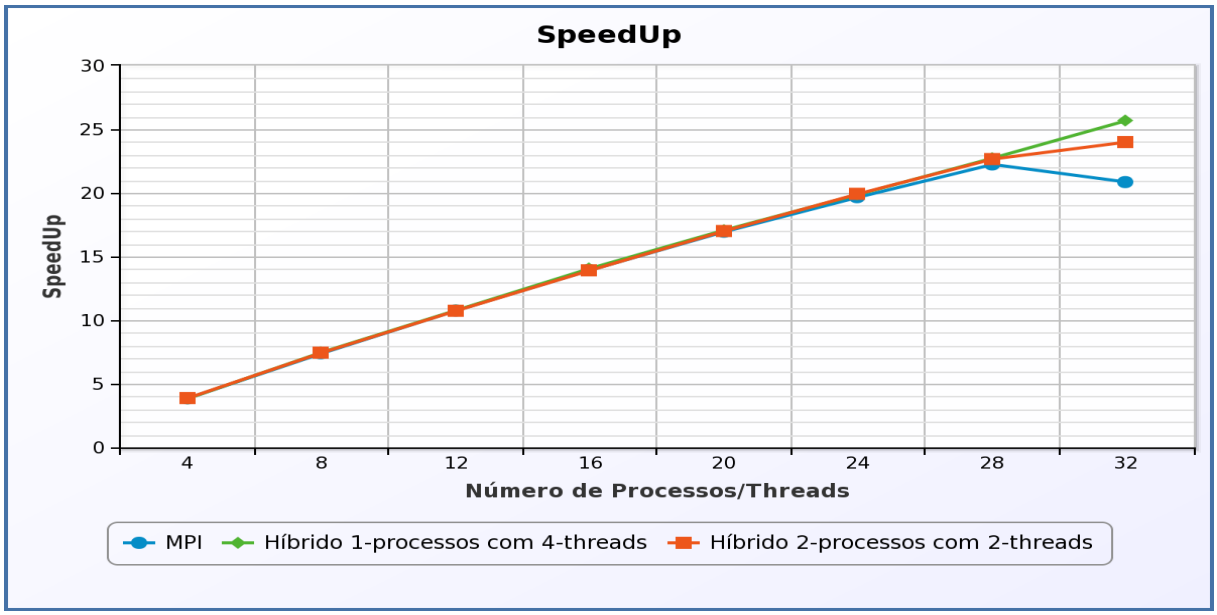


Figura 14 – *Speedup* do *testbed* com a aplicação de processamento de imagem.

Ao adicionar processos trabalhadores no nó mestre (32 processos/*threads*) o ganho de desempenho obtido com a abordagem que utiliza apenas processos MPI sofre uma queda, demonstrando menor ganho de desempenho do que com a sua execução anterior com menos nós. Já a abordagem híbrida que utiliza apenas 1 processo alcança o melhor ganho de desempenho ao executar o algoritmo, chegando a ser 25 vezes mais rápido que a abordagem serial.

Na Figura 15 foi possível observar o mesmo comportamento visto com a Figura

14, com a utilização de 6 nós trabalhadores houve uma redução da eficiência idêntica para as abordagens, porém mesmo utilizando a capacidade máxima testada pela aplicação a eficiência ficou acima dos 60%. A partir da utilização de 7 nós trabalhadores, a abordagem que utiliza apenas processos MPI acabou tendo uma queda mais acentuada que as outras abordagens na adição de processos trabalhadores no nó que aloca o processo mestre, porém a abordagem híbrida com 1 processo ficou com uma eficiência por volta dos 80%, mostrando que mesmo quando o nó que possui o processo mestre recebe processos trabalhadores, tem-se um bom aproveitamento dos recursos computacionais testados utilizando uma abordagem de *threads* ao invés de processos.

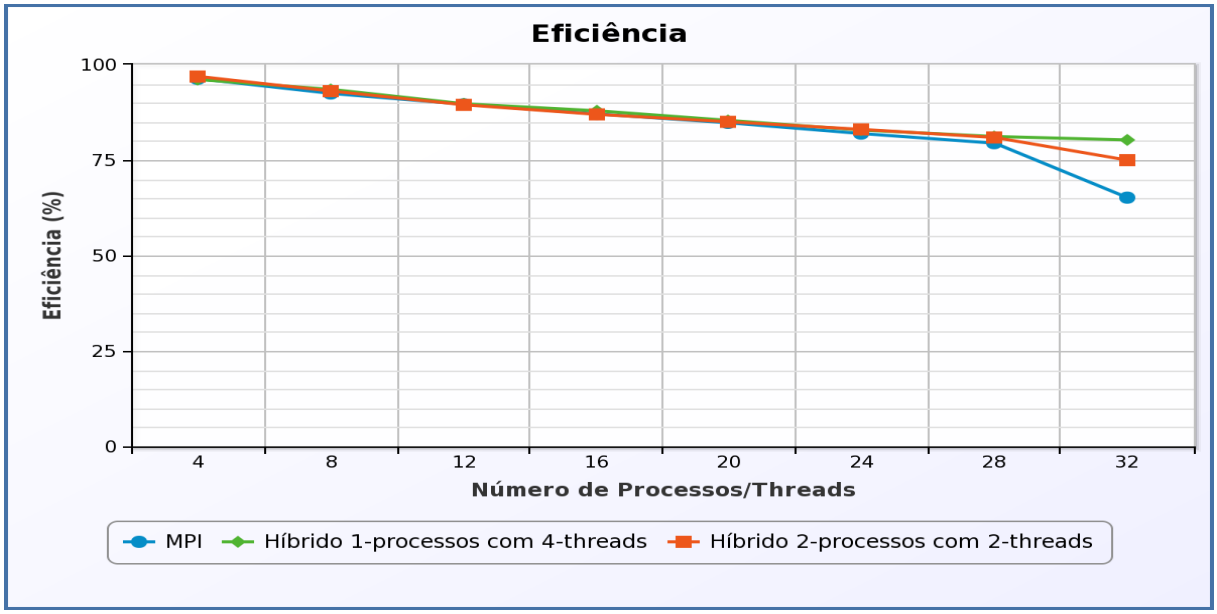


Figura 15 – Eficiência do *testbed* com a aplicação de processamento de imagem.

Tanto o ganho de desempenho como a eficiência obtida pelo algoritmo na arquitetura, denotam que a abordagem utilizando apenas processos MPI sofre quando o nó mestre aloca processos trabalhadores na execução do problema. Ao executar o problema desta maneira, é possível perceber que o processo mestre responsável pelo envio e recebimento de dados alocado no nó mestre sofre com a interferência dos trabalhadores a medida que o sistema operacional promove trocas de contextos entre os processos no processador.

Na execução do *benchmark* anterior, o *benchmark* HPL, foi possível perceber que a arquitetura em questão sofre com um limite imposto pelo enlace de rede que conecta o mestre aos trabalhadores, pois este *benchmark* é uma aplicação que utiliza bastante comunicação. Desta maneira, sua execução acabou sendo prejudicada pela baixa largura de banda admitida pelo enlace de rede, obtendo assim baixos índices de desempenho medidos pela aplicação.

Já com a aplicação de processamento de imagem, de acordo com as Figuras 14 e 15, percebe-se que mesmo com esse limite imposto pelo enlace, a aplicação consegue elevados

speedups e uma eficiência alta. Isso deve-se ao fato de que a aplicação em sua execução distribui totalmente os dados no início da execução entre os trabalhadores, não havendo assim comunicação para a troca de resultados intermediários, apenas comunicação no inicío e fim do processamento. Esse padrão de comunicação pode ser entendido observando as Figuras seguintes.

A partir da Figura 16, percebe-se a primeira vista que a abordagem híbrida que utiliza 1 processo possui o maior percentual de comunicação em relação as outras abordagens, seguida pela outra abordagem híbrida e depois pela abordagem de processos MPI. É importante resaltar que comunicação aqui quer dizer o tempo total em que os processos estão em alguma primitiva de envio ou recebimento de dados do padrão MPI em relação ao tempo total de processamento.

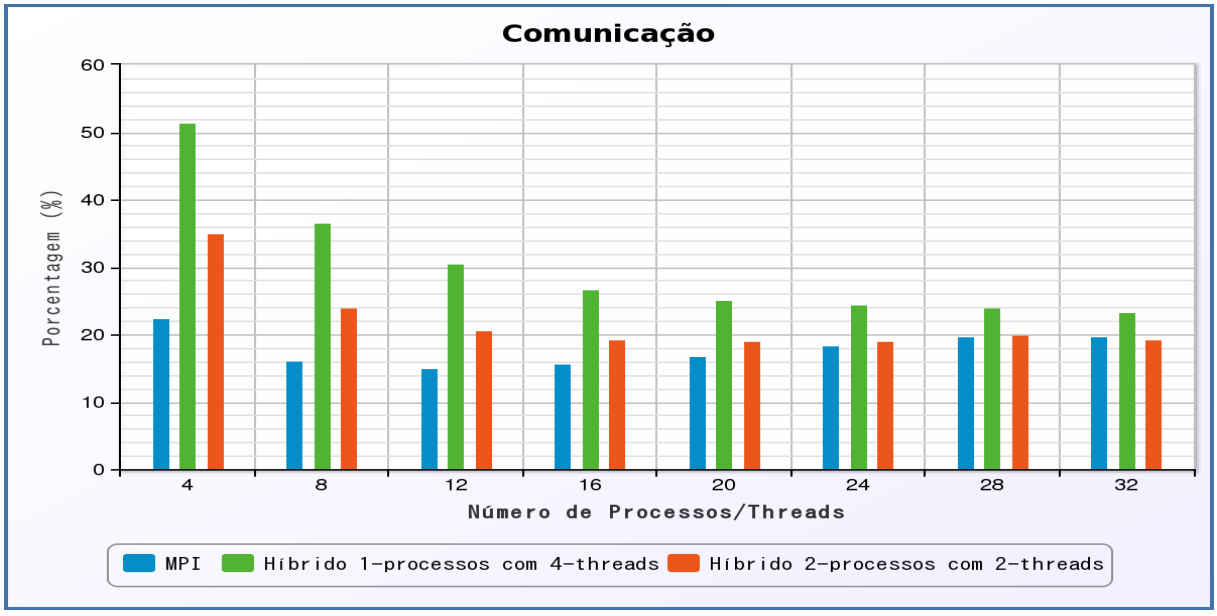


Figura 16 – Comunicação no *testbed* com a aplicação de processamento de imagem.

Isso quer dizer que, observando o comportamento com um nó trabalhador (4 processos/*threads*) na abordagem híbrida com 1 processo, no tempo em que o processo trabalhador está executando, temos o processo mestre em uma primitiva de recebimento, aguardando o resultado deste trabalhador. Isso resulta em um percentual de comunicação de 51,42%. Esse resultado pode ser mais detalhado se observarmos as Figuras 17 e 18, que representam respectivamente os percentuais de tempo em que os processos estão enviando e recebendo dados. Percebe-se que o tempo de espera para receber dados domina a porcentagem total de comunicação no caso de teste.

Isto deve-se ao fato de que nas abordagens híbridas, como não existe comunicação para a troca de resultados intermediários na execução do algoritmo, não existe comunicação entre as *threads* e o processo mestre. Por isso, até que o processo trabalhador termine sua execução e tenha o resultado o processo mestre fica bloqueado esperando o resultado

deste processo trabalhador. Já nas outras abordagens que utilizam um número maior de processos, assim que o processo finaliza sua execução o mesmo já está pronto para enviar os dados para o processo mestre, o que explica o percentual menor de envio de dados pela abordagem com processo MPI seguida pela abordagem híbrida que utiliza 2 processos.

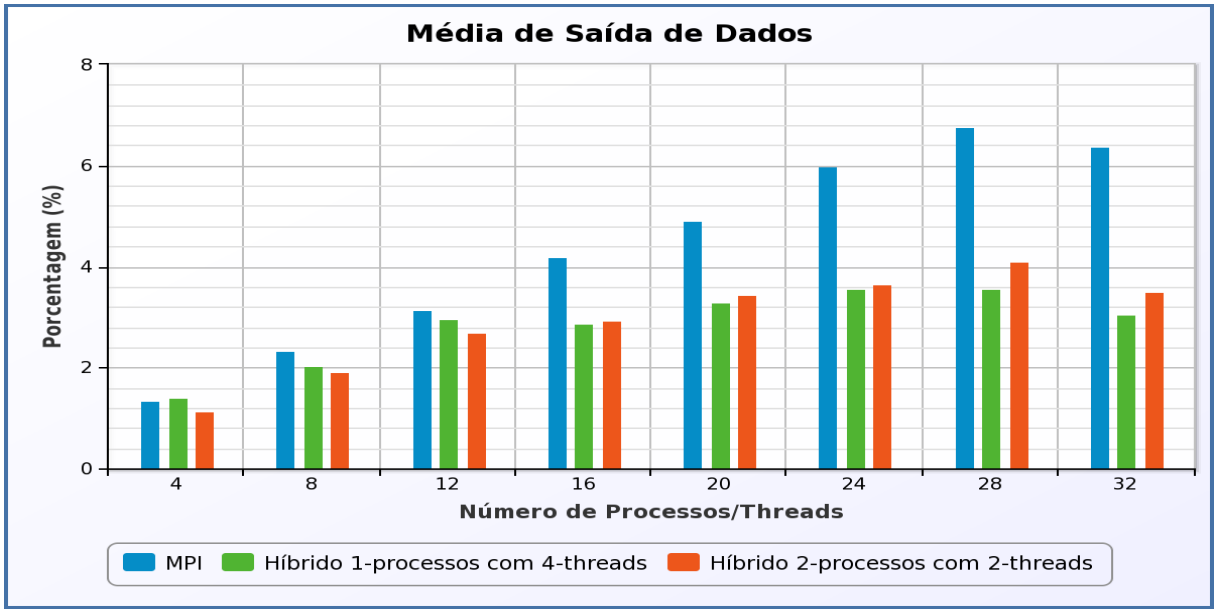


Figura 17 – Porcentagem do tempo de envio de dados no *testbed* com a aplicação de processamento de imagem.

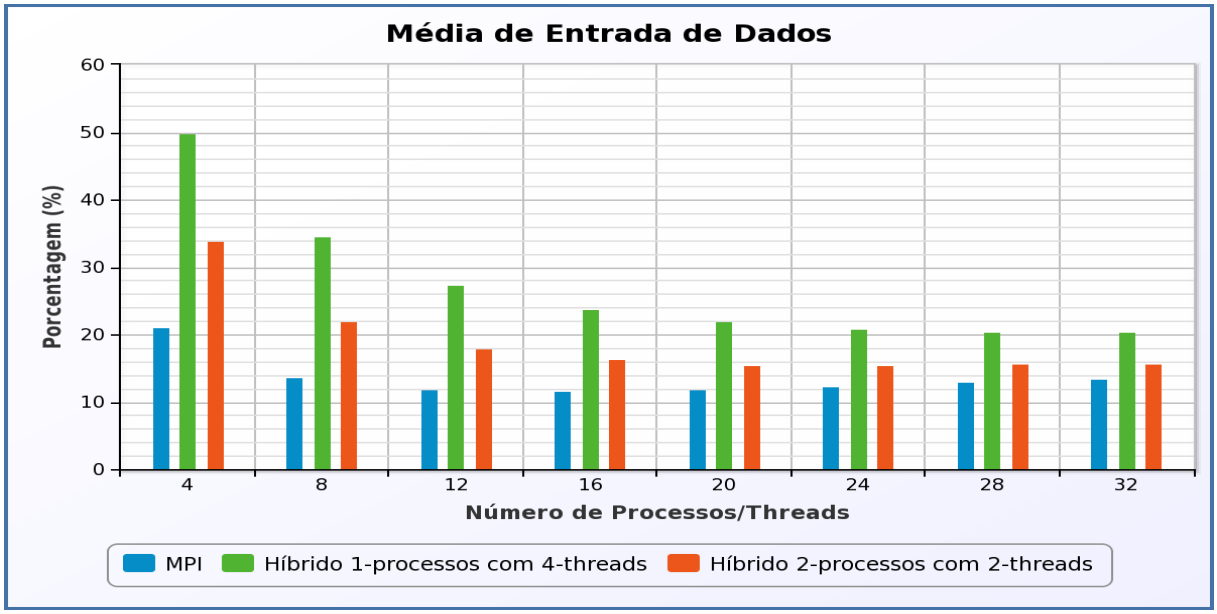


Figura 18 – Porcentagem do tempo de espera de recepção de dados no *testbed* com a aplicação de processamento de imagem.

Com a adição de mais nós na execução, as abordagens híbridas vão diminuindo os seus percentuais totais de comunicação. Já a abordagem que utiliza apenas processos

MPI na sua execução, ao chegar a um determinado número de processos trabalhadores, começa aumentar um pouco seu percentual de comunicação total. Esta questão pode ser vista na Figura 17, na qual é claramente perceptível o aumento do percentual do tempo de comunicação, a medida que o processo mestre passa a enviar dados para mais processos trabalhadores adicionados na execução do problema.

A redução do percentual total de comunicação das abordagens híbridas, principalmente a que utiliza de 1 processo trabalhador, se justifica com a diminuição do percentual do tempo de espera na hora de receber dados dos trabalhadores conforme visto na Figura 18. Isso porque a medida que mais processos vão sendo adicionados na execução, o processo mestre passará a esperar por dados de um número maior de processos que receberão uma carga menor de trabalho, consequentemente receberá dados de um processo em um tempo menor.

Já na abordagem híbrida que utiliza 2 processos trabalhadores por nó, apesar do percentual de recebimento de dados diminuir a medida que processos vão sendo adicionados, seu percentual de envio de dados aumenta, pois assim como a abordagem que utiliza apenas processos MPI, o processo mestre agora terá que distribuir dados para um número maior de processos.

4.3 Complemento Genérico

Para testar o complemento genérico, que permite ao usuário fazer o *upload* de códigos paralelos que possua e obter seus resultados conforme os obtidos com a aplicação de processamento de imagem, foi utilizado um algoritmo de multiplicação de matriz. Este algoritmo possui uma distribuição estática da carga de trabalho entre os processos participantes da computação. No início do processamento, o nó mestre divide a matriz A pelo número de processos participantes, então cada bloco é enviado para os processos juntamente com a matriz B inteira. Após distribuir todos os blocos, o processo mestre começa a processar sua parte. A execução deste algoritmo contou com os parâmetros listados na Tabela 5. Neste caso, a carga é definida pelo usuário que faz o *upload* do código conforme mencionado no Capítulo 3 Seção 3.5.

Tabela 5 – Parâmetros da aplicação de multiplicação de matriz.

Matriz	Serial	MPI	Híbrido	
Número de Testes	4	4	4	
Número de Máquinas Utilizadas	1	8	8	
Tamanho da Matriz	12000x12000	12000x12000	12000x12000	
Processos por Máquina	1	4	1	2
Threads por Processo	1	1	4	2

De acordo com a Tabela 5, o complemento genérico continua a configurar a execução dos algoritmos enviados pelo usuário de maneira a explorar o nível máximo de *threads* suportadas pelo sistema. Deste modo, a configuração dos cenários híbridos exploram o balanceamento entre processos e *threads* para dar uma melhor resposta de qual cenário executa melhor na arquitetura testada.

Observando a Figura 19, pode-se notar que o tempo de processamento da multiplicação diminuiu a medida que se aumenta o número de nós, mas diferente da aplicação anterior, a redução do tempo ocorre de forma diferente para as abordagens testadas. Das três abordagens executadas pela aplicação, a abordagem híbrida com 1 processo por nó é a que apresenta os menores tempos de execução, chegando a cerca de 1108 segundos a menos que a abordagem que utiliza apenas processos MPI. A abordagem híbrida com 2 processos por nó obteve cerca de 706 segundos a menos que a abordagem com processos MPI.

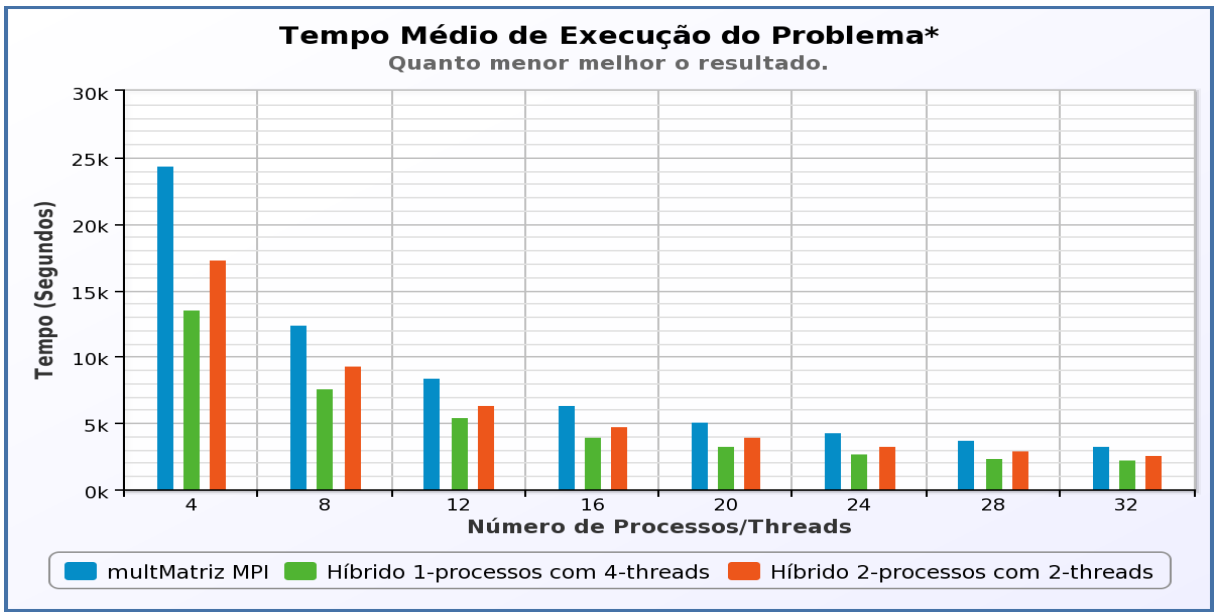


Figura 19 – Tempo de execução do *testbed* com a aplicação de multiplicação de matriz.

A redução no tempo é visível no *speedup* e eficiência obtidos com a aplicação, Figuras 20 e 21. Desde a utilização de apenas um nó (4 processos/*threads*), o *speedup* alcançado pela abordagem híbrida que utiliza apenas um processo se destaca em relação as outras abordagens, alcançando o speedup de 18x quando são utilizadas todos os nós no processamento. Contudo, as outras abordagens demonstram um crescimento linear, bastante parecido, mas ainda assim com uma vantagem da abordagem híbrida que utiliza dois processos em relação a abordagem que utiliza apenas processos MPI.

Na Figura 21, para a carga requisitada pelo usuário, é possível observar que mesmo utilizando todos os nós no processamento, a eficiência ficou acima dos 50% para as abordagens híbridas. Já para a abordagem que utiliza apenas processos MPI, sua eficiência

acabou demonstrando uma queda menos acentuada que as outras abordagens na adição de nós, porém a abordagem híbrida com 1 processo acabou ficando com uma eficiência de cerca de 60% quando todos os nós estavam trabalhando.

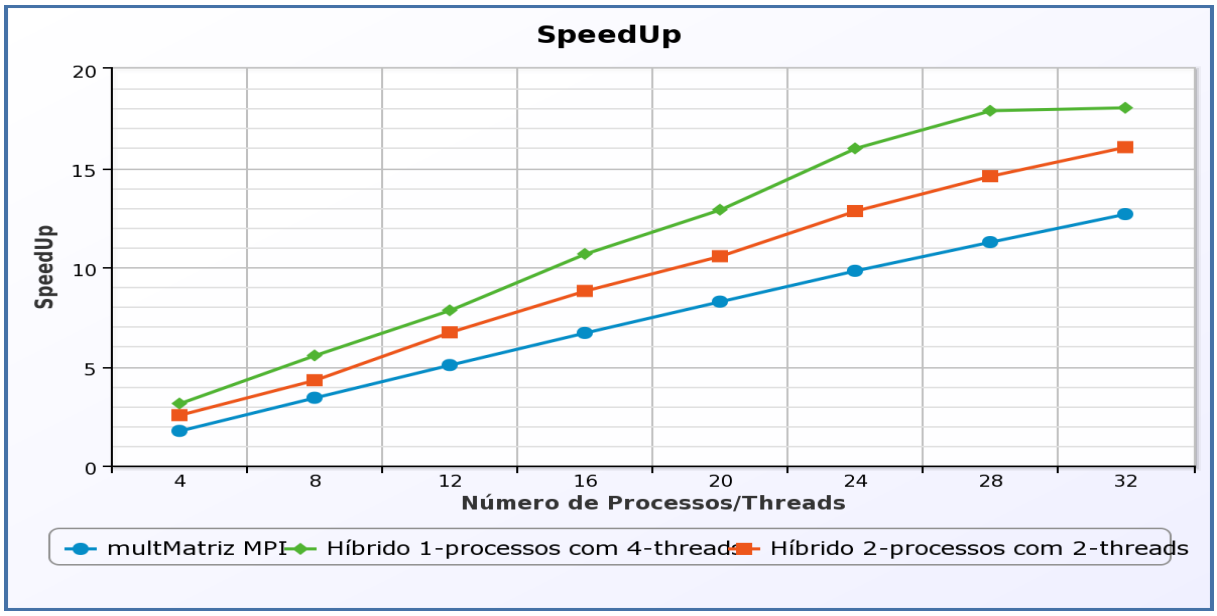


Figura 20 – *Speedup* do *testbed* com a aplicação de multiplicação de matriz.

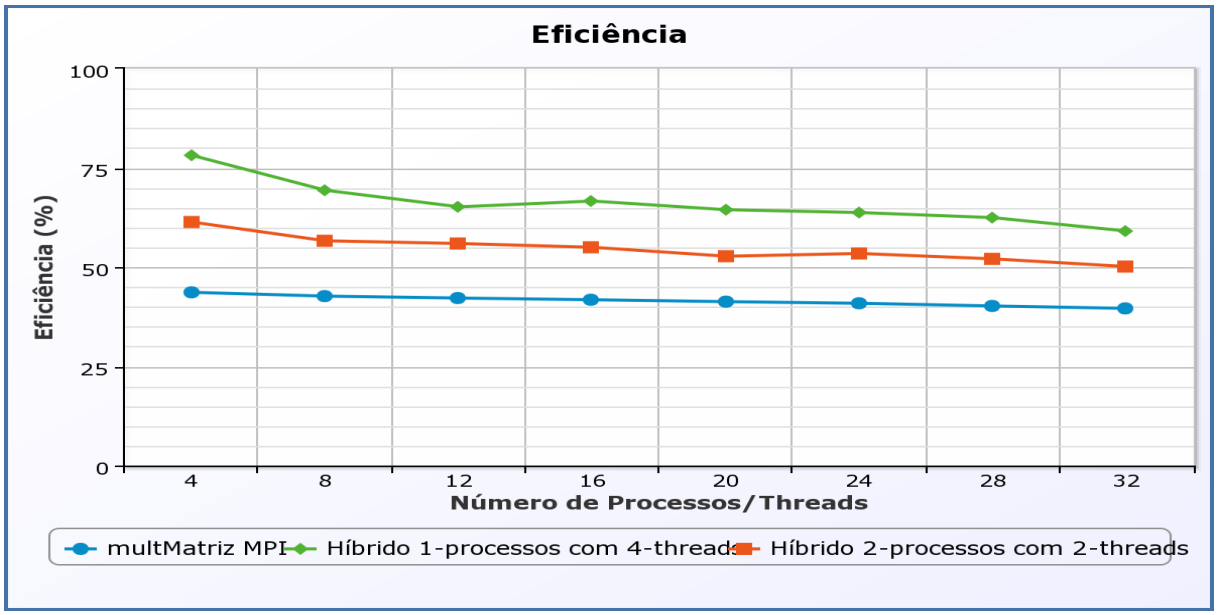


Figura 21 – Eficiência do *testbed* com a aplicação de multiplicação de matriz.

Observando a Figura 22, pode-se perceber a primeira vista que a abordagem híbrida que utiliza 1 processo aumenta seu percentual de comunicação em relação as outras abordagens. Também é possível notar que a comunicação com um nó trabalhador (4 processos/*threads*) é menor que as outras abordagens, pois neste caso, o processo mestre também é o processo trabalhador.

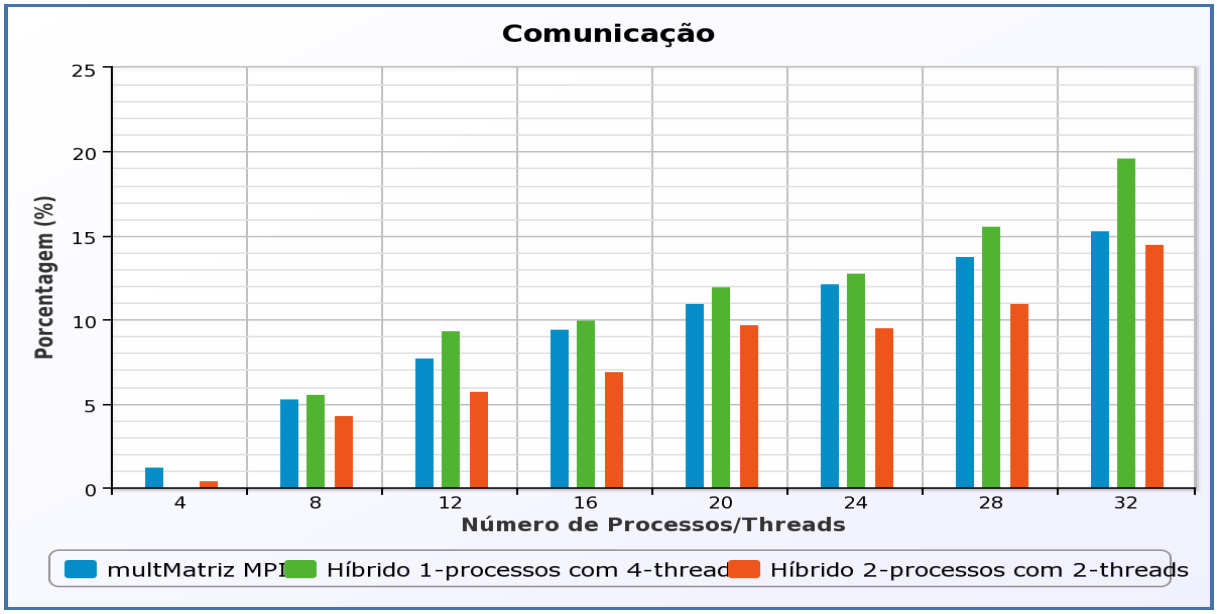


Figura 22 – Comunicação no *testbed* com a aplicação de multiplicação de matriz.

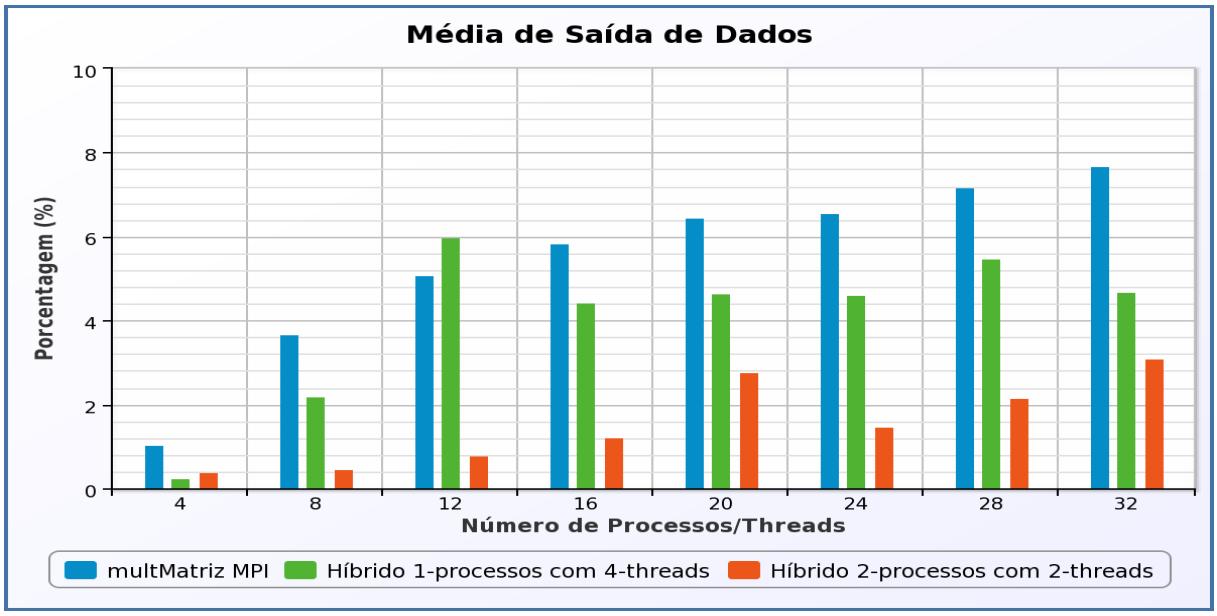


Figura 23 – Porcentagem do tempo de envio de dados no *testbed* com a aplicação de multiplicação de matriz.

Na Figura 23 percebe-se claramente que a abordagem utilizando apenas processos MPI sofre com a adição de nós ao processamento. Conforme o número de nós aumenta, seu percentual de envio de dados aumenta, superando o percentual de envio das outras abordagens. Este padrão também atinge a abordagem híbrida que utiliza 2 processos por nós e se inverte na abordagem híbrida que utiliza apenas 1 processo.

Tanto a Figura 23, quanto a Figura 24, demonstram que com a adição de processos a execução, as abordagens que utilizam mais processos tendem a ter o tempo de

comunicação dominado pela distribuição dos dados, enquanto que para abordagens que utilizem menos processos esse tempo será dominado pelo recebimento de dados.

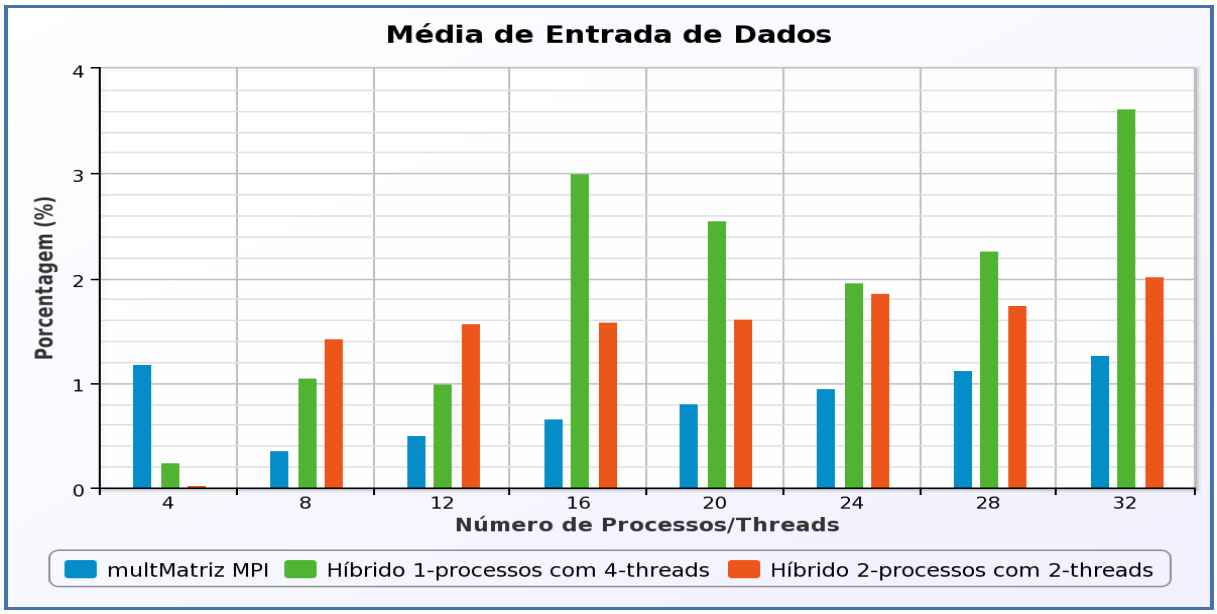


Figura 24 – Porcentagem do tempo de espera de recepção de dados no *testbed* com a aplicação de multiplicação de matriz.

5 Conclusões e Trabalhos Futuros

A ferramenta de *benchmark* proposta nesta pesquisa, através dos experimentos realizados no Capítulo 4, mostrou ser possível a apresentação de indicadores de desempenho consistentes para análise e avaliação do desempenho de *cluster* de computadores de memória distribuída, por meio de uma execução mais simples. A partir da utilização da interface Web é possível disponibilizar aos usuários um *benchmark* automático, sem a necessidade de intervenções que requeram bastante conhecimento da arquitetura ou permissões avançadas, e um resultado em gráfico das aplicações utilizadas como carga no sistema.

Outras aplicações apresentam indicadores de desempenho isolados ou muitas vezes informações pontuais, que pouco ajudam o usuário a formular hipóteses sobre suas arquiteturas ou descobrir oportunidades de melhoria. Por isso, esta ferramenta de *benchmark* utiliza de outras aplicações, para que ao relacionar suas saídas possam ser obtidos indicadores que compreendam fatores como operação em ponto flutuante, comunicação, taxa de transferência e tempo de processamento. Além disso, através da interface Web, o usuário pode enviar algum código paralelo que deseje para que a ferramenta execute esse código e apresente o resultado conforme o objetivo deste trabalho.

No *testbed* utilizado pela aplicação, com a versão multi*thread* do *benchmark* HPL foi possível obter um desempenho equivalente a 19% do desempenho teórico que poderia ser atingido com a arquitetura, que se apresenta como sendo 1,6 vezes o desempenho e a eficiência obtidos com a versão multi processos do *benchmark*. Isso porque, apesar do *benchmark* ter um alto grau de comunicação, a abordagem multi processos, por utilizar muitos processos, acabou sofrendo com a limitação da largura de banda do enlace de rede do nó mestre da computação, onde a máxima taxa de transferência foi de 94.8073 Mbits.

Ao executar a aplicação de processamento de imagem no *testbed*, foi possível perceber que por ser uma aplicação que distribui a carga de trabalho no inicío do processamento, tanto as abordagens híbridas que fazem um balanceamento entre processo e *thread* quanto a abordagem utilizando apenas processos MPI tiveram desempenho semelhantes. As diferenças foram vistas quando na máquina que aloca o processo mestre também foram alocados processos trabalhadores. Nessa situação a abordagem utilizando apenas processos MPI teve uma queda sensível no desempenho, enquanto que nas abordagens por *threads* não houve queda de desempenho.

Apesar do *benchmark* HPL ter demonstrado haver um limite no enlace de rede da arquitetura, com a aplicação de processamento de imagem foi possível obter desempenhos 20 vezes melhores que a solução serial mantendo uma eficiência acima dos 65% com

a abordagem utilizando apenas processos MPI. Utilizando alguma das abordagens com *threads*, obteve-se uma eficiência acima de 74% com desempenhos 23 vezes melhor.

Já com a aplicação de multiplicação de matriz, foi possível perceber que o complemento genérico, implementado para que o usuário possa executar algoritmos que deseje, configura e executa os códigos da mesma maneira na qual a aplicação de processamento de imagem foi executada. Essa configuração de execução permite dispor os resultados do processamento para o usuário de maneira mais cômoda para a sua avaliação de desempenho. No *testbed* utilizado, a partir da carga utilizada, foi possível obter uma execução 18 vezes mais rápida do que a execução serial com eficiência de cerca de 60% quando se utilizou uma abordagem híbrida.

Como trabalhos futuros para esta pesquisa, sugere-se expandir o número de fatores que afetam o desempenho de sistemas paralelos de memória distribuída, agregando indicadores que compreendam dispositivos como o subsistema de memória. Também sugere-se implementar um mecanismo de tolerância a falhas, para que em caso de interrupções oriundas de natureza distintas, a ferramenta possa continuar o *benchmark* de onde parou. Por fim, sugere-se uma extensão para o complemento genérico, para que possa suportar aplicações de diferentes modelos, com diferentes modelos de desempenho.

Referências

BORATTO, M.; ALONSO, P.; VIDAL, A. A Threaded Divide and Conquer Symmetric Tridiagonal Eigensolver on Multicore Systems. In: *ISPDC*. [S.l.: s.n.], 2008. p. 464–468. Citado na página 19.

BORATTO, M.; COELHO, L.; BARRETO, M. Distributed and parallel computing on multicore and multi-gpu systems. *XIII Simpósio em Sistemas Computacionais (WSCAD-SSC)*, 2012. Citado 2 vezes nas páginas 15 e 17.

BOTTA, A.; PESCAPE, A.; VENTRE, G. On the performance of bandwidth estimation tools. In: *Systems Communications, 2005 (ICW '05)*. [S.l.: s.n.], 2005. p. 287–292. Citado na página 27.

COELHO, S. A. Introduçãro a computação paralela com o OpenMPI. In: . [S.l.: s.n.], 2012. p. 24–44. Citado 2 vezes nas páginas 24 e 25.

COSTA, A. L. L. da; SOUZA, J. R. de. APCM: An Auto-Parallellism Computational Model. In: . [S.l.: s.n.], 2013. Citado na página 21.

COULOURIS, G.; DOLLIMORE, J.; KINDBERG, T. *Sistemas Distribuidos: Conceitos e Projetos*. 4. ed. [S.l.]: Addison-Wesley Publishers Limited, 2007. Citado na página 24.

DANTAS, M. *Computação Distribuída de Alto Desempenho: Redes, Clusters e Grids Computacionais*. 1. ed. Rio de Janeiro: [s.n.], 2005. Citado na página 25.

DONGARRA, J.; WHALEY, R. *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*. 2008. Version 2.0. Disponível em: <http://www.netlib.org/benchmark/hpl/>. Citado 4 vezes nas páginas 15, 28, 29 e 33.

EL-REWINI, H.; ABD-EL-BARR, M. *Advanced Computer Architecture and Parallel Processing*. Wiley, 2005. 288 p. (Wiley Series on Parallel and Distributed Computing). ISBN 9780471478393. Disponível em: <http://books.google.com.br/books?id=7JB-u6D5Q7kC>. Citado 4 vezes nas páginas 24, 25, 26 e 27.

FILHO, J. N. *Espacialização da Temperatura para o Pólo de Desenvolvimento Petrolina/Juazeiro, utilizando computação de alto desempenho*. 2012. Trabalho de Conclusão de Curso apresentado na Universidade Federal do Vale do São Francisco como requisito parcial para obtenção do título de Engenheiro de Computação, Juazeiro, 2012. [Orientador: Prof Msc. Murilo do Carmo Boratto, Co Orientador: Dsc. Brauliro Gomes Leal]. Citado na página 20.

FLYNN, M. J. Some computer organizations and their effectivness. *IEEE Transactions on Computers*, v. 21, p. 948–960, 1972. Citado na página 17.

GRAMA, A. et al. *Introduction to Parallel Computing (2nd Edition)*. 2. ed. Addison Wesley, 2003. 656 p. Hardcover. ISBN 0201648652. Disponível em: <http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20&path=ASIN/0201648652>. Citado 5 vezes nas páginas 19, 21, 22, 23 e 25.

KOGGE, P. M.; DYSART, T. J. Using the top500 to trace and project technology and architecture trends. In: *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*. New York, NY, USA: ACM, 2011. (SC '11), p. 28:1–28:11. ISBN 978-1-4503-0771-0. Disponível em: <http://doi.acm.org/10.1145/2063384.2063421>. Citado na página 15.

MILANI, C. R. *Computação verificada aplicada á resolução de sistemas lineares intervalares densos em arquiteturas multicore*. 2010. Dissertação apresentada como requisito parcial á obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2010. [Orientador: Prof Dr. Luiz Gustavo Leão Fernandes, Co Orientadora: Prof Dra. Mariana Luderitz Kolberg]. Citado na página 29.

MPI, F. *Message passing interface forum*. 2013. Disponível em: <www.mpi-forum.org>. Acesso em: 10 Set 2013. Citado na página 20.

NAS, P. B. *NASA Advanced Supercomputing Division - NAS Parallel Benchmarks*. 2013. Disponível em: <http://www.nas.nasa.gov/publications/npb.html>. Acesso em: 10 Set 2013. Citado na página 27.

NAVAUX, P. O. A.; ROSE, C. A. F. D.; PILLA, L. L. Fundamentos das arquiteturas para processamento paralelo e distribuído. *XI Escola Regional de Alto Desempenho do Estado do Rio Grande do Sul - 2011 - Porto Alegre, RS*, p. 22 – 59, 2011. Citado 3 vezes nas páginas 23, 24 e 25.

NULL, L.; LOBUR, J. *Princípios Básicos de Arquitetura e Organização de Computadores*. 2. ed. Bookman, 2010. 822 p. ISBN 9788577807666. Disponível em: <http://books.google.com.br/books?id=vn-ISIU82t4C>. Citado 5 vezes nas páginas 17, 18, 19, 23 e 25.

OLIVEIRA, A. L. F. de; SOUZA, U. dos S. *Algoritmos Paralelos De Ordenação*. 2008. Trabalho de Conclusão de Curso submetido ao Curso de Tecnologia em Sistemas de Computação da Universidade Federal Fluminense como requisito parcial para obtenção do grau de Tecnólogo em Sistemas de Computação, Niterói, 2008. [Tutor Orientador: Prof Msc. Maise Dantas da Silva]. Citado na página 19.

OPENMP. *The openmp api specification for parallel programming*. 2013. Disponível em: <http://openmp.org/>. Acesso em: 10 Set 2013. Citado na página 20.

ORELLANA, E. T. V. *Introdução ao Processamento de Alto Desempenho*. 2011. Minicurso palestrado ERAD Nordeste 2011. Citado 2 vezes nas páginas 18 e 19.

PACHECO, P. *An Introduction to Parallel Programming*. Elsevier Science, 2011. 392 p. (An Introduction to Parallel Programming). ISBN 9780080921440. Disponível em: <http://books.google.com.br/books?id=SEmfraJjvfwC>. Citado 2 vezes nas páginas 17 e 19.

PINHEIRO, O. R. *Um ambiente computacional tolerante a falhas para acplicações paralelas*. Dissertação (Mestrado), Fevereiro 2013. Dissertação de mestrado apresentada ao Programa de Pós-graduação em Modelagem Computacional e Tecnologia Industrial, Curso de Mestrado em Modelagem Computacional e Tecnologia Industrial do Senai Cimatec, como requisito aprcial para obtenção do título de Mestre em Modelagem Computacional e Tecnologia industrial. Citado 2 vezes nas páginas 19 e 20.

PITANGA, M. *Construindo Supercomputadores com Linux*. 3. ed. Rio de Janeiro: BRASPORT, 2008. 400 p. ISBN 9788574523729. Disponível em: <http://books.google.com.br/books?id=PbMKWMXeZgsC>. Citado 4 vezes nas páginas 17, 18, 19 e 20.

PRESSMAN, R. *Engenharia de software*. McGraw-Hill, 2006. ISBN 9788586804571. Disponível em: <http://books.google.com.br/books?id=MNM6AgAACAAJ>. Citado na página 41.

QUINN, M. J. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill Higher Education, 2004. 529 p. (McGraw-Hill Higher Education). ISBN 9780072822564. Disponível em: <http://books.google.com.br/books?id=tDxNyGSXg5IC>. Citado 3 vezes nas páginas 18, 20 e 21.

RAUBER, T.; RÜNGER, G. *Parallel Programming: For Multicore and Cluster Systems*. Springer, 2010. 455 p. ISBN 9783642048173. Disponível em: <http://books.google.com.br/books?id=wWogxOmA3wMC>. Citado 8 vezes nas páginas 15, 18, 21, 22, 23, 24, 26 e 27.

RIBEIRO, N. S. *Explorando programação híbrida no contexto de clusters de máquinas NUMA*. 2011. Dissertação apresentada como requisito parcial á obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre, 2011. [Orientador: Prof Dr. Luiz Gustavo Leão Fernandes]. Citado na página 20.

SINDI, M. *How To - High Performance Linpack (HPL)*. 2013. Disponível em: <http://hpl-calculator.sourceforge.net/HPL-HowTo.pdf>. Acesso em: 10 Set 2013. Citado 2 vezes nas páginas 22 e 43.

SKINNER, D. *Performance monitoring of parallel scientific applications*. [s.n.], 2005. Disponível em: <http://www.osti.gov/scitech/servlets/purl/881368>. Citado na página 33.

SUTTER, H. The free lunch is over: A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, v. 30, n. 3, p. 202–210, 2005. Disponível em: <http://www.gotw.ca/publications/concurrency-ddj.htm>. Citado na página 17.

VELÁSQUEZ, K.; GAMESS, E. A comparative analysis of network benchmarking tools. *Proceedings of the World Congress on Engineering and Computer Science*, Vol I, p. 299–305, Outubro 2009. Citado na página 28.

WATANABE, K. et al. A network interface controller chip for high performance computing with distributed pcs. *IEEE Transactions on Parallel and Distributed Systems*, v. 18, p. 1282–1295, Setembro 2007. Citado na página 25.

WHALEY, R. C.; PETITET, A. Minimizing development and maintenance costs in supporting persistently optimized blas. *Softw. Pract. Exper.*, John Wiley & Sons, Inc., New York, NY, USA, v. 35, n. 2, p. 101–121, fev. 2005. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.v35:2>. Citado na página 29.

YONG, C. K.; ABDULLAH, A.; ABDULLAH, M. T. Securing video streaming over internet protocol version 6(ipv6). In: *IJCSNS International Journal of Computer Science and Network Security*. [s.n.], 2012. v. 12, n. 11, p. 76–83. Disponível em: <http://paper.ijcsns.org/07_book/201211/20121113.pdf>. Citado na página 33.

ZACARIAS, F. V. et al. Avaliação de desempenho do cluster gabi. *ERBASE 2013, Itabaiana, Sergipe*, v. 1, 2013. Citado na página 15.