UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ELTON PENICHE QUARESMA

IMPLANTAÇÃO DE UM CLUSTER DO TIPO BEOWULF: UM LABORATÓRIO PARA O ENSINO DA PROGRAMAÇÃO PARALELA EM AMBIENTE NÃO VIRTUALIZADO

Belém
2019

ELTON PENICHE QUARESMA

IMPLANTAÇÃO DE UM CLUSTER DO TIPO BEOWULF: UM LABORATÓRIO PARA O ENSINO DA PROGRAMAÇÃO PARALELA EM AMBIENTE NÃO VIRTUALIZADO

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal do Pará como parte dos requisitos necessários para obtenção de Título de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Josivaldo de Souza Araújo

Belém
2019

Quaresma, Elton

IMPLANTAÇÃO DE UM CLUSTER DO TIPO BEOWULF: UM LABORATÓRIO PARA O ENSINO DA PROGRAMAÇÃO PARALELA EM AMBIENTE NÃO VIRTUALIZADO/ ELTON PENICHE QUARESMA. – Belém, 2019.

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

Orientador: Prof. Dr. Josivaldo de Souza Araújo

Monografia – UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO, 2019.

1. Cluster Beowulf. 2. OpenMP. 3. Open MPI. 4. Computação Paralela. I. Título.

ELTON PENICHE QUARESMA

IMPLANTAÇÃO DE UM CLUSTER DO TIPO BEOWULF: UM LABORATÓRIO PARA O ENSINO DA PROGRAMAÇÃO PARALELA EM AMBIENTE NÃO VIRTUALIZADO

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal do Pará como parte dos requisitos necessários para obtenção de Título de Bacharel em Ciência da Computação.

Data da Defesa: 11 de Julho de 2019
Conceito:

Banca Examinadora

Prof. Dr. Josivaldo de Souza Araújo
Faculdade de Computação - ICEN - UFPA
Orientador

Prof^{a}. Dr^{a}. Regiane Silva Kawasaki Francês
Faculdade de Computação - ICEN - UFPA
Membro da Banca

Prof. Dr. Raimundo Viégas Jr.
Faculdade de Computação - ICEN - UFPA
Membro da Banca

Belém
2019

RESUMO

Com a crescente demanda por poder computacional e com o aumento de aplicações que necessitam ser executadas em um curto intervalo de tempo, tanto no meio científico, quanto no meio acadêmico, algumas simulações não podem ou não conseguem ser executadas de forma sequencial, seja pela grande quantidade de memória principal requerida ou pelo longo tempo de processamento, este trabalho apresenta como forma de solucionar essa carência computacional por meio do desenvolvimento de um *Cluster Beowulf*, que é sistema computacional de baixo custo, pois engloba dois ou mais computadores comuns, interconectados por uma tecnologia de rede, no qual trabalham em conjunto para executar aplicações, de tal forma que para os usuários que os utilizam tenham a impressão que há somente um único sistema respondendo a eles. Esta arquitetura foi implantada no LABCOMP-02, da Faculdade de Computação, no sentindo de aproveitar o tempo ocioso dos computadores do laboratório em alguns períodos, para paralelizar algoritmos, aumentando a eficiência e diminuindo o tempo de execução. Por fim, foram executados dois algoritmos, de baixa complexidade, utilizando as bibliotecas OpenMP e Open MPI, com o objetivo de testar seu funcionamento e eficiência, e os resultados obtidos por meio destes testes foram considerados satisfatórios.

Palavras-chave: Cluster Beowulf. OpenMP. Open MPI. Computação Paralela.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

COMA	*Cache-Only Memory Access*
COW	*Cluster of Workstation*
CPU	*Central Processing Unit*
E/S	Entrada e Saída
FACOMP	Faculdade de Computação
HPC	*High-performance computing*
IDE	*Integrated Development Environment*
LAN	*Local Area Network*
MIMD	*Multiple-instruction Multiple-data*
MISD	*Multiple-instruction Single-data*
MPI	*Message Passing Interface*
NOW	*Network of Workstations*
NUMA	*Non-Uninform Memory Access*
PVM	*Parallel Virtual Machine*
SIMD	*Single-instruction Multiple-data*
SISD	*Single-instruction Single-data*
SMP	*Symmetric MultiProcessors*
UC	Unidade de Controle
ULA	Unidade Lógica e Aritmética
UMA	*Uniform Memory Access*

SUMÁRIO

1 INTRODUÇÃO

1.1 Considerações Iniciais

O uso de computadores digitais dentro da nossa sociedade se desenvolveu desde o primeiro uso no fim da 2ª Guerra (1945). O principal objetivo, ao ser desenvolvido, era o de auxiliar na realização dos cálculos necessários para o projeto da primeira bomba atômica. Até o meados da década de 80, os computadores ainda eram grandes e caros e, por conta disso, muitas organizações ficavam limitadas a apenas alguns poucos microcomputadores. Esses microcomputadores também eram limitados na forma como as organizações poderiam utilizá-los, pois operavam independentemente uns dos outros, havendo uma capacidade limitada de conectá-los em si (MORRISON, 2003).

Ainda segundo Morrison (2003), em meados dos anos 80, dois avanços na tecnologia começaram a mudar essa situação. O primeiro foi o desenvolvimento de microprocessadores poderosos, muitos deles tinham o poder computacional dos sistemas de *mainframe* (computador de grande porte e desempenho da época), mas a uma fração do preço. O segundo desenvolvimento foi a invenção da *Local Area Network* (LAN) de alta velocidade. Os sistemas de redes permitiam que dezenas a centenas de máquinas fossem interconectadas de tal maneira que pequenas quantidades de informações pudessem ser transferidas entre máquinas em apenas alguns milissegundos. O resultado dessas duas tecnologias é a capacidade de construir redes de computadores compostas por um grande número de unidades centrais de processamento (CPU) interconectadas por meio de redes de alta velocidade.

Redes de computadores podem ser definidas como conjunto de computadores autônomos interconectados por uma única tecnologia e que dois computadores estão interconectados quando podem trocar informações entre si. A conexão pode ser feita por um fio de cobre; fibras ópticas, micro-ondas, ondas de infravermelho e satélites de comunicações. A principal diferença entre redes de computadores e um sistema distribuído é que, em um sistema distribuído, um conjunto de computadores independentes parecem ser, para usuários, um único sistema coerente. Em geral, possui um único modelo ou paradigma que é apresentado aos usuários. Com frequência, uma camada de software sobre o sistema operacional, chamada *middleware*, é responsável pela implementação desse modelo (TANENBAUM, 2003).

Sistemas distribuídos facilitam aos usuários, e às aplicações, o acesso aos recursos remotos e seu compartilhamento de maneira controlada e eficiente. com as vantagens de ser um sistema computacional mais barato, ter poder de processamento e armazenamento similar aos supercomputadores e por serem tolerantes à falhas (COULOURIS et al., 2013).

Atualmente pode-se observar que tem crescido o número de aplicações que necessitam de grande poder computacional, como aplicações de mapeamento genético, computação gráfica, previsões meteorológicas e até mesmo programas que exigem um grande número de variáveis de entrada. Os supercomputadores são ideais para esses tipos de aplicações, porém possuem um custo elevado e, portanto não são economicamente viáveis. Uma alternativa de mais baixo custo, para essa demanda, são os chamados *clusters* de computadores, pois processam as tarefas paralelas de forma a parecer um único sistema para os usuários (BACELLAR, 2009).

Clusters são sistemas de computação distribuídos que podem ser definidos, genericamente, como um conjunto de estações de trabalho ou computadores semelhantes, conectados por uma rede local de alta velocidade, além dos computadores executarem o mesmo sistema operacional (COULOURIS et al., 2013).

1.2 Justificativa

Atualmente, tem-se um grande número de aplicações que necessitam cada vez mais uma grande quantidade de processamento de dados, porém ainda não existe na Faculdade de Computação (FACOMP), da UFPA, um sistema computacional que possa satisfazer essa necessidade por alto desempenho. Para solucionar essa demanda por poder computacional, tem-se como uma alternativa viável, a utilização de um cluster de computadores para aproveitar o tempo ocioso dos computadores do laboratório de computação da FACOMP. A implantação do *Cluster Beowulf* no Laboratório de Computação, tem como objetivo disponibilizar uma estrutura de alto desempenho, inicialmente, aos pesquisadores da Faculdade, e posteriormente, para todos do Instituto de Ciências Exatas e Naturais (ICEN) que necessitam de alto poder de processamento de dados em pesquisas acadêmicas, e também podendo ser utilizado no auxílio do ensino de programação paralela.

1.3 Objetivos

1.3.1 Gerais

Este trabalho tem por objetivo geral a montagem, configuração e instalação de um *Cluster Beowulf*, no Laboratório 02, da Faculdade de Computação, para ser utilizado em pesquisas e estudos que necessitem de computação de alto desempenho.

1.3.2 Específicos

1. Analisar aspectos conceituais e de implementação de um *Cluster Beowulf*;
2. Apresentar uma visão dos componentes de um *Cluster Beowulf*;
3. Discutir a importância de uma infraestrutura para auxiliar no ensino de programação paralela;
4. Testar a configuração proposta através do estudo de desempenho;

1.4 Metodologia

1. O desenvolvimento deste projeto iniciou-se com uma revisão da literatura sobre os temas: *clusters Beowulf*; computação paralela; Processamento paralelo; Bibliotecas para programação paralela; abordagens de ensino da programação paralela;

2. Em seguida foi apresentado um modelo para a implementação de um *cluster Beowulf*;

3. Para a implementação do *cluster Beowulf* foi utilizada a infraestrutura do Laboratório 2 da FACOMP. Vale ressaltar que estas máquinas são utilizadas para as aulas de graduação dos cursos da Faculdade, bem como, para cursos de extensão e aulas da Pós-graduação;

4. Após a implementação, foram realizados testes com a finalidade de analisar o seu funcionamento. Os testes foram realizados a partir da execução de algoritmos paralelos, utilizando bibliotecas de comunicação, como o *OpenMP* e o *Open MPI*;

5. O processo de validação se deu através da análise de desempenho dos resultados dos testes;

6. Os recursos acima mencionados foram alocados pela UFPA, através da Faculdade de Computação (FACOMP), onde já se encontram máquinas para uso acadêmico.

1.5 Trabalhos Relacionados

Existem diversos trabalhos que abordam a implantação de um *Cluster Beowulf* que se diferenciam por utilizar distintas componentes que vão de: configuração de hardware das máquinas, número de máquinas, sistema operacional, rede interna e middleware, Dentre os trabalhos relacionados encontrados na pesquisa realizada, não se encontrou algum idêntico ao proposto neste trabalho.

No trabalho de (PUTTINI et al., 2016) é apresentada uma proposta para criação de um laboratório virtual de computação paralela. O laboratório consiste de um cluster de processamento paralelo utilizando-se de 18 máquinas de arquitetura Intel. O sistema operacional utilizado foi o *Red Hat* 6.2. Para validar o laboratório virtual de computação foi criado um ambiente de teste que consistia em executar uma aplicação de renderização de imagens. O autor conclui que o projeto de laboratório virtual possibilita que instituições de ensino, com um único computador conectado na Internet, possam utilizar o poder elevado de processamento de um cluster e que também incentiva tanto o ensino quanto a pesquisa no desenvolvimentos de ferramentas paralelas na medida em que torna acessível uma tecnologia cara.

No trabalho de (MAIA, 2016) é apresentado uma ferramenta para o ensino de programação paralela e distribuída, para que estudantes possam praticar os conceitos aprendidos dentro da sala de aula, em um ambiente virtual. Para validar a ferramenta foi criado um estudo de caso na Universidade Federal do Ceará, Campus Quixadá, onde foi feita uma análise do tempo de criação de *clusters* virtuais sobre 6 máquinas reais para diferentes números de usuários. O sistema operacional utilizado no nó mestre foi Ubuntu 15.04 LTS, enquanto que nos restantes dos nós foi o Ubuntu 14.04 LTS. Observou-se que durante os testes algumas máquinas travaram com muitos usuários, sendo necessário até o reinicio das máquinas, porém, analisando os resultados obtidos, a média de tempo de criação de *clusters* com 12 usuários, acessando simultaneamente a ferramenta, pode ser considerado como um tempo hábil. O autor conclui que tanto a ferramenta, quanto a infraestrutura utilizada pela ferramenta, responderam de forma eficaz a criação de *clusters* virtuais, podendo então ser utilizada para aulas práticas das disciplinas de programação paralela e distribuída.

Em (COSTA, 2014) é apresentado uma implementação de *Cluster* Beowulf em 4 máquinas, utilizando o sistema operacional Debian 4.0 "Etch" sem qualquer interface gráfica, IPs estáticos, sistema de arquivos distribuídos *Network File System* (NFS), *OpenSSH*, pacote *Parallel Virtual Machine* (PVM) e o ANSYS CFX. Para a análise de desempenho, o autor utiliza simulações dos métodos numéricos aplicados a CFD (Computational Fluid Dynamics) que demandam grande poder de processamento e conclui que a utilização de *clusters* é uma alternativa, que realmente proporciona alto poder computacional e baixo custo, pois, como no primeiro teste, houve uma melhora de desempenho conforme foram adicionados novos nós.

No trabalho de (PEREIRA, 2016) é apresentado uma implementação de *Cluster* Beowulf em 12 máquinas obsoletas, utilizando o sistema operacional Ubuntu 12.04 LTS de 32 bits com interface gráfica somente do nó mestre, IPs estáticos, sistema de arquivos distribuídos NFS, *OpenSSH* e o pacote *Open MPI*. Para a análise de desempenho, o autor utiliza o algoritmo que implementa o método de Monte Carlo para calcular o valor aproximado da constante π e obtém como resultados que o aumento do número de nós usados no cálculo, diminui o tempo total do mesmo, embora verifica-se que a partir de 4 nós o tempo tende a estabilizar e não se obtém uma melhoria significativa. O autor explica que uma razão provável para a estabilização dos tempos de execução do programa se dá pelo aumento de tráfego de dados pelo barramento que ocorre quando aumenta-se o número de nós usados no processamento.

A proposta deste trabalho se diferencia dos demais trabalhos por propor a implementação de um *Cluster Beowulf* em ambiente não virtualizado, aproveitando-se do tempo ocioso dos computadores. Possibilitando aos alunos a prática dos conceitos de programação paralela e a pesquisas acadêmicas.

1.6 Estrutura do Trabalho

Este trabalho está organizado em cinco capítulos, os quais são constituídos, além da Introdução, dos seguintes capítulos:

Capítulo 2 - Computação Paralela: Apresenta os conceitos básicos das arquiteturas de computadores paralelos. Aborda, também, os conceitos de *Cluster* de Computadores e seus principais tipos, além dos conceitos de de bibliotecas de comunicações, assim como, as implementações *Open MPI* e *OpenMP*, além das métricas que podem ser utilizadas para avaliar o desempenho desse tipo de sistema.

Capítulo 3 - Laboratório para o Ensino da Programação Paralela: aborda a importância da programação paralela e seu ensino. Discute vantagens e desvantagens de se montar ambientes reais e virtualizados, bem como as especificações técnicas de hardware, software e rede de conexão usados na implantação do *Cluster* Orã.

Capítulo 4 - Aplicações e Resultados: descreve os dois problemas que foram propostos para a validação do *Cluster* Orã, apresentando uma análise e discussão dos resultados obtidos.

Capítulo 5 - Conclusão: Apresenta as considerações finais do trabalho e também propostas para futuros trabalhos.

No Apêndice A estão descritos o passo-a-passo seguido para a montagem do *Cluster* Orã e no Apêndice B é apresentando o *script* que foi utilizado para a automação de alguns passos da instalação e configuração do *cluster*.

Nos Anexos A e B são apresentados os algoritmos paralelos utilizados na validação deste trabalho e no Anexo C é apresentado o algoritmo que foi utilizado para testar o funcionamento do *cluster*.

2 COMPUTAÇÃO PARALELA

2.1 Introdução

A programação estruturada foi desenvolvida para ser executada de forma sequencial, ou seja, um conjunto de instruções que são executados sequencialmente em um único processador, uma após a outra. Contudo, com a crescente necessidade por poder computacional, baseados no aumento de complexidade dos programas e das dificuldades para conseguir o aumento de *clock* dos processadores, juntamente por problemas de dissipação térmica, verificou-se como alternativa, uma programação que pudesse ser executada de forma paralela, ou seja, o problema poderia ser dividido em partes menores e, estabelecendo comunicações quando necessário, pudessem ser resolvidos simultaneamente em diferentes CPUs. A computação paralela tem sido usada em aplicações que exigem o processamento de grandes quantidades de dados. Por exemplo: *Big Data*, Inteligência Artificial, ferramentas de pesquisa na web, serviços empresariais baseados na Web, projeto farmacêutico, modelagem financeira e econômica, gestão de corporações nacionais e multinacionais, gráficos avançados e realidade virtual, particularmente na indústria do entretenimento, tecnologias de vídeo e multimídia em rede e exploração de óleo (BARNEY et al., 2010).

De acordo com Tokhi, Hossain e Shaheed (2012) um grande número de tipos de arquiteturas paralelas foi desenvolvido ao longo dos anos. Consequentemente, não é fácil desenvolver um sistema de classificação simples para arquiteturas paralelas. Além disso, vários tipos de arquiteturas paralelas têm características sobrepostas a diferentes extensões. No entanto, as várias formas de arquiteturas podem ser distinguidas sob as seguintes categorias amplas:

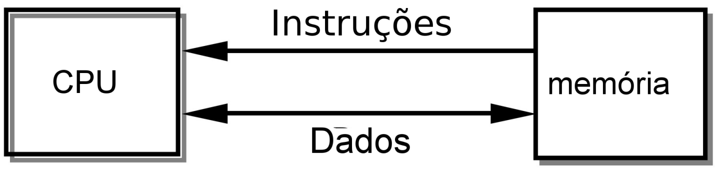
1. Classificação de Flynn.
2. Classificação por Arquitetura de memória.

2.2 Classificação de Flynn

Tanenbaum (2007) explica que apesar da classificação dos diferentes tipos de sistemas paralelos de Flynn não ser completamente satisfatória, ainda assim é muito usado. A classificação de Flynn é baseado nos conceitos de fluxos instruções e fluxos de dados. Um fluxo de instruções corresponde a um contador de programa. Um sistema com *n* CPUs tem *n* contadores de programa, por conseguinte, *n* fluxos de instruções. O fluxo de dados consiste em um conjunto de operandos.

A Figura 1 mostra que durante a execução do programa, a CPU busca instruções e dados da memória principal, processa os dados de acordo com as instruções e envia os resultados para a memória principal após o processamento ter sido concluído. As instruções são referidas como um fluxo de instruções, que flui da memória principal para o CPU, e os dados são chamados de fluxo

Figura 1 – Fluxo de instruções e fluxo de dados



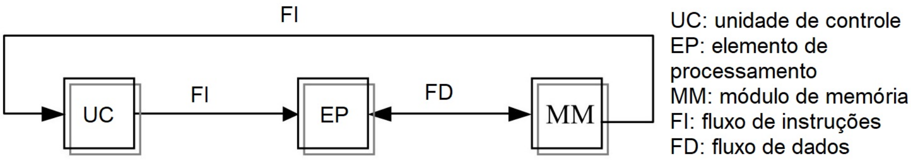
Fonte: Adaptado de Tokhi, Hossain e Shaheed (2012)

de dados, fluindo para e do CPU. Com base nesses fluxos, o Flynn classificou os computadores em quatro classes principais, descritas abaixo.

Arquitetura *Single-instruction Single-data Stream* (SISD)

A arquitetura do computador SISD possui uma CPU, unidade de memória e dispositivos de entrada e saída (E/S). A CPU consiste de uma unidade lógica e aritmética (ULA) para realizar operações aritméticas e lógicas, unidade de controle (UC) para executar operações de controle e registra para armazenar pequenas quantidades de dados. Os computadores SISD são computadores sequenciais e são incapazes de realizar operações paralelas (TOKHI; HOSSAIN; SHAHEED, 2012).

Figura 2 – Estrutura geral da arquitetura SISD



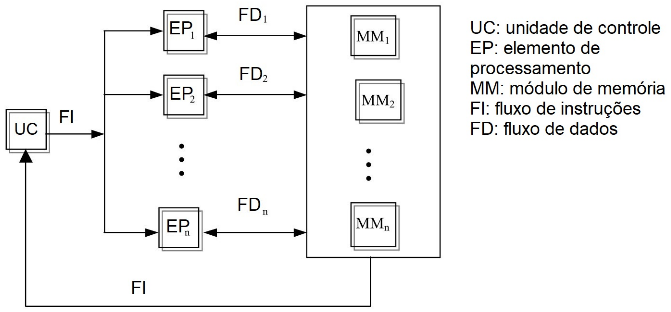
Fonte: Adaptado de Tokhi, Hossain e Shaheed (2012)

Arquitetura *Single-instruction Multiple-data* (SIMD)

Esta arquitetura possui um único fluxo de instruções para processar toda a estrutura de dados (fluxo de dados múltiplos). Em outras palavras, nesta configuração, uma única unidade de controle de programa ou processador de controle controla várias unidades de execução ou processadores de execução. Cada processador de execução é equipado com uma memória local para armazenar os dados nos quais ele irá trabalhar. Cada processador de execução executa a mesma instrução emitida pelo processador de controle em seus dados locais. Os processadores de execução são capazes de se comunicar uns com os outros quando necessário. A arquitetura SIMD é boa para problemas em que a mesma operação é executada em vários objetos diferentes, por exemplo, processamento de imagem. Algumas outras aplicações adequadas de arquiteturas

SIMD incluem operações de matriz e ordenação. A programação é bastante simples e direta para esta arquitetura. A arquitetura SIMD pode ser dividida em duas subclasses de acordo com as interconexões entre os CPUs. São eles: arquitetura vetorial e arquitetura de matriz (TOKHI; HOSSAIN; SHAHEED, 2012). A estrutura geral da arquitetura SIMD é mostrada na Figura 3 abaixo.

Figura 3 – Estrutura geral da arquitetura SIMD

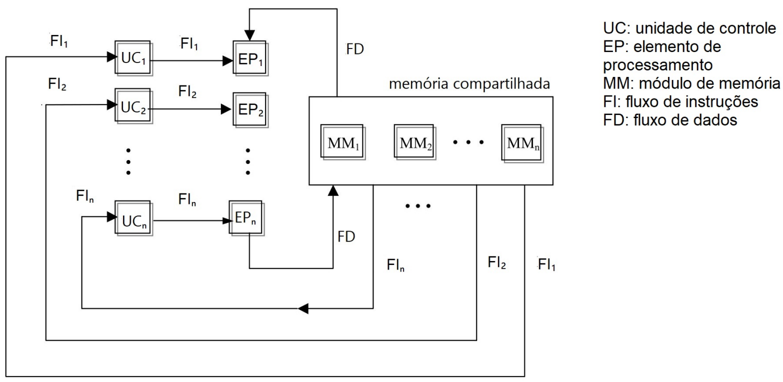


Fonte: Adaptado de Tokhi, Hossain e Shaheed (2012)

Arquitetura *Multiple-instruction Single-data* (MISD)

Essa arquitetura envolve múltiplos fluxos de instruções e um fluxo de dados único, conforme ilustrado na Figura 4. De modo geral, nenhuma arquitetura é classificada como MISD, isto é, não existem representantes desta categoria. Embora alguns autores considerem como MISD as máquinas com pipeline (TANENBAUM, 2007).

Figura 4 – Estrutura geral da arquitetura MISD

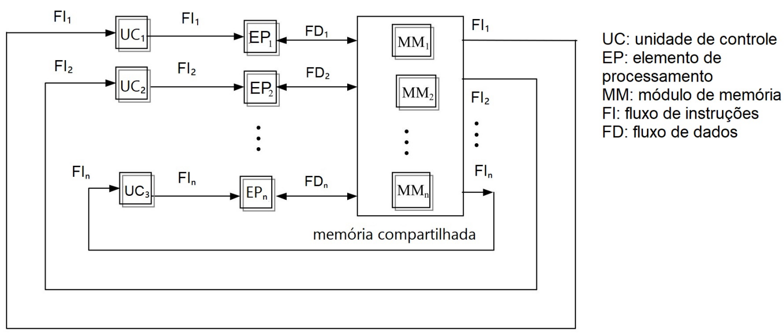


Fonte: Adaptado de Tokhi, Hossain e Shaheed (2012)

Arquitetura *Multiple-instruction Multiple-data* (MIMD)

Essa arquitetura é a forma mais comum e amplamente utilizada de arquiteturas paralelas. Ele compreende várias CPUs, cada um dos quais é capaz de executar fluxos de instrução independentes em fluxos de dados independentes. As CPUs no sistema geralmente compartilham recursos como recursos de comunicação, dispositivos de E/S, bibliotecas de programas e bancos de dados. Todos os CPUs são controlados por um sistema operacional comum. Os múltiplos CPUs no sistema melhoram o desempenho e aumentam a confiabilidade. O desempenho aumenta devido ao fato de que a carga computacional é compartilhada pelos CPUs no sistema. Teoricamente, se houver n CPUs no sistema, o desempenho aumentará em n vezes em comparação a um único sistema baseado em CPUs. A confiabilidade do sistema é aumentada pelo fato de que a falha de um CPUs no sistema não causa falha de todo o sistema (TOKHI; HOSSAIN; SHAHEED, 2012). A Figura 5 mostra a estrutura geral da arquitetura MIMD.

Figura 5 – Estrutura geral da arquitetura MIMD



Fonte: Adaptado de Tokhi, Hossain e Shaheed (2012)

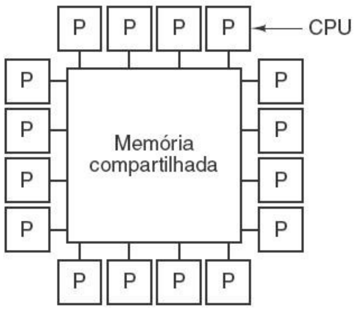
2.3 Tipos de arquiteturas de memórias

Arquiteturas paralelas podem ser classificadas em duas categorias principais em termos de arranjo de memória. São eles: multiprocessadores (memória compartilhada) e multicomputadores (memória distribuída). De fato, essas arquiteturas constituem uma subdivisão da arquitetura paralela do MIMD. Arquiteturas de memória compartilhada e memória distribuída também são chamadas de arquiteturas fortemente acopladas e fracamente acopladas respectivamente. Cada tipo de arquitetura tem suas vantagens e desvantagens (TOKHI; HOSSAIN; SHAHEED, 2012).

2.3.1 Compartilhada

Um computador paralelo no qual todas as CPUs compartilham uma memória comum é denominado um multiprocessador - como indicado simbolicamente na Figura 6. Há somente uma cópia do sistema operacional e todos os processos que funcionam juntos em um multiprocessador podem compartilhar um único espaço de endereço virtual mapeado para a memória comum. qualquer processo pode ler ou escrever uma palavra de memória apenas executando uma instrução *load* ou *store*. Dois processos podem se comunicar pelo simples ato de um deles escrever dados para a memória e o outro os ler de volta. São modelos populares, pois é de fácil de entendimento pelos programadores e é aplicável a ampla faixa de problemas. Em alguns sistemas multiprocessadores, somente certas CPUs têm acesso aos dispositivos de E/S, e, por isso, tem uma função de E/S especial. Em outros, toda CPU tem igual acesso a todo dispositivo de E/S. quando cada CPU tem igual acesso a todos os módulos de memória e a todos os dispositivos de E/S e é tratada pelo sistema operacional como intercambiável com as outras, o sistema é denominado *Symmetric MultiProcessors* (SMP) (TANENBAUM, 2007).

Figura 6 – Multiprocessador com 16 CPUs



Fonte: Adaptado de Tanenbaum (2007)

Em relação ao tipo de acesso às memórias do sistema, multiprocessadores podem ser classificados como:

1. *Uniform Memory Access* (UMA);
2. *Non-Uninform Memory Access* (NUMA);
3. *Cache-Only Memory Access* (COMA).

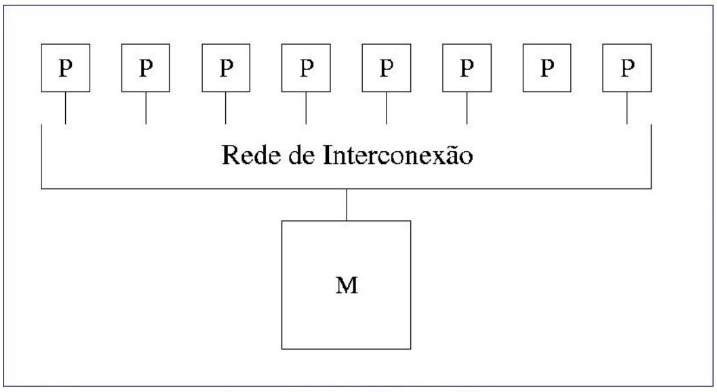
UMA

Classificam-se por serem máquinas que usam memória centralizada e encontra-se à mesma distância de todos os processadores, por conseguinte a latência de acesso à memória é igual para todos os processadores do sistema. Como o barramento é a rede de interconexão mais usada nessas máquinas e suporta apenas uma transação por vez, a memória principal é normalmente implementada como um único bloco. Também se enquadram nessa categoria máquinas com outras redes de interconexão se mantiverem o tempo de acesso à memória uniforme para todos os processadores do sistema (NAVAUX; ROSE; PILLA, 2011).

NUMA

Classificam-se por serem máquinas de memória distribuída, implementada com vários módulos que são associados um a cada processador com espaço de endereçamento é único. Cada processador pode endereçar toda a memória do sistema. Se o endereço gerado pelo processador encontrar-se no módulo de memória ligado a ele diretamente (local), o tempo de acesso a ele será menor que o tempo de acesso a um módulo que está ligado a outro processador diretamente (remoto), que só pode ser acessado através da rede de interconexão. Por esse motivo, essas

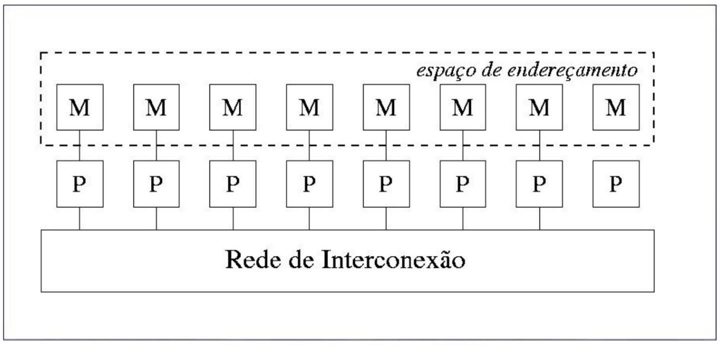
Figura 7 – Arquitetura UMA



Fonte: Adaptado de Navaux, Rose e Pilla (2011)

máquinas possuem um acesso não uniforme à memória, visto que a distância à memória não é sempre a mesma e depende do endereço desejado (NAVAUX; ROSE; PILLA, 2011).

Figura 8 – Arquitetura NUMA



Fonte: Adaptado de Navaux, Rose e Pilla (2011)

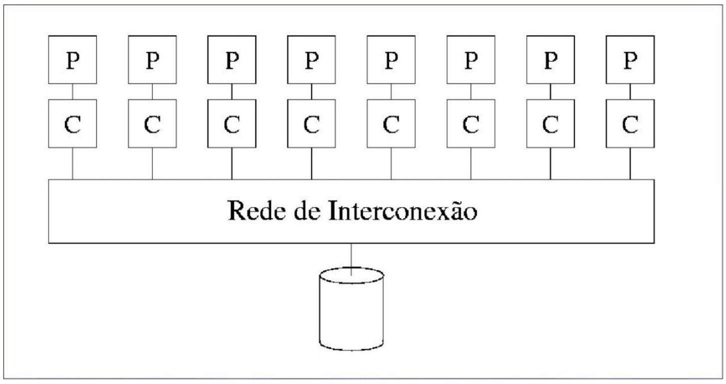
COMA

Classificam-se por serem máquinas com todas as memórias locais estruturadas como memórias cache e são chamadas de COMA caches. Essas caches têm muito mais capacidade que uma cache tradicional. Arquiteturas COMA têm suporte de hardware para a replicação efetiva do mesmo bloco de cache em múltiplos nós fazendo com que essas arquiteturas sejam mais caras de implementar que as máquinas NUMA (NAVAUX; ROSE; PILLA, 2011).

2.3.2 Distribuída

Segundo Tanenbaum (2007) o aspecto fundamental de um Multicomputador que o diferencia de Multiprocessador é que cada unidade dessa arquitetura é um bloco de construção de computadores completo, incluindo o processador, a memória e o sistema de E/S e cada CPU tem sua própria memória local privada acessível somente a ela mesma e nenhuma outra. Para se comunicar elas passam mensagens uma para outra usando a rede de interconexão.

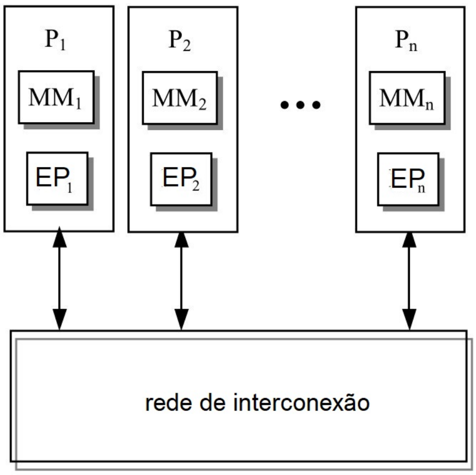
Figura 9 – Arquitetura COMA



Fonte: Adaptado de Navaux, Rose e Pilla (2011)

Por exemplo, se um processador precisar de dados de outro processador, ele enviará um sinal para esse processador por meio de uma rede interconectada, exigindo os dados necessários. A comunicação entre processos costuma usar primitivas de software tais como *send* e `receive`, isso dá ao software uma estrutura muito mais complicada do que de um multiprocessador, ou seja, é muito mais complicado programar em multicomputador em comparação a multiprocessador. Porém é mais simples e barato construir grandes multicomputadores do que multiprocessadores com o mesmo número de CPUs. Uma forma geral de arquitetura de memória compartilhada é mostrada na Figura 10.

Figura 10 – Multicomputador



Fonte: Adaptado de Navaux, Rose e Pilla (2011)

Os multicomputadores podem ser divididos em duas categorias, a primeira chame-se *Massively Parallel Processors* (MPP), são supercomputadores caros que consistem em muitas CPUs fortemente acopladas por uma rede de interconexão proprietária de alta velocidade. A outra categoria consiste em PCs ou estações de trabalho comuns, possivelmente montados em estantes e conectados por tecnologia de interconexão comercial, de prateleira. Em termos de lógica, não há muita diferença, mas supercomputadores enormes que custam muitos milhões de dólares são usados de modo diferente das redes de PCs montadas pelos usuários por uma fração do preço de um MPP. Essas máquinas caseiras são conhecidas por vários nomes, entre eles *Network of Workstations* (NOW), *Cluster of Workstation* (COW) ou apenas *cluster* (TANENBAUM, 2007).

2.4 *Cluster* de Computadores

O conceito fundamental de cluster refere-se a arquitetura de sistema que engloba dois ou mais computadores, chamados de nós, onde um deles é responsável por controlar e gerenciar os demais nós, chamado de mestre. interconectados por uma tecnologia de rede, como a Ethernet, no qual trabalham em conjunto para executar aplicações ou realizar alguma tarefa, de tal forma que para os usuários que os utilizam tenham a impressão que há somente um único sistema respondendo à eles, criando assim uma ilusão de um recurso único (PITANGA, 2003). Esse tipo de sistema começou a ser utilizado em 1960, na IBM, a gigante de informática norte-americana. O principal intuito sempre foi aumentar a eficiência dos processadores. Desde então os *clusters* continuam passando por renovações para melhorar o seu desempenho (TELES, 2018).

Segundo Alecrim (2013) há várias aplicações que devido a sua alta complexidade, só são atendidas de maneira satisfatória com a computação de alto desempenho: sistemas meteorológicos, ferramentas de mapeamento genético, simuladores geotérmicos, programas de renderização de imagens tridimensionais, entre tantos outros. Com o surgimento da computação em nuvens, este cenário se torna ainda mais amplo: pode-se ter uma infraestrutura tecnológica respondendo a vários clientes simultaneamente de maneira remota, por exemplo. Em todos estes casos citados acima e em aplicações do tipo crítica - que não deve parar de funcionar ou não pode perder dados - os *clusters* mostram-se como uma solução viável, desde que o tipo mais adequado seja escolhido. Há vários tipos de *clusters*, porém os principais são: cluster de alto desempenho, cluster de alta disponibilidade e cluster de balanceamento de carga.

2.4.1 *Cluster* de Alta Disponibilidade

Nesse tipo de cluster, o objetivo é prover a disponibilidade de serviços e recursos de forma ininterruptas através do uso de redundância implícitas ao sistema. Se um nó do cluster vier a falhar por qualquer que seja o motivo, as aplicações ou serviços devem estar disponíveis em outro nó. Normalmente são utilizados para base de dados de missões críticas, servidores de arquivos e aplicações (PITANGA, 2003).

2.4.2 *Cluster* de Balanceamento de Carga

Nesse tipo de cluster, as requisições são distribuídas entre os nós de forma mais uniforme possível. O objetivo é fazer com que cada nó processe a uma requisição e não, obrigatoriamente, seja dividida entre outros nós. Não basta ao cluster ter algum mecanismo capaz de partilhar as requisições, é necessário a este procedimento que ao ser executado, consiga manter o equilíbrio no sistema. Para tal, há tipicamente um *daemon* monitorando constantemente os nós para verificar, por exemplo, qual nó está com a maior quantidade de recursos disponíveis e direcionar uma nova requisição para este. Vários tipos de aplicações podem ser utilizadas com o balanceamento de carga, mas é comum o uso em servidores Web, já que soluções do tipo têm maior tolerância ao aumento instantâneo do número de requisições, justamente por causa do equilíbrio oriundo da distribuição de tarefas (ALECRIM, 2013).

2.4.3 *Cluster* de Alto Desempenho

Esse tipo de cluster é direcionado a aplicações que exigem bastante poder de processamento. O objetivo deste tipo de cluster é o de permitir que o processamento de aplicações forneça resultados satisfatórios no menor tempo possível, sistemas utilizados em pesquisas científicas, por exemplo, se beneficiam deste tipo de cluster pela necessidade de analisar uma grande variedade de dados rapidamente e realizar cálculos bastante complexos (ALECRIM, 2013).

O *Cluster* de Alto Desempenho funciona dividindo os trabalhos e distribuindo-os para diferentes nós do cluster. Ter vários computadores trabalhando em uma única tarefa tende a ser mais eficiente do que um computador trabalhando na mesma tarefa. O cluster é configurado com um nó mestre que passa trabalhos para nós escravos. Um nó mestre geralmente será a única máquina que a maioria dos usuários vê; ele protege o resto da rede por trás dele. O nó mestre pode ser usado para agendar trabalhos e monitorar processos, enquanto os nós escravos permanecem inalterados (BOOKMAN, 2003).

Cluster Beowulf

Segundo Bookman (2003) Beowulf é um projeto de *clusters* de computadores para computação paralela criados por Thomas Sterling e Don Becker em 1994 na NASA-Goddard (CESDIS, o Centro de Excelência em Dados Espaciais e Ciências da Informação). Foi construído principalmente devido à insatisfação de fornecedores estabelecidos e à falta de suporte. Tipicamente, um *Cluster Beowulf* é composto por um conjunto de componentes de commodities. Bookman (2003) explica que normalmente um *cluster Beowulf* atende aos seguintes requisitos:

1. Os nós são compostos de software estritamente de código aberto;
2. Os nós e a rede são dedicados ao cluster e são usados apenas para computação de alto desempenho (computação paralela);
3. Os nós são compostos apenas por componentes de commodity e de prateleira.

O alto custo dos supercomputadores da época e a possibilidade de encontrar hardwares de commodities a baixo custo ajudaram a tornar o projeto *Beowulf* bem-sucedido. A disponibilidade de software livre, como o Linux, o PVM e o *Message Passing Interface* (MPI), e compiladores, como o GCC, também serviram para tornar esse tipo de computação acessível (BOOKMAN, 2003).

2.5 Bibliotecas de Comunicação Paralela

As bibliotecas de comunicação paralela possibilitam o desenvolvimento de programas paralelos em ambientes com memória compartilhada e distribuída. Bibliotecas como *openMP*, PVM e MPI possibilitam escrever programas paralelos em C, C++ e Fortran para serem executados em sistemas paralelos. O PVM foi o padrão de fato no desenvolvimento paralelo em sistemas distribuídos até que o MPI surgiu, embora ambos ainda sejam amplamente utilizados. O MPI é padronizado e agora assumiu a liderança em programas portáteis em todos os computadores paralelos (BOOKMAN, 2003).

2.5.1 OpenMP

OpenMP[1] é uma interface de programação (API), portável, de aplicativos para escrever programas *multithread* em Fortran ou C/C++ de memória compartilhada. Sua primeira versão foi lançada em 1997 e tornou-se uma das principais APIs para o desenvolvimento de aplicações pararelas. Assim como no MPI, o paralelismo é explícito. o OpenMP é composto de um conjunto de diretivas de compilador que descrevem o paralelismo no código-fonte, junto com uma biblioteca de suporte de sub-rotinas disponível para aplicativos e variáveis de ambiente. Pode-se combinar o *openMP* com o MPI para ser executado em um cluster de computadores, a fim de otimizar a utilização dos recursos disponíveis e diminuir o número de comunicações realizadas entre os diferentes nós (MATTSON, 2003).

2.5.2 Open MPI

MPI é uma especificação para os desenvolvedores e usuários de bibliotecas de passagem de mensagens. Por si só, não é uma biblioteca — mas sim a especificação do que tal biblioteca deveria ser, baseado no consenso do MPI Forum[2], que tem muitas organizações participantes, incluindo fornecedores, pesquisadores, desenvolvedores de bibliotecas de software e usuários. O objetivo do MPI é estabelecer um padrão portátil, eficiente e flexível para a transmissão de mensagens, que será amplamente usado para escrever programas de transmissão de mensagens. Como tal, o MPI é a primeira biblioteca de passagem de mensagens padronizada, independente do fornecedor. As vantagens de desenvolver software de envio de mensagens usando o MPI correspondem às metas de design de portabilidade, eficiência e flexibilidade. MPI não é um

[1] https://www.openmp.org

[2] comitê composto por uma seção transversal entre representantes da indústria e de pesquisa com o objetivo de discutir a padronização do MPI

padrão IEEE[3] ou ISO[4], mas tornou-se o "padrão da indústria" para escrever programas de transmissão de mensagens em plataformas *High-performance computing* (HPC) (BARNEY, 2018).

As implementações reais da biblioteca MPI diferem em qual versão e recursos do padrão MPI elas suportam. Hoje, o MPI é executado em plataformas de memória distribuída, memória compartilhada e híbrido. Tradicionalmente, é implementado para as linguagens C, C++ e Fortran, porém há implementações menos comuns para Java, Python e IDL. O modelo de programação permanece claramente um modelo de memória distribuída, independentemente da arquitetura física subjacente da máquina. Todo paralelismo é explícito, ou seja, o programador é responsável por identificar corretamente o paralelismo e implementar algoritmos paralelos usando construções de MPI (BARNEY, 2018).

O Open MPI[5] é uma implementação do MPI em código aberto desenvolvida e mantida por um consórcio de parceiros acadêmicos, de pesquisa e do setor. Tendo sua primeira versão lançada em 2005, pode-se dizer que o *Open MPI* é combina os conhecimentos, tecnologias e recursos de toda a comunidade de Computação de Alto Desempenho, com o objetivo de construir uma implementação do MPI completa, gratuita, de código aberto, revisada por pares e com qualidade de produção, oferecendo vantagens para fornecedores de sistemas e software, desenvolvedores de aplicativos e pesquisadores de ciência da computação (Open MPI, 2019).

O Open MPI é inicialmente a fusão entre três implementações de MPI bem conhecidas: FT-MPI da Universidade do Tennessee, LA-MPI do Laboratório Nacional de Los Alamos, LAM/MPI da Universidade de Indiana e com contribuições da equipe PACX-MPI da Universidade de Estugarda. Cada uma dessas implementações de MPI se destacou em uma ou mais áreas. O Open MPI é junção para uma base de código inteiramente nova das melhores ideias dessas quatro implementações MPI. Isso também teve o efeito simultâneo de permitir o descarte do código antigo, que só foi mantido por razões históricas de cada projeto. Como tal, o Open MPI também contém muitos novos designs e metodologias baseados em anos de experiência em implementação de MPI (Open MPI, 2019).

Arquitetura da camada de abstração do Open MPI

Segundo Squyres (2016) o *open MPI* possui três camadas principais de abstração, mostradas na figura 11:

Open, Portable Access Layer (OPAL): é a camada mais inferior das abstrações do Open MPI. Suas abstrações são focadas em processos individuais. Ela fornece funcionalidades como manipulação de strings, controles de depuração e outras funcionalidades básicas. O OPAL também fornece a portabilidade básica do *Open MPI* entre diferentes sistemas operacionais,

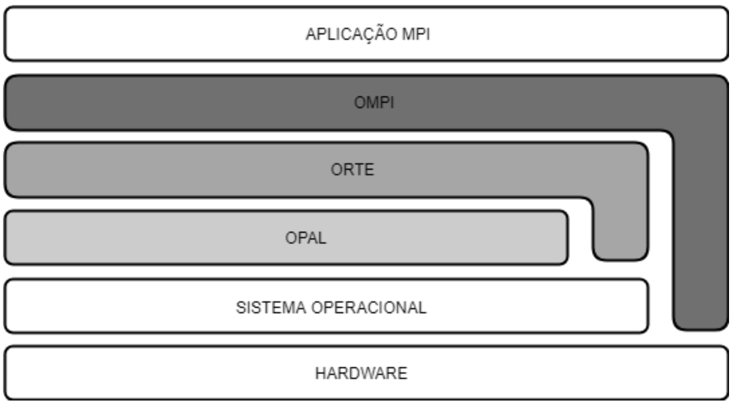
[3] https://www.ieee.org
[4] https://www.iso.org
[5] https://www.open-mpi.org

como a descoberta de interfaces IP, compartilhamento de memória entre processos do mesmo servidor, afinidade de processador e memória, temporizadores de alta precisão, etc.

Open MPI Run-Time Environment (ORTE): fornecer um sistema de tempo de execução para iniciar, monitorar e eliminar tarefas paralelas. No caso do Open MPI, uma tarefa paralela é composta por um ou mais processos que podem abranger várias instâncias do sistema operacional e são unidos para atuar como uma unidade única e coesa. Em ambientes simples com pouco ou nenhum suporte computacional distribuído, ORTE usa rsh ou ssh para iniciar os processos individuais em trabalhos paralelos. Em ambientes mais avançados, os ambientes dedicados a HPC normalmente têm agendadores e gerenciadores de recursos para compartilhar recursos computacionais entre muitos usuários. Esses ambientes geralmente fornecem APIs especializadas para iniciar e regular processos em servidores de computação.

Open MPI (OMPI): A camada MPI é a camada de abstração mais alta e é a única exposta aos aplicativos. A API MPI é implementada nessa camada, assim como toda a semântica de passagem de mensagem definida pelo padrão MPI. Como a portabilidade é um requisito principal, a camada MPI suporta uma ampla variedade de tipos de rede e protocolos subjacentes. Algumas redes são semelhantes em suas características e abstrações subjacentes; alguns não são.

Figura 11 – Arquitetura da camada de abstração do Open MPI



Fonte: Adaptado de Squyres (2016)

2.6 Avaliação de Desempenho

Pode-se pensar em primeiro momento que dividir uma tarefa em n unidades de processamento, os ganhos de desempenho seriam proporcionalmente em n vezes, porém isso nem sempre é possível, o motivo pelo qual nem sempre é possível obter tal resultado envolve vários aspectos tais como o tempo de decomposição, o tempo de processamento e o tempo de comunicação e sincronização entre os processadores (COELHO, 2012). Algumas métricas podem ser utilizadas para medir o desempenho de sistemas paralelos. Entre elas: o tempo de execução, fator de aceleração (*speedup*) e a eficiência (MATTOS, 2008).

O tempo de execução de um algoritmo paralelo é definido como o tempo que o algoritmo demora para resolver um problema computacional. Esse tempo é começa a ser medido no momento em que a execução do algorítimo paralelo inicia até o momento em que o último processador termina de executar, onde *Ts* indica o tempo de execução serial e *Tp* o tempo de execução paralelo (MATTOS, 2008).

A medida de desempenho de um sistema paralelo é o *speedup*, que tem por objetivo medir os ganhos que se obteve ao paralelizar algoritmos seriais, ou seja, a aceleração é utilizada para conhecer o quanto um algoritmo paralelo é mais rápido que sua implementação sequencial. Pode ser obtido com a razão entre o tempo necessário para solucionar um problema de maneira sequencial e o tempo necessário para resolver o mesmo problema de maneira paralela em *n* processadores:

$$S = \frac{T(1)}{T(N)} \tag{2.1}$$

Onde *S* é o speedup e *T(N)* é o tempo gasto para *N* processadores

O ganho de *speedup* deveria tender ao número de processadores utilizados, que seria o seu valor ideal (*speedup* linear), porém na maior parte das vezes o *speedup* é menor que a quantidade de processadores utilizados (*speedup* sublinear). Um estudo básico consiste em analisar qual o máximo ganho que se pode obter com o uso do paralelismo em um problema. Isso pode ser representado pela lei de Amdahl (COELHO, 2012).

A Lei de Amdahl ou também conhecida como limites de *speedup*, formalizada por Gene Amdahl, procura demonstrar que o ganho de velocidade não é infinito mesmo se existissem infinitos processadores em paralelo. Mas que na realidade, o ganho seria proporcional à razão entre as componentes, paralela e serial do programa, assim:

$$S = \frac{1}{(1-P)+\frac{P}{N}} \tag{2.2}$$

Onde, considerando o tempo de execução em série igual a 1, o número de processadores como *N*, parte do programa que não é paralelizável como (*1 – P*) e o percentual do programa que pode ser paralelizado como *P*, e de acordo com a Lei de Amdahl quando o número de processadores tende ao infinito, o *speedup* será no máximo de *1/(1 - P)* (MATTOS, 2008).

De acordo com Grama et al. (2003), somente em um sistema paralelo ideal com número de *N* processadores pode-se obter o valor de *speedup* igual a *N*. De fato, o valor *speedup* ideal não é obtido porque, ao executar um algoritmo paralelo, os processadores não podem dedicar-se 100% do seu tempo aos cálculos do algoritmo, devido a parte do tempo ser desperdiçada com outros fatores, como a comunicação. A eficiência é uma medida da fração de tempo para a qual um dos processadores são realmente utilizados para calcular e pode ser definida como a proporção entre o *speedup* e o número processadores. Em um sistema paralelo ideal, o valor da eficiência é igual a um. A eficiência é denotada pelo símbolo E. Matematicamente, é dado por:

$$E = \frac{S}{N} \tag{2.3}$$

3 LABORATÓRIO PARA O ENSINO DA PROGRAMAÇÃO PARALELA

3.1 Introdução

Nos últimos anos, a *High Performance Computing* (HPC) se tornou uma ferramenta importante para a sociedade moderna, pois passou a ser fonte de um número crescente de aplicativos e serviços. O estudo da HPC é um dos aspectos essenciais nos cursos de Computação, no entanto, o ensino universitário, muitas vezes, sofre com uma lacuna significativa entre os conceitos teóricos e a experiência prática dos alunos (ZARZA et al., 2012).

Um obstáculo presente no ensino de programação paralela, aos estudantes, são as linguagens de programação que são frequentemente utilizadas. Como a principal motivação para o uso deste tipo de programação é o aumento do desempenho, com a efetiva redução do tempo de processamento, as linguagens comumente usadas são aquelas focadas no desempenho, como C e C++, mas, também, Fortran, OpenCL e CUDA. Estas linguagens de programação, porém, são consideradas por muitos alunos, como “difíceis” de aprender.

Somando-se a “dificuldade” da linguagem, tem ainda a mudança no paradigma da programação, pois substitui-se uma programação estruturada, por uma que seja executada de forma paralela (FINLAYSON et al., 2015). Outro obstáculo para aprender programação paralela é a falta de depuradores ou *Integrated Development Environments* (IDE) especificamente voltados para a computação paralela. Aqueles que existem são tipicamente difíceis de usar, proprietários ou necessitam de recursos que ajudem os programadores a depurar código paralelo. Isso é especialmente um problema, pois a depuração de código paralelo é muito mais difícil do que o código sequencial devido à possibilidade de *race condition* e *deadlock* (FINLAYSON et al., 2015).

Pode-se ainda notar o pouco interesse dos alunos por estes tópicos, o que ocorre porque eles são ministrados principalmente focado na teoria dos conceitos, tendo pouco espaço para a prática. A pouca ocorrência da prática, por sua vez, é justificada pela falta de infraestrutura necessária para tal. (BESERRA et al., 2013). A experiência prática através do laboratório é um dos componentes-chave no ensino da Tecnologia da Informação, pois proporciona aos alunos a oportunidade de aprender e observar como aplicar os conceitos (KIM; JIANG; RAJPUT, 2016).

3.2 Ambientes Reais x Ambientes Virtuais

Devido à restrição de custos, as tecnologias de virtualização acabaram sendo amplamente adotadas em muitas áreas. Como a tecnologia virtual fornece o ambiente virtual isolado da máquina hospedeira ou de outras máquinas virtuais, ela se tornou muito popular para soluções de desenvolvimento e educação. A tecnologia de virtualização se torna um componente essencial na educação de TI, porque vários sistemas virtualmente isolados podem ser criados, modificados, testados e excluídos facilmente com pouco ou nenhum custo adicional (KIM; JIANG; RAJPUT, 2016).

Ghannoum e Rodrigues (2018) apontam as seguintes vantagens e desvantagens da virtualização como mostra a tabela 1.

Tabela 1 – Vantagens e desvantagens da virtualização

Vantagens	Desvantagens
Segurança: Máquinas isoladas, uma vulnerabilidade de uma forma não prejudicial como demais	Espaço em disco: O espaço de disco necessário é maior, pois cada máquina possui todos os arquivos de sistema instalados.
Facilidade de gerenciamento: gerenciamento centralizado das máquinas através de uma interface única	Sem acesso direto ao hardware: As máquinas não possuem acesso direto ao hardware, como algumas placas e dispositivos USB conectados no servidor físico.
Compatibilidade: Compatível com todos os tipos de aplicativos.	Desempenho: Com uma camada a mais de software implementada, o processamento necessário para o funcionamento das máquinas é maior do que um ambiente sem virtualização.
Aproveitamento: Maior aproveitamento dos recursos de hardware disponíveis.	Segurança: Uma camada a mais de software significa mais vulnerabilidade para o ambiente.
Ambiente de teste: Possível configurar ambientes de testes sem necessidade de aquisição de novos equipamentos para esse fim	Acúmulo de máquinas virtuais: a implementação de várias máquinas virtuais pode gerar um acúmulo de máquinas ociosas
	Performance de softwares ou sistemas: Alguns softwares não são totalmente compatíveis com sistemas virtualizados.

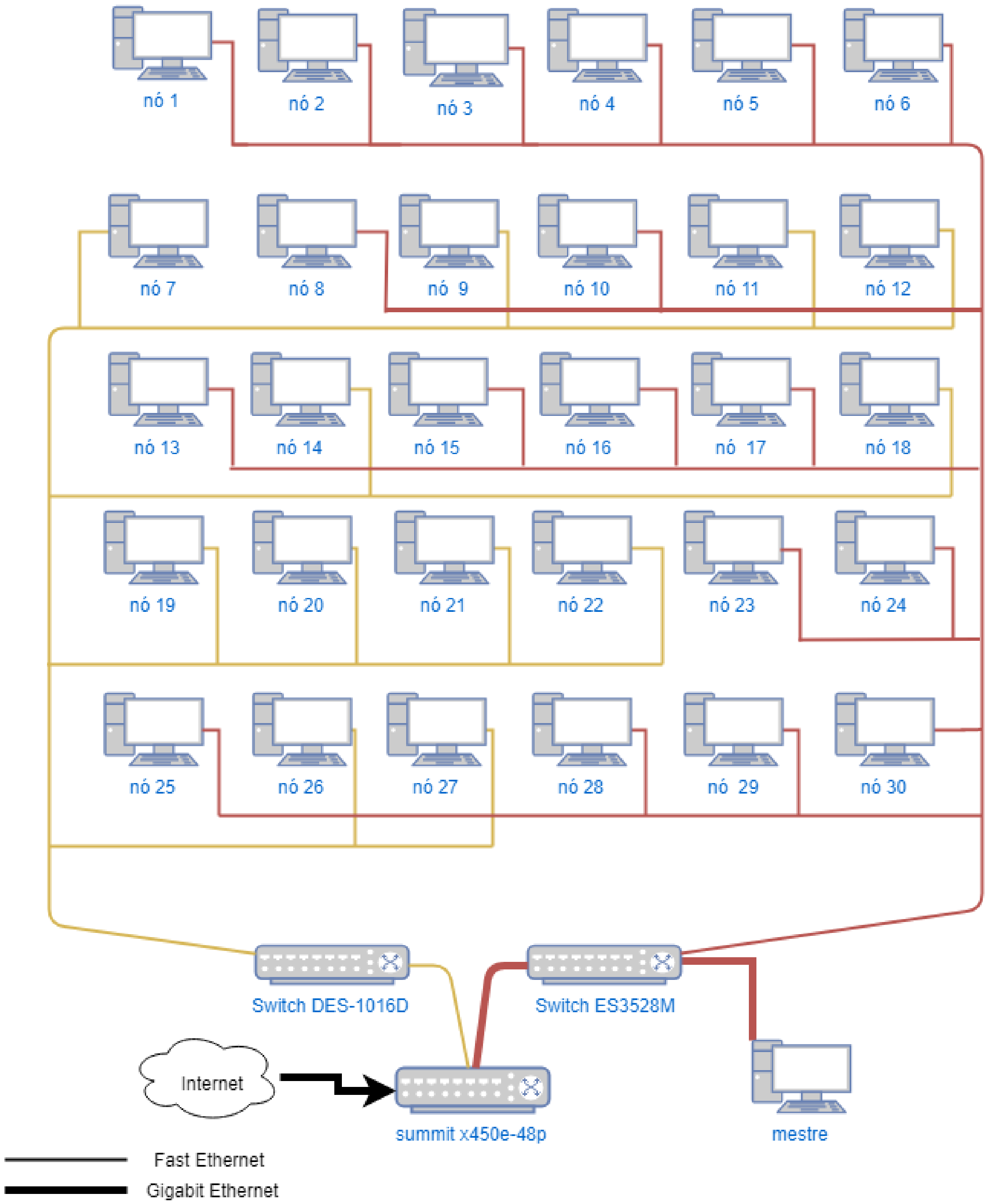
Fonte: Ghannoum e Rodrigues (2018)

3.3 Cluster Orã

A implantação de um *cluster beowulf* no Laboratório de computação da UFPa é uma solução alternativa viável para substituir um supercomputador, visando no auxílio do ensino e aprendizagem dos cursos de computação paralela e distribuída com experiências práticas no currículo de ciência da computação, como também para a pesquisas que necessitem de alto poder computacional.

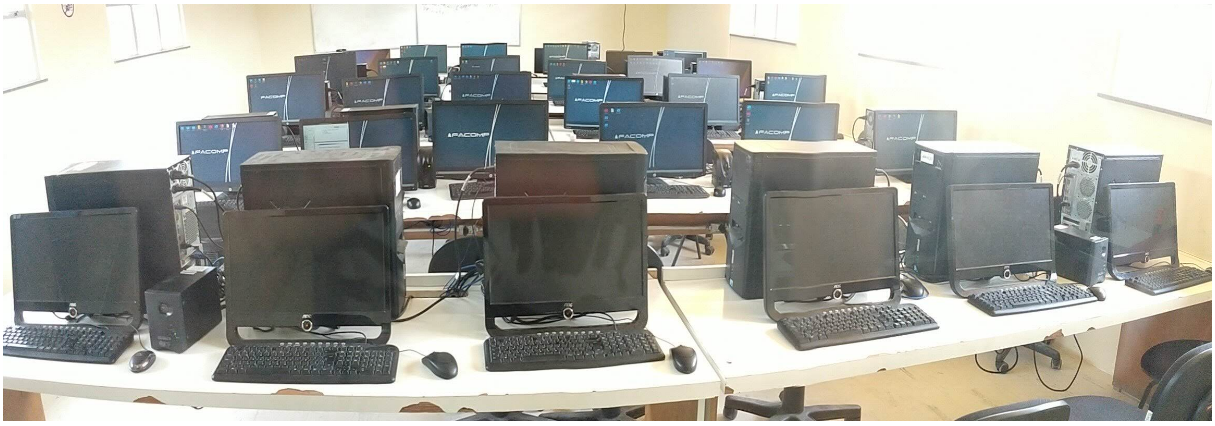
A seguir, a Figura 13 e 12 apresentam, respectivamente, o arranjo da arquitetura que forma o *cluster* do projeto e os computadores do Laboratório de informática da UFPa.

Figura 12 – Arquitetura do *cluster Beowulf*



Fonte: Elaborada pelo autor

Figura 13 – Imagem do Laboratório de Computação da UFPa



Fonte: Elaborada pelo autor

3.3.1 Hardware e Infraestrutura de rede

Como pode ser visto na Figura 12, foram utilizados no total 31 computadores com as seguintes configurações, como ilustra a Tabela 2:

Tabela 2 – Relação das configurações dos computadores utilizados no cluster

Nº de Computadores	Nó	configuracao
1	mestre	Processador Intel Core i7-3770 3.4GHz, 8 GB de Memória RAM DDR3, HD de 500 GB
6	1 - 6	Processador Intel Core i7-6700 3.4 GHz, 16 GB de Memória RAM DDR4, HD de 1 TB
18	7 - 24	Processador Intel Core i5-4570 de 3,2 GHz, 8 GB de memória RAM DDR3, HD de 1 TB
6	25 - 30	Processador Pentium E2180 de 2 GHz, 4 GB de memória RAM DDR2, HD de 160 GB

Fonte: Elaborada pelo autor

A infraestrutura de rede deste projeto é a mesma já utilizada no laboratório sem nenhuma modificação, no qual são utilizados 3 *switches* com as seguintes configurações como ilustra a Tabela 3:

Tabela 3 – Relação das configurações dos *switches* utilizados no *cluster*

Modelo	Configuração
Switch DES-1016D	16 portas 10/100 Mbits/s (RJ-45)
Switch ES3528M	24 portas 10/100 (RJ45) e 4 portas Gigabit (RJ-45/SFP)
Switch summit x450e-48p	48 Portas 10/100/1000 (RJ45) e 4 Portas SFP compartilhada

Fonte: Elaborada pelo autor

3.3.2 Software

Os software utilizados em todos os nós (incluindo o mestre):

1. SO GNU/Linux Kubuntu 18.04 LTS de 64 bits: é um sistema derivado do ubuntu que utiliza do ambiente gráfico KDE plasma com a proposta de ter uma Interface Leve e ao mesmo tempo de fácil utilização por qualquer pessoa;
2. OpenSSH: ferramenta de conectividade para login remoto com o protocolo SSH.
3. Sistema de arquivos de rede (NFS): protocolo que permite compartilhar arquivos e diretórios entre computadores conectados em rede;
4. OpenLDAP: é uma implementação de código aberto do serviço de diretório LDAP, que permiti o compartilhamento de informações sobre usuários através da rede;
5. Biblioteca Open MPI Versão 4.0.0

3.3.3 Instalação e Configuração do *Cluster* Orã

Nas etapas seguintes descreve-se todos os passos resumidos desde o início até o fim das configurações do cluster.

Passo 1 - Instalação do S.O Kubuntu

A etapa de configuração começou com a instalação do SO nos 31 computadores que foram usadas no *cluster*, visto que os computadores antes estavam com Windows 10. Depois de concluída a instalação, todos os computadores foram atualizados.

Passo 2 - Configuração de Rede

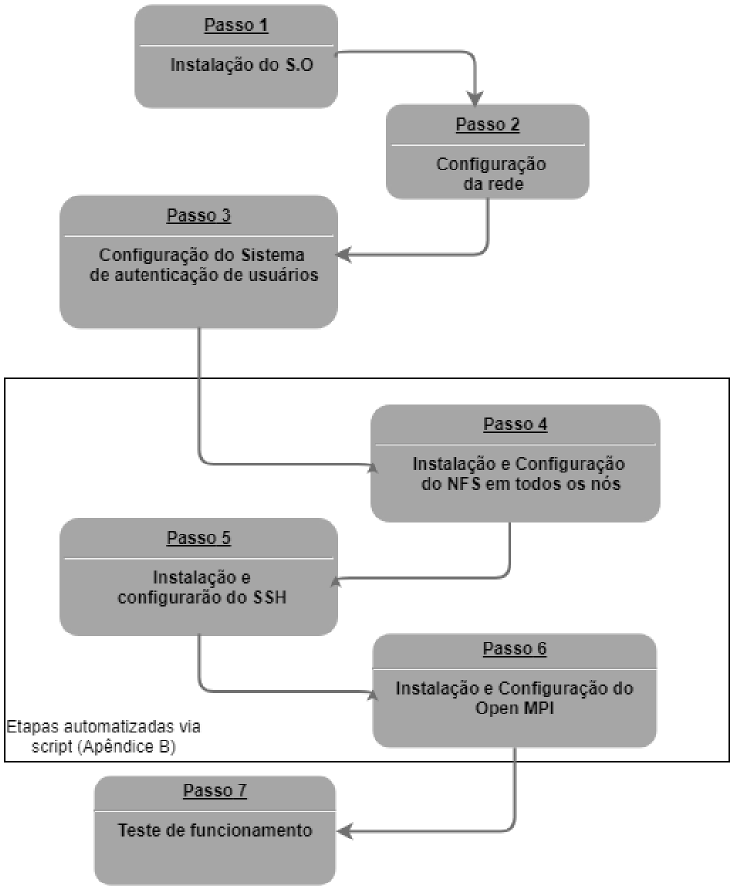
É mais fácil se os nós puderem ser acessados com o seu nome em vez de seu endereço IP. Por este motivo, foi adicionado o nome de todos os nós ao arquivo *hosts* de todos os nós. Porém seria um problema caso os endereços IPs dos nós mudem, o arquivo /etc/hosts ficaria desatualizado e os nós não iriam mais conseguir se conectar entre si. Então deve-se configurar os IPs dos nós para serem estáticos.

Passo 3 - Sistema de autenticação de usuários (LDAP)

Para que os usuários consigam executar trabalhos no *cluster*, alguns requisitos precisam ser atendidos:

1. Cada nó do *cluster* precisa compartilhar o mesmo diretório para que seja possível acessar o arquivo a ser executado, Neste projeto o diretório é compartilhado por meio do NFS.

Figura 14 – Fluxograma dos passos de instalação e configuração Do *Cluster* Orã



Fonte: Elaborada pelo autor

2. Os usuários precisam ter os mesmos nomes e ID de usuário porque as permissões de acesso ao diretório NFS são verificadas pelo ID do usuário.

Diante disso, seria muito dispendioso ter que criar uma nova conta de usuário em cada nó para cada novo usuário que for utilizar o *cluster*. Para contornar esse problema será utilizado o LDAP para a autenticação de usuários através da sua base de dados centralizada. Assim, cada conta de usuário só precisa ser criada uma única vez no nó mestre e cada nó escravo já terá acesso a ela.

A interface web para o gerenciamento de entradas (por exemplo, usuários e grupos) armazenadas em um diretório LDAP, *LDAP Account Manager*(LAM) também foi instalado. A ferramenta foi projetada para tornar o gerenciamento LDAP o mais fácil possível para o usuário, facilitando a administração de entradas LDAP, abstraindo os detalhes técnicos do LDAP e permitindo que administradores e usuários sem experiência técnica gerenciem o servidor LDAP. Se necessário, os usuários experientes podem editar diretamente as entradas LDAP por meio do navegador LDAP integrado (Josphat Mutai, 2018).

Passo 4 - Instalação e Configuração do NFS

Os arquivos e programas usados para tarefas que são executadas em paralelo no cluster precisam estar disponíveis para todos os nós. Portanto, damos a todos os nós acesso a uma parte do sistema de arquivos no nó principal. O *Network File System* (NFS) permite montar parte de um sistema de arquivos remoto para que você possa acessá-lo como se fosse um diretório local.

O *firewall* está ativado por padrão no Kubuntu e bloqueará o acesso quando um cliente tentar acessar um diretório compartilhado do NFS. Portanto, precisa-se adicionar uma regra ao UFW (ferramenta para gerenciar o *firewall*) para permitir o acesso de uma sub-rede específica.

Passo 5 - Instalação e configuração do SSH

Para o *cluster* funcionar, o nó mestre precisa ser capaz de se comunicar com os nós escravos e vice-versa. O *Secure Shell* (SSH) é geralmente usado para acesso remoto seguro entre computadores. Ao configurar o SSH sem senha entre os nós, o nó mestre é capaz de executar comandos nos nós escravos. Isso é necessário para executar os *daemons* MPI nos nós de computação disponíveis.

Passo 6 - Instalação e configuração do *Open MPI*

Nesta etapa, foi realizada a instalação e configuração do *Open MPI*. O pacote foi baixado na versão 4.0.0 em sua página oficial[1] no formato compactado tar.gz.

Passo 7 - Teste de Funcionamento

Para verificar o funcionamento do *Cluster* Orã, foi usado o algoritmo *hellow.c* (Anexo C). Este é um algoritmo simples, baseado em (MPICH, 2018b) que tem como resultado uma mensagem de “Olá Mundo” de cada processo criado, especificando qual é o sua identificação, números de processos executados no mesmo nó e o nome do nó aonde foi executado.

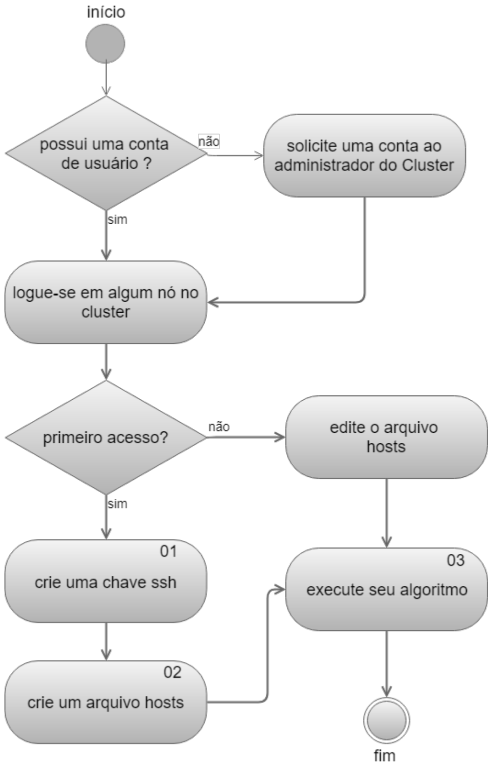
A Configuração passo a passo do *cluster* pode ser vista no Apêndice A.

3.3.4 Gerência e acesso do usuário ao *Cluster* Orã

A Figura 15 descreve o fluxograma das ações que deve-se ser realizada pelo usuário para utilizar o *Cluster* Orã .

[1] http://www.open-mpi.org/software/ompi

Figura 15 – Fluxograma de utilização do *Cluster* Orã



Fonte: Elaborada pelo autor

[01] Mais detalhes sobre como criar uma chave SSH é mostrado no Apêndice A.

[02] O arquivo *hosts* deve conter o nome das máquinas que serão utilizadas na execução de algum algoritmo. O nome das máquinas é LABCOMP2-PC"X", com "X" variando entre 01 e 30. aconselha-se criar o arquivo *hosts* no mesmo diretório do algoritmo a ser executado. Mais detalhes podem ser vistos na Figura 42 no Apêndice A.

[03] O algoritmo é compilado usando o comando "mpicc nome_do_algoritimo.c" e pode ser executado usando o comando "mpirun -np *Y* –hostfile hosts a.out". *Y* é o número de processos a serem criados e *hosts* é o arquivo contendo o nome dos nós que serão utilizadas na execução.

4 APLICAÇÕES E RESULTADOS

Para os testes e validações deste trabalho foram executados dois algoritmos editados de forma a executar necessitando de bastante tempo de processamento. O primeiro algoritmo a ser testado é o “cpi_montecarlo.c” (Anexo A) e o segundo foi o “numeros_primos.c” (Anexo B)

4.1 Algoritmo cpi_montecarlo.c

O algoritmo usado neste projeto foi baseado em (RIBAS, 2013) e (MPICH, 2018a) e tem o proposito de estimar o valor de PI usando o método de monte carlo.

Monte Carlo é um termo utilizado para se referir a qualquer método que resolve um problema utilizando amostragens aleatórias repetidas e observando se essa fração de números obedece a alguma propriedade estabelecida (WEISSTEIN, 2019a). A seguir é apresentada uma descrição do método a ser empregado na implementação do algoritmo.

Levando em consideração que a área da circunferência de raio r é:

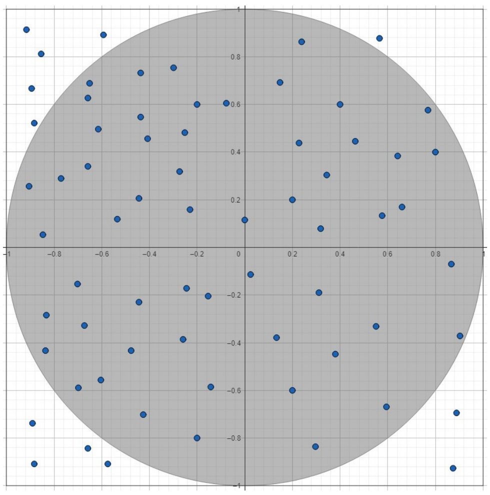
$$A = \pi * r^2 \tag{4.1}$$

e quando o r é igual 1, a área é igual ao valor da constante π

$$A(r = 1) = \pi \tag{4.2}$$

Então calculando o valor da área dessa circunferência, descobre-se o valor de π. Para calcular a área da circunferência, coloca-se a circunferência dentro de um quadrado – figura geométrica fácil de calcular a área – e sorteia-se pontos aleatórios dentro da figura.

Figura 16 – Distribuição aleatória de pontos dentro da área de circulo/quadrado



Fonte: Adaptado de Ribas (2013)

Se o ponto gerado aleatoriamente estiver dentro da circunferência, então somamos um ao contador (P_{in}). Ao final, espera-se que a área da circunferência seja proporcional a taxa de pontos dentro circulo e à área do quadrado. Esse valor será uma aproximação para o valor de π. Porém, para um grande número de pontos gerados, ele deve convergir para o valor real de π e é dado por:

$$\pi = 4 * \frac{P_{in}}{P_{total}} \tag{4.3}$$

Com 4 sendo o valor da área do quadrado, P_{in} sendo os pontos gerados aleatoriamente que ficaram dentro do circulo e P_{total} o número total de pontos gerados.

O algoritmo consiste em gerar pontos(x,y) aleatórios, sendo $0 < x < 1$ e $0 < y < 1$, identificando e contando P_{in} e aplicando a equação 4.3 para se obter o valor de π como mostrado no algoritmo 1.

Algoritmo 1: FRAGMENTO DO ALGORITMO CPI_MONTECARLO.C

```
1  MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
2  for (i = myid + 1; i <= N; i += numprocs) do
3  |   x = ((double)(rand())/RAND_MAX);
4  |   y = ((double)(rand())/RAND_MAX);
5  |   if (pow(x,2) + pow(y,2)) <=1) then
6  |   |   cont+=1;
7  |   end
8  end
9  mypi = 4.0 * (double) cont / N;
10 MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
```

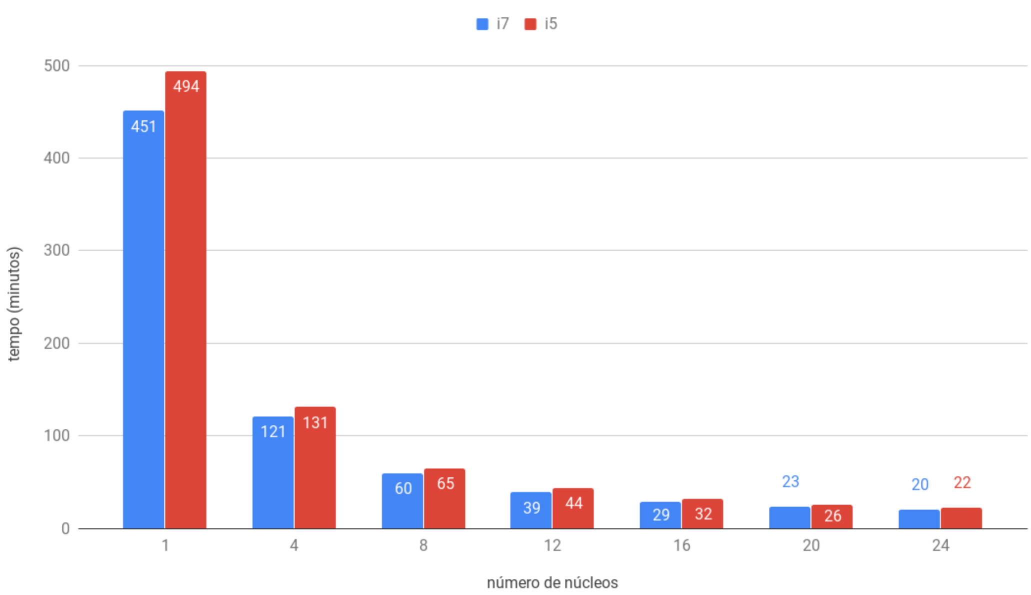
Linha 1: Transmite uma mensagem do processo com *rank* 0 para todos os outros processos;
Linha 2: Divide o intervalo de trabalho que cada processo executará igualmente;
Linha 3 e 4: Gera um número entre 0 e 1 para a coordenada x e y;
Linha 5 e 6: Verifica se o ponto gerado está dentro do circulo usando o teorema de Pitágoras e soma recebe + 1 se a condição for verdadeira;
Linha 9: aplica-se a equação 4.3 para se obter o valor de parte do valor de π;
Linha 10: junta as partes do valor de π de cada processo para obter o valor total;

O algoritmo completo pode ser visto no Anexo A.

Teste 01:

Foi mostrado no Capítulo 3 que o cluster dispõe de uma arquitetura com nós de diferentes configurações, o teste 01 foi aplicado com o objetivo de descobrir se existia e qual era a diferença de desempenho para diferentes quantidades de núcleos usados. O algoritmo 1 foi executado 5 vezes para cada quantidade de núcleo para os nós com processador I5 e I7. O tempo médio de execução foi observado e relatado no gráfico representado pela Figura 17. Todos os tempos foram registrados usando a função *MPI_Wtime()* do MPI.

Figura 17 – Comparação do tempo médio de execução do CPU I7 e I5



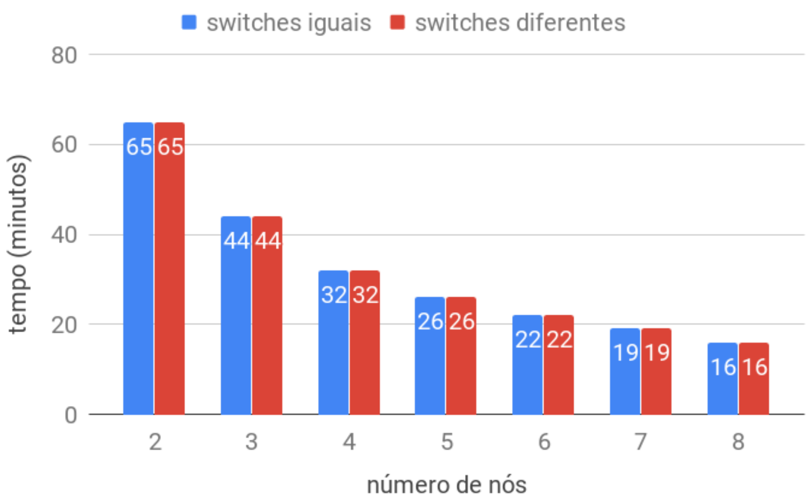
Fonte: Elaborada pelo autor

Como pode-se ver na Figura 17, o desempenho do processador I7 foi maior que o I5 para todas as quantidades de núcleos usados, resultado já esperado visto que o processador i7 é vendido pela fabricante como um processador mais potente, como também pelo fato do I7 (6ª geração) ser de uma geração mais nova que o I5 (4ª geração). O tempo médio de execução para o teste sequencial foi aquele que mais apresentou diferenças – 43 minutos – e foi diminuindo ao passo que mais núcleos eram adicionados até a diferença ser somente de 2 minutos.

Teste 02:

Também foi mostrado no capítulo 3 que o cluster dispõe de mais um *switch* conectando os nós, o teste 02 foi aplicado com o objetivo de descobrir se existia e qual era a diferença de desempenho para os nós que estavam conectados no mesmo *switch* e *switches* diferentes, o algoritmo 1 foi executado 5 vezes para cada quantidade de núcleo para os nós conectados no mesmo *switch* e *switches* diferentes. O tempo médio de execução foi observado e relatado na Figura 18. Todos os tempos também foram registrados usando a função *MPI_Wtime()* do MPI.

Figura 18 – Comparação do tempo médio de execução usando *switches* diferentes



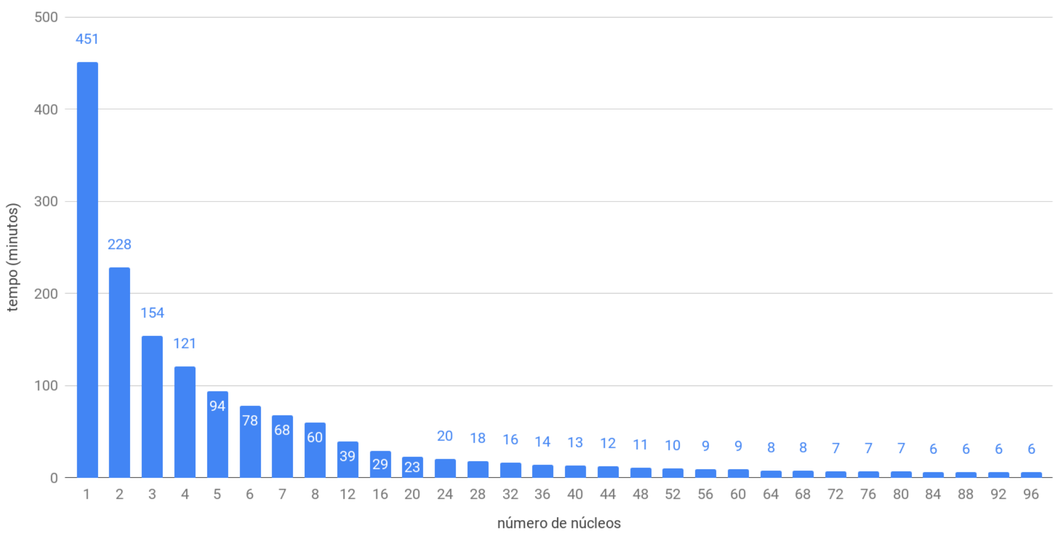
Fonte: Elaborada pelo autor

Pode-se ver na Figura 18, o tempo médio de execução dos nós conectados no mesmo *switch* e *switches* diferentes foram os mesmos, esse resultado pode ter sido obtido em razão do algoritmo 1 fazer pouca comunicação entre os processos durante a sua execução.

Teste 03:

Para o terceiro teste foi aplicado o algoritmo 1 para diferentes quantidades de núcleos, com o objetivo de mensurar o quanto um problema poderia ter ganho de desempenho ao ser executado no cluster. O algoritmo foi executado 5 vezes para cada quantidade de núcleo. O tempo médio de execução foi observado e relatado no gráfico representado pela Figura 19. Todos os tempos foram registrados novamente usando a função *MPI_Wtime()* do MPI.

Figura 19 – Tempo médio de execução (em minutos) do algoritmo cpi_montecarlo.c

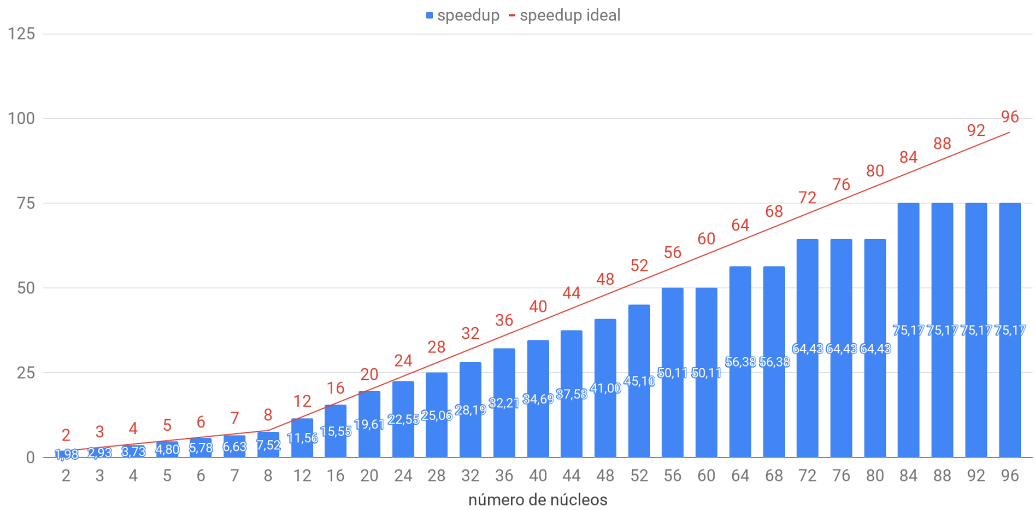


Fonte: Elaborada pelo autor

Pode-se observar na Figura 19 que à medida que um núcleo é adicionado, o tempo médio de execução cai de modo considerável. É possível analisar esse fenômeno até o momento em que os testes começam a utilizar 24 núcleos, a partir desse valor, o tempo de execução diminui sutilmente. Também é possível notar que ao utilizar 60 núcleos ou mais, a diferença desse tempo encontrada é mínima.

A Figura 20 mostra o *speedup* obtido a partir da relação do número de núcleos usados no teste 03. Note que *speedup* ideal é obtido quando ***speedup* = número de núcleos usados**. O *speedup* obtido chega a ser bem próximo do ideal até os 24 núcleos. A partir dos 25 núcleos, percebe-se que ainda há ganho de

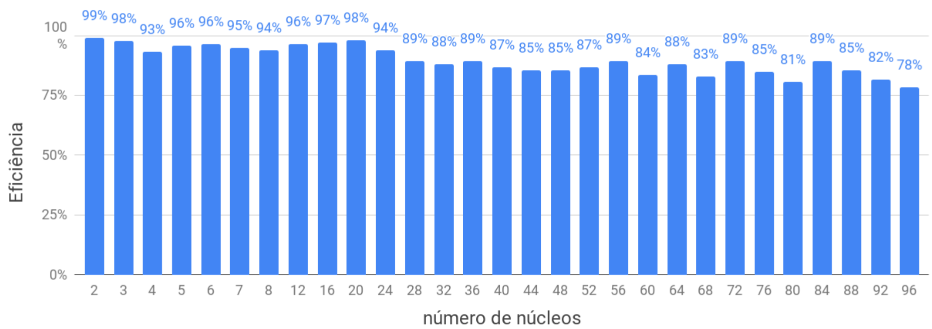
Figura 20 – *Speedup* para diferentes quantidades de núcleos do algoritmo cpi_montecarlo.c



Fonte: Elaborada pelo autor

A Figura 21 mostra a eficiência obtida a partir da relação do número de núcleos usados no teste 03. Note que a eficiência de 100% é alcançada somente quando o *speedup* é ideal. E como visto na Figura 20, o *speedup* chega a ser bem próximo do ideal até usar o 24.º núcleo, logo, a eficiência também ficou bem próximo dos 100% (eficiência ideal) quando executado até os 24 núcleos.

Figura 21 – Eficiência para diferentes quantidades de núcleos do algoritmo cpi_montecarlo.c



Fonte: Elaborada pelo autor

4.2 Algoritmo numeros_primos.c

Um número primo é um inteiro positivo com exatamente um divisor positivo diferente de 1, significando que é um número que não pode ser fatorado. Por exemplo, os únicos divisores de 13 são 1 e 13, tornando 13 um número primo, enquanto o número 24 tem divisores 1, 2, 3, 4, 6, 8, 12 e 24, fazendo 24 um número que não é primo (WEISSTEIN, 2019b).

Os números primos são fundamentais nos sistemas de criptografia modernos. É graças a estes números que é possível que tenhamos segurança em nossas transações financeiras, por exemplo. Devido as suas propriedades muito especiais para fatoração, embora não seja tão difícil encontrar números primos relativamente grandes, é inevitavelmente difícil de levar um número grande de volta para primos. Uma coisa é descobrir que 35 é (7 x 5), e outra bem diferente para descobrir que 2.244.354 é (2 x 3 x 7 x 53437). E é outra bem diferente novamente para encontrar os fatores primos de um número cinquenta dígitos. Um supercomputador pode levar até alguns milhões anos ou até mais para resolver um problema de fatoração de 256 bits (AKEL, 2016).

O algoritmo usado neste projeto foi baseado em (DEVDBI, 2016) e tem o proposito de calcular a quantidade existente de números primos no intervalo de 1 a 1000000000.

Algoritmo 2: FRAGMENTO DO ALGORITMO NUMEROS_PRIMOS.C

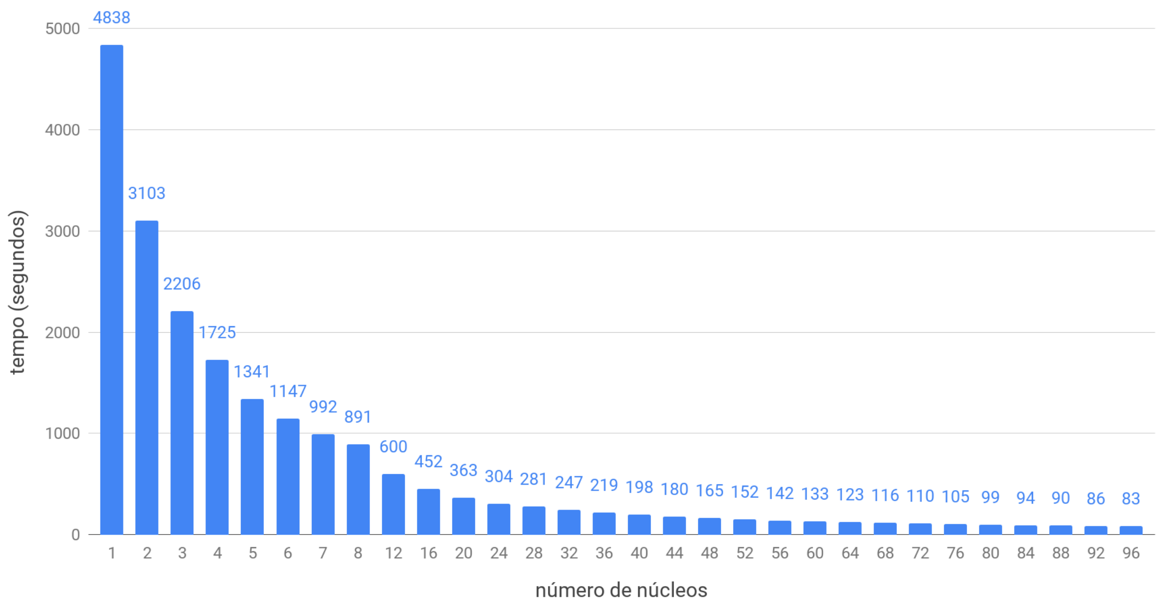
```
MPI_Init(&argc, &argv);
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
MPI_Comm_size(MPI_COMM_WORLD, &nprocs);
if (myrank == 0) then
    startwtime = MPI_Wtime();
end
split = ceil((float)(quant)/nprocs);
begin = split * myrank;
end = split * (myrank+1);
for (i = begin; i < end; ++i) do
    if (is_prime(i)) then
        localSum++;
    end
end
if (myrank == 0) then
    globalSum = localSum;
    for (i = 1; i < nprocs; ++i) do
        MPI_Recv(&tmp,1,MPI_INT,i,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
        globalSum += tmp;
    end
    endwtime = MPI_Wtime();
else
    MPI_Send(&localSum,1,MPI_INT,MASTER,0,MPI_COMM_WORLD);
end
```

Linha 1: Define e inicializa o ambiente necessário para executar o MPI;
Linha 2: Identifica o processo e coloca sua id em *myrank*;
Linha 3: Retorna o número de processos criados;
Linha 4, 5 e 6: Começa contar o tempo pelo processo que possui o id igual a "0";
Linha 7: Define o intervalo que cada processo vai calcular;
Linha 8: Define o inicio que cada processo vai começar a calcular;
Linha 9: Define até onde cada processo vai calcular;
Linha 10: *loop* começa e termina no intervalo que cada processo pode atuar
Linha 11 e 12: Verifica se o número é primo usando a função is_prime() [1]
Linha 15: Verifica se o processo é o de ID igual a "0";
Linha 17, 18 e 19: Executa um *loop* para receber os resultados de todos os outros processos e adiciona o resultado a variável *globalSum*;
Linha 23: todos os outros processos enviam seus resultados para o processo de ID igual a "0";

Teste 01:

O algoritmo 2 foi executado em diferentes quantidades de núcleos, com o objetivo de mensurar quanto um problema poderia ter ganho de desempenho ao ser executado no cluster. O algoritmo foi executado 5 vezes para cada quantidade de núcleo. O tempo médio de execução foi observado e relatado na Figura 22. Todos os tempos foram registrados novamente usando a função *MPI_Wtime()* do *MPI*.

Figura 22 – Tempo médio de execução (em segundos) do algoritmo numeros_primos.c



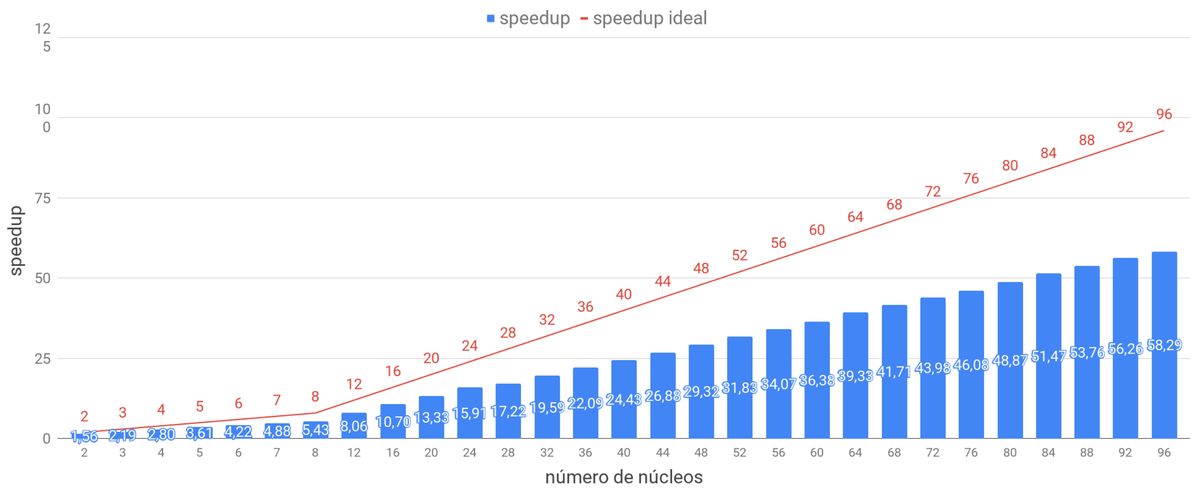
Fonte: Elaborada pelo autor

[1]Função pode ser conferida no algoritmo completo no Anexo B

Pode-se observar na Figura 22 que a medida que um núcleo é adicionado, o tempo médio de execução cai de modo considerável, é possível observar esse fenômeno até o número de núcleos igual a 5, que é quando o tempo começa a cair mais sensivelmente até começar a estabilizar aos 24 núcleos.

A Figura 23 mostra o *speedup* obtido a partir da relação do número de núcleos usados no teste 01. O *speedup* obtido chega a ser bem próximo do ideal até o 5.º núcleos utilizado. A partir do 8.º núcleo, o *speedup* ainda aumenta gradativamente, porém se afastando cada vez mais do *speedup* ideal.

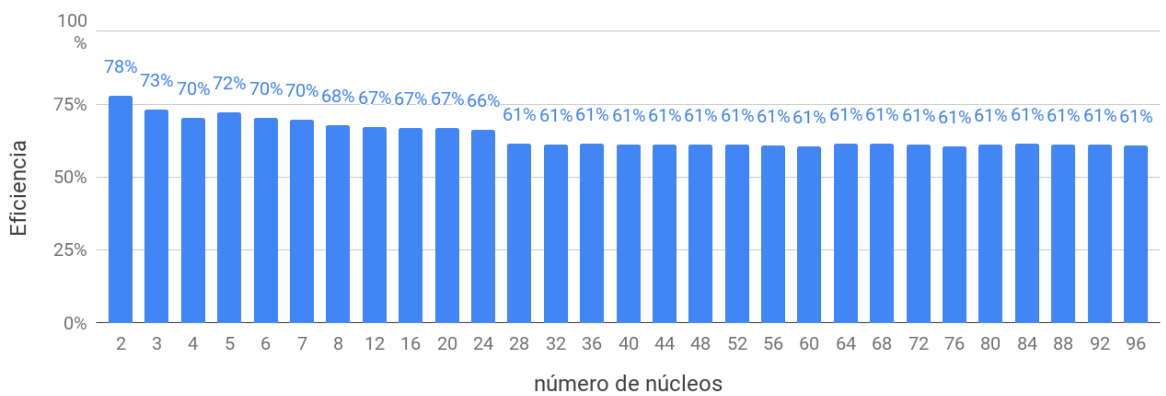
Figura 23 – *Speedup* para diferentes quantidades de núcleos do algoritmo numeros_primos.c



Fonte: Elaborada pelo autor

A Figura 24 mostra a eficiência obtida a partir da relação do número de núcleos usados no teste 01. Pode-se notar que há somente variação de eficiência até o 24.º núcleo usado e que a eficiência obtida é bem menor que a eficiência ideal.

Figura 24 – Eficiência para diferentes quantidades de núcleos do algoritmo numeros_primos.c



Fonte: Elaborada pelo autor

5 CONCLUSÃO

Considerações Finais

No presente trabalho discutiu-se alguns conceitos envolvendo computação de alto desempenho e a sua importância atualmente, vimos que o surgimento dos *clusters* computacionais foi uma forma de utilizar computadores comuns interligados através de uma rede para executarem tarefas que até então só eram executadas em tempo hábil por supercomputadores, demonstrando assim, a sua viabilidade de construção, tanto financeira como técnica, como alternativa barata, de alto desempenho, à resolução de problemas em computacionais.

Na etapa do desenvolvimento do projeto, foram encontradas dificuldades devido aos conhecimentos exigidos em Linux e programação paralela usando MPI para a realização do projeto, tornando cada etapa da configuração uma nova busca por conhecimento que viesse suprir a necessidade do projeto. Outra dificuldade foi o processo de instalação do sistema operacional nos computadores do laboratório, pois como os computadores são usados em aula durante o semestre, precisou haver certa "correria" quanto a troca de sistema operacional, já que tempo para a realização da instalação do novo sistema operacional tinha um período limitado e fixo de tempo.

Com os testes apresentados, os resultados finais se mostraram satisfatórios, provando que o processamento paralelo é vantajoso. Nos dois algoritmos testados no *Cluster* Orã, o processamento paralelo em vinte e quatro nós conseguiu cerca de um tempo de execução aproximadamente 20 vezes melhor, se comparado com a execução dos mesmos algoritmos em apenas um nó utilizando todos os seus núcleos físicos. Observa-se que, os algoritmos testados fazem pouca comunicação entre os processos, gerando assim pouco tráfego de rede, o que fez com que estes algoritmos se beneficiassem da paralelização.

Esse foi o primeiro projeto a implementar um *Cluster Beowulf* em um dos laboratórios da faculdade de computação da UFPa. E espera-se com ele que, vários estudantes possam fazer proveito dessa estrutura já construída para auxiliar na realização de seus projetos acadêmicos que necessitam de grande poder computacional e também podendo ser utilizado em aulas como uma ferramenta para apoiar a prática do ensino de programação paralela.

Este trabalho entrega como principais contribuições o desenvolvimento de um *Cluster Beowulf*, com a descrição em detalhes sobre todas as configurações necessárias para o seu funcionamento. Além disso, o trabalho ainda tem como aporte um script que auxilia fazendo a automação de alguns passos no desenvolvimento do cluster.

Trabalhos Futuros

Como trabalho futuro, sugere-se a criação de um ferramenta capaz de gerenciar o cluster. Apesar de o *cluster* Orã possuir uma quantidade significativa de nós, os algoritmos que forem enviados para a execução podem acabar ficando numa longa fila se um usuário enviar o seu algoritmo para ser executado nas mesmas máquinas que outro usuário já esteja usando. A ferramenta seria uma interface entre os usuários e o *cluster*, recebendo e enfileirando todas os solicitações e alocando nos nós que tivessem mais ociosos, monitorando o uso de memória RAM, CPU e Rede.

REFERÊNCIAS

AKEL, A. **A importância dos números primos – Unidades Imaginárias – Medium**. 2016. Disponível em: <https://medium.com/unidades-imaginárias/a-importância-dos-números-primos-1249a54cc57e>. 44

ALECRIM, E. **Cluster: conceito e características**. 2013. Disponível em: <https://www.infowester.com/cluster.php>. 24, 25

BACELLAR, H. V. Cluster: Computação de alto desempenho. **Instituto de Computação, Universidade Estadual de Campinas. Campinas, São Paulo**, 2009. 13

BARNEY, B. et al. Introduction to parallel computing. **Lawrence Livermore National Laboratory**, v. 6, n. 13, p. 10, 2010. 17

BARNEY, L. B. **Message Passing Interface (MPI)**. 2018. Disponível em: <https://computing.llnl.gov/tutorials/mpi/>. 27

BESERRA, D. et al. Ambiente virtualizado para ensino de programaçao paralela e computaçao em cluster. In: **XXXIII Congresso da Sociedade Brasileira de Computação/XXI Workshop sobre Educação em Computação, Maceió**. [S.l.: s.n.], 2013. 31

BOOKMAN, C. **Agrupamento de computadores em Linux**. [S.l.]: Ciência Moderna, 2003. 25, 26

COELHO, S. A. Introdução a computação paralela com o open mpi. **Simpósio Brasileiro de Computação Paralela com o Open MPI**, p. 24–44, 2012. 28, 29

COSTA, R. Anã lise de desempenho de um algoritmo paralelo implementado em um cluster beowulf. **REPOSITÓRIO DE RELATÓRIOS-Sistemas de Informação**, n. 2, 2014. 15

COULOURIS, G. et al. **Sistemas Distribuídos-: Conceitos e Projeto**. [S.l.]: Bookman Editora, 2013. 12, 13

DEVDBI. **Numbers primes in the range**. 2016. Disponível em: <https://pastebin.com/vSv8c3p0>. 44

FINLAYSON, I. et al. Introducing tetra: an educational parallel programming system. In: IEEE. **2015 IEEE International Parallel and Distributed Processing Symposium Workshop**. [S.l.], 2015. p. 746–751. 31

GHANNOUM, R. G.; RODRIGUES, F. B. Virtualização De Servidores : Vantagens E Desvantagens Virtualization of Servers : Advantages and Disadvantages. v. 6, 2018. 32

GRAMA, A. et al. **Introduction to Parallel Computing, Second Edition**. [S.l.: s.n.], 2003. 856 p. ISSN 09670661. ISBN 0130661023. 29

Josphat Mutai. **How to install and configure LDAP Account Manager on Ubuntu 18.04 / Ubuntu 16.04 LTS - Computingforgeeks**. 2018. Disponível em: <https://computingforgeeks.com/how-to-install-and-configure-ldap-account-manager-on-ubuntu-18-04-ubuntu-16-04-lts/>. 36

KIM, T.-H.; JIANG, K.; RAJPUT, V. Adoption of container-based virtualization in it education. In: **ASEE's 123rd Annual-Conference and Exposition, New Orleans**. [S.l.: s.n.], 2016. 31, 32

MAIA, R. M. S. Ferramenta para criação de clusters virtuais para o ensino de programação paralela e distribuída. **Repositório Institucional UFC: Página inicial**, 2016. Disponível em: <http://www.repositorio.ufc.br/handle/riufc/24785>. 14

MATTOS, G. D. O. **Aspectos de desempenho da computação paralela em Clusters e Grids para processamento de imagens**. Tese (Doutorado) — Universidade Federal de Pernambuco, 2008. 28, 29

MATTSON, T. G. How good is OpenMP. **Scientific Programming**, v. 11, n. 2, p. 81–93, 2003. ISSN 10589244. 26

MORRISON, R. S. Cluster computing architectures, operating systems, parallel processing and programming languages. **GNU General Public Licence**, v. 5, 2003. 12

MPICH. **cpi.c**. 2018. Disponível em: <https://github.com/pmodels/mpich/blob/master/examples/cpi.c>. 39

MPICH. **hellow.c**. 2018. Disponível em: <https://github.com/pmodels/mpich/blob/master/examples/hellow.c>. 37

NAVAUX, P. O.; ROSE, C. A. 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. 21, 22, 23

Open MPI, P. **FAQ: General information about the Open MPI Project**. 2019. Disponível em: <http://www.open-mpi.org/>. 27

PEREIRA, V. Instalação e configuração de um cluster beowulf. 2016. 15

PITANGA, M. **Computação em cluster - Clube do Hardware**. 2003. Disponível em: <http://www.clubedohardware.com.br/printpage/Computacao-em-cluster/153>. 24

PUTTINI, R. S. et al. Laboratório virtual de ensino em computação paralela. In: **Atas da Conferência da Associação Portuguesa de Sistemas de Informação**. [S.l.: s.n.], 2016. v. 3, n. 3. 14

RIBAS, A. **Estimando o valor de pi usando o método de Monte Carlo | Recologia**. 2013. Disponível em: <http://recologia.com.br/2013/07/estimando-o-valor-de-pi-usando-o-metodo-de-monte-carlo/>. 39

SQUYRES, J. M. **The Architecture of Open Source Applications (Volume 2): Open MPI**. 2016. Disponível em: <http://aosabook.org/en/openmpi.html>. 27, 28

TANENBAUM, A. S. Redes de computadores (2003). **Tradução da Quarta Edição, Editora Campus**, 2003. 12

TANENBAUM, A. S. **Organização Estruturada de Computadores, 2007, 5ª Edição**. [S.l.]: Prentice Hall, 2007. 17, 19, 21, 22, 24

TELES, F. **Cluster: o que é e como essa estrutura pode ser benéfica para você**. 2018. Disponível em: <https://blog.deskmanager.com.br/cluster/>. 24

TOKHI, M. O.; HOSSAIN, M. A.; SHAHEED, M. H. **Parallel computing for real-time signal processing and control**. [S.l.]: Springer Science & Business Media, 2012. 17, 18, 19, 20

WEISSTEIN, E. W. Monte Carlo Method. Wolfram Research, Inc., 2019. Disponível em: <http://mathworld.wolfram.com/MonteCarloMethod.html>. 39

WEISSTEIN, E. W. Prime Number. Wolfram Research, Inc., 2019. Disponível em: <http://mathworld.wolfram.com/PrimeNumber.html>. 44

ZARZA, G. et al. An innovative teaching strategy to understand high-performance systems through performance evaluation. **Procedia Computer Science**, Elsevier, v. 9, p. 1733–1742, 2012. 31

Apêndices

APÊNDICE A – INSTALAÇÃO E CONFIGURAÇÃO DO CLUSTER

Nesta etapa descreve-se todos os passos desde o início até o fim das configurações do cluster.

Passo 1 - Instalação do S.O Kubuntu

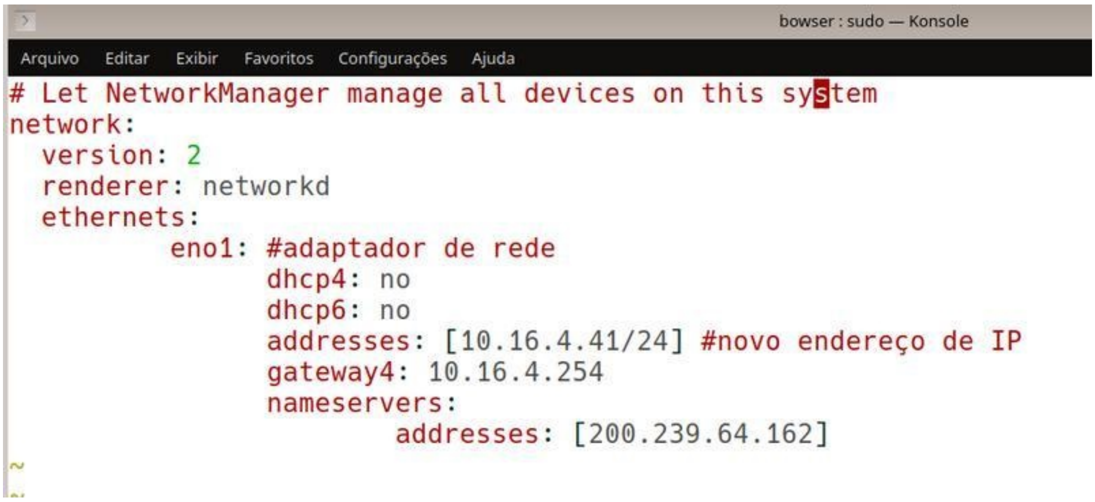
A etapa de configuração começa com a instalação do SO nas 31 máquinas que foram usadas no cluster. Depois de concluída a instalação, os seguintes comandos foram executados via terminal para atualizar os nós:

```
$ sudo apt update
$ sudo apt -y upgrade
```

Passo 2 - Configuração de Rede

Na versão do SO utilizado nesse projeto, o endereço de IP pode ser modificado para estático acessando o seguinte arquivo (sudo vim /etc/netplan/01-network-manager-all.yaml) e modificando-o de acordo com a Figura 25:

Figura 25 – Configuração do IP estático




```
# Let NetworkManager manage all devices on this system
network:
  version: 2
  renderer: networkd
  ethernets:
          eno1: #adaptador de rede
                dhcp4: no
                dhcp6: no
                addresses: [10.16.4.41/24] #novo endereço de IP
                gateway4: 10.16.4.254
                nameservers:
                        addresses: [200.239.64.162]
~
```

Fonte: Elaborada pelo autor

A configuração de rede é aplicada usando o comando abaixo:

```
$ sudo netplan apply
```

Configuração do arquivo /etc/hosts

O arquivo *hosts* foi editado (sudo vim /etc/hosts) como na Figura 26 abaixo:

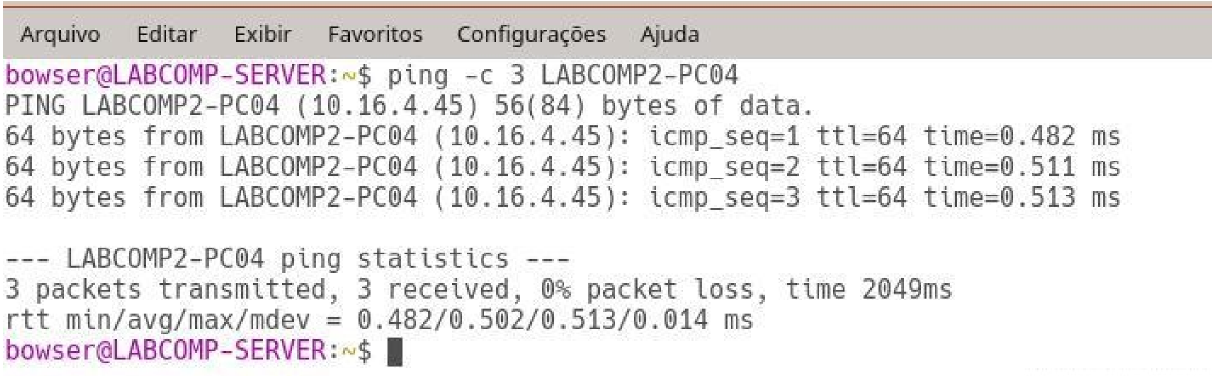
Figura 26 – Configuração do Arquivo *hosts*

```
Arquivo   Editar   Exibir   Favoritos   Configurações   Ajuda
127.0.0.1       localhost

10.16.4.41  LABCOMP-SERVER.ora.ufpa.br  LABCOMP-SERVER

10.16.4.42  LABCOMP2-PC01
10.16.4.43  LABCOMP2-PC02
10.16.4.44  LABCOMP2-PC03
10.16.4.45  LABCOMP2-PC04
10.16.4.46  LABCOMP2-PC05
10.16.4.47  LABCOMP2-PC06
10.16.4.48  LABCOMP2-PC07
10.16.4.49  LABCOMP2-PC08
10.16.4.50  LABCOMP2-PC09
10.16.4.51  LABCOMP2-PC10
10.16.4.52  LABCOMP2-PC11
10.16.4.53  LABCOMP2-PC12
10.16.4.54  LABCOMP2-PC13
10.16.4.55  LABCOMP2-PC14
10.16.4.56  LABCOMP2-PC15
10.16.4.57  LABCOMP2-PC16
10.16.4.58  LABCOMP2-PC17
10.16.4.59  LABCOMP2-PC18
10.16.4.60  LABCOMP2-PC19
10.16.4.61  LABCOMP2-PC20
10.16.4.62  LABCOMP2-PC21
10.16.4.63  LABCOMP2-PC22
10.16.4.64  LABCOMP2-PC23
10.16.4.65  LABCOMP2-PC24
10.16.4.66  LABCOMP2-PC25
10.16.4.67  LABCOMP2-PC26
10.16.4.68  LABCOMP2-PC27
10.16.4.69  LABCOMP2-PC28
10.16.4.70  LABCOMP2-PC29
10.16.4.71  LABCOMP2-PC30
```

Fonte: Elaborada pelo autor

Depois de salvar o arquivo, pode-se usar os nomes dos nós para se conectar aos nós, como demonstrado na figura 27:

Figura 27 – Saída do comando ping



Fonte: Elaborada pelo autor

Esta etapa foi realizada em todas as nós que compõem o cluster.

Passo 3 - Sistema de autenticação de usuários (LDAP)

No nó mestre, os pacotes LDAP podem ser instalados executando o comando abaixo:

```
$ sudo apt -y install slapd ldap-utils
```

O pacote *slapd* contém o servidor LDAP e o pacote *ldap-utils* inclui ferramentas de linha de comando para interação com os servidores LDAP.

Durante a instalação, será solicitado a inserção de uma senha de administrador LDAP, deve-se digitar a senha e pressionar <OK>

É possível confirmar se a instalação foi bem-sucedida usando o comando “*slapcat*” para gerar o conteúdo do banco de dados SLAPD.

```
$ sudo slapcat
```

A Figura 28 mostra que que o processo de instalação ocorreu sem falhas.

Figura 28 – saida do comando slapcat

```
bowser@LABCOMP-SERVER:~$ sudo slapcat
dn: dc=ora,dc=ufpa,dc=br
objectClass: top
objectClass: dcObject
objectClass: organization
o: ora.ufpa.br
dc: ora
structuralObjectClass: organization
entryUUID: 6558d914-e784-1038-800c-d79c1bcdbbb8
creatorsName: cn=admin,dc=ora,dc=ufpa,dc=br
createTimestamp: 20190330221039Z
entryCSN: 20190330221039.281728Z#000000#000#000000
modifiersName: cn=admin,dc=ora,dc=ufpa,dc=br
modifyTimestamp: 20190330221039Z

dn: cn=admin,dc=ora,dc=ufpa,dc=br
objectClass: simpleSecurityObject
objectClass: organizationalRole
cn: admin
description: LDAP administrator
userPassword:: e1NTSEF9cmZvT3pwQmNodnZSTEdZV3pkUEQ3Y3d3MVJTT3ZLWmI=
structuralObjectClass: organizationalRole
entryUUID: 65596744-e784-1038-800d-d79c1bcdbbb8
creatorsName: cn=admin,dc=ora,dc=ufpa,dc=br
createTimestamp: 20190330221039Z
entryCSN: 20190330221039.285398Z#000000#000#000000
modifiersName: cn=admin,dc=ora,dc=ufpa,dc=br
modifyTimestamp: 20190330221039Z

bowser@LABCOMP-SERVER:~$
```

Fonte: Elaborada pelo autor

A próxima etapa é adicionar um DN (*Distinguished Name*) base para usuários e grupos. Cria-se um arquivo chamado “basedn.ldif” com o mesmo conteúdo mostrado na Figura 29:

Figura 29 – Arquivo basedn

```
bowser : sudo — Konsole
Arquivo  Editar  Exibir  Favoritos  Configurações  Ajuda
dn: ou=people,dc=ora,dc=ufpa,dc=br
objectClass: organizationalUnit
ou: people

dn: ou=groups,dc=ora,dc=ufpa,dc=br
objectClass: organizationalUnit
ou: groups
```

adiciona-se o arquivo executando o comando:

```
$ ldapadd -x -D cn=admin,dc=ora,dc=ufpa,dc=br -W -f basedn.ldif
```

Gerenciador de Contas LDAP

Antes de instalar o Gerenciador é preciso instalar suas dependências:

```
$ sudo apt -y install apache2 php php-cgi libapache2-mod-php
  php-mbstring php-common php-pear
$ sudo a2enconf php7.2-cgi
$ sudo systemctl reload apache2
```

O Gerenciador de Contas LDAP é instalado com o seguinte comando:

```
$ sudo apt -y install ldap-account-manager
```

Após o término da instalação, restringi-se o acesso ao painel da Web, permitindo apenas sub-redes locais confiáveis editando o arquivo a seguir.

```
$ sudo vim /etc/apache2/conf-enabled/ldap-account-manager.conf
```

Edita-se a linha 12 como mostrado na Figura 30, comentando "*Require all granted*" e adiciona-se as sub-redes permitidas para acessar a interface de administração do Gerenciador de Contas LDAP.

Figura 30 – Restrição de acesso ao painel Web

Fonte: Elaborada pelo autor

Reinicia-se o servidor apache depois de fazer a alteração:

```
$ sudo systemctl restart apache2
```

Configuração do gerenciador de contas LDAP

Pode-se agora acessar a interface web do Gerenciador de contas LDAP a partir de uma máquina com a rede confiável:

```
http://(nome do servidor ou IP)/lam
```

O formulário de login do Gerenciador de contas LDAP será exibido. deve-se definir nosso perfil do servidor LDAP clicando no <*LAM configuration*> no canto superior direito.

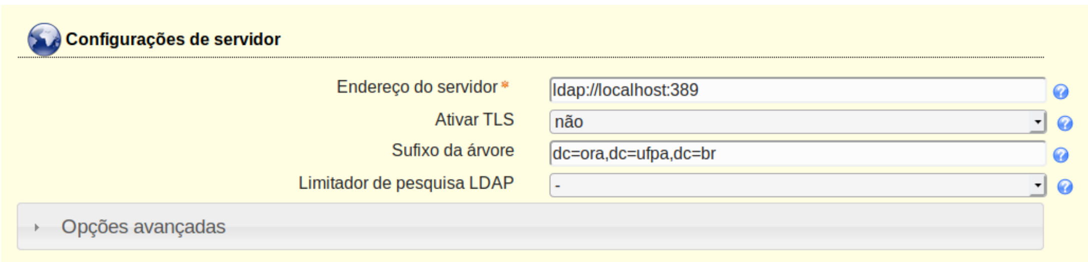
Então clica-se em <*Edit server profiles*>

Será então pedido a senha do perfil LAM, a senha padrão é "lam".

A primeira coisa a se mudar é a senha do perfil, isso é no final da página em Configurações gerais.

Em seguida, defini-se o endereço do servidor e o sufixo da árvore usando os componentes de domínio como definido no nome do *host* do servidor, como mostrado na Figura 31.

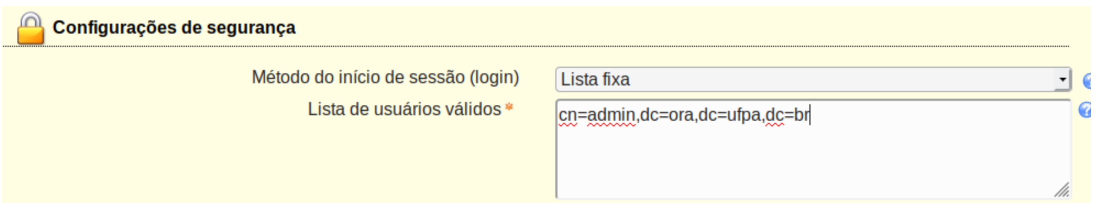
Figura 31 – Configurações do servidor LDAP



Fonte: Elaborada pelo autor

Define-se também o login do Painel especificando a conta do usuário administrador e os componentes do domínio na seção "Configurações de segurança" como mostrado na Figura 32

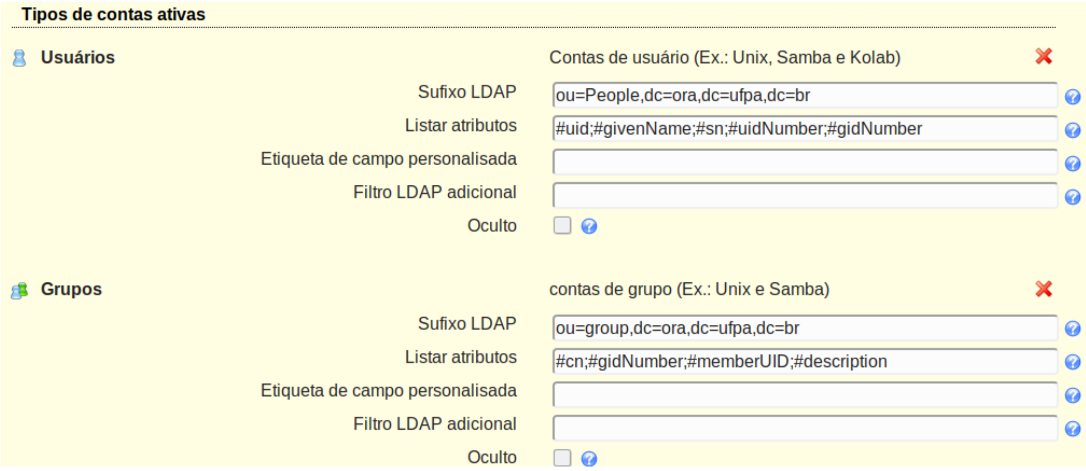
Figura 32 – Configurações de segurança do servidor LDAP



Fonte: Elaborada pelo autor

Em seguida, muda-se para a página "Tipos de conta" e configura-se os campo Sufixo LDAP como mostrado na Figura 33

Figura 33 – Tipos de contas ativas



Fonte: Elaborada pelo autor

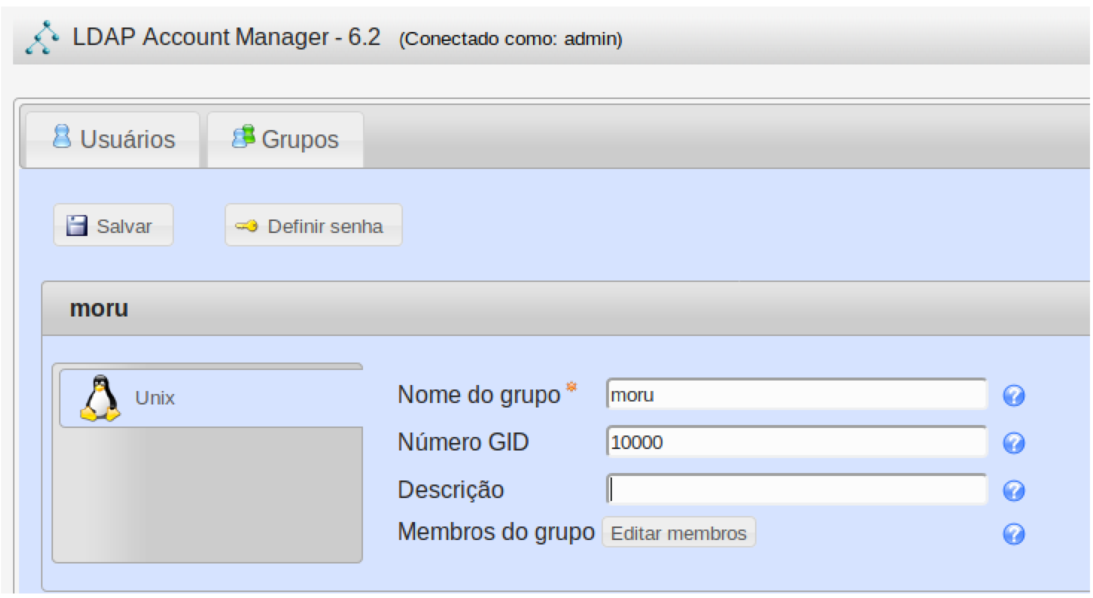
Após o término das configurações, salva-se as modificações feitas clicando no botão [Salvar] na parte inferior da página.

Adicionando contas de usuários e grupos com o Gerenciador de contas LDAP

A primeira coisa a se fazer é realizar login com a conta "admin" no painel do LAM para começar a gerenciar contas e grupos de usuários. Antes de adicionar a conta de usuário, precisa ser adicionado um grupo de usuários. Clica-se em <Grupos> e depois <Novo Grupo>

O nome do grupo capaz de realizar trabalhos no cluster foi dado "moru" como mostrado na Figura 34, o ID de grupo e descrição o preenchimento são opcionais.

Figura 34 – Criação de grupos no servidor LDAP

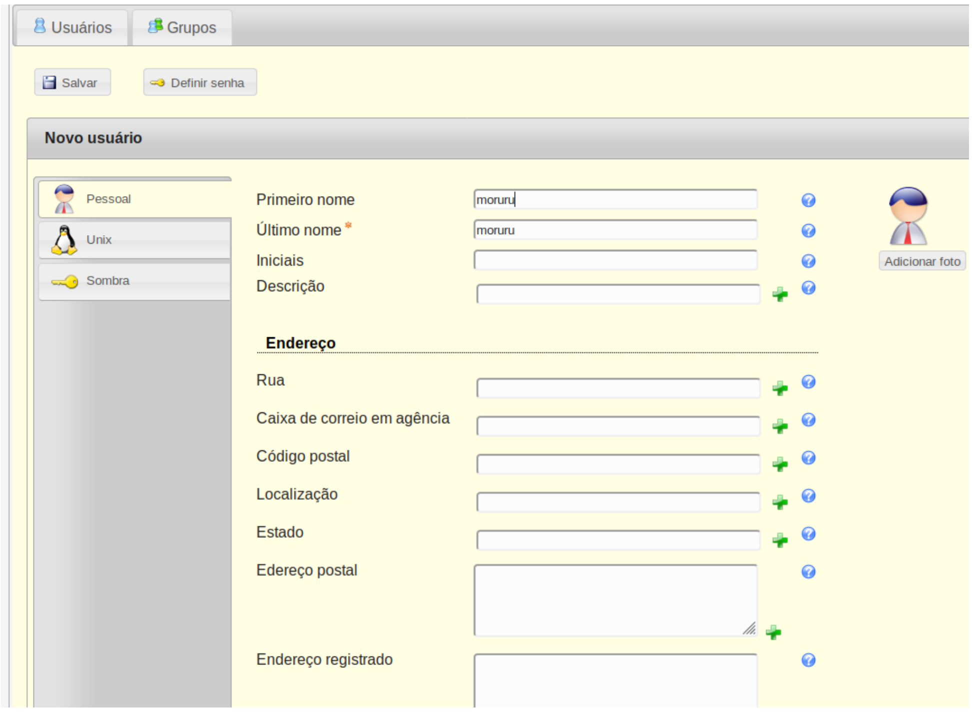


Fonte: Elaborada pelo autor

O mesmo é feito para adicionar outros grupos.

Depois de ter os grupos de contas de usuários adicionados, pode-se adicionar uma nova conta de usuário clicando em <Usuários> e depois <Novo usuário>. Tem-se três seções para gerenciamento de usuários como pode ser visto na Figura 35:

Figura 35 – Criação de usuários no servidor LDAP



Fonte: Elaborada pelo autor

Pessoal - possui informações pessoais do usuário, como nome, sobrenome, e-mail, telefone, departamento, endereço, etc.

Unix: aqui define-se o nome de usuário, o número UID, o grupo primário do usuário e os o diretório principal, etc.

shadow: aqui defini-se vencimento da conta de usuário, expiração da senha, etc.

Configuração do nós escravos para o acesso do servidor LDAP

Nos nós escravos, os pacotes LDAP podem ser instalados executando o comando abaixo:

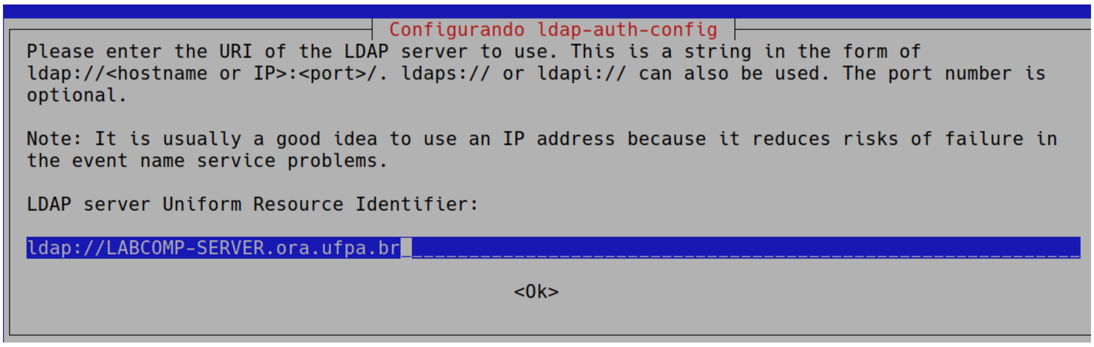
```
$ sudo apt install libpam-ldap nscd
```

libpam-ldap: este pacote fornece uma interface entre um servidor LDAP e o sistema de autenticação do usuário PAM.

nscd: é um daemon que lida com consultas *passwd*, de grupo e de *host* para executar programas e armazena em cache os resultados para a próxima consulta.

Durante a instalação será solicitado o preenchimento do endereço do servidor LDAP como mostrado na Figura 36.

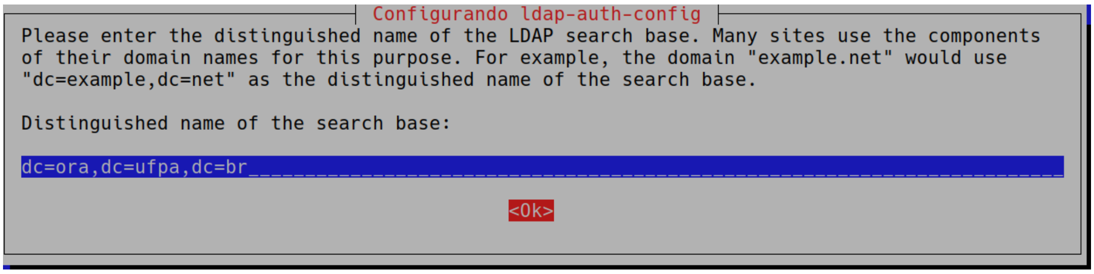
Figura 36 – Instalação do cliente LDAP - parte 1



Fonte: Elaborada pelo autor

depois será solicitado o preenchimento dos componentes de domínio definido no processo de configuração do servidor LDAP, como mostrado na Figura 37.

Figura 37 – Instalação do cliente LDAP - parte 2



Fonte: Elaborada pelo autor

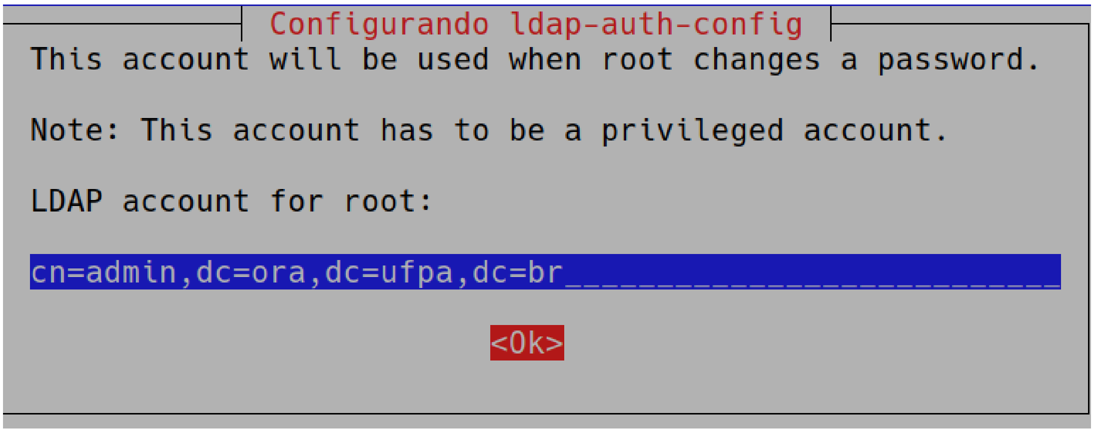
A versão 3 do LDAP foi escolhida.

foi escolhido [sim] para *make local root Databse admin*.

foi escolhido [não] para *Does the LDAP database require login*.

A Figura 38 mostra que a conta LDAP para *root* foi definida como na configuração do servidor LDAP.

Figura 38 – Instalação do cliente LDAP - parte 3



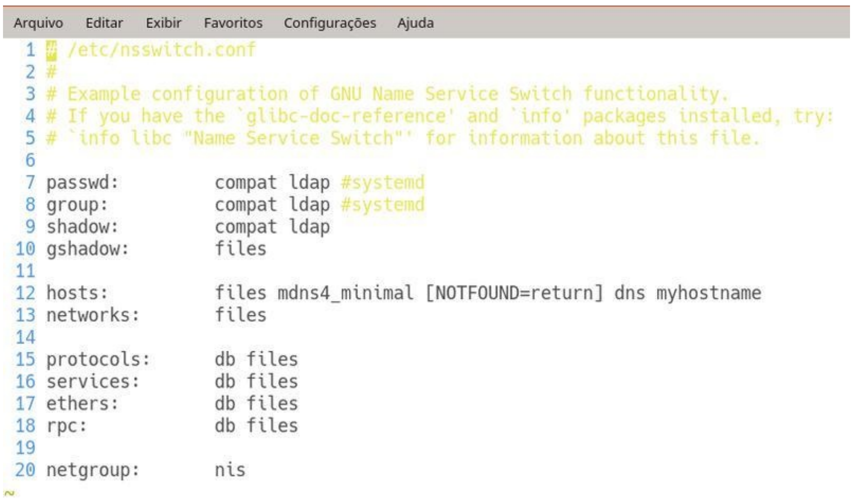
Fonte: Elaborada pelo autor

o ultimo passo da instalação é digitar a senha do servidor LDAP.

Após a instalação, o arquivo /etc/nsswitch.conf é modificado nas linhas 7, 8 e 9 como mostrado na Figura 39 para ser possível autenticar usando um usuário criado no servidor LDAP:

```
$ sudo vim /etc/nsswitch.conf
```

Figura 39 – Modificações do arquivo /etc/nsswitch.conf

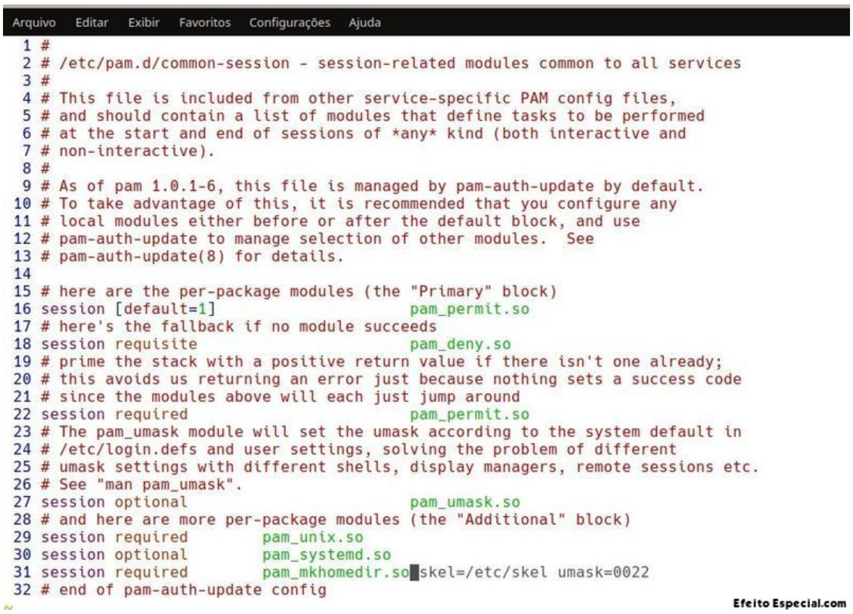



```
1 # /etc/nsswitch.conf
2 #
3 # Example configuration of GNU Name Service Switch functionality.
4 # If you have the `glibc-doc-reference' and `info' packages installed, try:
5 # `info libc "Name Service Switch"' for information about this file.
6
7 passwd:         compat ldap #systemd
8 group:          compat ldap #systemd
9 shadow:         compat ldap
10 gshadow:        files
11
12 hosts:          files mdns4_minimal [NOTFOUND=return] dns myhostname
13 networks:       files
14
15 protocols:      db files
16 services:       db files
17 ethers:         db files
18 rpc:            db files
19
20 netgroup:       nis
```

Fonte: Elaborada pelo autor

Em seguida, deve editar o arquivo /etc/pam.d/common-session adicionando a linha 31 como mostrado na Figura 40, O comando adicionado criará um diretório pessoal de usuários se ele não existir quando a sessão começar.

Figura 40 – Modificações do arquivo /etc/pam.d/common-session



Arquivo Editar Exibir Favoritos Configurações Ajuda

```
#
# /etc/pam.d/common-session - session-related modules common to all services
#
# This file is included from other service-specific PAM config files,
# and should contain a list of modules that define tasks to be performed
# at the start and end of sessions of *any* kind (both interactive and
# non-interactive).
#
# As of pam 1.0.1-6, this file is managed by pam-auth-update by default.
# To take advantage of this, it is recommended that you configure any
# local modules either before or after the default block, and use
# pam-auth-update to manage selection of other modules.  See
# pam-auth-update(8) for details.

# here are the per-package modules (the "Primary" block)
session [default=1]                     pam_permit.so
# here's the fallback if no module succeeds
session requisite                       pam_deny.so
# prime the stack with a positive return value if there isn't one already;
# this avoids us returning an error just because nothing sets a success code
# since the modules above will each just jump around
session required                        pam_permit.so
# The pam_umask module will set the umask according to the system default in
# /etc/login.defs and user settings, solving the problem of different
# umask settings with different shells, display managers, remote sessions etc.
# See "man pam_umask".
session optional                        pam_umask.so
# and here are more per-package modules (the "Additional" block)
session required        pam_unix.so
session optional        pam_systemd.so
session required        pam_mkhomedir.so skel=/etc/skel umask=0022
# end of pam-auth-update config
```


Fonte: Elaborada pelo autor

Por ultimo deve-se reiniciar o serviço ncsd com o comando abaixo:

```
$ sudo service nscd restart
```

o comando abaixo é usado para testar se está funcionando:

```
$ getent passwd
```

A etapa de configuração dos nós para acesso do servidor LDAP. deverá ser realizada também no nó mestre.

Passo 4 - Instalação e Configuração do NFS

No nó mestre foi instalado o *nfs-server*:

```
$ sudo apt install nfs-kernel-server
```

O NFS é usado para compartilhar o diretório inicial dos usuários que utilização o *cluster*, isto é o */cluster* neste projeto, com os nós escravos. Como esse diretório ainda não existe, ele foi criado e modificado usando os comandos abaixo

```
$ sudo mkdir /cluster && sudo chmod 777 /cluster
```

Agora deve-se compartilhar o diretório */cluster* do nó mestre com todos os outros nós. Para isso, o arquivo /etc/exports no nó mestre precisa ser editado. É Adicionado a seguinte linha a este arquivo:

```
/cluster *(rw, sync, no_subtree_check)
```

/cluster: diretório dos usuários do cluster

rw: permissão de leitura e gravação no diretório

sync: garante que os dados do arquivo em cache na memória é automaticamente gravado em disco após a conclusão da transferência dos mesmos.

no_subtree_check: todos os diretórios dentro do diretório compartilhado também são compartilhados

Por fim, termina-se a configuração do NFS no mestre reiniciando-o:

```
$ sudo service nfs-kernel-server restart
```

o comando abaixo exporta os diretórios listados em /etc/exports:

```
$ sudo exportfs -a
```

Para adicionar uma regra ao UFW, execute:

```
$ sudo ufw allow from 10.16.4.0/24
```

Em seguida nos nós escravos foi instalado o *nfs-client*:

```
$ sudo apt install nfs-common
```

Para montar o diretório */cluster* nos nós escravos, o arquivo /etc/fstab foi editado com o comando:

```
$ sudo vim /etc/fstab
```

e foi adicionado no final do arquivo a seguinte linha:

```
LABCOMP-SERVER:/cluster /cluster nfs
```

LABCOMP-SERVER: nome do nó onde encontra o diretório */cluster*

Em seguida foi executado o comando *mount* que fará a releitura do arquivo fstab, montando assim o diretório "/cluster" imediatamente:

```
$ mount -a
```

Verifica-se o diretório foi realmente montado com o comando:

```
$ df -h
```

Passo 5 - Instalação e configuração do SSH

Instala-se o servidor SSH em todos os nós com o comando:

```
$ sudo apt install openssh-server
```

Agora precisa-se gerar uma chave SSH para todos os nós. Por padrão, a chave SSH é criada no diretório pessoal do usuário. Como o diretório inicial do usuário que utilizará o cluster é o mesmo diretório para todos os outros nó. Então, se for gerado uma chave SSH para um usuário em um dos nós, todos os nós terão automaticamente uma chave SSH.

As chaves deverão ser geradas por cada usuário que utilizará o cluster com o comando:

```
$ ssh-keygen
```

Quando perguntado por uma senha, foi deixado-a vazia.

```
$[enter]
$[enter]
$[enter]
```

Quando terminar, todos os nós devem ter uma chave SSH (exatamente a mesma chave). O nó mestre precisa ser capaz de efetuar login automaticamente nos nós escravos. Para habilitar o login sem senha, executa-se os seguintes comandos no nó mestre:

```
$ cd ~/.ssh
$ ssh-copy-id -i id_rsa.pub localhost
```

A chave SSH pública do mestre foi copiada para ~/.ssh/authorized_keys. E como todo o diretório do usuário é compartilhado com todos os nós via NFS, todos os outros nós agora devem ter a chave pública SSH do mestre na lista de *hosts* conhecidos. Isso significa que agora pode-se fazer login nos nós escravos a partir do nó mestre sem precisar digitar uma senha.

Passo 6 - Instalação e configuração do *Open MPI*

Antes de começar instalar o *open MPI*, alguns pacotes precisam ser instalados.

```
$ sudo apt-get update
$ sudo apt-get install build-essential manpages-dev
```

build-essential: é um pacote que reúne diversas aplicações para compilar/instalar aplicações com base no seu código fonte. Este pacote inclui, por exemplo, o `make`, o g++, gcc, etc.

Para instalar o *Open MPI* os seguintes passos foram seguidos:

1. O arquivo baixado foi descompactado com:

```
$ tar -vzxf openmpi-*
```

2. dentro do diretório descompactado, configura-se o arquivo de instalação com o comando.

```
$  ./configure --enable-orterun-prefix-by-default
```

4. Para a instalação foi feito o uso da ferramenta *make*.

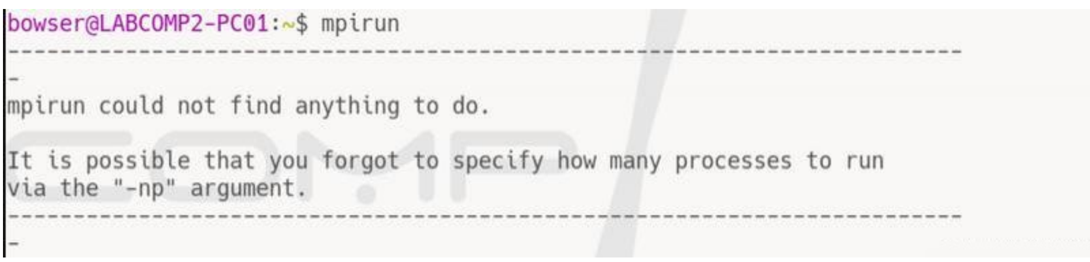
```
$ make
$ sudo make install
```

5. Para configurar a varável de ambiente do *open MPI*, é preciso editar o arquivo /etc/bash.bashrc e adicionar no final do arquivo as linhas abaixo:

```
export PATH=/usr/local/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

6. Após reiniciar o terminal, ao executar o comando mpirun, deve ser mostrado a mensagem *"mpicc could not find anything to do"*, como mostrado na Figura 41 demonstrando que não houve falhas na instalação.

Figura 41 – Saída do comando *"mpirun"*



Fonte: Elaborada pelo autor

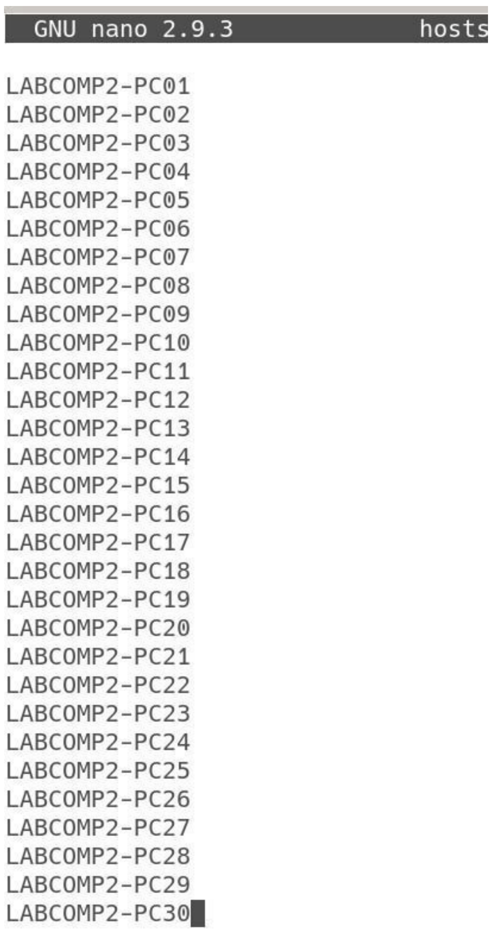
Passo 7 - Teste de Funcionamento

O primeiro passo para testar se o *cluster* está executando corretamente é criar o arquivo *hosts* contendo o nome de todas as máquinas que serão utilizadas na execução. Vale lembrar que este arquivo *hosts* não tem relação com o arquivo */etc/hosts* que foi citado no passo de configuração de rede e pode receber qualquer outro nome.

```
$ vim ~/hosts
```

A Figura 42 mostra o arquivo *hosts* depois de adicionado todos os nomes dos nós que serão utilizados.

Figura 42 – Arquivo *hosts*



Fonte: Elaborada pelo autor

O comando a seguir foi executado para compilar e executar o algoritmo *hellow.c* (Anexo C):

```
$ mpicc hellow.c -o hello && mpirun --hostfile ~/hosts hello
```

APÊNDICE B – SCRIPT DE AUTOMAÇÃO DOS PASSOS 4, 5 E 6 DA INSTALAÇÃO E CONFIGURAÇÃO DO *CLUSTER* ORÃ

```
#!/bin/bash
master="LABCOMP-SERVER"
rede="10.16.4.0/24"
cam=$(pwd)

sudo apt update
clear

ms=3
configurarArquivoHosts(){
        echo "---------------Passo 1 -----------------"
        echo "CONFIGURANDO O ARQUIVO /etc/hosts         "
        #Para que está etapa ocorra bem, crie um arquivo host com a
        #lista de IPs e nome de hosts de todos os computadores que
        #farão parte do cluster (dúvidas, consultar Apêndice A)
        sudo cp -r hosts /etc/hosts
        sleep 2
}

clear
instalarNFS(){
        echo "---------------Passo 2 -------------------"
        echo "----------- Instalando o NFS--------------"
        sleep 2
        read -p "A maquina é slave(1) ou master(0)? " ms
        echo ""

        #Configuração do(s) slave(s)
        if [ $ms -eq 1 ]; then

        sudo apt install nfs-common -y
        # criando o ponto de montagem
        sudo mkdir /cluster
```

```
        sudo mount $master:/cluster /cluster
        echo "$master:/cluster /cluster nfs" | sudo tee -a /etc/fstab
        # Agora vamos verificar se as pastas foram montadas corretamente
        sudo mount -a
        #echo ""
        #df -h


        #Configuração do Master

    elif [ $ms -eq 0 ]; then

        sudo apt-get install nfs-server -y
        sudo mkdir /cluster
        #Agora vamos editar o arquivo /etc/exports
        echo "/cluster *(rw,sync,no_subtree_check)" | sudo tee -a /etc/exports
        sudo service nfs-kernel-server restart
        sudo exportfs -a
        sudo ufw allow from $rede
        fi
        sleep 5
}

instalarSSH(){
        echo "---------------Passo 3 ------------------"
        echo "-----------Instalando o SSH--------------"
        sleep 2
        if [ $ms -eq 3 ]; then
                read -p "A maquina é slave(1) ou master(0)? " ms
                echo ""
        fi

        #Configuração do(s) slave(s)
        if [ $ms -eq 1 ]; then
                sudo apt install openssh-server -y

        #Configuração do Master
        elif [ $ms -eq 0 ]; then
                sudo apt-get install openssh-server -y
```

```
        #ssh-keygen
                #cd ~/.ssh && ssh-copy-id -i id_rsa.pub localhost
        fi

        sleep 5
}

instacaoOpenMpi(){
        echo ""
        echo "---------------Passo 4-------------------"
        echo "------Instalando o OPEN MPI--------------"
        sleep 2
        sudo apt install build-essential manpages-dev -y
        cp $cam/openmpi-4.0.0.tar.gz ~/Documentos
        cd ~/Documentos
        tar -vzxf openmpi-4.0.0.tar.gz
        cd openmpi-4.0.0
        ./configure --enable-orterun-prefix-by-default
        sudo make all install

        echo 'export PATH=/usr/local/bin:$PATH' | sudo tee -a
        /etc/bash.bashrc

        echo 'export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH'
        | sudo tee -a /etc/bash.bashrc

        sudo rm -Rf ~/Documentos/openmpi-4.0.0
                    ~/Documentos/openmpi-4.0.0.tar.gz

        sleep 3

}


while true; do

        echo ""
        echo " CONFIGURAR O ARQUIVO /etc/hosts------------(1)"
        echo " INSTALAR E CONFIGURAR O NFS----------------(2)"
```

```
        echo " INSTALAR E CONFIGURAR o SSH----------------(3)"
        echo " INSTALAR E CONFIGURAR  O OPEN MPI----------(4)"
        echo " TODAS AS OPÇÕES----------------------------(5)"
        echo " SAIR---------------------------------------(0)"
        read -p "escolha uma opção de 0 a 5:  " opcao
        clear



        if [ $opcao -eq 1 ]; then
                configurarArquivoHosts
        elif [ $opcao -eq 2 ]; then
                instalarNFS
        elif [ $opcao -eq 3 ]; then
                instalarSSH
        elif [ $opcao -eq 4 ]; then
                instacaoOpenMpi
        elif [ $opcao -eq 5 ]; then
                configurarArquivoHosts
                instalarNFS
                instalarSSH
                instacaoOpenMpi
        elif [ $opcao -eq 0 ]; then
                break
        else
                echo "ERRO! TENTE NOVAMENTE..!!"
        fi

done
```

Anexos

ANEXO A – ALGORITMO CPI_MONTECARLO.C

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

int main(int argc, char *argv[]){
    int myid, numprocs;
    double mypi,x, y, pi = 0.0;
    double i, cont = 0, N = 700000000000.0;
    double startwtime = 0.0, endwtime;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

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

    fflush(stdout);

    if (myid == 0)
        startwtime = MPI_Wtime();

    MPI_Bcast(&N, 1, MPI_INT, 0, MPI_COMM_WORLD);
    srand((unsigned)time(NULL) * myid);
    for (i = myid + 1; i <= N; i += numprocs) {
        x = ((double)(rand())/RAND_MAX);
        y = ((double)(rand())/RAND_MAX);
        if((pow(x,2) + pow(y,2)) <=1)
            cont+=1;
    }
    mypi = 4.0 * (double) cont / N;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);

    if (myid == 0) {
        endwtime = MPI_Wtime();
```

```
        printf("\n\nNúmero de processos %d\n", numprocs);
        printf("PI é aproximandamente %.10f \n", pi);
        printf("Tempo usado = %f\n", (endwtime - startwtime)/60);
        fflush(stdout);
    }
    MPI_Finalize();
    return 0;
}
```

ANEXO B – ALGORITMO NUMEROS_PRIMOS.C

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <mpi.h>

//método para verificar se um número é primo
int is_prime(int num){
        int max_divisor, d;
        if(num<1)
                return 0;
        else if(num==1)
                return 1;
        else if(num>3){
                if(num%2==0)
                        return 0;
                max_divisor = sqrt(num);
                for(d = 3; d<=max_divisor; d+=2){
                        if(num%d==0) return 0;
                }
        }
        return 1;
}

int main(int argc, char** argv){
    double startwtime = 0.0, endwtime;
    int quant = 1000000000,
        i,
        localSum = 0,
        globalSum = 0, tmp;

    // comm
    int myrank, nprocs;

    // split
    int split,
        begin,
        end;
```

```
    MPI_Init(&argc, &argv);
    MPI_Comm_rank(MPI_COMM_WORLD,&myrank);
    MPI_Comm_size(MPI_COMM_WORLD,&nprocs);

    if (myrank == 0)
                startwtime = MPI_Wtime();
    // split tasks
    split = ceil((float)(quant)/nprocs);
    begin = split * myrank;
    end = split * (myrank+1);

    // if happen overflow
    if (end > quant)
        end = quant;

    for (i = begin; i < end; ++i){
        if (is_prime(i))
            localSum++;
    }



// receive all sums of others process
    if (myrank == 0){
        globalSum = localSum;
        for (i = 1; i < nprocs; ++i)
        {
            MPI_Recv(&tmp,1,MPI_INT,i,0,MPI_COMM_WORLD,MPI_STATUS_IGNORE);
            globalSum += tmp;
        }
        endwtime = MPI_Wtime();

        printf("\n\nNúmeros de processos %d\n", nprocs);
        printf("%i números primos foram encontrados entre %i e %i\n",
                                        globalSum, 0, quant);
        printf("Tempo usado = %f\n", endwtime - startwtime);
        fflush(stdout);
    }
```

```
    // send the local sum to master process
    else{
        MPI_Send(&localSum,1,MPI_INT,0,0,MPI_COMM_WORLD);
    }

    MPI_Finalize();
    return 0;
}
```

ANEXO C – ALGORITMO HELLOW.C

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

int main(int argc, char *argv[])
{
    int myid;
    int numprocs;
    int namelen;
    char processor_name[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(0, 0);
    MPI_Comm_rank(MPI_COMM_WORLD, &myid);
    MPI_Comm_size(MPI_COMM_WORLD, &numprocs);
    MPI_Get_processor_name(processor_name, &namelen);


    fprintf(stdout, "Olá mundo do Processo %d de %d de %s\n",
            myid, numprocs, processor_name);
    MPI_Finalize();
    return 0;
}
```