PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO GRANDE DO SUL
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PROPOSTA DE UMA LINGUAGEM ESPECÍFICA DE DOMÍNIO DE PROGRAMAÇÃO PARALELA ORIENTADA A PADRÕES PARALELOS: UM ESTUDO DE CASO BASEADO NO PADRÃO MESTRE/ESCRAVO PARA ARQUITETURAS MULTI-CORE

DALVAN JAIR GRIEBLER

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Ciência da Computação na Pontifícia Universidade Católica do Rio Grande do Sul.

Orientador: Prof. Dr. Luiz Gustavo Fernandes

Porto Alegre
2012

G848p Griebler, Dalvan Jair
Proposta de uma linguagem específica de domínio de programação paralela orientada a padrões paralelos : um estudo de caso baseado no padrão mestre/escravo para arquiteturas Multi-Core / Dalvan Jair Griebler. – Porto Alegre, 2012.
168 f.

Diss. (Mestrado) – Fac. de Informática, PUCRS.
Orientador: Prof. Dr. Luiz Gustavo Fernandes.

1. Informática. 2. Arquitetura de Computador. 3. Arquitetura Paralela. I. Fernandes, Luiz Gustavo. II. Título.

CDD 004.22

Ficha Catalográfica elaborada pelo
Setor de Tratamento da Informação da BC-PUCRS

Pontifícia Universidade Católica do Rio Grande do Sul
FACULDADE DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

TERMO DE APRESENTAÇÃO DE DISSERTAÇÃO DE MESTRADO

Dissertação intitulada ***"Proposta de uma Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos: Um Estudo de Caso Baseado no Padrão Mestre/Escravo para Arquiteturas Multi-Core"***, apresentada por Dalvan Jair Griebler como parte dos requisitos para obtenção do grau de Mestre em Ciência da Computação, Processamento Paralelo e Distribuído, aprovada em 19/03/2012 pela Comissão Examinadora:

Prof. Dr. Luiz Gustavo Leão Fernandes - Orientador — PPGCC/PUCRS

Prof. Dr. César Augusto Fonticielha De Rose - — PPGCC/PUCRS

Prof. Dr. Philippe Olivier Alexandre Navaux - — UFRGS

Prof. Dr. Marcelo Blois Ribeiro — GE- Brasil

Homologada em..22../..06../..2012, conforme Ata No. ...013... pela Comissão Coordenadora.

Prof. Dr. Paulo Henrique Lemelle Fernandes
Coordenador.

PUCRS
Campus Central
Av. Ipiranga, 6681 - P32- sala 507 - CEP: 90619-900
Fone: (51) 3320-3611 - Fax (51) 3320-3621
E-mail: ppgcc@pucrs.br
www.pucrs.br/facin/pos

Dedico este trabalho a minha família,
pelo apoio e incentivo em todos
os dias da minha vida.

AGRADECIMENTOS

Em nossa vida buscamos atingir metas e realizar os nossos sonhos. A minha fé foi uma aliada para atingir mais uma meta na minha vida, sou muito grato a Deus por ter me proporcionado tudo isso. Pelos amigos que fiz e pelas pessoas maravilhosas que conheci durante este período. Por tudo, pela vida, saúde e conquistas.

À família, em especial meus pais Sérgio e Maria, que sempre estiveram comigo, me apoiando e incentivando. Obrigado por acreditar no meu potencial e investir em mim. Minha vó Cecilha, a mais querida, que sempre me apoiou em minhas decisões e aos seus conselhos sábios da experiência de vida. As irmãs Ana e Andréia, que de uma forma ou de outra me apoiaram. E, ao final desta caminhada, apareceu a Litiéle, uma das pessoas que mais amo neste mundo, obrigado por estar do meu lado e me apoiar na conclusão deste trabalho, te amo.

Ao orientador Luiz Gustavo Fernandes, pela oportunidade, paciência, conselhos, disponibilidade e pelas observações enriquecedoras. Além disso, por todo conhecimento repassado nas orientações eu agradeço.

Aos meus colegas de trabalho, pela paciência e troca de ideais. Meus agradecimentos ao Mateus e a Andriele pelas correções textuais. Ao Eduardo e a Victoria pelo suporte na execução dos experimentos e testes de desempenho.

Quero também agradecer a todos os meus amigos que sempre estiveram presente na minha vida e a todos professores que tive até o momento, pois todos contribuíram para o meu crescimento profissional e como pessoa. Muito obrigado!

PROPOSTA DE UMA LINGUAGEM ESPECÍFICA DE DOMÍNIO DE PROGRAMAÇÃO PARALELA ORIENTADA A PADRÕES PARALELOS: UM ESTUDO DE CASO BASEADO NO PADRÃO MESTRE/ESCRAVO PARA ARQUITETURAS MULTI-CORE

RESUMO

Este trabalho propôs uma Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP). O principal objetivo é reduzir o esforço e induzir o programador a desenvolver algoritmos paralelos guiando-se através de padrões que são implementados pela interface da linguagem, evitando que ocorram grandes perdas de desempenho nas aplicações. Anteriormente estudados, os padrões são soluções especializadas e utilizadas para resolver um problema frequente. Assim, padrões paralelos são descritos em um alto nível de abstração para organizar os algoritmos na exploração do paralelismo, podendo ser facilmente interpretados por programadores inexperientes e engenheiros de *software*. Como ponto de partida, este trabalho realizou um estudo de caso baseando-se no padrão Mestre/Escravo, focando na paralelização de algoritmos para arquiteturas *multi-core*. Através de experimentos para medição de esforço e desempenho, a implementação de estudo de caso foi avaliada obtendo bons resultados. Os resultados obtidos mostram que houve uma redução no esforço de programação paralela em relação a utilização da biblioteca Pthreads. Já com relação ao desempenho final das aplicações paralelizadas, foi possível comprovar que a paralelização com LED-PPOPP não acarreta perdas significativas com relação a paralelizações com OpenMP na quase totalidade das aplicações testadas.

Palavras-chave: Programação Paralela Orientada a Padrões Paralelos, Padrões Paralelos, Programação Paralela, Linguagem Específica de Domínio.

PROPOSAL OF AN DOMAIN-SPECIFIC LANGUAGE FOR PARALLEL PATTERNS ORIENTED PARALLEL PROGRAMMING: A CASE STUDY BASED ON MASTER/SLAVE PATTERN FOR MULTI-CORE ARCHITECTURES

ABSTRACT

This work proposes a Domain-Specific Language for Parallel Patterns Oriented Parallel Programming (LED-PPOPP). Its main purpose is to provide a way to decrease the amount of effort necessary to develop parallel programs, offering a way to guide developers through patterns which are implemented by the language interface. The idea is to exploit this approach avoiding large performance losses in the applications. Patterns are specialized solutions, previously studied, and used to solve a frequent problem. Thus, parallel patterns offer a higher abstraction level to organize the algorithms in the exploitation of parallelism. They also can be easily learned by inexperienced programmers and software engineers. This work carried out a case study based on the Master/Slave pattern, focusing on the parallelization of algorithms for multi-core architectures. The implementation was validated through experiments to evaluate the programming effort to write code in LED-PPOPP and the performance achieved by the parallel code automatically generated. The obtained results let us conclude that a significant reduction in the parallel programming effort occurred in comparison to the Pthreads library utilization. Additionally, the final performance of the parallelized algorithms confirms that the parallelization with LED-PPOPP does not bring on significant losses related to parallelization using OpenMP in most of the all experiments carried out.

Keywords: Parallel Patterns Oriented Parallel Programming, Parallel Patterns, Parallel Programming, Domain-Specific Language.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE SIGLAS

GPGPU	*General-Purpose Computing on Graphics Processing Units*
MPSoC	*Multiprocessor System-on-Chip*
SO	Sistema Operacional
SA	Seminário de Andamento
NUMA	*No Uniform Memory Access*
MPI	*Message Passing Interface*
SPMD	*Single Program Multiple Data*
OpenMP	*Open MultiProcessor*
D&C	*Divide and Conquer*
ID	IDentificador
API	*Aplication Programming Interface*
DSL	*Domain Specific Language*
XML	*Extensible Markup Language*
SQL	*Structured Query Language*
IDE	*Integrated Development Environment*
OpenMP	*Open Multi-Processing*
TBB	*Threading Bulding Blocks*
SWARM	*SoftWare and Algorithms for Running on Multi-core*
MIT	*Massachusetts Institute of Technology*
IPS	*Intel Parallel Studio*
SPSC	*Single-Producer-Single-Consumer*
LED-PPOPP	Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos
PPOPP	Programação Paralela Orientada a Padrões Paralelos
GNU	*General Public License*
GCC	*GNU Compiler Collection*
PPGCC	Programa de Pós-Graduação em Ciência da Computação
PUCRS	Pontifícia Universidade do Rio Grande do Sul
SPSS	*Statistical Package for the Social Sciences*
LAD	Laboratório de Alto Desempenho
EI_2D	*Estimate an Integral 2D*
FFT	*Fast Fourier Transform*
MD	*Molecular Dynamics*
MM	*Matrix Multiplication*
PN	*Prime Numbers*

SUMÁRIO

1. INTRODUÇÃO

A indústria de processadores tem aumentado a capacidade computacional em ambientes de memória compartilhada através de diferentes tipos de arquiteturas paralelas, tais como: processadores *multi-core*, GPGPU (*General-Purpose Computing on Graphics Processing Units*) e MPSoC (*Multiprocessor System-on-Chip*). A maioria delas, são encontradas nos principais equipamentos ou plataformas computacionais atuais. Estações de trabalho, servidores, supercomputadores, ou até mesmo sistemas embarcados fazem parte desta exploração de paralelismo para prover maior desempenho em suas respectivas plataformas.

Interfaces de programação paralela estão disponíveis por meio de Bibliotecas e *Frameworks*, possibilitando uma abstração otimizada de parte dos detalhes que envolvem a exploração de paralelismo [3]. No entanto, é necessário que o programador entenda da arquitetura alvo para que sejam possíveis maiores níveis de escalabilidade e eficiência. Igualmente, os ambientes arquiteturais estabelecem diferentes formas e abordagens, desde o controle de acesso a dados, para o nível de memória compartilhada, até a comunicação entre processos, para o nível de memória distribuída [4].

Normalmente, o paralelismo direciona o programador a usar mecanismos de programação mais complexos e com pouca portabilidade. No contexto *multi-core* é necessário considerar fatores como: sincronização, escalonamento, balanceamento de carga e controle de acesso à seção crítica. A maioria das interfaces de programação paralela implementam estes mecanismos realizando a decomposição de dados e tarefas, entretanto, estas abordagens não são facilmente entendidas por programadores inexperientes e engenheiros de *software* [1].

Metodologias de programação paralela têm sido criadas para facilitar o entendimento e relatar as experiências de programadores na paralelização de algoritmos . Assim como, também descrevem os padrões paralelos em um alto nível, servindo de suporte para estruturar computações de maneira a obter vantagem das arquiteturas [5]. Isso tende a direcionar o programador a usar soluções especializadas usadas anteriormente na exploração de paralelismo, evitando erros e proporcionando aplicações de alto desempenho.

1.1 Motivação

Atualmente, em ambientes multitarefa, o Sistema Operacional (SO) é responsável por escalonar os processos nas unidades de processamento, beneficiando-se apenas através da execução paralela ou concorrente das aplicações, pois o código fonte é sequencial. Nos últimos anos, é notório o aumento da disponibilidade de computadores com alto poder computacional, mas somente uma pequena porcentagem das aplicações que executam em ambientes *multi-core* consegue obter vantagem do paralelismo disponível [6], [5], [7].

Percebe-se que a programação paralela ainda não está tão popular quanto as arquiteturas *multi-core*. As interfaces de programação paralela ainda não são um atrativo aos programadores, pois exigem um certo esforço no desenvolvimento de *softwares* e na aprendizagem [1], [4]. Portanto, é de suma importância a disponibilidade de soluções que sejam facilmente entendidas entre engenheiros de *software* e programadores inexperientes, fornecendo: uma interface intuitiva, uma garantia de bom desempenho e uma maneira otimizada de explorar o paralelismo disponível.

Este tipo de solução pode ser uma Linguagem Específica de Domínio [8] de programação paralela orientado a padrões, cujo o objetivo é fornecer uma interface de programação que implementa os detalhes que envolvem a exploração de paralelismo da arquitetura alvo (*i.e.*, sincronização, modelagem e acesso aos dados) e prover aplicações com alto desempenho. Com isso, a realização deste

trabalho torna-se um atrativo para a comunidade científica e aos desenvolvedores de *software*, pois poderão focar na resolução do problema e reduzir seus esforços para melhor aproveitar os recursos das arquiteturas.

1.2 Objetivos

O objetivo geral deste trabalho é a proposta, implementação e validação de uma Linguagem Específica de Domínio de programação paralela orientada a padrões paralelos, realizando um estudo de caso com o padrão Mestre/Escravo em arquiteturas *multi-core* para reduzir o esforço no desenvolvimento de *software* paralelo sem comprometer o desempenho das aplicações.

1.2.1 Objetivos Específicos

Motivado pelo objetivo geral deste trabalho, foram definidos os seguintes objetivos específicos:

- estudar os padrões no contexto de programação paralela;
- estudar as interfaces de programação, principalmente, Linguagens Específicas de Domínio;
- propor uma abordagem de programação paralela para implementar uma Linguagem Específica de Domínio;
- escolher um padrão paralelo para realizar um estudo de caso;
- estudar uma metodologia para avaliação dos resultados e elaborar um planejamento para realizar os experimentos;
- construir a Linguagem Específica de Domínio para programação paralela:
  - criar uma interface para a linguagem;
  - criar um compilador para a linguagem;
- realizar experimentos para medir o esforço de programação;
- realizar experimentos para medir o desempenho;
- avaliar os resultados utilizando a metodologia estudada.

1.2.2 Questões de Pesquisa

O presente trabalho busca resolver as duas seguintes questões de pesquisa:

1. Qual é o impacto no desenvolvimento de *software* paralelo, utilizando a LED-PPOPP (Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos) em relação a uma das soluções disponíveis atualmente na programação paralela?
2. Qual é o impacto no desempenho de uma aplicação que utiliza a LED-PPOPP comparado a soluções disponíveis atualmente na exploração de paralelismo?

1.3 Hipóteses Informais

Este estudo foi motivado com auxilio de hipóteses, as quais foram testadas com a finalidade de aceitar ou rejeitar [9]. Para isso, formulou-se duas hipóteses neste trabalho. A **(HP1)** tem como objetivo contribuir na resolução da primeira questão de pesquisa e a **(HP2)** está associada com a segunda questão de pesquisa:

- **(HP1)** Para o desenvolvimento de aplicações paralelas é necessário um esforço de programação. Sugere-se que o esforço de programação utilizando a interface proposta seja menor do que utilizando uma existente.
- **(HP2)** Na exploração de paralelismo é importante que as aplicações obtenham desempenho nas arquiteturas paralelas. Sugere-se que a abordagem proposta não possua uma diferença significativa de desempenho em relação ao uso de uma das abordagens para exploração de paralelismo otimizada.

1.4 Etapas da Pesquisa

Uma boa pesquisa sempre é acompanhada de um plano bem elaborado. Para este, foram formuladas 4 etapas (estudo, planejamento, desenvolvimento e avaliação) e representadas no desenho de pesquisa ilustrado na Figura 1.1.

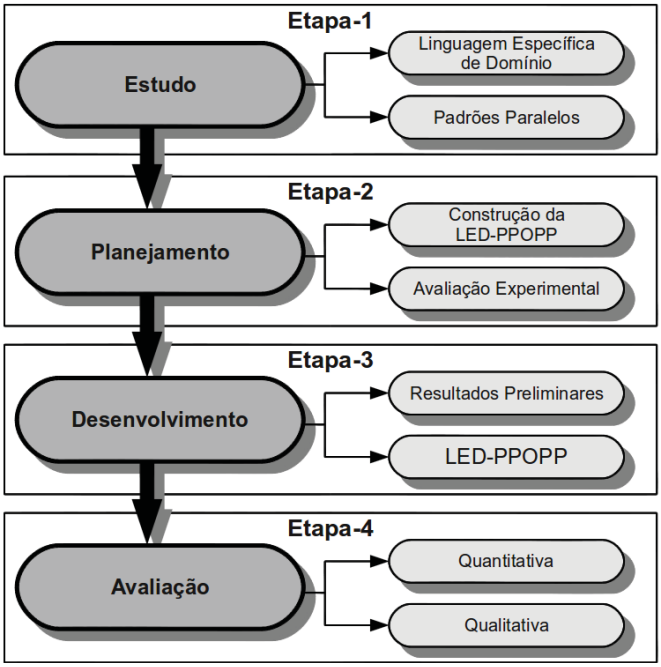


Figura 1.1: Desenho de Pesquisa.

A **Etapa-1** teve como objetivo realizar um estudo exploratório. Nela foram estudadas as principais interfaces de programação (Capítulo 3) e os padrões usados na programação paralela (Capítulo 2), analisando qualitativamente as suas características e benefícios. Além disso, estudou-se alguns trabalhos que se correlacionam com o tema de pesquisa proposto a fim de identificar as contribuições deste trabalho em relação aos existentes atualmente (apresentado no Capítulo 4).

Para a **Etapa-2** um planejamento de construção e avaliação foi prevista. Muito mais do que um planejamento, realizou-se um estudo para conhecer as metodologias de avaliação e validação

das interfaces de programação. Isso possibilitou o desenvolvimento do plano de execução da metodologia escolhida, que foi conduzida através de experimentos (Seções 7.1 e 7.2). O outro estudo concentrou-se na elaboração de um conceito (o PPOPP é descrito na Seção 5.1) para desenvolvimento de algoritmos paralelos com intuito de ser implantado na Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP), que posteriormente possibilitou projetar a construção dela.

Na **Etapa-3**, concentrou-se a maioria dos esforços para o desenvolvimento desta pesquisa. Primeiramente, escolheu-se um padrão dos que foram estudos para realizar uma breve implementação com a abordagem PPOPP e OpenMP, apresentada no SA (Seminário de Andamento) conforme está protocolado no sistema de avaliação da dissertação. Com estes resultados preliminares conseguiu-se demonstrar que o padrão Mestre/Escravo implementado em memória compartilhada pode fornecer aplicações com bom desempenho e similar em relação a utilização do OpenMP. Então, o próximo passo foi a construção da LED-PPOPP (Capítulo 5). Neste processo, foi criada uma interface de programação (Seção 5.2) e um compilador (Seção 5.4) para a linguagem, tarefa esta, que exigiu um estudo aprofundado de DSL, Pthread, implementação de compiladores, linguagem C e do padrão Mestre/Escravo.

Concluindo a pesquisa, a **Etapa-4** tratou da avaliação qualitativa e quantitativa dos resultados. Além de analisar os resultados (Capítulo 7), esta etapa também contempla a execução dos experimentos controlados e da medição de desempenho. Nos experimentos controlados foram realizadas a seleção de indivíduos e a aplicação de questionários para a formação de dois grupos, nos quais, para cada um deles avaliou-se o tempo necessário para o desenvolvimento de uma aplicação com o auxílio de um formulário contendo informações dos resultados obtidos. No entanto, a medição do desempenho das duas abordagens houve apenas a intervenção do pesquisador, sendo este, o responsável por paralelizar cinco aplicações diferentes usando a LED-PPOPP e compará-las com as mesmas aplicações paralelizadas utilizando outra abordagem, porém, desenvolvidas por outro profissional da área.

1.5 Organização do Trabalho

Após a introdução da pesquisa realizada, o Capítulo 2 deste trabalho apresentará o embasamento teórico sobre programação paralela com padrões paralelos, realizando um estudo sobre padrões de projeto, metodologia de programação paralela e os principais padrões paralelos. Assim como, o Capítulo 3 define e caracteriza as interfaces de programação utilizadas para abstrair trechos de código. Na sequência, o Capítulo 4 faz um apanhado geral do estado da arte referente as interfaces de programação disponíveis no contexto de arquiteturas *multi-core*, relacionando-as com a proposta deste trabalho.

A implementação de estudo de caso da Linguagem Específica de Domínio de programação paralela orientada a padrões paralelos proposta é descrita no Capítulo 5, detalhando a abordagem e a interface de programação usando o padrão Mestre/Escravo. A seguir, o Capítulo 6 apresenta o planejamento e a execução dos experimentos para medição do esforço de programação paralela e a medição do desempenho obtido na exploração de paralelismo em algumas aplicações de teste. Finalizando o estudo experimental, o Capítulo 7 descreve os resultados obtidos para os experimentos realizados.

2. PROGRAMAÇÃO PARALELA COM PADRÕES

Padrão de projeto é uma solução genérica para resolver problemas encontrados frequentemente na computação em um domínio específico. Foi inicialmente introduzido na Engenharia de *Software* no cenário de programação orientada a objetos. Uma das necessidades era estruturar classes em estruturas de dados heterogêneas. Esta descrição em alto nível possibilita que soluções inteligentes sejam utilizadas por outros desenvolvedores de *software* [4], [10], [1].

Partindo desta premissa de reutilização de soluções inteligentes, metodologias de programação paralela foram propostas no contexto de exploração do paralelismo. Esta metodologia está dividida em quatro espaços de projeto (Procura da Concorrência, Estrutura de Algoritmo, Estrutura de Apoio e Mecanismos de Implementação) a fim de tornar mais fácil o entendimento do problema a ser resolvido. Portanto, guiando-se através de uma metodologia de programação paralela é possível obter uma avaliação qualitativa de diferentes padrões paralelos [1], [11], [7].

Padrões paralelos definem a estrutura de um programa paralelo em um nível mais alto de abstração. No cenário atual, eles são utilizados para modelar algoritmos, cuja a finalidade é de explorar o paralelismo disponível das arquiteturas paralelas. A abordagem de paralelismo é implementada por meio de padrões que exploram a decomposição de dados e tarefas. Para tanto, a definição genérica permite que os padrões paralelos sejam implementados em um vasto conjunto de aplicações [12], [13], [14].

2.1 Padrões de Projeto

Segundo [4], cada padrão é responsável por implementar um problema que precisa ser resolvido frequentemente em um determinado ambiente. Basicamente, o padrão descreve o núcleo de uma solução para este problema, de modo que esta solução possa ser usada várias vezes sem precisar fazer a mesma coisa. Isso é comum na engenharia de *software*, onde os problemas frequentes são formalmente documentados e se tornam um padrão para outros que necessitam trabalhar com algo semelhante. Assim, os desenvolvedores podem basear-se no padrão criado e poupar os esforços para resolver este problema, focando-se apenas em desenvolver a aplicação final.

Com o passar dos anos, a criação de padrões tem sido comum na área de engenharia de *software*. Igualmente, outras áreas passaram a investir neste contexto para facilitar e direcionar o desenvolvedor para melhores práticas de programação. Entretanto, os padrões operam da mesma forma nas diferentes áreas, possuindo um único objetivo, fornecer uma solução para um problema em um determinado contexto. Logo, os padrões devem possuir quatro elementos:

- Nome do Padrão: o nome do padrão tem por objetivo aumentar o vocabulário do projeto que descreve um problema, suas soluções e consequências em uma ou duas palavras. O nome deve ser bom a fim de facilitar o entendimento do desenvolvedor sobre o comportamento do padrão e como ele é implementado.

- Problema: usado para descrever quando o padrão é aplicado, explicando o problema e qual é o seu contexto. As vezes, o problema pode conter uma lista de condições que devem ser realizadas antes de poder aplicar o padrão.

- Solução: descreve os elementos que compõem o projeto, os relacionamentos, responsabilidades e colaborações. A solução não descreve um projeto ou uma implementação concreta, pois os padrões são como um *template* que pode ser aplicado em várias situações diferentes. Basicamente, a solução do padrão descreve como um problema de projeto é resolvido.

- Consequência: diz respeito aos resultados de quando o padrão é aplicado. As consequências do padrão não são tão claras para o projeto de decisão, mesmo assim, são importantes para a avaliação de projeto e para o entendimento dos custos e benefícios na utilização do padrão.

Embora existam algumas variações na definição do conceito de padrões de projeto, é importante adotar uma caracterização clara dos quatro elementos descritos anteriormente obtendo assim uma definição formal utilizável por programadores de diferentes áreas. Assim, os padrões de projeto (*Design Patterns*) podem fornecer soluções inteligentes com um alto nível de abstração e guiar os programadores no seu projeto de *software*.

2.2 Metodologia para Programação Paralela

Padrões de programação apareceram com o surgimento das primeiras linguagens de programação no fim da década de 60/início da década de 70 [5]. Os padrões sequenciais são tradicionalmente divididos em dois grupos: padrões de controle de fluxo e padrões de gerenciamento de dados. Dentre os padrões de controle de fluxo, destacam-se: sequência, seleção, iteração e recursão. Os padrões de gerenciamento de dados mais usados são: leitura de acessos aleatório, alocação de pilha e alocação dinâmica de memória.

No contexto de programação paralela, os padrões são descritos ou implementados a partir de uma metodologia. A metodologia ajuda a explorar várias abordagens para a computação, levando à solução de problemas frequentemente encontrados no desenvolvimento de programas paralelos [4]. Enfim, auxilia os programadores em seus projetos de *software*, realizando a paralelização a partir de soluções especializadas na exploração de paralelismo através dos quatro espaços de projeto: procura da concorrência, estrutura de algoritmo, estrutura de apoio e mecanismos de implementação.

2.2.1 Procura da Concorrência

O desenvolvimento de algoritmos paralelos não é uma tarefa simples e fácil [1], [5]. Antes de modelar um algoritmo paralelo, o levantamento da aplicação se faz necessário para identificar o problema a ser resolvido. Na sequência, se paralelizar for vantajoso, estuda-se quais padrões paralelos podem ser implementados na aplicação. Então, o primeiro passo no projeto de algoritmos paralelos é a procura da concorrência em uma determinada computação. A representação deste espaço de projeto está ilustrado na Figura 2.1, onde os padrões estão classificados em três grupos.

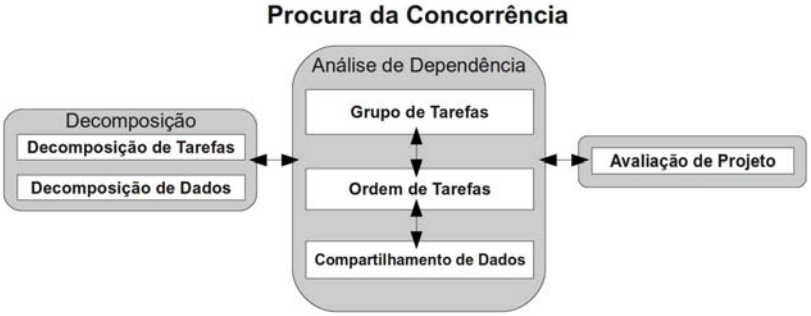


Figura 2.1: Encontrando Concorrência. Adaptado de [1]

O grupo de **decomposição** tem por objetivo decompor um problema em pequenos fragmentos para que possam ser executados em paralelo ou concorrentemente. A **análise de dependência** opera entre si e possivelmente pode revisar a decomposição para encontrar as dependências de uma computação. E, o grupo de **avaliação de projeto** direcionam o projeto de algoritmo para uma

avaliação (testando qual é a melhor solução a ser aplicada), antes de encaminhar para o próximo espaço de projeto.

Decomposição de Tarefas

Na decomposição de tarefas, o problema é visto como um fluxo de instruções que podem ser quebrados em tarefas que são executadas concorrentemente [15]. O programador deverá identificar quais são os pontos (que geram tarefas) a serem paralelizados no algoritmo, levando em conta os seguintes aspectos:

- Flexibilidade: permitir que o número e o tamanho de tarefas possam ser parametrizados para diferentes números de processadores;
- Eficiência: garantir que não ocorram problemas como, *overhead* (sobrecarga de trabalho) pelo gerenciamento de dependência das tarefas e que o número de tarefas não seja maior que o número de unidades de processamento;
- Simplicidade: tornar possível a depuração e a manutenção o mais simples possível.

A decomposição de tarefas pode ser usada em aplicações como: imagem médica; multiplicação de matrizes e dinâmica molecular.

Decomposição de Dados

A decomposição de dados foca-se nos dados exigidos pelas tarefas e como podem ser decompostos em pedaços separados. Isso só é eficiente se os pedaços de dados podem operar independentemente. Algoritmos de decomposição de dados fazem o uso eficiente da memória e (em ambientes com memória distribuída) usam menor largura de banda. Mas, ocorre maior sobrecarga de comunicação durante a execução de partes concorrentes, sendo também mais complexa a sua decomposição. Máquinas com arquitetura NUMA (*No Uniform Memory Access*) se mostram vantajosas utilizando esta decomposição de dados [11].

No projeto de algoritmos paralelos, o primeiro passo utilizando esta decomposição é identificar as estruturas de dados definidas pelo problema e verificar o que pode ser quebrado em segmentos [1]. Um exemplo prático para este cenário é a computação baseada em vetores (quebrar em segmentos) e estrutura de dados recursiva (decompor em sub-árvores). Os seguintes aspectos devem ser fortemente considerados ao utilizar este padrão:

- Flexibilidade: o tamanho e a quantidade dos segmentos de dados devem ser flexíveis a fim de suportar um sistema paralelo amplo, onde a partir da parametrização define-se a granularidade, podendo variar conforme o *hardware* utilizado;
- Eficiência: a decomposição torna-se eficiente quando a quantidade de pedaços não causa a sobrecarga no gerenciamento de dependência. Para tanto, um algoritmo paralelo deve balancear a carga entre as *threads* ou processos;
- Simplicidade: decompor dados é complexo, o que torna a depuração difícil. Esta decomposição geralmente requer um mapeamento global dentro de uma tarefa local, beneficiando-se deste mapeamento abstrato para isolar e testar.

Aplicações como, imagens médicas, multiplicação de matrizes e dinâmica molecular, utilizam frequentemente a decomposição de dado, pois é possível atingir uma granularidade mais fina se comparado com a decomposição de tarefas.

Grupo de Tarefas

As abordagens de decomposição de dados e de tarefas são correspondentes ao grupo de tarefa, que descreve a primeira etapa na análise de dependências. Em um cenário de aplicação é possível dividi-la em vários problemas, para cada um, uma tarefa é usada para resolvê-lo. Por outro lado, é possível criar grupos de tarefas que são agrupados correspondentes a seus devidos problemas. Isso se torna um benefício na análise de dependência das tarefas, pois o controle passa a ser realizado nos grupos (quando um grupo termina o outro grupo pode começar a ler). Todavia, se um grupo de tarefas trabalha juntamente a uma estrutura de dados compartilhado, necessita-se de sincronização em todo grupo.

Quando existem restrições entre as tarefas podem ocorrer casos como: dependência temporal; coleção de tarefas executando ao mesmo tempo e tarefas que podem ser independentes umas das outras dentro do grupo [16]. Para identificar as restrições e grupos de tarefas, deve-se olhar primeiramente como o problema original foi decomposto. Em seguida, verificar se outro grupo compartilha a mesma restrição, identificando as restrições entre os grupos de tarefas [1].

Ordenação de Tarefas

A ordenação de tarefas consiste na segunda etapa de análise de dependências, cujo o objetivo é identificar como os grupos de tarefas podem ser ordenados para atender as restrições entre as tarefas. Assim como, a ordenação deve ser restrita o suficiente para satisfazer todos os requisitos que resultam em um projeto correto. Neste sentido, a ordenação não deve ser mais restrita que ela precisa ser.

De forma a contribuir na ordenação das restrições, é importante observar a necessidade do dado para um grupo de tarefas antes executá-lo. Igualmente, também podem ser considerados serviços externos que promovem a ordenação de restrições e identificar quando uma ordenação não existe.

Compartilhamento de Dados

Um algoritmo paralelo é composto de: uma coleção de tarefas que executam concorrentemente, uma decomposição de dados correspondente a uma solução de tarefas concorrentes e também sobre a dependência entre as tarefas, que devem ser gerenciadas para permitir uma execução segura [1]. O objetivo desta abordagem é o compartilhamento de dados entre os grupos de tarefas e determinar o acesso de dados compartilhados de maneira correta e eficiente. Para garantir estes fatores, o algoritmo deve prever questões como: condição de corrida, geração de sincronização excessiva e *overhead* de comunicação (em sistemas distribuídos).

Comumente, os dados são compartilhados entre as tarefas (padrão de decomposição de dados), decompondo os dados em blocos e o compartilhamento de dados é realizado entre estes blocos. Na decomposição de tarefas se torna mais complicado, pois é necessário tratar os dados passados dentro ou fora da tarefa e tratar quando um dado é atualizado no corpo da tarefa. O programador é responsável por conhecer as formas de uso do compartilhamento de dados, para as quais inclui-se: somente leitura, efetivamente local (operações com matrizes), leitura e escrita, acumulação e múltipla leitura/somente escrita.

Avaliação de Projeto

A avaliação de projeto consiste na última etapa no processo de encontrar concorrência, preparando o programa para o próximo espaço de projeto. A decomposição do problema original e a análise pode ser realizada com: uma decomposição de tarefas que identifica tarefas que podem

executar concorrentemente; uma decomposição de dados que identifica o local dos dados para cada tarefa; uma maneira de agrupamento de tarefas e uma ordenação de grupos para satisfazer restrições temporais que analisam as dependências entre as tarefas. Baseando-se nestes termos, esta abordagem tem por objetivo encontrar qual é a melhor decomposição do problema para produzir um projeto de ótima qualidade, avaliando três aspectos:

- Adequação para a plataforma alvo: para a escolha da plataforma alvo devem ser levados em consideração: a quantidade de processadores disponíveis, como são compartilhadas as estruturas de dados entre os elementos de processamento, identificar como a arquitetura alvo implica sobre o número de *threads* ou processos e como as estruturas de dados são compartilhadas entre elas. Por último, analisar o tempo gasto trabalhando em uma tarefa, sendo que o tempo deve ser adequado quando estiver lidando com as dependências;
- Qualidade de projeto: na qualidade de projeto, requer-se que as características da plataforma alvo sejam mantidas, avaliando as dimensões de flexibilidade, eficiência e simplicidade;
- Preparação para a próxima fase de projeto: o projetista quando se deparar com avaliação deste aspecto deve considerar a regularidade das tarefas e suas dependências de dados, interação entre as tarefas (síncrono ou assíncrono) e se as tarefas estão agrupadas da melhor maneira.

2.2.2 Estrutura de Algoritmo

Uma estrutura de algoritmo requer eficiência, simplicidade, portabilidade e escalabilidade [1]. Porém, estes termos causam conflitos, como por exemplo, eficiência com portabilidade. Alguns programas necessitam ser escritos com características específicas, o que não torna possível a portabilidade. A eficiência com a simplicidade também é um caso de conflito, pois geralmente aplica-se o paralelismo de tarefas, necessitando o uso de algoritmos de escalonamento complexos que dificultam o entendimento. No entanto, um bom projeto de algoritmo deve estabelecer um equilíbrio entre abstração e portabilidade para uma determinada arquitetura.

O princípio de organização implica na concorrência, sendo assim definidos os termos de tarefas, grupos de tarefas e ordenação entre grupos de tarefas. Para isso, no projeto de algoritmo paralelo usa-se múltiplas estruturas de algoritmos a fim de encontrar um algoritmo estruturado que representa o mapeamento da concorrência dentro das *threads* ou processos. Em consequência, este espaço de projeto é dividido em três grupos (Organização por Tarefas, Decomposição de Dados e Fluxo de Dados) de padrões representados através da Figura 2.2.

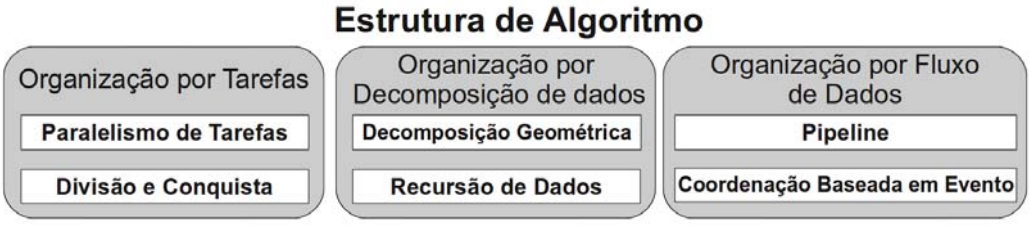


Figura 2.2: Estruturas de Algoritmos. Adaptado de [1].

Esta separação de grupos de padrões permite uma classificação dos padrões para este espaço de projeto de algoritmos paralelos. Portanto, o grupo de Organização por Tarefa implementa os padrões que são baseados em tarefas para exploração de paralelismo. O grupo de Organização por Decomposição de Dados aborda os padrões que se utilizam da concorrência para resolver um problema recursivamente. E, o grupo de Organização por Fluxo de Dados caracteriza os padrões que definem como o fluxo de dados é apresentado na ordenação em grupo de tarefas.

2.2.3 Estruturas de Apoio

O espaço de projeto de estruturas de apoio descreve construções ou estruturas de *software* que comportam os algoritmos paralelos. Conforme ilustra a Figura 2.3, o espaço de projeto está representado em dois grupos de padrões: os que representam a abordagem de estrutura dos programas e os que representam o uso de estrutura de dados. Nada impede o uso de outro padrão para implementar uma estrutura, como por exemplo, Mestre/Escravo utilizar *Fork/Join* ou SPMD. Isso significa que estes padrões não representam unicamente uma maneira de estruturar um programa paralelo.

Figura 2.3: Estruturas de Apoio. Adaptado de [1]

Para um programador MPI (*Message Passing Interface*), todos os programas de padrões estruturados são derivados do padrão SPMD(Single Program Multiple Data). Entretanto, para um programador OpenMP (*Open MultiProcessor*) existe uma enorme diferença entre programas que utilizam identificadores de *threads* (padrão SPMD) e programas que expressam a concorrência em nível de laço (padrão de paralelismo de laço) [16]. Geralmente todos os programas que utilizam padrões estruturados enfrentam alguns problemas, tais como:

- clareza e abstração: refere-se a forma que o algoritmo está escrito no código fonte. Uma abstração clara é importante para escrever um código corretamente e auxiliar na depuração;
- escalabilidade: diz respeito a quantidade de processos que um programa paralelo pode eficientemente utilizar. Ao restringir a concorrência disponível no algoritmo, a quantidade de processadores será limitada. Da mesma forma, o *overhead* pode contribuir para baixar e limitar a escalabilidade;
- eficiência: está diretamente relacionada com o desempenho do programa em relação ao programa sequencial;
- sustentabilidade: concentra-se na facilidade para modificar, depurar e verificar um *software* e em relação a sua qualidade;
- afinidade do ambiente: envolve a relação do programa com o ambiente de programação e com a escolha do *hardware*;
- equivalência sequencial: esta situação corre quando um programa produz resultados equivalentes, quando executa com vários processos assim como em um único processo.

2.2.4 Mecanismos de Implementação

O mecanismo de implementação é a última etapa do espaço de projeto de algoritmos paralelos, sendo o mais alto nível de construção para organizar programas paralelos. Este, conforme ilustra a

Figura 2.4: Mecanismos de Implementação. Adaptado de [1]

Figura 2.4, está dividido em três categorias: gerenciamento de *threads* e processos, sincronização e comunicação.

As próximas seções relatam os três mecanismos de implementação. Estes, não são considerados padrões, mas fazem parte do projeto de algoritmos paralelos, referindo-se ao gerenciamento de *threads* ou processos, sincronização e comunicação.

Gerenciamento de *Threads* e Processos

O processo é um objeto que carrega um contexto em algum lugar no sistema, isso inclui memória, registradores e *buffers*. Em um sistema, diferentes processos pertencem a diferentes usuários. Ao contrário, uma *Thread* é um novo fluxo criado a partir de um processo, podendo compartilhar o mesmo espaço de endereçamento e a comunicação entre as *threads* pertencer ao mesmo processo.

Processos e *threads* podem ser criados e destruídos. *Threads* são mais simples e não exigem muito poder computacional para serem criadas. Por outro lado, os processos são mais custosos, pois quando ele é criado, todas as informações necessárias para definir um lugar de atuação no sistema devem ser carregadas. Interfaces de programação como OpenMP, Pthread, Java e MPI disponibilizam funções de criação de *threads* e processos, abstraindo grande parte da complexidade em lidar com esta abordagem de programação [17].

Sincronização

No gerenciamento de *threads* e processos, a sincronização ocorre quando se deseja manter uma ordem na execução dos eventos [18]. As interfaces de programação como OpenMP, Pthread, Java e MPI implementam a sincronização em suas funções, incluindo os seguintes aspectos:

- Sincronização e barreira de memória: no ambiente computacional de memória compartilhada, *threads* e processos podem executar uma sequência de instruções que são lidas e escritas na memória compartilhada atomicamente. Também pode ser visto como uma sequência de eventos atômicos, os quais intercalam a partir de diferentes *threads* ou processos. As barreiras de memória são usadas na sincronização para garantir que processos ou *threads* tenham uma visão consistente da memória. No contexto de *threads*, um programa com várias condições de corrida possivelmente não será vantajoso;

- Barreiras: são o mais alto nível para tratar a sincronização de *threads* e processos. As barreiras são usadas para garantir que um conjunto de *threads* ou processos não proceda antes que todos estejam em um determinado ponto;

- Exclusão mútua: o objetivo é garantir que *threads* ou processos não interfiram entre si, pois quando os recursos são compartilhados dois ou mais processos ou *threads* podem tentar atualizar um dado compartilhado, gerando conflitos e inconsistências. O acesso a seção crítica é controlado usando exclusão mútua, onde uma *thread* ou processo avança e enquanto isso, os outros ficam esperando até que a seção crítica seja liberada.

Comunicação

Para que a troca de informações ocorra entre as *threads* e processos é necessário a comunicação, que pode ser realizada usando mensagens através de uma rede ou de uma memória. Uma mensagem pode conter dados sobre a mensagem e outras informações e, ainda, a troca pode ser realizada de duas formas: de uma origem específica para um destino específico e múltipla troca de mensagens entre processos ou *threads* em uma única comunicação (comunicação coletiva).

Na comunicação coletiva estão envolvidas diferentes operações, como por exemplo, o mecanismo de *broadcast*, barreiras e redução [16]. Esta, consiste em reduzir uma coleção de itens de dados para um único item de dados, para então combinar os itens de dados com uma operação binária, associativa ou comutativa (*i.e.*, soma, produto, maior elemento, menor elemento, entre outras).

2.2.5 Considerações

Este capítulo apresentou uma metodologia para programação paralela para projetar algoritmos paralelos e descrever soluções especializadas para exploração de paralelismo. Do mesmo modo, apresentou-se uma maneira qualitativa de entender o processo que envolve o desenvolvimento de *software* paralelo através dos espaços de projeto. A Figura 2.5 ilustra uma visão global da programação paralela utilizando os espaços de projeto e como ocorre a implementação destes padrões.

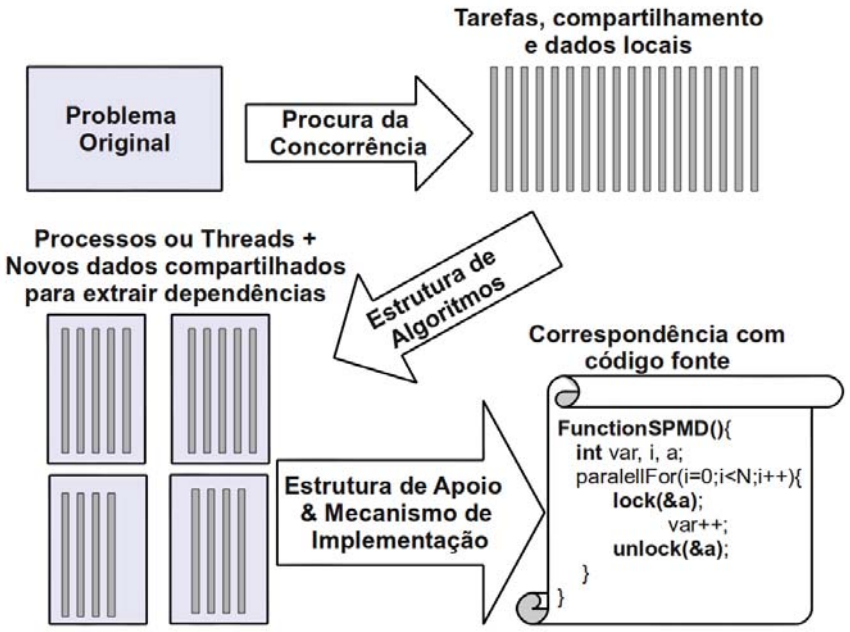


Figura 2.5: Exploração do paralelismo com padrão de linguagem para programação paralela.

A exploração de paralelismo com a metodologia para programação paralela começa com o espaço de projeto Procura da Concorrência, estruturando o algoritmo para obter vantagem do potencial de paralelismo que um problema pode oferecer. Esta primeira etapa, tem por objetivo analisar a computação encontrando possíveis tarefas, compartilhamentos e dados locais. Logo o próximo passo é estruturar os algoritmos em um modelo de processos ou *threads*, verificando as dependências e os dados compartilhados, para os quais a metodologia descreve as estratégias de implementação.

Com a definição das *threads* ou processos realizada na Estrutura de Algoritmo, os programadores podem guiar a exploração do paralelismo utilizando os padrões do espaço de projeto de Estruturas de Apoio, sendo descritas as formas para realizar a modelagem da computação anteriormente estruturada. Contudo, os Mecanismos de Implementação preocupam-se como os padrões de alto nível

são mapeados em um determinado ambiente de programação.

2.3 Padrões de Programação Paralela

Um padrão de programação paralela é uma solução genérica para problemas frequentemente encontrados na exploração de paralelismo [4], [1]. Igualmente, o seu objetivo é fornecer uma descrição em um alto nível de abstração para estruturar a computação obtendo vantagem das arquiteturas paralelas. Portanto, estes padrões são utilizados na organização de uma computação decomposta em dados ou tarefas, estruturando a execução de forma paralela ou concorrentemente em um sistema computacional.

Programas paralelos eficientes e confiáveis podem ser projetados em torno de padrões paralelos [5]. Estes, enquanto melhoram a produtividade através de padrões específicos, especializados e a fusão entre eles, ao mesmo tempo podem guiar os usuários inexperientes a desenvolver algoritmos eficientes com boa escalabilidade. Por outro lado, a metodologia de programação paralela direciona o desenvolvimento de *software* para melhores práticas na exploração do paralelismo.

A experiência em desenvolver programas paralelos mostrava que vários dos problemas eram frequentes [4]. Isso alavancou o formalismo destas práticas, criando-se padrões de programação paralela. Mesmo que alguns autores os denominem de modelos ou estratégias de programação paralela, eles são usados com a mesma finalidade, fornecer uma descrição para estruturar programas que obtém vantagem sobre uma arquitetura paralela [16], [19]. No entanto, conforme a literatura descreve nas Seções 2.1 e 2.2, o conceito de padrão de programação paralela é mais apropriado para o formalismo de uma solução genérica no projeto de *software* paralelo [1], [20], [5], [21].

Nas seções 2.3.1 e 2.3.2 são descritos os padrões de programação paralela comumente utilizados e aplicáveis para um vasto conjunto de aplicações. Vários destes são introduzidos no contexto de arquitetura de memória compartilhada e distribuída. Por exemplo, quando o padrão Mestre/Escravo é implementado em ambiente de memória compartilhada, o mestre comunica-se com os escravos através da memória e quando se está em um ambiente de memória distribuída, a comunicação é realizada com troca de mensagens.

2.3.1 Padrões Paralelos de Estrutura Algorítmica

O propósito geral destes padrões é caracterizado na metodologia de programação paralela (Seção 2.2). Esta seção apresenta os padrões paralelos que determinam a estrutura algorítmica dos programas.

Padrão de Paralelismo de Tarefas

Quando se trata de paralelismo de tarefas, ao desenvolver uma aplicação deve-se observar e analisar como as tarefas são definidas, as dependências entre elas e o escalonamento delas. As tarefas, na decomposição de um problema, deveriam existir pelo menos tantas quanto unidades de processamento, de preferência mais, para obter maior flexibilidade no escalonamento [22]. Também é importante que a computação associada com as tarefas seja larga o suficiente para equilibrar a sobrecarga com o gerenciamento e manipulação de dependências.

Um dos maiores impactos no padrão de paralelismo de tarefas é a dependência, isso envolve questões como o compartilhamento de dados e grupos de tarefas. Na maioria dos casos os algoritmos tornam-se mais complexos e perdem eficiência [6]. Para isso, ao desenvolver um programa, é importante que o programador esteja preocupado com estas questões, procurando remover dependências e separando-as se possível.

O escalonamento parte da ideia de balancear a carga entre os processadores, mantendo um equilíbrio de tarefas entre as *threads* ou processo. Esta atribuição de tarefas aos processadores, pode ser realizada estaticamente (determinando no início da computação como e quanto é usado a *thread* ou processo, e esta se mantem até o fim) ou dinamicamente (variando a distribuição entre as *threads* ou processos, conforme procede a computação).

Este padrão pode ser usado para explorar o paralelismo de laços `for`, de preferência, dividindo as interações do laço em tarefas que executam em paralelo. Entretanto, deve-se tomar cuidado quando existe a dependência entre as tarefas ou se existe uma grande quantidade delas, pois isso poderá prejudicar o desempenho da aplicação. Além disso, quando ocorre a replicação de dados entre as tarefas, é necessário um controle maior sobre eles [22].

Padrão de Divisão e Conquista

O padrão D&C (*Divide and Conquer*) é uma estratégia utilizada em vários algoritmos sequenciais [1]. Consiste em dividir o problema em subproblemas menores, resolvendo-os independentemente e fundindo todas as subsoluções em uma solução para o problema [16]. Isso é resolvido de forma recursiva, tendo como objetivo explorar a eficiência na resolução de uma determinada computação.

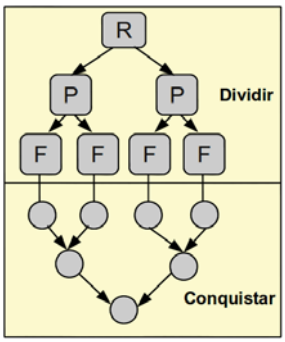


Figura 2.6: Padrão Divisão e Conquista.

Um exemplo de modelagem usando o padrão D&C é apresentado na Figura 2.6. Neste padrão, o primeiro passo é dividir o problema em subproblemas. Neste sentido, o resultado desta modelagem será um grafo de tarefas independentes. Depois da divisão, ocorre a conquista. Os resultados retornados do nível abaixo são fundidos e, logo em seguida, devolvidos a um nível acima do grafo. Desta forma, as tarefas filhas (**F**) retornam o resultado para a tarefa pai (**P**), logo após, estas tarefas devolverão o seu problema resolvido para a raiz (**R**), que obterá a solução.

Padrão de Decomposição Geométrica

Este padrão tem por objetivo realizar a divisão de uma região geométrica em sub regiões geométricas [20]. Para vetores, esta decomposição pode ser de uma ou mais dimensões, e os sub vetores resultantes são chamados de blocos e denominados de pedaços, as sub estruturas e as sub regiões. A decomposição de dados em pedaços implica que as operações nas tarefas sejam atualizadas, onde cada tarefa representa a atualização de um pedaço e estas executam concorrentemente. Em certos casos, alguns pontos necessitam da atualização de outros pedaços, sendo assim, a informação deve ser compartilhada entre os pedaços para completar a atualização.

A maior parte dos problemas que são resolvidos usando o padrão de decomposição geométrica são: soluções de equações diferenciais e álgebra linear computacional. Estes, geralmente implementam rotinas paralelas em suas bibliotecas matemáticas. Assim como, este padrão pode ser aliado com o padrão de decomposição de tarefas (quando a atualização para cada pedaço é realizada sem

dados a partir dos pedaços) e com o D&C (quando a estrutura de dados pode ser distribuída e recursiva).

Padrão de Recursão de Dados

Para os problemas de lista, árvores ou grafos, o padrão de recursão de dados é uma boa opção, pois busca o explorar paralelismo decompondo processos ou *threads* em elementos individuais. Da mesma maneira, o padrão de D&C também opera usando recursão de dados, mas sem grande potencial de concorrência. O objetivo deste padrão é reformular estas operações para que o programa possa operar concorrentemente em todos os elementos da estrutura de dados sem perder a forma recursiva de resolver uma computação [1].

Um dos principais desafios deste padrão é aplicar a mudança do algoritmo original para explorar a concorrência, isso implica que em todos os níveis da estrutura exista concorrência entre os processos. Para isso, cada nó comunica-se com o raiz ou com o pai dele. O padrão de recursão de dados é semelhante ao D&C, porém, acrescenta a concorrência na estrutura de dados.

Padrão Pipeline

O padrão *pipeline* consiste de uma computação baseada em estágios, visto também como uma sequência de dados através de uma sequência de estágios. Na definição de estágios de um *pipeline*, cada processo deve conhecer a quantidade de estágios existentes. A partir da contagem do número de elementos, cada estágio saberá o momento de parar e se o dado já foi processado [6]. Este padrão assume que no *pipeline* existe: um longo fluxo de dados, que toda entrada é processada e cada estágio pode obter uma entrada diferente ao mesmo tempo [23].

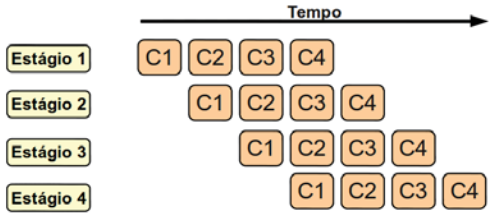


Figura 2.7: Padrão *Pipeline*.

Através da Figura 2.7, é possível observar o funcionamento e o paralelismo fornecido de um *pipeline* com quatro estágios. Embora o fluxo seja contínuo e o problema é executado sequencialmente, o paralelismo é obtido através da execução de mais de um estágio. Basicamente, cada estágio recebe uma entrada, processa e passa o resultado para o próximo estágio até concluir a computação. Na estruturação da computação, pode-se usar o ID (IDentificador) de cada processo para escolher uma opção, onde cada caso corresponde a um estágio do *pipeline* (*e.g.*, processar o resultado). Os estágios podem ser conectados explicitamente com canais bufferizados, implementando por exemplo, filas compartilhadas entre os que enviam e os que recebem de tarefas.

Padrão de Coordenação Baseada em Eventos

Ao contrário do padrão *pipeline*, a coordenação baseada em eventos não opera estritamente em uma estrutura linear, não possui restrições que um fluxo de dados seja apenas de uma maneira e as iterações são irregulares com intervalos imprevisíveis. Um exemplo disso é uma garagem de lava-carros. Esta por sua vez, possui duas máquinas de lavar carro e uma fila de atendimento. Cada carro poderá ficar um determinado tempo usando o recurso e se ainda a limpeza não for concluída,

o carro volta para a fila de atendimento, passando novamente pela máquina de lavar carro e assim sucessivamente, até que o carro esteja limpo. Os carros que estão na fila ganham o recurso quando a máquina não está ocupada e quando chegou a sua vez.

Para fluxos de dados usa-se eventos, onde cada evento contém uma tarefa que gera o evento e uma tarefa que processa o evento. Isso porquê, um evento deve ser gerado antes que ele seja processado, definindo a restrição de ordenação entre as tarefas, onde a computação de cada tarefa consiste de processamento de eventos [1]. Assim sendo, a estrutura básica de cada tarefa consiste no: recebimento de um evento, processamento dele e possivelmente gerá-lo. Na representação do fluxo de evento, estão associados a comunicação e a computação de sobreposição. De modo geral, a comunicação é assíncrona dos eventos, na qual, uma tarefa pode criar (enviar) um evento e continuar sem esperar por um destinatário para receber um evento.

Em ambientes de memória distribuída, um evento pode ser representado por uma mensagem enviada assincronamente a partir da tarefa, gerando o evento para a tarefa que irá processá-lo. Já em ambientes de memória compartilhada, uma fila pode ser usada para simular a troca de mensagens, onde uma fila pode ser acessada por mais de um tarefa e deve ser implementada de forma a permitir o acesso seguro e concorrente. Deve-se ter cuidado com problemas de *deadlock*, escalonamento e alocação de processo (um processo por elemento de processamento) e uma eficiente comunicação de eventos.

2.3.2 Padrões Paralelos de Estruturas de Apoio

Os padrões paralelos de estruturas de apoio são são bastante usado para modelar e organizar uma computação decomposta em *threads* ou processo. Estes, são abordados no espaço de projeto de Estrutura de Apoio na metodologia de programação paralela (Seção 2.2).

Padrão SPMD (*Single Program, Multiple Data*)

Em programas que utilizam o padrão SPMD, os processos executam o mesmo programa, onde cada processo tem seu próprio conjunto de dados [6]. Isso significa que os processos podem seguir caminhos diferentes dentro de um programa. Devido a sua versatilidade, normalmente este padrão é usado para descrever e estruturar os padrões do espaço de projeto da estrutura de algoritmos paralelos. Mesmo assim, o padrão SPMD é melhor aplicado quando usado em casos de integração numérica e dinâmica molecular. Este é constituído por alguns elementos básicos que compõem sua estrutura:

- inicialização;
- obtenção de um único identificador;
- execução do mesmo programa em cada processo, usando o identificador único para diferenciar comportamento em diferentes processos;
- dados distribuídos;
- finalização.

Padrão Mestre/Escravo

O padrão Mestre/Escarvo é uma solução baseada no paralelismo de tarefas, onde uma tarefa mestre possui instâncias de uma ou mais tarefas escravas [1]. Conforme o exemplo da Figura 2.8, a tarefa mestre (**M**) inicializa a computação dividindo-a em tarefas independentes. Isso resultará

em um saco de tarefas que são enviadas aos escravos (**S**) para serem computadas. Ao resolverem a computação os escravos retornam o resultado para o mestre que une os resultados e processa a solução final do problema.

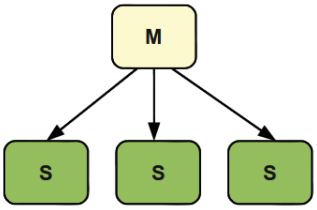


Figura 2.8: Padrão Mestre/Escravo.

Soluções que possibilitam modelar a computação em tarefas independentes (*e.g.*, paralelismo de laço) podem apresentar bom desempenho na exploração de paralelismo com este padrão paralelo. No entanto, para que a implementação apresente bons resultados é preciso dividir o problema adequadamente. No padrão Mestre/Escravo existe um comunicador global para resolver dependências entre as tarefas e distribuí-las. Em alguns casos, quando a comunicação é bastante requisitada é comum acontecer uma sobrecarga de comunicação, o que afeta o desempenho do sistema. A solução para este tipo de problema é aumentar o tamanho das tarefas escravas para diminuir o número de acessos ao saco de tarefas.

Padrão de Paralelismo de Laço

O paralelismo de laço tem como propósito encontrar maneiras de explorar o paralelismo através de programas que possuem estruturas baseadas em laços, ou seja, transformar um programa serial, onde o tempo de execução é determinando por um conjunto intensivo de computações baseadas nas interações de laço, executando-as em paralelo com diferentes índices. Segundo [1], é particularmente relevante usar o paralelismo de laço para programas OpenMP em computadores com memória compartilhada e problemas que operam em padrões de paralelismo de tarefas e decomposição geométrica. A implementação deste padrão com OpenMP implica nas seguintes etapas:

- encontrar gargalos: o programador é quem identifica os laços no código para combinar e encontrar o desempenho necessário de cada subproblema;
- eliminar dependência na interação do laço: consiste em eliminar operações de escrita e leitura que provocam uma sessão crítica;
- paralelizar o laço: dividir as iterações entre as *threads* ou processos;
- otimizar o escalonamento do laço: as iterações devem ser escalonadas em *threads* ou processos, obtendo o balanceamento de carga;
- mesclar laços: um sequencia de laços que possuem limites consistentes podem ser mesclados em um único laço com iterações mais complexas;
- reunir laços alinhados: normalmente é possível reunir laços alinhados em um único laço, gerando uma grande combinação de iterações.

Padrão Fork/Join

O padrão *Fork/Join* é baseado no conceito de processo pai e processos filhos. Um processo pai pode criar vários processos filho dinamicamente, quando um processo terminar ele executa *join* e termina, os demais processos filhos continuam sua execução [19]. Este padrão é usado em exemplos que usam recursividade, como o padrão D&C. Com o produto da execução do programa o problema é divido em subproblemas e novas tarefas são recursivamente criadas para executar concorrentemente os subproblemas, desta maneira, cada uma destas tarefas pode se tornar um subdivisor. Quando todas as tarefas foram criadas ao realizar uma divisão elas são finalizada e posteriormente se juntam com a tarefa pai, para então, a tarefa pai continuar a computação.

Parafraseando [1], o padrão *fork/join* é relevante para programas Java executando em computadores com memória compartilhada e para problemas que usam padrões como D&C e Recursão de Dados. O OpenMP pode ser usado efetivamente com padrões quando o seu ambiente suporta alinhamento de regiões paralelas. Os problemas que usam o padrão *fork/join* possuem o mapeamento de tarefas em *threads* ou processos de duas maneiras:

- mapeamento direto de tarefa: nesse mapeamento existe uma tarefa por processo/*thread*. Para cada nova sub tarefa criada, novas *threads*/processos são criados para manipulá-las. Na maioria dos casos, existe um ponto de sincronização onde a tarefa principal espera pelo término das sub tarefas, isto é conhecido como *join*;

- mapeamento indireto de tarefa: neste mapeamento existe uma certa quantidade de *threads*/processos trabalhando em um conjunto de tarefas. A ideia é a criação estática de *threads* ou processos antes de começar as operações de *fork/join*. Então, o mapeamento de tarefas em *threads* ou processos ocorre dinamicamente usando uma fila de tarefas. Neste tipo de mapeamento, não é possível que *threads* ou processos criam e destruam a si mesmos, mas podem simplesmente mapear para tarefas criadas dinamicamente conforme necessitarem. Em alguns casos, a implementação é complicada e geralmente resultam em programas eficientes com bom balanceamento de carga.

Como exemplo de aplicação que são implementadas com este padrão através do mapeamento direto e indireto de tarefas, tem-se o algoritmo de ordenação de vetores *mergesort*.

Padrão de Compartilhamento de Dados

A manipulação de dados com este padrão é compartilhada normalmente por mais de um processo, onde o compartilhamento de dados implica em um conjunto de tarefas operando concorrentemente ou em paralelo. Além disso, trabalhar com dados compartilhados propicia o programador a cometer erros no projeto de algoritmos paralelos. Por isso, é importante uma boa abordagem que enfatiza a simplicidade e uma boa abstração, possibilitando que abordagens mais complexas sejam exploradas para obter maior desempenho.

A primeira etapa no projeto do algoritmo é identificar se este padrão é realmente necessário. Se este é importante, começa-se a definir uma abstração e os tipos de dados, como por exemplo, no padrão de compartilhamento de fila, apenas são incluídas operações de colocar e tirar um elemento na fila quando as operações foram concluídas, para posteriormente incluir novas operações de manipulação na fila compartilhada [16]. Por outro lado, também é necessário que o algoritmo paralelo implemente um protocolo de controle de concorrência apropriado, considerando: um conjunto de operações sem interferência, leitura/escrita, redução da seção crítica e alinhamento de `locks`.

Padrão de Compartilhamento de Fila

O padrão de compartilhamento de fila tem por objetivo tratar o problema da execução concorrente de *threads* e processos em uma fila compartilhada. A implementação de algoritmos paralelos requer uma fila que é compartilhada entre processos ou *threads*. Uma das situações mais comuns é a necessidade de um fila de tarefas implementada no padrão Mestre/Escravo.

Para não haver erros na programação é importante prover uma abstração bem clara e tornar a verificação da fila compartilhada simples, identificando se esta está corretamente implementada. Em determinadas situações, podem ser crescentes as chances de processos e *threads* permanecerem bloqueados esperando acesso à fila e limitar a concorrência disponível. É importante destacar que, mantendo uma única fila para sistemas com hierarquia de memória mais complicada (por exemplo, máquinas NUMA e *clusters*) pode causar *overhead* na comunicação, no entanto, este problema é resolvido usando filas múltiplas ou distribuídas.

Na implementação do projeto de uma fila compartilhada, segue-se os mesmos requisitos de projeto de padrão de compartilhamento de dados [1]. Inicialmente é necessário prover a abstração do tipo de dados, definindo os valores que esta fila pode receber e qual é o conjunto de operações. Em seguida, deve-se considerar o protocolo de controle de concorrência, começando da maneira mais simples, um em um tempo de execução, para então, aplicar os refinamentos direcionando esta metodologia neste padrão.

Padrão de vetor Distribuído

O padrão de vetor distribuído é usado frequentemente na computação científica para dividir um determinado vetor de maneira a processá-lo em paralelo entre processos e *threads*. Na maioria da vezes, necessita-se de vetores com grandes proporções para calcular equações diferenciais neste cenário [19].

A implementação deste padrão se torna interessante quando o algoritmo utiliza o padrão de decomposição geométrica e o programa pode ser estruturado com o padrão SPMD. Não é necessário especificar a distribuição do array para cada plataforma, entretanto, é importante gerenciar a hierarquia de memória. Devido a isso, em máquinas NUMA os programas MPI podem ser melhores do que algoritmos similares que implementam *multithread* para manter páginas de memória próximos aos processadores que irão trabalhar com este *array*. Questões como balanceamento de carga, gerenciamento de memória (pode trazer bom desempenho quando feio o uso correto da hierarquia de memória) e abstração (clareza em como os vetores estão divididos entre os processo ou *threads* e mapeados) podem resultar em uma implementação com bons resultados, aproveitando bem os recursos disponíveis.

A abordagem para este padrão é particionar o vetores em blocos e mapeá-los em processos ou *threads*. Cada processo ou *thread* tem a mesma quantidade de trabalho e todos as *threads* ou processos deixam compartilhado um único endereço. Assim, cada bloco do processo ou *thread* armazena em um vetor local, do mesmo modo, para acessar os elementos do vetor distribuído, utiliza-se os índices no vetor local.

3. INTERFACES DE PROGRAMAÇÃO

Interfaces de programação são utilizadas para abstrair linhas de código, simplificando o desenvolvimento para uma ampla quantidade de aplicações. Este capítulo realiza um apanhado geral sobre interfaces tais com: bibliotecas, *Application Programming Interface* (API), *framework* e *Domain Specific Language* (DSL).

3.1 Biblioteca de Programação

Bibliotecas de programação são uma boa alternativa para reutilização de código e redução do esforço na implementação de uma aplicação. O uso delas se torna interessante quando um código em uma implementação já existe, evitando que ele se repita várias vezes no mesmo código. Portanto, através de módulos pré-fabricados são disponibilizadas bibliotecas do programa (também chamados de modulo, classe ou biblioteca de código).

Desde que a computação começou, as bibliotecas têm sido criadas, contendo vários módulos reusáveis para diferentes propósitos [3]. A grande maioria dos programas desenvolvidos em qualquer linguagem de programação importam bibliotecas que fornecem suporte a implementações de baixo nível. Em geral, elas implementam sub rotinas de aritmética, entrada/saída e funções do SO, sendo abstraídas por uma função de programação em um programa de aplicação.

No contexto de programação paralela é comum encontrar bibliotecas para reduzir o esforço no desenvolvimento. Estas, implementam sub rotinas para exploração de paralelismo e funções do próprio SO. Isso incluí funções para Para demonstrar o funcionamento desta primit abstrair a troca de mensagens, criação de *threads* ou processos, sincronização, exclusão mútua, escalonamento e dentre outros [24]. Em resumo, uma biblioteca sempre está vinculada a uma determinada linguagem de programação, fornecendo funcionalidades para um domínio.

3.2 API (*Aplication Programming Interface*)

Uma API define blocos de construção reutilizáveis que permitem modular pequenas funcionalidades incorporadas em uma aplicação de usuário final [25]. Neste contexto, fornecem uma abstração para um problema e especificam como o usuário deve interagir com os componentes de *software* que implementam a solução do problema. Para tanto, os componentes são distribuídos em bibliotecas de programação, permitindo várias aplicações os utilizam.

Resumidamente, a API é uma interface bem definida que prove serviços específicos para outros *softwares*. Ela pode ser tanto pequena quanto uma única função ou envolver várias classes, métodos, tipos de dados e constantes. O desenvolvimento de uma API não é muito claro, seu objetivo é fornecer uma interface lógica enquanto também oculta os detalhes de implementação.

Embora os conceitos se relacionam entre biblioteca e API, estas abordagens se diferenciam. Uma API possui uma interface lógica, para a qual, uma ou mais bibliotecas implementam as sub rotinas. Em um código de aplicação que instancia APIs é possível que elas realizem a instanciação de outras para resolver um problema. Assim, a API está uma camada a cima de uma biblioteca.

3.3 *Framework*

Framework consiste de uma técnica de reutilização orientada a objeto [26]. Esta definição diverge entre alguns autores, entretanto, uma definição frequentemente utilizada define que "*framework* é

um projeto reutilizável de todo ou parte de um sistema que é representado por um conjunto de classes abstratas e a maneira como as instâncias interagem". Por outro lado, definem que "*framework* é um esqueleto de uma aplicação que pode ser customizada por um desenvolvedor de aplicação". Ambos os conceitos não se conflitam, pelo contrário, eles se complementam, pois o primeiro descreve a estrutura e o segundo descreve a proposta deles.

Tipicamente eles são implementados com uma linguagem orientada a objeto. Cada objeto no *framework* é descrito por uma classe abstrata. Uma classe abstrata é uma classe sem instâncias, sendo usada como uma superclasse. Ainda que, uma classe abstrata sem instâncias pode ser usada como *template* para criar subclasses ao invés de um *template* para criar objetos. Enfim, o *framework* descreve: os objetos, como eles (objetos) interagem, a interface de cada um deles, o fluxo de controle entre eles e as responsabilidades do sistema também são mapeadas nos objetos.

Com isso, *frameworks* tiram proveito dos conceitos da linguagem orientada a objeto (*i.e*, abstração de dados, polimorfismo e herança). Em um tipo de dado abstrato, uma classe abstrata pode representar uma interface por três, onde uma implementação geralmente podem mudar. Polimorfismo orientado a objeto permite o desenvolvedor misturar e combinar componentes, permitindo um objeto trocar suas colaborações em tempo de execução e possibilitar a criação de objetos genéricos que podem trabalhar com um intervalo grande de componentes. E, a Herança facilita a criação de um novo componente [26], [4].

A caracterização de um *framework* é dado pelo conceito de inversão de controle, pois o controle do comportamento das classes do *framework* é gerenciado por ele e não pelas classes da aplicação. Esta arquitetura permite que uma aplicação canônica processe em etapas para ser customizada por objetos manipuladores de evento invocados pelo mecanismo de expedição reativa ao *framework*. Quando os eventos ocorrem, o *framework* invoca métodos nos objetos manipuladores pré-registrados, realizando o processamento específico da aplicação dos eventos. Isso permite que o *framework* determine qual conjunto de métodos específicos da aplicação são invocados na resposta de um evento externo. Além desta característica, são fornecidos benefícios de modularidade, reusabilidade e extensibilidade.

3.4 Linguagem Específica de Domínio

DSL (*Domain Specific Language*) é uma linguagem de programação direcionada para um domínio específico [8]. As linguagens normalmente utilizadas são de propósito geral, como por exemplo C, C++ e Java. Uma DSL contém sintaxe e semântica no mesmo nível de abstração que o domínio do problema oferece. Um exemplo no contexto de programação paralela é fornecer uma linguagem para programação paralela para um determinado domínio. Neste sentido, a DSL é responsável pela abstração dos detalhes do domínio e implementação da linguagem.

O autor de [2] define a DSL como uma linguagem de programação de expressivas limitações focada em um domínio. Categorizando assim, também é possível ter três tipos de DSL:

- **DSL externa** é uma linguagem separada da linguagem principal da aplicação que trabalha com ela. Uma DSL externa possui uma sintaxe personalizada e também normalmente utiliza XML (*Extensible Markup Language*) como sintaxe. Além disso o texto geralmente é analisado por um código na aplicação do hospedeiro usando técnicas de análise de texto. Exemplo disso são: expressões regulares, SQL (*Structured Query Language*), Awk e configuração de arquivos XML.

- **DSL interna** é uma maneira de utilizar uma linguagem de propósito geral. Um texto neste tipo de DSL é validado pela linguagem de propósito geral, mas somente utiliza alguns recursos

da linguagem em um estilo particular para lidar com um aspecto pequeno de um sistema global. O resultado deve ser uma linguagem personalizada ao invés de uma linguagem de hospedeiro.

- **linguagem de bancada** é uma IDE (*Integrated Development Environment*) especializada para criar DSL. No entanto, uma linguagem de bancada não serve somente para determinar a estrutura de uma DSL, mas também personalizar o ambiente de edição para as pessoas escreverem o texto da DSL.

A arquitetura básica para a implementação de uma DSL é composta por um processo de análise e geração de código, como ilustrado na Figura 3.1. O texto na arquitetura refere-se ao programa que é descrito em uma linguagem. O analisador lê o texto e gera um modelo semântico (um modelo de objeto que combina dados e processamento), para que o gerador, a partir de uma determinada técnica de geração de código crie o código alvo para ser interpretado computacionalmente.

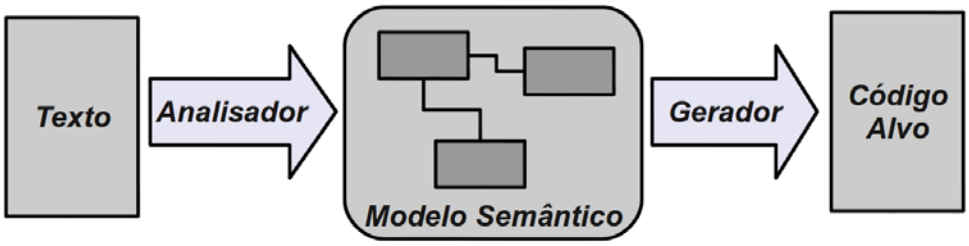


Figura 3.1: Arquitetura básica de uma DSL. Adaptado de [2]

Embora existam diferentes tipos de DSL, a arquitetura permanece a mesma para ambos os tipos. A diferença entre DSL interna e externa está na etapa do analisador (tanto na análise quanto como ela é feita). Em uma DSL externa existe uma separação clara entre o texto, o analisador e o modelo semântico. O texto é escrito em uma linguagem separada, o analisador lê este texto e preenche o modelo semântico. Porém, na interna é normal que estas coisas se misturem, pois é possível ter uma chamada explícita de objetos, sendo que o objetivo é fornecer uma interface para atuar como uma linguagem. Desta forma, o texto executa através da invocação de métodos em uma construção de expressões que preenche o modelo semântico. Em resumo, na DSL interna a análise do texto é realizado pela combinação de análise de linguagem hospedeira e construção de expressões.

3.5 Conclusão

Este capítulo apresentou um estudo sobre as principais interfaces de programação paralela utilizadas para abstrair detalhes de programação. Nenhuma das interfaces foram descartadas na implementação de uma abordagem de exploração de paralelismo, porém, não encontrou-se um interface de programação paralela atualmente disponibilizada e caracterizada como uma DSL. Isso motivou o estudo sobre a implementação e seus benefícios, pois, as demais são frequentemente encontradas neste contexto.

Além do mais, com uma DSL é possível obter flexibilidade em nível de usuário e de implementação, podendo modelar o comportamento das características de paralelismo em um alto nível de abstração, utilizando bibliotecas, APIs ou *frameworks*. Também, usando uma interface amigável que envolve fatores como geração e análise de código é possível introduzir o programador a trabalhar com uma determinada abordagem, direcionando o desenvolvimento de *software*.

4. TRABALHOS RELACIONADOS

A Computação de Alto Desempenho está em busca de novas alternativas para tornar a programação paralela algo comum aos desenvolvedores de *software*. Arquiteturas paralelas não são facilmente exploradas. Pesquisadores têm se preocupado com isso há muito tempo, dedicando seus esforços em fornecer novas soluções e metodologias, com o propósito de abstrair a complexidade em lidar com diferentes tipos de arquiteturas. O primeiro passo foi a exploração de paralelismo através de interfaces de programação de baixo nível como: ***Pthreads*** [23], **OpenMP** (*Open Multi-Processing*) [27], o **TBB** (*Threading Bulding Blocks*) [28], **Cilk** [29] e **SWARM** (*SoftWare and Algorithms for Running on Multi-core*) [30]. Mais tarde, juntamente com as linguagens de programação orientadas a objetos, passou-se a criar os *frameworks*, os mais conhecidos são: **Intel Parallel Studio** [31], **Galois** [32] e o **FastFlow** [33]. Assim, neste capítulo, caracterizou-se as interfaces de programação para exploração de paralelismo de baixo nível na Seção 4.1 e os *frameworks* na Seção 4.2.

4.1 Interfaces de Programação Paralela

Esta seção apresenta as principais interfaces de programação paralela para arquiteturas *multi-core*. Neste contexto estão inseridas bibliotecas, APIs e linguagens, as quais tem por finalidade fornecer uma camada de abstração de desenvolvimento de aplicações paralelas. Existem várias destas tecnologias representando elas. Somente as mais populares e conhecidas serão descritas nas subseções a seguir.

4.1.1 Pthread

Quando se trata de programação paralela no cenário de memória compartilhada, a biblioteca pthread é uma das que lida com paralelismo de mais baixo nível. Isso porque ao programar é necessário que o programador entenda os conceitos de *threads* e do SO. Assim como em sistemas operacionais multitarefa é possível realizar mais de uma coisa concorrentemente executando mais de um processo, um processo pode também fazer o mesmo executando mais de uma *thread*. Cada *thread* é vista como um fluxo de controle que pode executar suas instruções independentemente, sendo que elas podem ser desempenhadas para executar de forma concorrente ou paralela [34].

O uso da biblioteca tende a proporcionar maior flexibilidade ao programador e resultar em soluções paralelas com maior desempenho. Nela são disponibilizadas funções que lidam com a criação, sincronização e escalonamento de *threads*. Cabe ao programador implementar estas funções para melhor explorar o paralelismo disponível [23]. A obtenção de soluções inteligentes dependerá de como o programador fará uso das funções da biblioteca, o uso indevido pode resultar em implementações com grande sobrecarga, exemplo disso é quando as *threads* compartilham uma mesma região de memória.

4.1.2 OpenMP

O objetivo principal do OpenMP é facilitar o desenvolvimento de *software* para memória compartilhada. Não se trata de uma linguagem de programação e sim de uma interface de programação de aplicações [27]. O OpenMP fornece uma variedade de diretivas de compilação, rotinas de bibliotecas e variáveis de ambiente. As diretivas podem ser usadas para estender linguagens sequenciais, (por exemplo, Fortran, C e C++) oferecendo construções: SPMD (*Single Program Multiple Data*), de tarefas, de trabalho compartilhado e para sincronização [19].

A sua popularidade atualmente se deve pela simplicidade em implementar o paralelismo de laços e também pela portabilidade oferecida. Além disso, é enfatizado a programação paralela estruturada, podendo ser utilizado em diferentes plataformas de memória compartilhada. O modelo de programação é baseado na cooperação de execução simultânea de *threads* em multiprocessadores ou *multi-core*. Para criar e destruir as *threads* o OpenMP utiliza o padrão *Fork-Join*. Assim, em um programa OpenMP, uma *thread* é inicializada e esta executa o programa sequencial até encontrar uma região paralela, somente então a operação *fork* é realizada implicitamente [16].

4.1.3 TBB

O TBB (*Threading Bulding Blocks*) é uma abordagem para expressar o paralelismo em C++. A biblioteca ajuda a melhorar o desempenho de aplicações em arquiteturas *multi-core*, sendo esta uma abstração de alto nível que é baseada no paralelismo de tarefas, abstraindo detalhes da plataforma e mecanismos de *threads* para fornecer maior escalabilidade e desempenho [28]. No entanto, o desenvolver é quem deve encontrar os pontos de paralelismo do algoritmo e o TBB é quem vai fazer o mapeamento das tarefas em *threads* fornecendo o balanceamento de carga.

Assim como a maioria das interfaces de programação paralela, o TBB também implementa o paralelismo de laços, mecanismos de sincronização, diretivas de escalonamento de *threads* e alguns padrões paralelos. Além disso, a TBB permite que outras interfaces que implementam *multithread* também sejam utilizadas, por exemplo, OpenMP. Desta forma, o programador pode adicionar diretivas OpenMP no código que instancia os mecanismos do TBB [35], [36]. O objetivo é tornar o TBB mais flexível possível ao desenvolver e deixá-lo optar pela melhor solução na sua aplicação.

4.1.4 Cilk

O Cilk é uma tecnologia e uma linguagem algorítmica de programação *multithread* criado no MIT (*Massachusetts Institute of Technology*), a qual é baseada na linguagem de programação C. A sua filosofia parte do princípio que um programador deve concentrar-se na estruturação do seu programa para expor o paralelismo e explorar a localidade, deixando que o sistema de execução seja responsável pelo escalonamento de uma determinada computação. O sistema de execução é responsável pelo balanceamento de carga e também pelo protocolo de comunicação. Diferente das outras interfaces de programação paralela, o sistema de execução do escalonador garante a eficiência e o desempenho dos algoritmos [29].

A linguagem Cilk é simples. Ela possui três palavras chave para indicar o paralelismo e a sincronização: `cilk` para declarar funções que podem ser transformadas em *threads*; `spawn` para chamar funções declaradas com `cilk` a fim de criar uma nova *thread* e `sync` é usado para forçar a sincronização, ou seja, o programador identifica pontos no programa aonde a execução continua somente quando a dependência for resolvida e só assim a próxima instrução é executada. O Cilk estende a semântica da linguagem C, então se o objetivo é executar o programa em um único processador, basta deletar estas palavras chave [37].

4.1.5 SWARM

SWARM (*SoftWare and Algorithms for Running on Multicore*) é uma biblioteca para programação paralela em ambientes de memória compartilhada (arquiteturas *multi-core*) [38]. Ela é construída usando o padrão POSIX *threads*, isso permite que os usuários utilizem primitivas já desenvolvidas ou primitivas de *threads*. Além disso, a biblioteca fornece construções para o paralelismo, controle de

restrições de *threads*, alocação e desalocação de memória compartilhada e primitivas de comunicação para sincronização, como replicação e *broadcast*.

O principal objetivo da SWARM é minimizar o esforço na modificação de códigos sequencias. Para isso, é importante que o programador tenha a capacidade de identificar rotinas de computação intensiva no programa, o trabalho poderá ser atribuído em cada núcleo usando um algoritmo eficiente para organizar a computação. As operações que podem ser computadas independentemente, a biblioteca fornece o paralelismo funcional (paralelismo de laços), no qual, cada *thread* resolve parte da computação em paralelo [30].

4.2 *Frameworks* de Programação Paralela

Diferente do cenário das ferramentas de interfaces de programação paralela, os *Frameworks* são caracterizados pela abstração e reutilização de soluções inteligentes (métodos e classes). Normalmente *frameworks* de programação paralela fornecem métodos/classes que implementam o comportamento do paralelismo, sendo, posteriormente, reutilizados em outros tipos de aplicações. Assim como, metodologias de desenvolvimento de *software* podem ser facilmente associadas para direcionar o desenvolvedor a obter soluções com alta produtividade. Nem todos estes conceitos são introduzidos nos *frameworks* utilizados em programação paralela para ambientes *multi-core*, no entanto, esta seção busca apresentá-los e caracterizá-los no que se refere a implementação e utilização.

4.2.1 Intel Parallel Studio

A indústria de processadores também está se preocupando com a exploração de paralelismo disponibilizado em suas arquiteturas *multi-core*. Exemplo disso, a Intel está comercializando um *framework* conhecido como **Intel Parallel Studio** (IPS) [31]. Este, disponibiliza uma interface intuitiva com depuração otimizada para mecanismos de sincronização a fim de prover maior agilidade na programação para linguagem C++. A ideia é direcionar o desenvolvedor através de um metodologia de projeto de *software*. A metodologia proposta é dividida em quatro fases de projeto: projetar, construir e depurar, verificar e ajustar.

Para cada uma das fases de projeto o *framework* fornece uma ferramenta. A primeira é o "**Parallel Advisor**", este possui funcionalidades de detecção de conflitos, como por exemplo, condições de corrida e mecanismos de bloqueios (efetua o bloqueio de uma região crítica). As otimizações no código são realizadas com a ferramenta "**Parallel Composer**", a qual dispõem de opções avançadas de compilador e bibliotecas, bem como o suporte ao paralelismo simples e complexo (bibliotecas *pre-threaded* e *thread-safe*) [31]. Na fase de verificação é usada a ferramenta "**Parallel Inspector**", esta tem a finalidade de encontrar problemas com relação a construção do programa (memória e *threads*) através de uma interface de depuração. Encerrando o ciclo do *framework*, o "**Parallel Amplifier**" ajuda a construir aplicativos de escala *multi-core* e *many-core* e garantir o desempenho do aplicativo identificando concorrência, *hotspots* (unidades do programa que mais consomem tempo) e analisando *locks* e *waits* (*threads* que ficam esperando para continuar a computação).

4.2.2 Galois

Um dos recentes trabalhos de pesquisa na comunidade científica é o *framework* **Galois** [32]. Sua proposta é o paralelismo de dados, cujo o foco é operar em códigos irregulares que são organizados em torno de estruturas de dados baseadas em ponteiros, como grafos e árvores. Os detalhes de paralelismo são resolvidos através da sua biblioteca de programação, a qual utiliza o padrão *Worklist*, onde um algoritmo interativo obtém trabalho através de uma lista de tarefas. O algoritmo chega ao final somente quando a lista de tarefas estiver vazia [39].

No Galois, o usuário expressa o algoritmo usando código sequencial Java, mas deve especificar os laços que podem ser paralelizados usando a interface Galois através do `foreach`. O código resultante terá a mesmo aspecto semântico de código sequencial, enquanto isso, permite o paralelismo eficiente. O *framework* também disponibiliza um conjunto de *benchmarks* com a finalidade de permitir que possam ser feitas avaliações e comparações de desempenho da aplicação [40].

4.2.3 FastFlow

Outro recente trabalho de pesquisa é o *framework* **FastFlow**. Seu objetivo é fornecer suporte ao desenvolvimento de aplicações de *streaming* utilizando mecanismos de sincronização não bloqueantes em plataformas *multi-core* [33]. FastFlow fornece uma biblioteca paralela em linguagem C++, sendo que esta implementa o padrão *pipeline* com filas para um único consumidor e produtor. Também são tratadas as questões de sincronização e fluxo de dados de um para muitos, muitos para um e muitos para muitos.

O sistema de execução do FastFlow suporta duas camadas com duas características: **exploração de paralelismo**, por exemplo, a criação, destruição e controle do ciclo de vida de diferentes fluxos de controle de memória compartilhada; **canais de comunicação um para um assíncronos**, suportando a sincronização de diferentes fluxos de controle. Estas características são implementadas em filas SPSC (*Single-Producer-Single-Consumer*) equipadas com operações `push` e `pop` não bloqueantes. Nesta abordagem, a operação `push` referente ao produtor sempre lê e escreve usando `pwrite` (ponteiro corrente), no entanto, a operação `push` referente ao consumidor somente utiliza o `pread` (ponteiro chefe) para leitura e escrita. Isso é difere das outras abordagens, onde produtor e consumidor acessam o mesmo ponteiro corrente e chefe causando inconsistências na memória [41].

4.3 Síntese

Neste breve estudo identifica-se o esforço da comunidade para facilitar o desenvolvimento de *software* paralelo e melhorar a exploração do paralelismo das arquiteturas. Existe uma diversidade de ferramentas para exploração de paralelismo. As interfaces de programação paralela (bibliotecas e API's) se aplicam no contexto de flexibilidade para obter soluções de alto desempenho. Os *Frameworks* não são tão flexíveis para a exploração de paralelismo, entretanto, têm por objetivo abstrair os detalhes de programação paralela e são frequentemente implementados em linguagens orientadas a objetos, permitindo a reutilização de soluções paralelas anteriormente construídas. O desafio inicial deste trabalho de pesquisa é aprender com estas abordagens e propor uma maneira de obter as características de flexibilidade, facilidade e alto desempenho em uma única solução através de uma DSL.

Nota-se que, DSLs não são encontradas no cenário de interfaces de programação paralela. Isso se torna um diferencial com relação a proposta de pesquisa deste trabalho, onde o objetivo é prover uma interface orientada a padrões paralelos para um domínio específico, o qual, refere-se a programação paralela em ambientes *multi-core*.

De modo a complementar o relacionamento entre os trabalhos elencados nesta pesquisa, tabulou-se algumas das principais características importantes na exploração de paralelismo em arquiteturas *multi-core*, ranqueando-as de 0 a 3 estrelas (★) e associando com a afinidade do trabalho, conforme é expressado na Tabela 4.1. Através desta representação, buscou-se diferenciar os trabalhos atualmente utilizados, identificando as possíveis contribuições neste cenário para a implementação da LED-PPOPP.

O paralelismo em laços é uma necessidade comum para obter vantagem sobre uma arquitetura *multi-core*, pois as computações que exigem alto poder de processamento geralmente estão em

Tabela 4.1: *Avaliação das características dos trabalhos.*

Características	TBB	OpenMP	Pthread	Cilk	SWARM	IPS	FastFlow	Galois
Paralelismo em laços	★ ★	★ ★ ★			★ ★	★	★	★
Paralelismo em tarefas	★	★	★ ★ ★	★ ★	★	★	★	★ ★
Paralelismo em Fluxo (*pipeline*)	★					★	★ ★ ★	
Paralelismo incremental		★ ★ ★		★ ★	★			
Criação implícita do paralelismo	★	★ ★		★	★	★		
Programação orientada a padrões paralelos								

torno deles. A grande maioria dos trabalhos implementam diretivas para paralelizar os problemas de laços `for`. Naturalmente, as interfaces de programação paralela de mais baixo nível são baseadas no paralelismo de tarefas, o que torna mais difícil resolver este tipo de problema em paralelo.

Alguns trabalhos implementam a técnica de paralelismo em fluxo, permitindo que as computações sejam paralelizadas através de estágios em um *pipeline*. Enquanto isso, outros trabalhos focam no paralelismo incremental, evitando que grandes alterações sejam realizadas em um programa sequencial. Evidentemente, as interfaces que implementam abstração de alto nível proporcionam o paralelismo mais implícito, ocultando os detalhes de programação paralela.

Padrões paralelos são soluções especializadas para estruturar e modelar algoritmos que obtém vantagem sobre as arquiteturas paralelas. Observa-se que nenhum dos trabalhos implementa uma interface introduzindo este conceito. Em vista disso, está se propondo uma DSL que forneça todas as características elencadas através de uma interface que introduza a abordagem de Programação Paralela Orientada a Padrões Paralelos (PPOPP), sendo elaborada e apresentada na Seção 5.1.

5. LINGUAGEM ESPECÍFICA DE DOMÍNIO DE PROGRAMAÇÃO PARALELA ORIENTADA A PADRÕES PARALELOS (LED-PPOPP)

A LED-PPOPP é uma Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos. Primeiramente, realizou-se um estudo de caso implementando o padrão Mestre/Escravo baseado no modelo PPOPP, representado na Figura 5.2 e discutido na Seção 5.1. Do mesmo modo, suporta o desenvolvimento de programas com a linguagem C e exploração de paralelismo em arquiteturas *multi-core*.

Esta, por sua vez, dispõe de uma interface de núcleo e subnúcleo do padrão Mestre/Escravo, onde mecanismos de sincronização e proteção de dados compartilhados também são fornecidos. A interface (descrita na Seção 5.2) é reconhecida através de um compilador da linguagem, o qual é responsável pela análise e geração de código paralelo (descrito na Seção 5.4). O compilador é o principal elemento que compõe o projeto de algoritmos paralelos para o ciclo de desenvolvimento ilustrado na Figura 5.1.

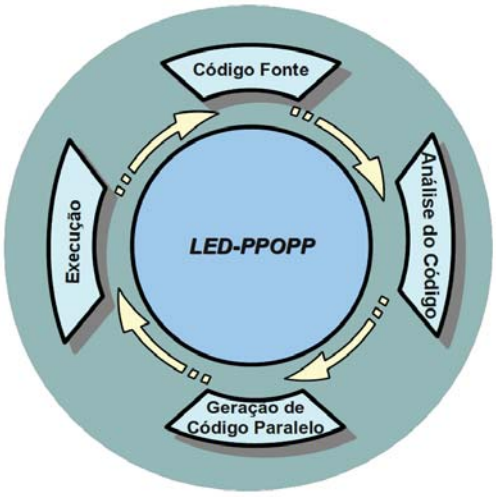


Figura 5.1: Ciclo de desenvolvimento de *software* paralelos com LED-PPOPP.

Caracterizando-se em um ciclo de desenvolvimento de *software* é possível identificar todo o processo que envolve a criação de soluções paralelas a partir da LED-PPOPP. Assim, classificou-se em quatro fases de projeto, verificando separadamente os papeis de cada uma delas neste ciclo:

- a fase de **Código Fonte** refere-se ao ambiente de desenvolvimento, onde são utilizados os recursos da linguagem LED-PPOPP para criar uma solução paralela;
- na **Análise do Código** é utilizado o compilador da linguagem, no qual é realizado a análise sintática e semântica do código da solução.
- **Geração de Código Paralelo** não está explícito ao programador, pois esta fase faz parte do processo de compilação e é realizada automaticamente com base no modelo semântico;
- **Execução** é a fase onde o programador executa a aplicação e realiza testes para avaliar o funcionamento e o desempenho.

De acordo com a Figura 5.1, o desenvolvimento de *software* paralelo inicia com a fase de código fonte. Após concluir a escrita da solução, a fase de análise do código é inicializada com a intervenção

do compilador da linguagem. Se houver algum erro, o compilador efetua um bloqueio, impedindo que a programação avance para a próxima fase, caso contrário, se não foram constatados erros, avança-se para a fase seguinte. Então, acontece a geração automática do código paralelo, a qual faz parte do processo de compilação. Com o código binário gerado, o desenvolvedor pode testar (executar) a aplicação. Se esta não apresentar resultados satisfatórios, novas otimizações no código poderão ser efetuadas, iniciando-se novamente o ciclo de desenvolvimento.

5.1 Programação Paralela Orientada a Padrões Paralelos

Programação Paralela Orientada a Padrões Paralelos (PPOPP) é uma abordagem que se baseia em soluções inteligentes para exploração de paralelismo. O objetivo é induzir o programador a desenvolver *software* com padrões paralelos. As aplicações que são paralelizadas utilizando este tipo de abordagem podem proporcionar alta produtividade, pois os padrões derivam a partir de aplicações que já foram exploradas no passado.

Como se trata de uma abordagem baseada em padrões conceitualmente definidos, esta pode ser facilmente entendida por desenvolvedores inexperientes e profissionais de diferentes áreas. A implementação de uma interface ou linguagem de PPOPP é uma maneira de direcionar o desenvolvimento de *software* nesta linha de raciocínio. A vantagem está no desenvolvimento, pois o programador ao projetar sua aplicação pode focar na resolução do problema e reduzir seus esforços na paralelização. Consequentemente, não é necessário conhecer as bibliotecas de programação paralela para explorar o paralelismo das arquiteturas.

A proposta é trabalhar com núcleos de padrões paralelos, definindo-os através de uma função da linguagem e nela são fornecidos blocos correspondentes ao padrão paralelo da função. Como é de conhecimento, diferentes tipos de computações podem estar contidas em um programa, acredita-se que nem sempre é possível resolvê-las com o mesmo padrão paralelo e obter um bom desempenho ao mesmo tempo. Para isso, está se propondo um modelo com subnúcleos que podem implementar uma diversidade de padrões paralelos e também combiná-los entre si. Este modelo de programação é ilustrado na Figura 5.2, o qual servirá de base para a construção da LED-PPOPP.

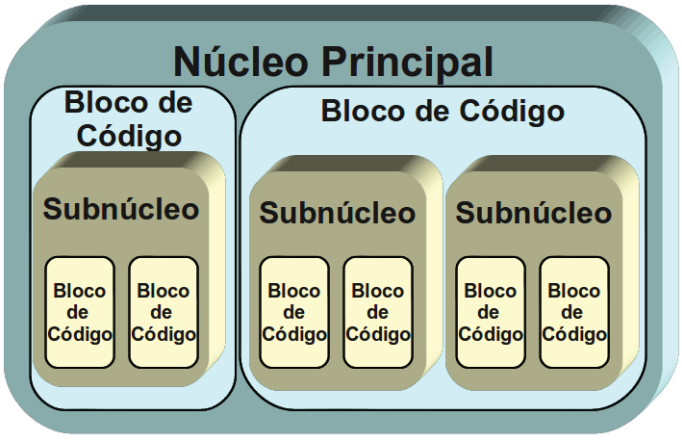


Figura 5.2: Proposta de modelo de programação com PPOPP.

Neste modelo, o núcleo principal pode suportar diferentes padrões, entretanto, apenas um deles é declarado uma única vez em todo programa, pois trata-se da função principal desta abordagem. Em uma visão global, a aplicação é paralelizada com blocos de código para efetuar a modelagem da computação. Os subnúcleos (correspondem a um determinado padrão paralelo disponibilizado por uma função da linguagem), se necessários, podem paralelizar uma computação declarada em

um bloco de código do núcleo principal. O objetivo geral é fornecer uma abordagem genérica de PPOPP, sendo esta, composta de vários padrões que exploram o paralelismo em diferentes tipos arquiteturais.

5.2 Interface de Programação da Linguagem

Em linguagens específicas de domínio uma interface nada mais é do que uma abstração de linguagem expressada em um texto com uma gramática mais próxima da realidade humana [2]. De tal forma, a interface foi definida utilizando palavras do vocabulário inglês (linguagem padrão da computação), atribuindo-se nomes aos núcleos (funções da linguagem), aos blocos de código (blocos pertencentes as funções da linguagem) e para as primitivas. A Tabela 5.1 descreve a interface de programação da LED-PPOPP.

Tabela 5.1: *Interface da LED-PPOPP*

Interface	**Descrição da Interface**
Funções "$"	
$mainMasterSlavePattern($int\ num_threads$){}	Núcleo principal que implementa o padrão *Master/Slave*
$MasterSlavePattern($int\ num_threads$){}	Subnúcleo que implementa o padrão *Master/Slave*
Blocos de código "@"	
@Master_{}	Bloco que implementa o Mestre no núcleo principal
@Slave_{}	Bloco que implementa o Escravo do núcleo principal
@Master{}	Bloco que implementa o Mestre do subnúcleo
@Slave{}	Bloco que implementa o Escravo do subnúcleo
Primitivas "_&"	
&SynchronizeSlaves;	Primitiva para sincronizar os blocos *Slave* do núcleo principal
_&SynchronizeSlaves;	Primitiva para sincronizar os blocos *Slave* do subnúcleo
_&ProtecData($tipo$, $nome\_var$){}	Primitiva global de proteção de dados compartilhados

A interface da LED-PPOPP foi projetada pensando em uma possível ampliação de funcionalidades seguindo o princípio de núcleos de padrões paralelos, ou seja, futuramente novos padrões poderão ser implementados sem perder o formato das declarações. Assim, definiu-se que: os núcleos de um padrão devem iniciar com o caractere "$", os blocos são inicializados com o caractere "@" e as primitivas com "_&". Para diferenciar o núcleo principal do subnúcleo, as funções de núcleo principal iniciam a construção do nome com `$main`. Os blocos pertencentes ao núcleo principal devem conter o caractere "_" no final do nome, bem como as primitivas pertencentes a ele.

O reconhecimento da interface é realizado através do compilador da linguagem (compilador PPOPP descrito na Seção 5.4) que requer a instanciação da biblioteca `ppoppLinux.h` no inicio do *script* de texto (código fonte). Como a LED-PPOPP esta associada a linguagem C, esta biblioteca possui suporte a todas bibliotecas desta linguagem, fazendo-se necessário apenas a declaração dela. Um exemplo de código é descrito no Algoritmo 5.1, apresentando um esqueleto do núcleo principal com o processo mestre imprimindo uma mensagem na tela logo após as duas *threads* escravas.

Algoritmo 5.1: Exemplo de código LED-PPOPP

```
1  #include<ppoppLinux.h>
2  $mainMasterSlavePattern(3){
3      @Master_{
4          _&SynchronizeSlaves_;
5          printf("Mestre terminou");
6      }
7      @Slave_{
8          printf("Escravo terminou");
9      }
10 }
```

Esta seção teve como objetivo apresentar a interface e entender como ela é expressada em um *script* de texto, exemplificado através do Algoritmo 5.1. Para melhor esclarecer o funcionamento da interface da LED-PPOPP, as próximas subseções descreverão a Estrutura da Linguagem (Seção 5.2.1), como é a Organização dos Dados (Seção 5.2.2) e como ocorre a Exploração do Paralelismo (Seção 5.2.4).

5.2.1 Estrutura da Linguagem

Para toda linguagem existe uma maneira de estruturar os algoritmos. Como descrito anteriormente, a LED-PPOPP baseia-se na abordagem do modelo de PPOPP para o desenvolvimento de aplicações paralelas. A representação da estrutura de combinações possíveis e mínimas é apresentado na Figura 5.3, através de um modelo estrutural em formato de árvore.

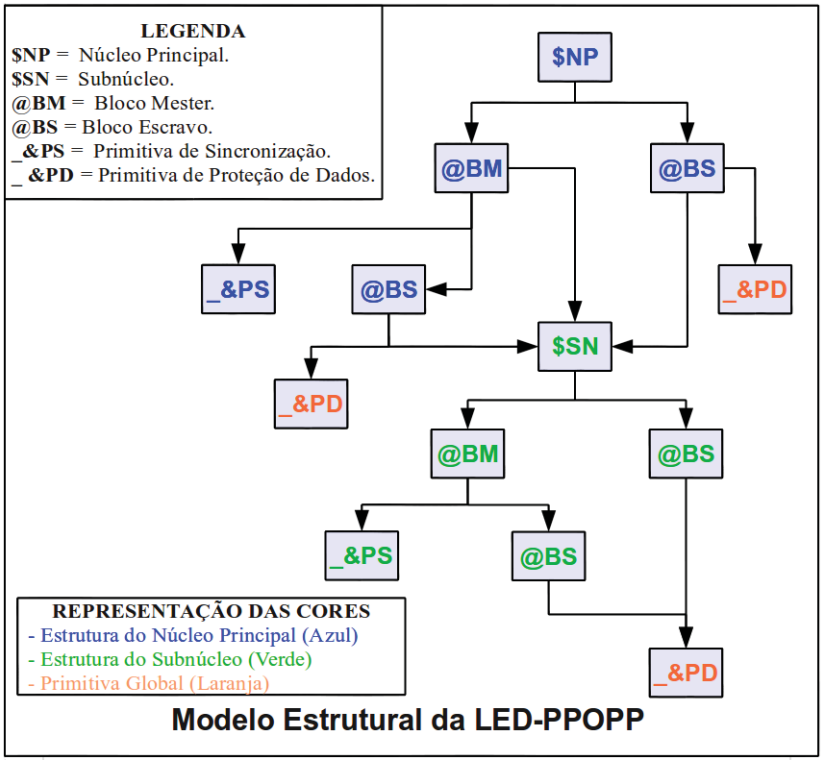


Figura 5.3: Modelo estrutural da LED-PPOPP.

Quando declarado um núcleo principal é necessário no mínimo um bloco **Master** e um **Slave**. Os blocos **Slave** podem ser declarados dentro ou fora do bloco **Master** tantos quantos são necessários. A mesma regra se aplica para construções de subnúcleos, no entanto, o núcleo principal deve ser declarado uma vez no programa, pois trata-se da função principal da LED-PPOPP. No núcleo principal podem ser declarados subnúcleos, porém, subnúcleos não podem conter subnúcleos, pois eles são o último nível da representação estrutural. As primitivas de sincronização somente são necessárias quando existirem blocos **Slave** fora do bloco **Master**. E, a outra primitiva refere-se a proteção de dados, sendo necessária quando um determinado dado é compartilhado entre as tarefas escravas, garantindo que os dados não sofram inconsistências.

Neste breve relato, nota-se que a estrutura da linguagem é bastante flexível, permitindo diversos tipos de combinações de blocos e subnúcleos. Mas é importante ressaltar que quando declarado um núcleo (núcleo principal ou subnúcleo) o código deve ser especificado dentro dos blocos **Master** e **Slave** pertencentes a ele. Um exemplo de código usando uma construção de subnúcleo para resolver um problema em uma função global é apresentado no Algoritmo 5.2. Nela, também é possível observar como o código é expressado nas declarações.

Algoritmo 5.2: Exemplo de código LED-PPOPP usando subnúcleo

```
1  #include<ppoppLinux.h>
2  void ProcessaTrabalho(){
3      $MasterSlavePattern(3){
4          @Master{
5              _&SynchronizeSlaves;
6          }
7          @Slave{
8              int i;
9              for(i=0;i<5000; i++);
10         }
11     }
12 }
13 $mainMasterSlavePattern(3){
14     @Master_{
15         _&SynchronizeSlaves_;
16         ProcessaTrabalho();
17     }
18     @Slave_{
19         int i;
20         for(i=0; i<5000; i++);
21     }
22 }
```

Nesta seção foram demonstradas as maneiras de se estruturar um código com a LED-PPOPP. Esta foi a primeira etapa para entender a linguagem como um todo e o quanto ela pode ser flexível. O próximo passo é entender como os dados são vistos, interpretados e organizados. Para isso, a próxima seção (Seção 5.2.2) esclarecerá estes detalhes.

5.2.2 Organização dos Dados

A LED-PPOPP permite que diversas combinações sejam expressadas em um programa. O objetivo desta seção é simplificar o entendimento de como os dados devem ser vistos e como eles são tratados. Antes de tudo, é importante ter a percepção de que os dados declarados dentro do bloco de código são declarações utilizadas somente por ele (*i.e.*, variáveis). E, os dados que são declarado fora do núcleo principal são os dados globais (acessível para todas as construções da linguagem). Essa representação é esclarecida com a ajuda do Algoritmo 5.3.

Algoritmo 5.3: Organização dos dados na LED-PPOPP

```
1  #include<ppoppLinux.h>
2  <dados globais>
3  $mainMasterSlavePattern(3){
4      @Master_{
5          <dados do bloco Master>
6          @Slave_{
7              <dados do bloco Slave>
8          }
9          _&SynchronizeSlaves_;
10     }
11     @Slave_{
12         <dados do bloco Slave>
13     }
14 }
```

A medida que são utilizados os subnúcleos, mais complexa ficará a interpretação dos dados. Uma boa prática para reduzir esta complexidade é implementar funções específicas globalmente. Com isso, as construções de subnúcleo não poluem tanto o código do núcleo principal e simplifica a interpretação dos dados. Outro cuidado a ser tomado na interpretação dos dados está na declaração de blocos **Slaves** dentro do bloco **Master**. Como dito anteriormente, os dados são localmente usados em seus blocos, no entanto, os que estão contidos no bloco **Slave** são replicados, se este estiver no bloco **Master**, uma cópia também é mantida nele, o qual ajudará na resolução da computação. Isso implica que o programador deve tomar cuidado na manipulação das variáveis, pois estas não poderão ser iguais em ambos os blocos nesta ocasião. A próxima seção (Seção 5.2.4) descreve em mais detalhes o comportamento na exploração do paralelismo.

5.2.3 Proteção dos Dados

Em ambientes multi-core os dados estão acessíveis ou compartilhados através da memória, isso implica que os processadores podem acessar a mesma região de memória simultaneamente. Normalmente, nestes ambientes, o uso de mecanismos de bloqueio são necessários para evitar que duas threads/processos escrevam na mesma região de memória, assim, não ocorrem inconsistências nos dados. A LED-PPOPP fornece uma primitiva (**ProtectData**) para assegurar o acesso na memória.

O objetivo é fornecer uma primitiva com uma interface padrão para suportar diferentes mecanismos de sincronização, no caso, utilizou-se o *Mutex* que é implementado na biblioteca pthread. Uma demonstração do funcionamento desta primitiva foi ilustrado no Algoritmo 5.4. Este pseudo código nos apresenta uma simples estrutura com núcleo principal usando um bloco mestre e outro escravo, do qual, originarão duas *threads* que dividirão a carga de trabalho entre elas. Além disso, como o bloco escravo foi declarado fora do bloco mestre, necessitou-se aplicar a primitiva de sincronização.

Algoritmo 5.4: Exemplo de código LED-PPOPP usando a primitiva de proteção de dados

```
1  #include<ppoppLinux.h>
2  int total_pares;
3  $mainMasterSlavePattern(3){
4      @Master_{
5
6          _&SynchronizeSlaves_;
7      }
8      @Slave_{
9          int i;
10         for(i=0; i<5000; i++){
11             if (i%2 == 0){
12                 ProtecData(MUTEX, m_var){
13                     total_pares++;
14                 }
15             }
16         }
17     }
18 }
```

A primitiva **ProtecData** foi utilizada neste exemplo do Algoritmo 5.4 para evitar que a operação de soma sofra inconsistências, pois duas *threads* escravas estarão compartilhando a mesma variável (`total_pares`). Esta, por sua vez, é apenas uma interface facilitadora da linguagem LED-PPOPP, no caso, usando Mutex o comportamento dele não se altera nesta linguagem. Neste sentido, a variável declarada na primitiva é interpretada como sendo global do tipo *mutex*, possibilitando que esta seja utilizada pelas *threads* escravas.

5.2.4 Exploração do Paralelismo

Um dos maiores desafios na programação paralela é a exploração de paralelismo em um alto nível de abstração. A proposta de trabalhar com padrões paralelos surge como alternativa para fornecer programação guiada através de uma linguagem que os implementa. Nesta pesquisa, optou-se pela utilização do padrão Mestre/Escravo, pois ele é facilmente introduzido na maioria das aplicações e obtêm soluções paralelas com alto desempenho na maioria dos casos.

A interface da LED-PPOPP tem como objetivo fornecer flexibilidade ao programador no desenvolvimento de *software* sem restringir as construções de núcleos. Entretanto, ela não garante que o programador sempre obterá uma solução de alto desempenho. Para isso, é importante entender claramente como acontece a exploração do paralelismo desta linguagem e como se comporta o padrão Mestre/Escravo no cenário de arquiteturas *multi-core*. Através do modelo de execução da Figura 5.4 é possível identificar como isso é efetuado em um núcleo da linguagem.

Em um núcleo é necessário a especificação do número de *threads*, sendo que a primeira unidade sempre será o processo mestre e as restantes a quantidade de escravos para cada bloco **Slave** declarado no programa. Isso é simplificado na Figura 5.4, na qual tem-se um processo mestre (resultante do bloco **Master**) e três *threads* escravas (resultante do bloco **Slave**) que processam em paralelos uma determinada computação. O processo mestre na LED-PPOPP é responsável pela criação, sincronização das *threads* escravas e processar o resultado. Todo este procedimento é abstraído através da linguagem, no entanto, a comunicação deve ser controlada pelo programador. Esta comunicação

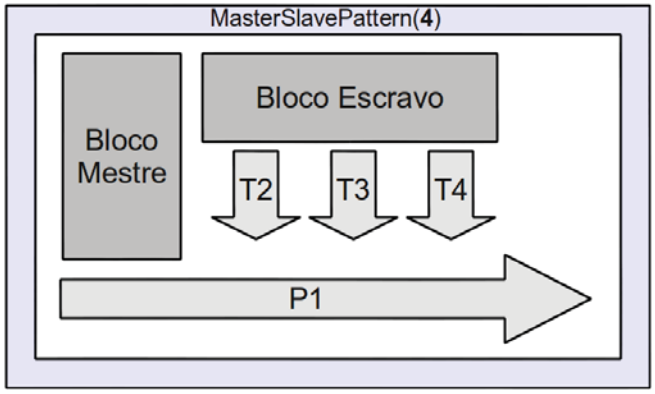


Figura 5.4: Modelo de execução da LED-PPOPP.

é realizada através da memória compartilhada (ex. um dado declarado globalmente e usado nas declarações de blocos).

Na organização dos dados (Seção 5.2.2) foi demonstrado que os dados do bloco **Slave** são replicados tantas vezes quantas tarefas escravas são criadas a partir dele. Se replicas são criadas, como é possível obter desempenho? Realmente, instruções que não podem ser divididas dificilmente apresentarão desempenho (*i.e.*, variáveis e funções). Neste contexto, o uso de subnúcleos nos blocos **Slave** também apresenta o mesmo problema, isso pode ser visto na Figura 5.5, na qual está ilustrando duas *threads* escravas (T2 e T3) processando a mesma computação. Além disso, o mestre também exemplifica o comportamento da declaração de subnúcleo, processando-o sequencialmente (P1) e deixando a cargo do subnúcleo a exploração do paralelismo.

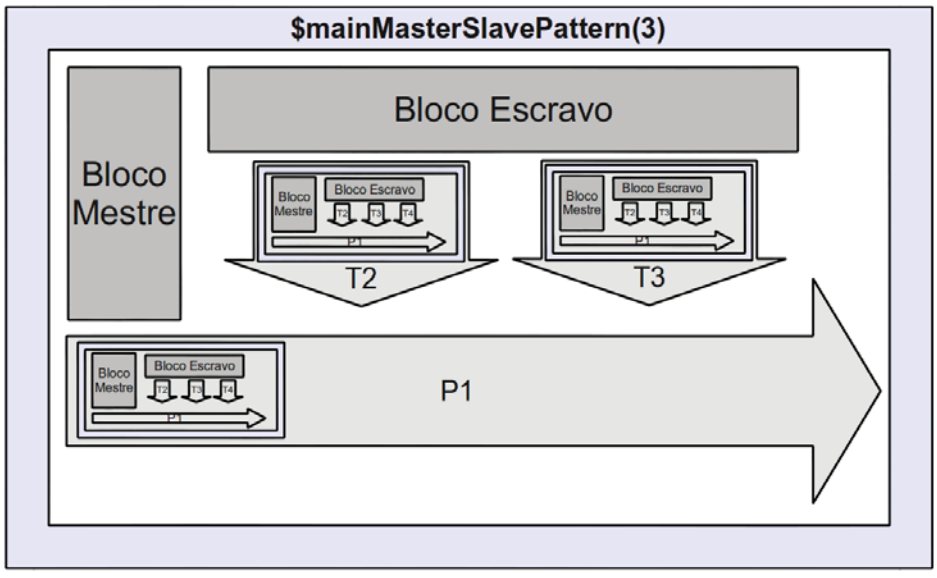


Figura 5.5: Exploração do paralelismo com subnúcleo na LED-PPOPP.

Como visto anteriormente, algumas construções não fornecem bom desempenho quando não permitem a sua divisão em tarefas, mas isso não impede que o programador implemente construções avançadas para obter maior desempenho. Uma alternativa é controlar o comportamento da computação que foi declarada no bloco **Slave**, fazendo com que cada *thread* escrava processe instruções diferentes ou parte dela em paralelo e aproveitando-se da maneira que o padrão Mestre/Escravo trabalha. Este tipo de implementação requer a comunicação entre as *threads* escravas e a utilização da primitiva de proteção de dados, a qual é fornecida pela LED-PPOPP.

A maioria das computações estão em torno de laços, muitas delas exigem muito do processador para resolver um determinado problema. A grande maioria das bibliotecas de programação paralela possuem suporte ao paralelismo de laços `for` (por exemplo, OpenMP e TBB), com técnicas de balanceamento de carga e escalonamento otimizado. No entanto, o paralelismo implementado por estas bibliotecas é realizado com o padrões *Fork/Join*, o que inviabiliza a utilização das mesmas na filosofia de programação da LED-PPOPP, fazendo-se necessário um estudo para encontrar a melhor maneira de explorar o paralelismo em laços sem perder a originalidade e o comportamento do padrão Mestre/Escravo.

Conforme pode ser visto na Figura 5.6, na LED-PPOPP o paralelismo de laços `for` é abstraído, basta declará-lo no bloco **Slave** para que a divisão do processamento aconteça. No entanto, somente construções simples de laços são possíveis, contendo apenas um parâmetro de inicialização, um de controle e um contador utilizando a mesma variável. E, a expressão de controle de parada pode ser criada apenas com os operadores lógicos $>$, $>=$, $<$, e $<=$, caso estas condições não forem satisfeitas, o compilador da linguagem (Seção 5.4) irá reportar um erro. Estas restrições também são encontradas nas bibliotecas que suportam o paralelismo de laço, isso porque estruturas complexas de laços não são facilmente paralelizadas.

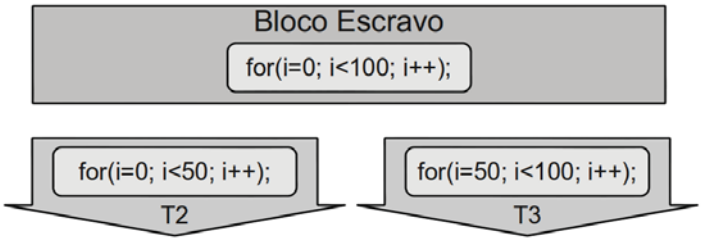


Figura 5.6: Paralelismo de laços na LED-PPOPP.

Esta seção apresentou diferentes formas de implementar o paralelismo e obter soluções paralelas que podem proporcionar alto desempenho através da LED-PPOPP. Demonstrou-se também que a linguagem não garante que qualquer construção paralela resulte em uma solução com alto desempenho, os bons resultados na implementação estarão diretamente ligados a utilização correta das funcionalidades e o quanto a aplicação é adaptável ao padrão Mestre/Escravo.

5.3 Cenário de Implementação

Um exemplo prático de implementação da LED-PPOPP está ilustrado na Figura 5.7. Nela, o algoritmo de multiplicação de matrizes é exemplificado em três cenários diferentes (Sequencial, LED-PPOPP e Pthread). O objetivo com estes cenários de implementação é demonstrar um ambiente real de programação, entendendo como um código é criado com esta linguagem específica de domínio e comparando as diferenças em relação a biblioteca Pthread.

Analisando os cenários, nota-se que para paralelizar o algoritmo de multiplicação de matrizes:

- a LED-PPOPP requer poucas modificações no código sequencial;
- a quantidade de código necessário utilizando LED-PPOPP é menor do que em Pthread;
- usando a LED-PPOPP o desenvolvedor não precisa saber como quebrar um problema, pois a linguagem se encarrega de fazer isso;
- a sincronização de threads é necessária em ambas as abordagens. No entanto, usando a LED-PPOPP isto é abstraído por meio da primitiva de sincronização de blocos escravos.

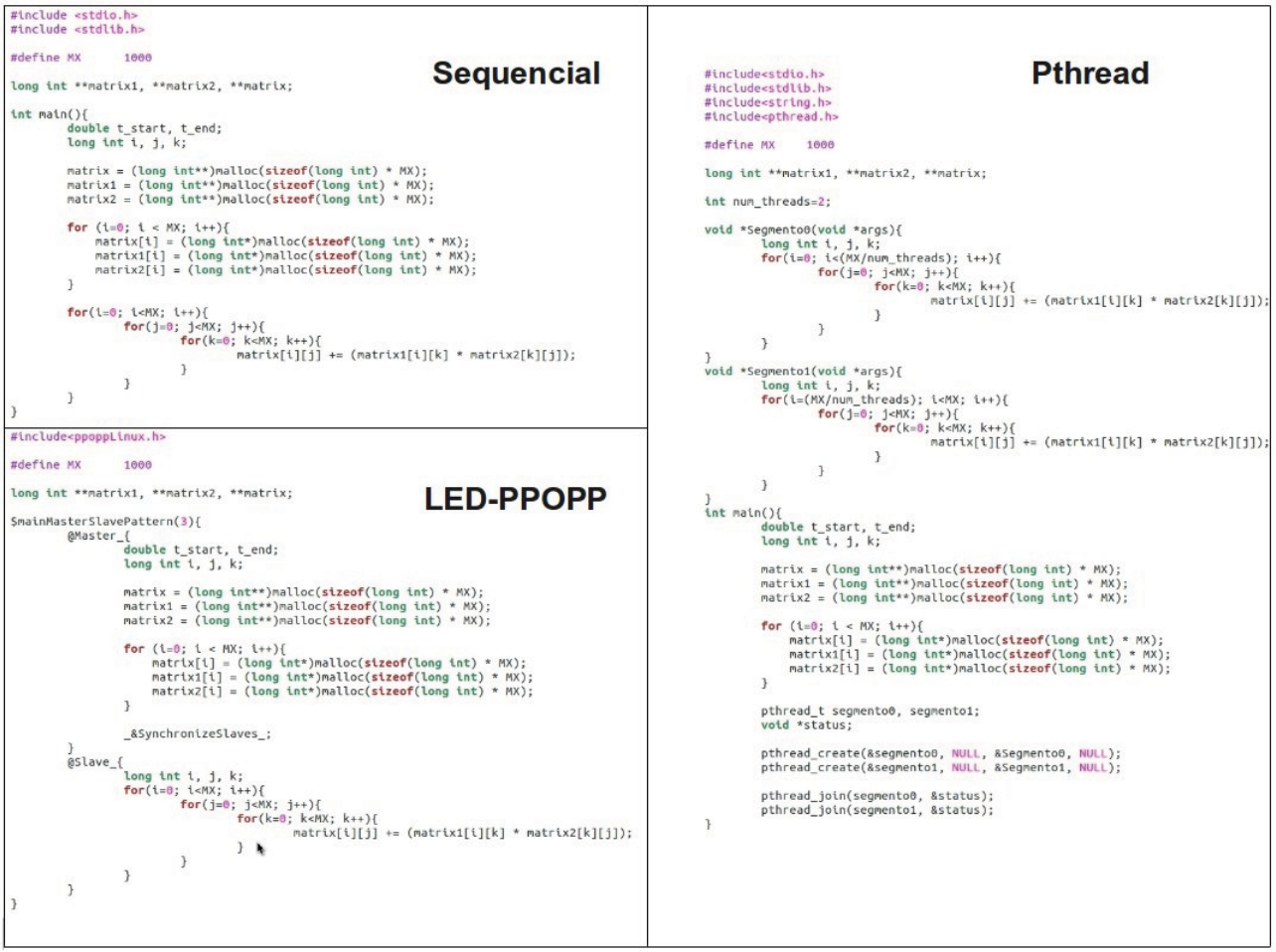


Figura 5.7: Comparação das implementações do algoritmo de multiplicação de matrizes.

5.4 Compilador

Um compilador tem por objetivo transformar um código fonte de uma representação em outra [42]. Não existe um formalismo para representar e determinar a construção, mas é possível classificá-lo em fases no processo de execução. Mais precisamente, um compilador deve conter uma fase de análise e uma de geração de código, as quais podem ser alimentadas por um tratador de erros e um gerenciador de tabelas e símbolos. Mantendo-se caracterizadas as duas principais fases, não existe um restrição que impeça de quebrá-las em um nível maior de detalhamento. A implementação é bastante particular do construtor do compilador, este pode tanto personalizar da sua maneira a analise e a geração, quanto usar ferramentas que auxiliam neste processo.

Nomeado como PPOPP é o compilador criado para a LED-PPOPP. Diferente dos tradicionais compiladores, este tem como objetivo reconhecer e gerar código paralelo orientado ao padrão Mestre/Escravo para arquiteturas *multi-core*. Igualmente, permitir que os algoritmos possam ser escritos com a sintaxe e semântica da linguagem C, sendo reconhecida com o auxilio do compilador GCC (GNU (*General Public License*)*Compiler Collection*) integrado ao compilador PPOPP, para o qual, a sua estrutura pode ser vista através da Figura 5.8. Desta forma, o programador será induzido desde o inicio a desenvolver algoritmos que exploram o paralelismo utilizando o padrão Mestre/Escravo.

O programa fonte deve ser escrito em um arquivo com extensão "*.c*" para que o compilador possa realizar a leitura inicializando o processo de compilação. Então, primeiramente acontece a análise de erros e inconsistências no código fonte relacionados a LED-PPOPP (detectados pelo PPOPP)

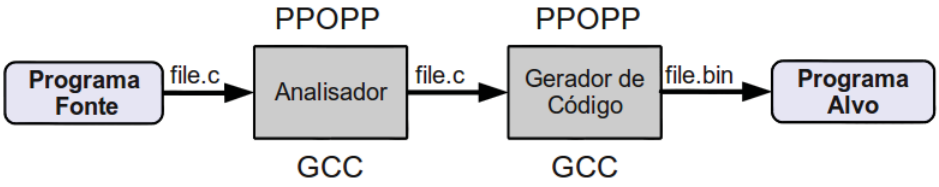


Figura 5.8: Estrutura do compilador PPOPP.

e a linguagem C (detectados pelo GCC). O tratador de erros retorna mensagens indicando a linha e o tipo de erro e, enquanto estes persistirem, a próxima fase de compilação não é processada. Portanto, na geração de código ocorre a transformação do código fonte especificado através da LED-PPOPP em um código paralelo orientado ao padrão Mestre/Escravo utilizando a biblioteca "`pthread.h`" em linguagem C. Em seguida, este código paralelo é transformado (utilizando o GCC) em um programa alvo (código binário), pronto para ser executado no ambiente em que foi gerado.

Atualmente o compilador PPOPP suporta a instalação em plataformas Linux 64/32 bits. Após a instalação, a compilação é realizada em um terminal através do comando **`ppopp <nome_codigo_fonte.c> <nome_codigo_binario>`**, onde o primeiro argumento do comando é o programa fonte com extensão "`.c`" e o segundo é um nome atribuído ao programa alvo, ou seja, o arquivo de saída do compilador.

5.5 Fatores Considerados

O projeto da LED-PPOPP ambiciona reduzir o esforço de programação paralela e induzir o programador a explorar o paralelismo das arquiteturas *multi-core*. A escolha deste tipo de arquitetura paralela se deve pela demanda que o cenário se encontra, sendo bastante comum em estações de trabalho e *desktops*, entretanto, a maior parte dos aplicativos que executam nelas não obtém vantagem sobre o paralelismo disponível. A perspectiva é que estas arquiteturas forneçam processadores com centenas de núcleos. Para isso, é importante que desenvolvedores de diferentes áreas seja induzidos e que as interfaces de programação paralelas abstraem o paralelismo de baixo nível, usando uma abordagem de alto nível a fim de que programadores inexperientes também consigam programar.

Centrando-se neste objetivo, a abordagem PPOPP proposta visa a implementação de vários padrões paralelos e a combinação entre eles. Entretanto, é preciso ter cautela, porque uma proposta desta magnitude é bastante complexa e custosa. Em virtude disso, implementou-se o padrão Mestre/Escravo, por ser o padrão paralelo mais genérico, simples e conhecido pela grande maioria dos desenvolvedores. Por outro lado, os relatos demonstram que a utilização deste padrão tende a fornecer aplicações com bom desempenho para uma boa parte dos problemas encontrados na computação.

Mesmo que a prioridade tenha sido diminuir o esforço por meio da LED-PPOPP, é preciso fornecer uma interface que obtenha vantagem sobre a arquitetura paralela sem grandes perdas de desempenho. Logo, questões de otimização de desempenho não foram implementadas, como por exemplo, balanceamento e escalonamento. As *threads* são simplesmente criadas e sincronizadas no contexto do padrão Mestre/Escravo e o SO se encarrega de escaloná-las entre os elementos de processamento.

6. PLANEJAMENTO E EXECUÇÃO DOS EXPERIMENTOS

Este capítulo apresenta o planejamento e a execução dos experimentos para validação da LED-PPOPP em um estudo de caso com o padrão Mestre/Escravo. O objetivo é demonstrar claramente como a metodologia de experimentação foi conduzida na medição do esforço de programação paralela e na medição do desempenho obtido na paralelização. Tratando-se de objetivos diferentes, estes foram categorizados em duas seções experimentais: a Seção 6.1 demonstra o processo experimental na medição do esforço e a Seção 6.2 sobre a medição do desempenho.

6.1 Experimento para Medição do Esforço de Programação Paralela

Neste trabalho de pesquisa, um dos objetivos é reduzir o esforço de programação através da LED-PPOPP. Experimentos são usados na área da computação quando existe o fator humano na operacionalização com *software*, sendo este, um método científico aplicado para avaliar os benefícios de determinada abordagem ou teoria relacionada ao *software*. Para isso, os experimentos são baseados em hipóteses estatísticas, tratando-se de uma suposição formulada, referindo-se a distribuição da probabilidade de uma ou mais populações. Em seguida, com os resultados da execução, realiza-se o teste de hipótese, validando-a ou rejeitando-a. A experimentação é caracterizada em um processo com quatro fases (definição, planejamento, execução e avaliação), das quais, esta seção irá apresentar o planejamento e a execução.

6.1.1 Medidas para Avaliar o Esforço

A métrica associada a primeira questão de pesquisa corresponde ao esforço medido pela relação do tempo gasto em minutos por cada participante na paralelização de um algoritmo com a LED-PPOPP e com Pthread. Para isso, será comparado as médias de tempo dos participantes na paralelização de uma aplicação para cada uma das abordagem.

A análise é realizada usando o teste estatístico de hipótese. Para este, efetua-se a suposição de uma hipótese nula ($H_0,\ AbordagemA = AbordagemB$) que será rejeitada ou não, estipulando-se um valor crítico (conjunto de valores que podem rejeitar a hipótese nula). Como neste experimento deseja-se obter uma confiabilidade de 95%, o nível de significância (probabilidade máxima para rejeitar a hipótese) adotado é de 5% (o mesmo que 0.05), ou seja, para que a hipótese seja rejeitada é necessário que o resultado do nível de significância seja menor que 0.05 (a literatura denominado este valor de p-`value`) [43].

6.1.2 Caracterização Formal da Hipótese

Esta seção tem por objetivo associar a hipótese informal **HP1** (formulada na Seção 1.3) com a métrica levantada, formulando uma hipótese formal que guiará a execução deste experimento. A tradução de uma hipótese informal é traduzida em indicadores numéricos para realizar a verificação estatística de sua validade. Assim, devido a natureza do teste estatístico, transformou-se a hipótese em uma hipótese nula (H_0), da seguinte forma:

1. **Hipótese Nula,** H_0: o esforço na paralelização (programação paralela) com Pthread é igual ao esforço utilizando LED-PPOPP.

   (a) Medidas: o esforço será avaliado pelo tempo gasto em minutos na paralelização de um algoritmo sequencial em cada abordagem (LED-PPOPP e Pthread):

i. $\mu T_{LED-PPOPms}$: representa a média do tempo gasto em minutos com a abordagem LED-PPOPP.

ii. $\mu T_{Pthread}$: representa a média do tempo gasto em minutos com a abordagem Pthread.

(b) H_0: $\mu T_{Pthread} = \mu T_{LED-PPOPms}$;

(c) Hipótese Alternativa, H_1: o esforço na paralelização com Pthread é maior do que utilizar a abordagem LED-PPOPP:

i. $\mu T_{Pthread} > \mu T_{LED-PPOPms}$;

(d) Hipótese Alternativa, H_2: o esforço na paralelização com LED-PPOPP é maior do que utilizar a abordagem Pthread:

i. $\mu T_{LED-PPOPms} > \mu T_{Pthread}$;

6.1.3 Seleção das Variáveis

Em um estudo experimental a causa ou variável independente é o previsor e o efeito ou variável dependente é o resultado. Esta terminologia em termos de trabalhos transversais, estatisticamente, pode usar uma ou mais variáveis para fazer previsões sobre outros sem a necessidade de implicar a casualidade [43]. Neste experimento, assumiu-se como variável dependente, o **Esforço** para paralelizar um algoritmo sequencial. Como variáveis independentes assumiu-se: o **Conhecimento** para realizar o experimento (variável de bloqueio) e as **Abordagens** utilizadas para paralelizar um algoritmo (LED-PPOPP e Pthread). A representação das variáveis é ilustrado na Figura 6.1.

Figura 6.1: Variáveis independentes e dependentes do estudo experimental.

A Tabela 6.1 sumariza a escala para cada uma das variáveis consideradas neste experimento.

Tabela 6.1: *Escala das variáveis.*

Variáveis	**Nome**	**Escala**
Dependente	Esforço	Razão
Independentes	Conhecimento	Nominal
	Abordagens	Nominal

6.1.4 Caracterização do Experimento

Para a condução deste experimento escolheu-se o ambiente de universidade. O objetivo é avaliar o esforço dos estudantes na programação paralela utilizando as abordagens LED-PPOPP e Pthread. Desta forma, esta seção demonstra o quão controlado é o experimento e quais são os indícios do ambiente em relação a representatividade dos resultados. Dentro deste contexto, caracterizam-se:

- processo: os participantes executaram o experimento em um ambiente controlado (laboratório);
- participantes: o experimento foi conduzido com mestrandos e doutorandos do Programa de Pós-Graduação em Ciência Computação (PPGCC) da Pontifícia Universidade do Rio Grande do Sul (PUCRS), os quais, como requisito deveriam estar cursando ou já ter cursado a disciplina de Programação Paralela;
- realidade: o problema estudado será um algoritmo conhecido no cenário de programação paralela, o qual será paralelizado pelo estudantes durante a disciplina de Programação Paralela do PPGCC. Esta escolha se deve pela aproximação do ambiente real sem que ocorra a intervenção do pesquisador;
- generalidade: o experimento é específico e só possui validade no escopo desta pesquisa.

6.1.5 Seleção dos Indivíduos

Esta seção apresenta os detalhes para seleção dos indivíduos que participarão do experimento. Como caracterizado anteriormente, escolheu-se a população acadêmica de nível de pós-graduação, pois é necessário que os participantes tenham conhecimento em linguagem C, conhecimentos básicos em modelagem de programas para ambientes *multi-core*, conhecimento básico em *linux* e uma experiência em desenvolvimento de *software* paralelo. O objetivo é que o experimento seja executado com programadores da área de processamento paralelo e distribuído.

Para garantir estes requisitos, aplicou-se um questionário de avaliação de conhecimentos dos participantes, podendo ser visto no Apêndice A.1. O critério utilizado para a seleção dos indivíduos foi através da conveniência, no caso, todos os que responderam o questionário eram convenientes a realizar o experimento, pois atendiam aos requisitos básicos. Apesar dos vários requisitos necessários, conseguiu-se selecionar 20 indivíduos aptos.

6.1.6 Critérios Observados para o Experimento

Os critérios observados para o projeto do experimento foram:

- aleatoriedade: atribuição do participante ao grupo que pertence (definição de dois grupos, os que usam primeiro Pthread e os que usam primeiro o método proposta);
- balanceamento: cada abordagem utilizou a mesma quantidade de participantes e com níveis de conhecimento balanceados;
- bloqueio: somente os indivíduos que atenderam aos requisitos participaram do experimento.

6.1.7 Tipo de Experimento e Unidades Experimentais

Nesta seção será indicado o tipo de experimento realizado e como foram montadas as unidades experimentais de acordo com os princípios escolhidos (aleatoriedade, balanceamento e bloqueio). Para isso, definiu-se a seguinte denotação:

- $\Delta_{LED-PPOPms}$: representa a abordagem de programação LED-PPOPP;
- $\Delta_{Pthread}$: representa a abordagem de programação Pthread;

O projeto apresentado busca investigar se $\Delta_{LED-PPOPms}$ possui o mesmo esforço que $\Delta_{Pthread}$. Para isso, o tipo de experimento utiliza um fator com dois tratamentos pareados. O fator, neste experimento, consiste no tempo que o participante levou para paralelizar um algoritmo e os tratamentos consistem nas abordagens LED-PPOPP e Pthread. Desta forma, os participantes desenvolveram a aplicação com as duas abordagens, sendo que a execução foi aleatória e balanceada.

Através do questionário de avaliação de conhecimento (Apêndice A.1), os indivíduos se autoavaliavam escolhendo uma das quatro opções oferecidas (zero, baixo, médio e alto). A codificação das opções foram traduzidas em unidades numéricas, calculando-se o peso da opção (PO) escolhida para cada questão respondida, utilizando a formula

$$PO = \left(\frac{PMX}{QOP}\right) \cdot NO,$$

onde PMX é o peso máximo definido (igual a 100), QOP a quantidade de opções pontuáveis (igual a 3 opção pontuáveis) e NO é o nível da opção que está representado na Tabela 6.2. Esta tabela apresenta também os pesos resultantes para cada uma das opções e utilizados posteriormente na classificação dos níveis de conhecimento dos indivíduos (Tabela 6.4).

Tabela 6.2: *Peso para as opções das questões.*

Opções de escolha			
Zero (NO_0)	Baixo (NO_1)	Médio (NO_2)	Alto (NO_3)
0	33	66	100

Tratando-se de um experimento com amostras pareadas é importante que a distribuição dos participantes esteja balanceada. Neste sentido, optou-se pela criação de dois grupos com níveis de conhecimento equilibrados. O processo iniciou-se classificando o conhecimento em razoável, bom, muito bom e ótimo, para os quais, foram determinados o peso máximo de classificação (PMX), utilizando a formula

$$PMX = \left(\left(\frac{TMA - TME}{TO}\right) \cdot OP\right) + TME,$$

onde o TMA (maior total do conhecimento) e TME (menor total do conhecimento) são obtidos através do somatório total dos pesos das questões. O TO (igual a 4) representa o total de opções para definir a classificação do indivíduo e a opção é representada por OP na Tabela 6.3. Com base no peso máximo de classificação (PMX), estipulou-se os intervalos para cada uma das opções de classificação. E, utilizando o critério de bloqueio (peso total de no mínimo 132 para o somatório das quatro primeiras questões) para determinar o peso mínimo aceitável. Por exemplo, para um indivíduo ter a classificação "Razoável", é necessário que o Tot (somatório total dos pesos das questões da Tabela 6.4) seja maior que 132 e menor ou igual a 257.

Tabela 6.3: *Classificação do conhecimento dos indivíduos.*

Opções de Classificação			
Razoável (OP_1)	Bom (OP_2)	Muito Bom (OP_3)	Ótimo (OP_4)
Tot>132 e Tot<=257	Tot>257 e Tot<=315	Tot>315 e Tot<=374	Tot>374 e Tot<=432

O resultado da classificação do nível de conhecimento é demonstrado na Tabela 6.4. A maior parte dos indivíduos que participaram do experimento possuíram conhecimento entre ótimo e bom. Do mesmo modo, identifica-se como é realizada a atribuição dos grupos, onde o nível de conhecimento é organizando de maneira decrescente e os indivíduos são escolhidos alternadamente, mantendo o balanceamento entre os grupos com a mesma quantidade de participantes.

Tabela 6.4: *Classificação dos indivíduos*

Participante	Q1	Q2	Q3	Q4	Q5	Q6	Total	Classificação	Grupo
1	100	100	66	100	66	0	432	Ótimo	1
2	100	100	66	66	100	0	432	Ótimo	2
3	100	100	66	66	66	0	398	Ótimo	1
4	100	100	66	66	66	0	398	Ótimo	2
5	100	100	66	66	33	0	365	Muito Bom	1
6	66	66	66	100	66	0	364	Muito Bom	2
7	66	100	66	66	66	0	364	Muito Bom	1
8	66	66	66	100	33	0	331	Muito Bom	2
9	66	66	66	66	66	0	330	Muito Bom	1
10	66	66	66	66	33	0	297	Bom	2
11	66	100	66	33	33	0	298	Bom	1
12	66	66	66	66	33	0	297	Bom	2
13	66	66	66	66	33	0	297	Bom	1
14	66	66	66	66	33	0	297	Bom	2
15	66	100	33	33	33	0	265	Bom	1
16	33	66	66	100	0	0	265	Bom	2
17	66	66	66	66	0	0	264	Bom	1
18	66	66	33	66	0	0	231	Razoável	2
19	66	33	33	66	33	0	231	Razoável	1
20	33	66	33	33	33	0	198	Razoável	2

A aleatoriedade no experimento e a eliminação do viés sobre uma determinada abordagem são garantidas através do modelo de execução do experimento ilustrado na Figura 6.2. Primeiramente, os participantes do Grupo 1 começaram o experimento paralelizando um algoritmo com $\Delta_{LED-PPOPms}$ e em um outro dia utilizando a abordagem $\Delta_{Pthread}$. No Grupo 2, as abordagens foram executadas inversamente.

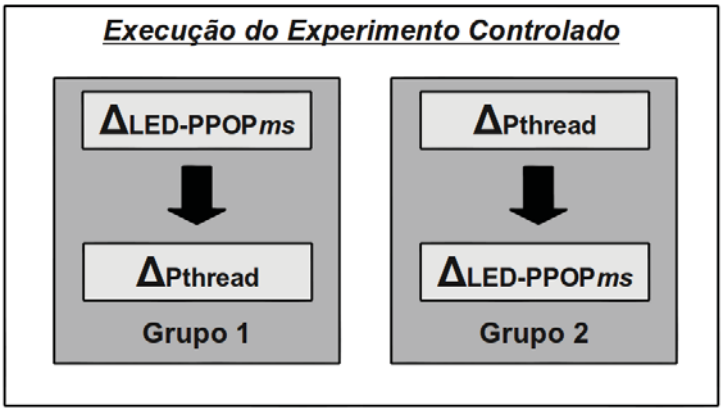


Figura 6.2: Modelo de execução do experimento controlado.

O teste de hipótese, em um contexto de um fator e dois tratamentos pareados, a literatura [43] sugere o teste de significância denominado de **Teste T pareado**, caso for realizado um teste paramétrico, ou **Wilcoxon**, caso o teste não seja paramétrico. O teste a ser aplicado será definido após a análise da normalidade (teste de Shapiro-Wilk e Kolmogorov-Smimov) e a variância dos dados obtidos pela execução do experimento (Teste de Levene).

6.1.8 Instrumentação

Os instrumentos utilizados para realizar este experimento são:

- ambiente: na execução do experimento, cada participante utilizou uma máquina com SO Ubuntu Linux, no qual estavam instalados a LED-PPOPP e a biblioteca *Pthreads*. Para realizar o teste de hipótese, utilizou-se a ferramenta SPSS (*Statistical Package for the Social Sciences*) versão 14.0;

- guias: os participantes puderam usar somente o material eletrônico fornecido: manual de utilização da biblioteca Pthread e da LED-PPOPms (ambos oferecidos apenas em suas execuções), podendo ser visto nos Apêndices A.3 e A.4. Além disso, um código do algoritmo sequencial (Apêndice A.5) é disponibilizado;

- métricas: através de um formulário impresso (Apêndice A.2), os participantes especificaram o horário de inicio e término, relatando também, o desempenho da aplicação paralelizada e a descrição do ambiente de execução.

6.1.9 Validação do Experimento

Em busca de uma definição formal do experimento para com a validade durante a execução, esta seção busca apresentar diferentes tipos de validade, com o propósito de facilitar a replicação e dar maior credibilidade aos resultados obtidos. Assim, a **validade interna** do experimento será avaliada pelos seguintes critérios:

- histórico: a data do experimento foi definida no período em que os participantes não sofreram influências externas;

- maturação: para que o interesse na condução do experimento se mantivesse, os participantes foram incentivados de forma positiva;

- seleção: foi usado o critério de nivelamento dos participantes, formando-se grupos com níveis de conhecimento similares (Seção 6.1.7);

- Difusão: durante a execução do experimento foi observado e evitado que os participantes se interagissem e influenciassem no resultado da pesquisa, restringindo-se o uso de celulares, internet e qualquer outro recurso que não seja a material eletrônico disponibilizado.

Para a **validade externa** do experimento, adotou-se um critério de escolha dos participantes. Estes deveriam ter um perfil adequado, tendo conhecimento prévio em: linguagem C, modelagem de programas para ambientes *multi-core*, Linux e experiência em desenvolvimento de *software* paralelo.

A **validade de construção** é caracterizada pela avaliação de:

- explicação pré-operacional: uma explicação operacional do experimento foi realizada, esclarecendo a condução do experimento e apresentando as abordagens LED-PPOPP e Pthread;

- hipóteses: o experimento é realizado com humanos, isso possibilita que estes interagem com o experimento, podendo surgir novas hipóteses. Para tanto, foram anotadas as observações para estudos posteriores, mas para este experimento foi mantido o foco;

- expectativas do condutor do experimento: antes da realização do experimento, foi realizada uma avaliação de todo material utilizado por um outro responsável (professor orientador), eliminando-se o viés do condutor da pesquisa.

Na **validade da conclusão** são avaliados os fatores de:

- manipulação dos dados: os resultados dos experimentos foram manipulados pelo pesquisador;
- confiabilidade das medidas: as medidas foram objetivamente definidas;
- confiabilidade na implementação: diz respeito as diferentes formas que os participantes podem desenvolver a aplicação. Nesta pesquisa não são dadas garantias de que o desenvolvimento de uma aplicação seja igual para todos os indivíduos, cada humano tem uma maneira de expressar e operacionalizar. Entretanto, para que a implementação seja confiável, é necessário que a implementação paralela ofereça desempenho em relação a sequencial;
- configuração do ambiente: o experimento foi executado em um laboratório sem comunicação externa do ambiente (*e.g.*, celulares, internet, etc). Apenas foram permitidos o uso de materiais disponibilizados.

6.1.10 Aspectos para Execução do Experimento

Esta seção apresenta uma visão geral de como foi aplicado o experimento, atenuando-se nos seguintes fatores:

- Consenso com o experimento: durante o experimento, os participantes tiveram o embasamento necessário sobre o experimento (objetivos e metas). Todos sabiam do que se tratava e ninguém foi obrigado a participar da pesquisa.
- Resultado sensitivos: neste experimento é possível que o resultado obtido influencie por questões de concorrência (*i.e.*, quem é o mais rápido). Para isso, foi mantido o anonimato dos participantes na descrição dos resultados dos experimentos.

Antes de executar o experimento, realizou-se um pré-teste, onde aplicou-se o experimento para um indivíduo com características compatíveis, simulando o ambiente experimental. Isso possibilitou a avaliação da documentação e a projeção aproximada do tempo necessário para executar o experimento, apontando alguns erros no processo experimental planejado, que posteriormente foram corrigidos.

No experimento, não estipulou-se um tempo limite para que o participante execute-o, sendo necessário concluí-lo. Antes de começar o experimento, os participantes tiveram uma explicação geral sobre a abordagem e como deveriam procedê-lo, esta foi em torno de 15 a 20 minutos. Durante a execução, em caso de dificuldades ou dúvidas, os participantes eram atendidos pelo pesquisador, mas somente quando se tratava do processo e não sobre como resolver o problema proposto. Na coleta de dados utilizou-se um formulário preenchido pelos participantes, para que o pesquisador não se envolva na definição dos resultados do experimento. Como mediu-se o esforço na paralelização de um algoritmo, o critério avaliado era o tempo, assim, os participantes foram instruídos a começar a contagem do tempo e a execução, somente depois da explicação sob um aviso.

Para verificar se o participante realmente concluiu o experimento, atentou-se para que a versão paralela desenvolvida com uma determinada abordagem, avaliando a obtenção de desempenho e o resultado da computação. Como garantia, o fomulário (Apêndice A.2) ofereceu campos para especificar os tempos de execução e a saída do programa.

6.2 Experimento para Medição de Desempenho

O experimento para medição de desempenho foi planejado usando uma abordagem estatística para validação dos resultados obtidos. Neste sentido, esta seção apresenta a utilização da abordagem de intervalos de confiança para estimar uma média do tempo de execução (Seção 6.2.3), identificação do tamanho da amostra necessário para estimar uma média de tempo (Seção 6.2.4), os instrumentos utilizados para conduzir o experimento (Seção 6.2.5) e quais foram os aspectos para a execução.

6.2.1 Medidas para Avaliação do Desempenho

A métrica relacionada com a segunda questão de pesquisa corresponde ao desempenho de uma aplicação paralela desenvolvida com as abordagens LED-PPOPP e OpenMP. Para isso, será medido o tempo de execução calculando o *Speed-Up* (fator de aceleração), no qual são analisados e comparados os ganhos em relação a aplicação sequencial para ambas as abordagens. O cálculo do fator de aceleração (S_p) de um programa paralelo com tempo de execução paralelo (T_p) se dá por

$$S_p = \frac{T_1}{T_p},$$

onde S representa o *Speed-Up*, T o Tempo de execução e $_p$ a quantidade de processos usados para resolver a computação. Este conceito de aceleração é usado tanto para uma análise teórica de algoritmos baseados na notação assintótica quanto para avaliação prática dos programas paralelos [19].

6.2.2 Formalização do Experimento

O experimento de medição de desempenho irá investigar a hipótese informal HP2, associando com a métrica levantada. Esta hipótese é aceita ou rejeitada com base nos indicadores numéricos dos resultados obtidos na paralelização de alguns algoritmos. Para isso, será comparado o desempenho (*Speed-Up*) de 5 algoritmos/programas paralelizados com a abordagem LED-PPOPP e OpenMP, analisando o comportamento ao executar com 2, 4, 6, 8, 10, 12, 14 e 16 núcleos. Devido a natureza do experimento (não busca resultados precisos na comparação), a hipótese vai ser aceita ou rejeitada subjetivamente:

1. Hipótese Informal, **HP2**: A LED-PPOPP (com Mestre/Escravo) não possui uma diferença significativa de desempenho em relação ao uso do OpenMP. Para considerar que uma diferença é significativa, estimou-se que um diferença de 10% ou mais é considerada significativa.

   (a) Medidas: será calculado o desempenho obtido usando LED-PPOPP e OpenMP para cada um dos núcleos com a formula do *Speed-Up*, através da coleta da média dos tempos de execução de cada algoritmo. Posteriormente, compara-se as diferenças de desempenho da LED-PPOPP em relação ao OpenMP.

6.2.3 Intervalos de Confiança

Uma estimativa pontual não fornece uma indicação de quanto ela é boa. Para isso, os estatísticos criaram o intervalo de confiança, que consiste em um intervalo de valores, em vez de apenas um único valor [44]. Com o nível de confiança é possível identificar uma taxa de sucesso de um procedimento experimental para calcular o intervalo de confiança. Enfim, para este experimento foi usado um nível de confiança de 95%, para o qual, o valor crítico ($z_{\alpha/2}$) é 1.96.

Antes de calcular o intervalo de confiança, calculou-se o desvio padrão (σ) de uma média de execuções do algoritmo, usando a seguinte formula:

$$\sigma = \sqrt{\frac{\Sigma(x-\mu)^2}{N}}.$$

Com o desvio padrão conhecido é possível calcular a margem de erro para a média populacional (E), através da seguinte formula:

$$E = z_{\alpha/2} \cdot \frac{\sigma}{\sqrt{n}}.$$

O intervalo de confiança é estimado a partir da margem de erro (E) calculada sobre a μ de execuções de um algoritmo, usando a seguinte formula:

$$\bar{x} - E < \mu < \bar{x} + E,$$

onde $\bar{x}$ é a média de execuções do algoritmo. Assim, é possível estar 95% confiante que o verdadeiro valor da μ esteja entre $\bar{x} - E$ e $\bar{x} + E$. Isso significa que, se fossem selecionadas várias amostras diferentes do mesmo tamanho e construindo-se os intervalos de confiança correspondentes, certamente 95% dos casos, iriam conter o valor da μ.

6.2.4 Tamanho da Amostra

O tamanho amostral está relacionado com a quantidade de execuções do algoritmo necessárias para determinar uma média. Esta estimativa garante maior confiabilidade da média e diminui o tempo gasto em execuções desnecessárias. Como lidou-se com o σ desconhecido, uma das alternativas é calculá-lo através de um estudo piloto, com base nos primeiros 31 valores amostrais selecionados aleatoriamente [44]. Desta forma, para cada algoritmo realizou-se aleatoriamente 40 execuções para determinar o desvio padrão (σ). O tamanho da amostra (n), calculou-se usando a formula

$$n = \left[\frac{z_{\alpha/2}\sigma}{E}\right]^2,$$

onde E é a margem de erro (calculado com base nas 40 execuções) e o $z_{\alpha/2}$ é 1.96 para 95% de nível de confiança. Normalmente, o resultado do cálculo é um número não inteiro, devendo este, usar a regra de arredondamento (arredondando o valor de n para o número inteiro mais próximo). Em conclusão, para as milhares de execuções possíveis do algoritmo, precisa-se apenas obter uma amostra aleatória de pelo menos n. Ainda que, será obtido 95% de confiança de que a média de execuções do algoritmo ($\bar{x}$) estará abaixo do desvio padrão (σ) da verdadeira média de execuções (μ).

6.2.5 Instrumentação

- Ambiente: realizou-se a execução do experimento uma máquina do Laboratório de Alto Desempenho (LAD) da PUCRS. Esta é um Dell PowerEdge R610, possuindo dois processadores Intel Xeon Quad-Core E5520 de 2.27GHZ com tecnologia *Hyper-Threading* (total de 16 núcleos). A memória concentra-se em uma arquitetura NUMA (*Non-Uniform Memory Access*), para a qual, cada processador possui 6Gb (total de 16Gb de memória por máquina). Um sistema de armazenamento com HD de 146.8GB, contendo o sistema operacional "Ubuntu-Linux-10.04-server-64bits".

- Recursos: foram usado os seguintes *benchmarks* desenvolvidos em C e paralelizados com OpenMP, disponibilizados por [45]:

  - *Estimate an Integral 2D*(EI_2D): programa que estima uma integral sobre um domínio retangular 2D usando uma técnica de média. O código fonte se encontra em [46].
  - *Fast Fourier Transform*(FFT): programa que demonstra a computação de um transformada rápida de Fourier. O código fonte está disponível em [47].
  - *Molecular Dynamics*(MD): programa que demonstra a simulação de dinâmica molecular. O código fonte foi extraído de [48].
  - *Matrix Multiplication*(MM): programa que cria um problema de multiplicação de matrizes $C = A * B$, o qual, está disponível em [49].
  - *Prime Numbers*(PN): programa que conta números primos de 1 até N. O código fonte se encontra em [50].

- Métricas: a coleta dos dados foi realizada a partir do *log* de execução dos algoritmos.

6.2.6 Aspectos para Execução do Experimento

O experimento foi executado levando em consideração os resultados sensitivos. A execução é realizada pelo pesquisador, sendo possível que ele influencie nos resultados. Para isso, optou-se por usar algoritmos desenvolvidos por outro pesquisador com a abordagem OpenMP, evitando que o desempenho na exploração do paralelismo seja influenciado.

Este experimento consiste na medição de desempenho para identificar e comparar as diferenças entre as abordagens comparadas. Então, paralelizou-se os mesmos programas de *benchmark* (EI_2D, FFT, MD, MM, PN) atentando-se principalmente no uso das funcionalidade da LED-PPOPP a partir da versão sequencial dos algoritmos. Assim, para demonstrar os resultados da implementação desta linguagem, as versões paralelizadas estão anexados no Apêndice A.6.

Os *benchmarks* forma executados aleatoriamente, evitando que qualquer influência do ambiente sobre os tempos de execução. Assim como, os resultados da execução dos programas foram registrados em arquivos de log. Verificando através destes, se os resultados obtidos eram corretos e, consequentemente, coletou-se o tempo de execução para calcular o fator de aceleração (*Speed-Up*) dos *benchmarks* em 2, 4, 6, 8, 10, 12, 14 e 16 núcleos.

7. RESULTADOS

Este capítulo apresenta a última etapa do processo de experimentação, realizando a análise e avaliação dos resultados obtidos na execução dos dois experimentos. A Seção 7.1 discute os resultados obtidos na medição do esforço de programação paralela usando a abordagem LED-PPOPP e Pthread. Na sequência, a Seção 7.2 realiza uma avaliação do desempenho medido na paralelização de 5 algoritmos, comparando as diferenças entre a LED-PPOPP e o OpenMP.

7.1 Análise e Avaliação da Medição do Esforço de Programação Paralela

Conforme o planejamento e a execução do experimento relatado na Seção 6.1, garantiu-se maior confiabilidade na obtenção dos resultados. Isso porquê, em um ambiente controlado é possível restringir fatores que poderiam influenciar e desvirtuar os participantes do foco experimental, simulando um ambiente real de trabalho.

O resultado da medição do esforço é demonstrado na Tabela 7.1. Nela, estão descritos: o tempo que cada indivíduo levou para paralelizar o algoritmo de multiplicação de matrizes, o nível de conhecimento (conhecimento geral) e o peso de conhecimento dos participantes sobre as abordagens. Nota-se que na classificação dos grupos, o nível de conhecimento ficou equilibrando, evitando que uma das abordagens obtivesse vantagem. Também, identifica-se que a maioria dos participantes já programou com Pthread, enquanto com LED-PPOPP, nenhum deles se quer conhecia esta abordagem.

Tabela 7.1: *Resultados do experimento controlado.*

ID	LED-PPOPP (Min.)	Pthread (Min.)	Nível	Peso LED-PPOPP	Peso Pthread
Grupo 1 - (LED-PPOPP $\Longrightarrow$ Pthread)					
1	29	18	Ótimo	Zero	Médio
2	23	32	Ótimo	Zero	Médio
3	13	30	Muito Bom	Zero	Baixo
4	25	29	Muito Bom	Zero	Médio
5	19	27	Muito Bom	Zero	Médio
6	33	65	Bom	Zero	Baixo
7	15	39	Bom	Zero	Baixo
8	31	81	Bom	Zero	Baixo
9	28	59	Bom	Zero	Zero
10	28	45	Razoável	Zero	Baixo
Grupo 2 - (LED-PPOPP $\Longleftarrow$ Pthread)					
11	15	33	Ótimo	Zero	Alto
12	17	54	Ótimo	Zero	Médio
13	12	65	Muito Bom	Zero	Médio
14	15	70	Muito Bom	Zero	Baixo
15	17	60	Bom	Zero	Baixo
16	15	71	Bom	Zero	Baixo
17	12	61	Bom	Zero	Baixo
18	21	63	Bom	Zero	Zero
19	24	61	Razoável	Zero	Zero
20	22	66	Razoável	Zero	Baixo

Realmente, os resultados mostraram que o esforço tende a ser menor quando já paralelizada a aplicação com outra abordagem, pois ao utilizar a abordagem posterior, os participantes já conheciam o problema. Portanto, a criação de grupos executando inversamente o experimento garantiu o balanceamento entre as abordagens, evitando que apenas uma seja beneficiada.

Embora uma abordagem seja beneficiada ao ser desenvolvida posteriormente, a maioria dos participantes obteve menos esforço paralelizando com a LED-PPOPP. No entanto, encontrou-se uma exceção no experimento, o indivíduo 1 da Tabela 7.1 paralelizou mais rápido o problema com a abordagem Pthread. Acredita-se que dois fatores contribuíram nesse acontecimento: o problema já era conhecido para o participante e possuía um conhecimento bastante apurado de Pthread, sendo modesto ao preencher o questionário de avaliação de conhecimento.

Com os resultados obtidos de cada um dos participantes na execução do experimento é possível extrair diversas informações sobre o comportamento na paralelização usando um método estatístico para indicar tendências. Alguns cálculos exigem a aplicação de formulas complexas, para evitar isso, usou-se a ferramenta SPSS de análise estatística. Uma análise descritivas dos resultados é ilustrada na Figura 7.1.

			Statistic	Std. Error
Pthread	**Mean**		51.45	4.048
	95% Confidence Interval for Mean	Lower Bound	42.98	
		Upper Bound	59.92	
	5% Trimmed Mean		51.67	
	Median		59.50	
	Variance		327.734	
	Std. Deviation		18.103	
	Minimum		18	
	Maximum		81	
	Range		63	
	Interquartile Range		33	
	Skewness		-.365	.512
	Kurtosis		-1.157	.992
LED_PPOPP	**Mean**		20.70	1.487
	95% Confidence Interval for Mean	Lower Bound	17.59	
		Upper Bound	23.81	
	5% Trimmed Mean		20.50	
	Median		20.00	
	Variance		44.221	
	Std. Deviation		6.650	
	Minimum		12	
	Maximum		33	
	Range		21	
	Interquartile Range		12	
	Skewness		.360	.512
	Kurtosis		-1.151	.992

Figura 7.1: Análise descritiva do experimento gerada pelo SPSS.

A análise descritiva mostrou que em média o tempo gasto com Pthread foi maior do que com a LED-PPOPP, estando 95% confiante que o verdadeiro valor da média em uma próxima experimentação (de 20 ou mais amostras) com o algoritmo de multiplicação de matrizes, a média ficará entre 42.98 e 59.92 usando Pthread e entre 17.59 e 23.81 utilizando LED-PPOPP.

Conforme foi previsto no planejamento do experimento, no teste da hipótese é preciso verificar se a distribuição dos resultados concentra-se em uma curva normal, identificando qual abordagem

deve ser utilizada para executar o teste. A Figura 7.2 demonstra o resultado do teste de normalidade no SPSS, podendo concluir que a distribuição não é normal. A normalidade é existente quando as abordagens **Kolmogorov-Smimov** e **Shapiro-Wilk** possuem o Sig. $>$ 0.05. Neste experimento, os resultados de Pthread não estão distribuídos normalmente, indicando que o teste de hipótese deve ser realizado usando a abordagem **Wilcoxon**.

	Kolmogorov-Smimov(a)			***Shapiro-Wilk***		
	Statistic	df	Sig.	Statistic	df	Sig.
Pthread	.212	20	.019	.920	20	.099
LED_PPOPP	.161	20	.185	.931	20	.158

a Lilliefors Significance Correction

Figura 7.2: Teste de normalidade.

O teste de Wilcoxon baseia-se na comparação das diferenças de scores de duas abordagens, ranqueando-as em *ranks* positivos e negativos [43]. Assim, o resultado do cálculo dos *ranks* é demonstrado na Figura 7.3, obtendo apenas um negativo e 19 favoráveis, tendenciando que o esforço com Pthread foi maior que com LED-PPOPP (**b** Pthread $>$ LED_PPOPP).

Ranks

		N	**Mean Rank**	**Sum of Ranks**
Pthread - LED_PPOPP	Negative Ranks	1(a)	4.00	4.00
	Positive Ranks	19(b)	10.84	206.00
	Ties	0(c)		
	Total	20		

a Pthread < LED_PPOPP
b Pthread > LED_PPOPP
c Pthread = LED_PPOPP

Figura 7.3: Ranks.

Para confirmar se o esforço é significativamente diferente entre as abordagens, é necessário que o nível de significância (Sig.) seja menor que 0.05. O cálculo realizado pelo SPSS é demonstrado na Figura 7.4, podendo concluir que o esforço é significativamente diferente. Logo, rejeita-se a hipótese nula (H_0) e baseando-se no resultado das médias, aceita-se a hipótese alternativa H_1, a qual, afirma que o esforço é maior paralelizando com Pthread do que com a LED-PPOPP.

Teste Estatístico(b)

	Pthread - LED_PPOPP
Z	-3.771(a)
Asymp. Sig. (2-tailed)	.000

a Based on negative ranks.
b Wilcoxon Signed Ranks Test

Figura 7.4: Teste estatístico.

7.1.1 Discussão

Este experimento demonstrou uma grande vantagem da LED-PPOPP em relação ao esforço na paralelização do algoritmo de multiplicação de matrizes, tendenciando maior facilidade ao utilizar esta abordagem. Como os participantes não conheciam esta abordagem, mas mesmo assim conseguiram desenvolvê-la mais rápido, conclui-se que, a aprendizagem e a programação da aplicação exigiram menos esforços com a LED-PPOPP (média de 20.70 minutos) do que programar tendo um conhecimento básico sobre Pthread (média de 51.45 minutos).

O objetivo foi disponibilizar uma interface em um contexto de alto nível (padrão de programação paralela) facilmente interpretada, utilizando a biblioteca Pthread para explorar o paralelismo da arquitetura. Isso alavancou a comparação do esforço com esta biblioteca, pois foi possível identificar o quanto a interface da LED-PPOPP reduziu o esforço do programador na paralelização de uma aplicação.

7.2 Análise e Avaliação da Medição de Desempenho

Na medição do desempenho são observados o *Speed-Up* (desempenho) e a eficiência (eficiência da aplicação em relação ao uso dos processadores) que uma aplicação consegue obter na exploração de paralelismo. Mesmo que o objetivo principal deste trabalho seja diminuir o esforço na programação paralela, é necessário que as aplicações possuam um desempenho aceitável, obtendo vantagem da arquitetura paralela. Portanto, realizou-se uma avaliação da LED-PPOPP identificando se existem grandes perdas de desempenho em relação a uma abordagem otimizada.

O experimento buscou um nível de confiança de 95% nos resultados obtidos. Sendo assim, para cada um dos *benchmarks*, calculou-se o desvio padrão (σ) e a margem de erro (E), determinando a quantidade de execuções necessárias (n) para estimar a média. Consequentemente, com as médias efetivas, os intervalos de confiança ($< \mu <$) para as médias dos tempos de execuções são calculados.

7.2.1 Panorama Geral de Resultados dos *Benchmarks*

O experimento utilizou 5 aplicações de *benchmark* paralelizadas com o OpenMP por [45]. Para a análise do desempenho, optou-se por aplicações com características diferentes e que possibilitam a implementação do padrão Mestre/Escravo. Desta forma, definiu-se uma entrada padrão para cada um dos respectivos programas:

- EI_2D: estima uma integral sobre uma área retangular 2D usando a regra de quadrado do produto. O tamanho da área definido é de *nx=91768* por *ny=91768*. Esta aplicação pode ser praticamente toda paralelizada.

- FFT: este programa realiza uma transformada rápida de *Fourier* em um vetor complexo de dados. Executando com interações de tamanho *nits=1000* dividindo este valor por 10 a cada 4 interações até completar *ln2_max=25* (*log* na base 2 de n). A maior parte da computação é sequencial, sendo pouco paralelizável.

- MD: o programa efetua uma simulação de dinâmica molecular. Efetuando com *np=4000* partículas em *step_num=4* etapas com tempo de *dt=0.000100*. Boa parte desta computação pode ser paralelizada.

- MM: programa que cria um problema de multiplicação de matrizes densa. Utilizando matrizes de ordem *n=1800*. Este programa pode ser praticamente todo paralelizado.

- PN: programa que encontra os números primos em um vetor de 1 até *N = 500000*. O paralelismo não é facilmente explorado nesta aplicação.

Para não influenciar nos resultados, assumiu-se que os *benchmarks* utilizando a abordagem OpenMP foram implementados da melhor maneira, realizando a experimentação com 2, 4, 6, 8, 10, 12, 14 e 16 núcleos. Os gráficos da Figura 7.5 demonstram os resultados obtidos de todos os *benchmarks*. O FFT teve um *Speed-Up* e uma eficiência muito abaixo do ideal. O mesmo acontece com o PN, entretanto, seu desempenho é melhor se comparado com o FFT. Em relação aos programas EI_2D, MD e MM, a curva manteve-se próxima do ideal até 8 núcleos e, posteriormente, observa-se uma queda no desempenho a partir de 10 núcleos. Acredita-se que o uso de núcleos lógicos afetou o desempenho destes *benchmarks*, pois após a queda, a curva do *Speed-up* mantém-se em uma curva crescente até 16 núcleos. Além disso, se utilizadas técnicas de afinidade de memória para melhor explorar a arquitetura NUMA do ambiente experimentado, poderiam ser obtidos melhores resultados para o algoritmo MM [51].

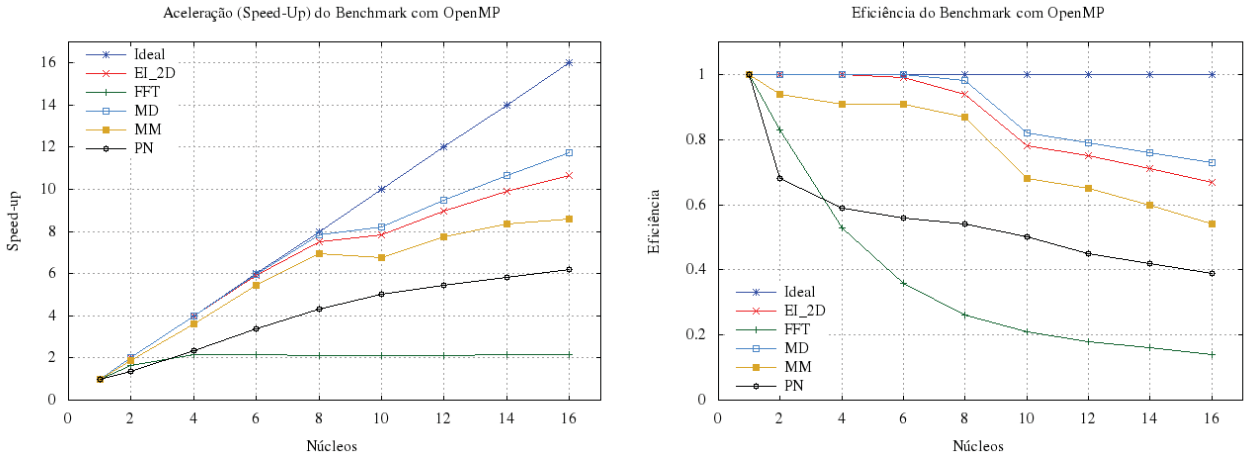


Figura 7.5: Desempenho dos programas de *benchmark* com OpenMP (*Speed-Up* e Eficiência).

Todos os *benchmarks* foram testados com diferentes granularidades, ajustando o tamanho do grão para obter o melhor desempenho possível. No caso do FFT, como dito anteriormente, seu código é pouco paralelizável, identificando que o seu comportamento foi semelhante (baixo desempenho) ao obtido em outros trabalhos, como por exemplo em [52]. O mesmo acontece com o algoritmo PN, pois o paralelismo não é facilmente explorado.

7.2.2 Comparação dos Resultados do *Benchmark* EI_2D (*Estimate an Integral* 2D)

Na comparação de desempenho do EI_2D também é analisado o quão eficiente é o algoritmo na utilização dos recursos de paralelismo da arquitetura. Normalmente, o tempo de execução de um algoritmo apresenta um comportamento irregular, pois fatores como ambiente de execução e a computação podem influenciar na medição do desempenho. Para garantir maior confiabilidade dos testes, utilizou-se a abordagem estatística de nível de confiança, conforme descrito na Seção 7.2.

Os resultados do experimento para o EI_2D estão descritos na Tabela 7.2. A estimativa do tamanho da amostra necessário (n) para estimar a média do tempo de execução abordou um nível de confiança de 95%, calculado a partir das 40 primeiras execuções nas versões LED-PPOPP e OpenMP. A quantidade de execuções realizadas para estimar a média do tempo (μTempo) é representada por N. Além disso, estão tabulados o *Speed-Up*, a eficiência e o intervalo de confiança.

Tabela 7.2: *Resultados com benchmark EI_2D.*

Núcleos	μTempo (s)	N	σ	E	Confiança de 95% (s)	*Speed-Up*	Eficiência	n
EI_2D - OpenMP								
1	242.85	40	0.05	0.05	$242.80 < \mu < 242.90$	1.00	1.00	4
2	121.50	40	0.21	0.10	$121.39 < \mu < 121.60$	2.00	1.00	17
4	60.73	40	0.01	0.03	$60.71 < \mu < 60.76$	4.0	1.00	2
6	41.01	140	2.61	0.36	$40.65 < \mu < 41.37$	5.92	0.99	133
8	32.31	190	3.12	0.39	$31.92 < \mu < 32.70$	7.52	0.94	185
10	30.96	40	0.36	0.13	$30.83 < \mu < 31.09$	7.84	0.78	29
12	27.01	40	0.49	0.16	$26.86 < \mu < 27.17$	8.99	0.75	39
14	24.52	70	1.86	0.30	$24.22 < \mu < 24.82$	9.90	0.71	68
16	22.78	90	0.96	0.22	$22.57 < \mu < 23.00$	10.66	0.67	81
EI_2D - LED-PPOPP								
1	243.82	40	0.05	0.05	$243.77 < \mu < 243.88$	1.00	1.00	5
2	125.38	140	1.71	0.29	$125.09 < \mu < 125.66$	1.94	0.97	140
4	67.40	120	1.41	0.26	$67.14 < \mu < 67.67$	3.62	0.90	115
6	46.01	100	1.08	0.23	$45.78 < \mu < 46.24$	5.30	0.88	99
8	33.48	70	0.50	0.16	$33.32 < \mu < 33.64$	7.28	0.91	68
10	31.33	60	0.70	0.19	$31.15 < \mu < 31.52$	7.78	0.78	56
12	27.19	40	0.23	0.11	$27.08 < \mu < 27.29$	8.97	0.75	18
14	24.02	40	0.17	0.09	$23.93 < \mu < 24.11$	10.15	0.73	14
16	21.79	40	0.10	0.07	$21.72 < \mu < 21.86$	11.19	0.70	8

Por meio destes testes, pode-se perceber que os algoritmos não possuem o mesmo comportamento em cada um dos conjuntos de núcleos testados, tornando-se necessário estimar o n em cada um deles, pois o desvio padrão (σ) e a margem de erro (E) não eram iguais nas primeiras 40 execuções. Os valores descritos na Tabela 7.2 de σ e E estão relacionados ao que foi obtido com N execuções do EI_2D para LED-PPOPP e OpenMP.

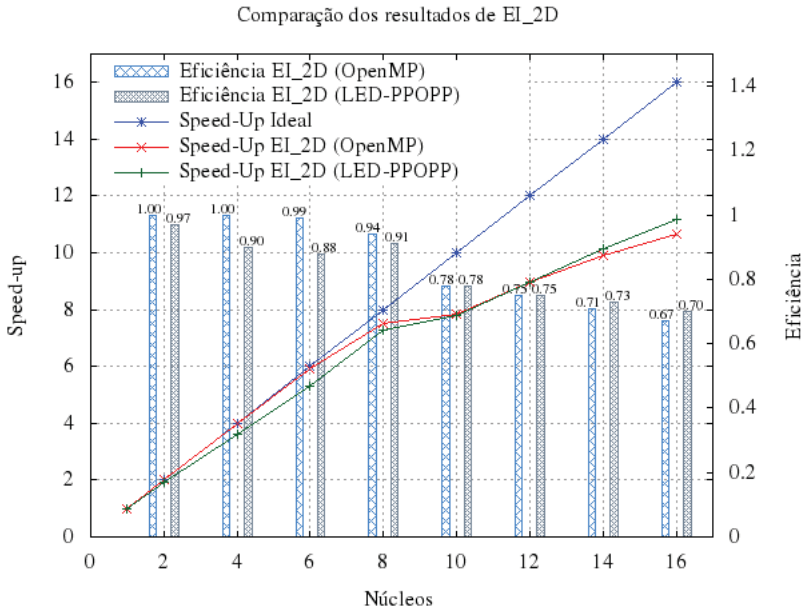


Figura 7.6: Gráfico do *Speed-Up* e Eficiência com EI_2D.

Na experimentação da LED-PPOPP realizada foi implementado apenas o núcleo principal com o bloco escravo fora do bloco mestre (código fonte disponível em [53]), apresentando bons resultados e parecidos aos obtidos com OpenMP. A versão OpenMP com conjunto de núcleos menores de 8 obteve desempenho e eficiência melhores que a versão paralelizada com a LED-PPOPP. No entanto, com 14 e 16 núcleos a LED-PPOPP foi melhor. Mesmo assim, o desempenho esteve aproximado em todos os conjuntos de núcleos testados, conforme pode ser visto no gráfico da Figura 7.6.

A queda provocada no desempenho a partir de 10 núcleos está associada à utilização dos núcleos lógicos do ambiente arquitetural. Já a vantagem obtida do OpenMP em relação a LED-PPOPP nos primeiros conjuntos de núcleos, pode estar associada à divisão do trabalho, pois não foi implementado um tratamento otimizado para o balanceamento de carga na LED-PPOPP, dividindo o trabalho apenas pela quantidade de escravos.

7.2.3 Comparação dos Resultados do *Benchmark* FFT (*Fast Fourier Transform*)

A implementação paralela do FFT com a LED-PPOPP necessitou o uso de subnúcleos para efetuar a paralelização das rotinas que foram implementadas em funções globais do algoritmo, sendo construções com escravos processando independentemente, ou seja, sem ajuda do Mestre (código fonte disponível em [53]).

Os resultados da paralelização do algoritmo FFT utilizando OpenMP e LED-PPOPP são demonstrados na Tabela 7.3. Como visto anteriormente na Seção 7.2.1, este algoritmo não é facilmente paralelizado com OpenMP, mostrando resultados ruins a partir de 4 núcleos. O mesmo aconteceu para a versão com LED-PPOPP, sendo apenas melhor em relação ao OpenMP com 2 núcleos.

Tabela 7.3: *Resultados com benchmark FFT.*

Núcleos	μTempo (s)	N	σ	E	Confiança de 95% (s)	*Speed-Up*	Eficiência	n
FFT - OpenMP								
1	146.42	40	0.16	0.09	$146.33 < \mu < 146.51$	1.00	1.00	13
2	88.63	40	0.27	0.12	$88.51 < \mu < 88.74$	1.65	0.83	22
4	68.55	150	2.33	0.34	$68.22 < \mu < 68.89$	2.14	0.53	100
6	67.83	150	2.03	0.32	$67.52 < \mu < 68.15$	2.16	0.36	94
8	69.19	170	2.53	0.35	$68.84 < \mu < 69.55$	2.12	0.26	165
10	70.22	170	1.99	0.31	$69.91 < \mu < 70.53$	2.09	0.21	157
12	69.08	150	1.67	0.29	$68.80 < \mu < 69.37$	2.12	0.18	95
14	67.40	40	0.43	0.15	$67.26 < \mu < 67.55$	2.17	0.16	34
16	67.67	40	0.43	0.15	$67.53 < \mu < 67.82$	2.16	0.14	34
FFT - LED-PPOPP								
1	146.42	40	0.16	0.09	$146.33 < \mu < 146.51$	1.00	1.00	13
2	83.06	50	0.62	0.17	$82.88 < \mu < 83.23$	1.76	0.88	47
4	74.38	190	1.88	0.30	$74.08 < \mu < 74.68$	1.97	0.49	186
6	77.83	40	0.51	0.16	$77.67 < \mu < 77.98$	1.88	0.31	40
8	77.65	190	2.62	0.36	$77.29 < \mu < 78.01$	1.89	0.24	190
10	82.24	60	1.74	0.29	$81.95 < \mu < 82.53$	1.78	0.18	58
12	84.16	190	2.04	0.32	$83.85 < \mu < 84.48$	1.74	0.14	159
14	84.55	190	1.78	0.29	$84.25 < \mu < 84.84$	1.73	0.12	175
16	84.91	50	0.57	0.17	$84.75 < \mu < 85.08$	1.72	0.11	48

A representação gráfica do desempenho obtido pode ser visto na Figura 7.7. Embora as duas abordagens possuem resultados não satisfatórios, a eficiência na utilização do paralelismo disponível e o desempenho são aproximados para todos os conjuntos de núcleos testados.

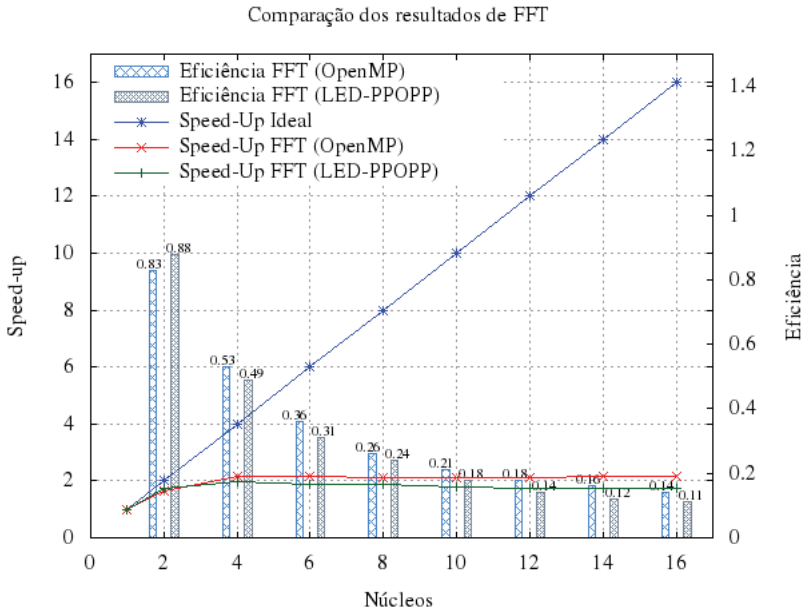


Figura 7.7: Gráfico do *Speed-Up* e Eficiência com FFT.

7.2.4 Comparação dos Resultados do *Benchmark* MD (*Molecular Dynamics*)

O algoritmo MD é quase inteiramente paralelizável. A versão implementada com LED-PPOPP necessitou o uso de uma construção de subnúcleo, além do núcleo principal (código fonte disponível em [53]). Em função disso, acredita-se os resultados não foram fortemente prejudicados, conforme demonstram os resultados da Tabela 7.4.

Tabela 7.4: *Resultados com benchmark MD.*

Núcleos	μTempo (s)	N	σ	E	Confiança de 95% (s)	*Speed-Up*	Eficiência	n
MD - OpenMP								
1	106.04	40	0.28	0.12	$105.92 < \mu < 106.16$	1.00	1.00	23
2	53.04	40	0.17	0.09	$52.95 < \mu < 53.13$	2.00	1.00	14
4	26.53	40	0.07	0.06	$26.47 < \mu < 26.59$	4.00	1.00	5
6	17.76	60	0.55	0.16	$17.60 < \mu < 17.92$	5.97	1.00	53
8	13.51	70	0.81	0.20	$13.31 < \mu < 13.70$	7.85	0.98	69
10	12.87	40	0.04	0.05	$12.82 < \mu < 12.92$	8.24	0.82	4
12	11.18	40	0.75	0.19	$10.99 < \mu < 11.37$	9.48	0.79	83
14	9.93	90	1.12	0.23	$9.69 < \mu < 10.16$	10.68	0.76	97
16	9.05	110	0.95	0.22	$8.83 < \mu < 9.26$	11.72	0.73	110
MD - LED-PPOPP								
1	105.21	40	0.28	0.12	$105.09 < \mu < 105.32$	1.00	1.00	23
2	52.64	40	0.14	0.08	$52.56 < \mu < 52.73$	2.00	1.00	12
4	26.43	40	0.07	0.06	$26.37 < \mu < 26.48$	3.98	1.00	6
6	17.83	80	0.11	0.07	$17.75 < \mu < 17.90$	5.90	0.98	80
8	13.63	40	0.40	0.14	$13.49 < \mu < 13.77$	7.72	0.96	32
10	13.03	70	0.86	0.21	$12.82 < \mu < 13.23$	8.08	0.81	68
12	11.74	150	1.50	0.27	$11.47 < \mu < 12.01$	8.96	0.75	142
14	10.58	170	1.17	0.24	$10.34 < \mu < 10.82$	9.94	0.71	161
16	9.63	130	1.08	0.23	$9.40 < \mu < 9.86$	10.93	0.68	121

Na versão LED-PPOPP a utilização de núcleos lógicos também prejudicou o desempenho da aplicação, conforme é ilustrado no gráfico da Figura 7.8. Constatou-se que o desempenho e a eficiência foi melhor utilizando o OpenMP, mas os resultados ficaram aproximados utilizando as duas abordagens.

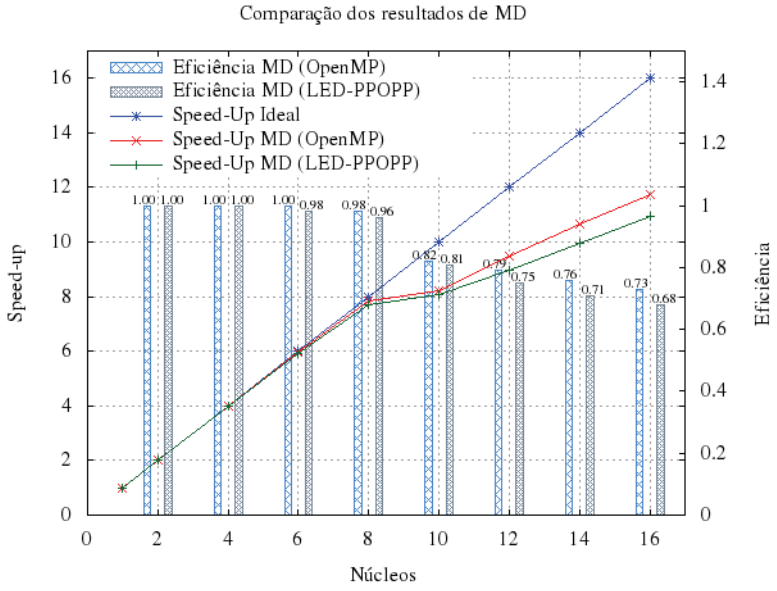


Figura 7.8: Gráfico do *Speed-Up* e Eficiência com MD.

7.2.5 Comparação dos Resultados do *Benchmark* MM (*Matrix Multiplication*)

A paralelização efetuada com a LED-PPOPP também obteve um desempenho satisfatório na aplicação de MM. Os resultados são comparados graficamente na Figura 7.9, demonstrando que as medidas de desempenho e eficiência foram aproximadas.

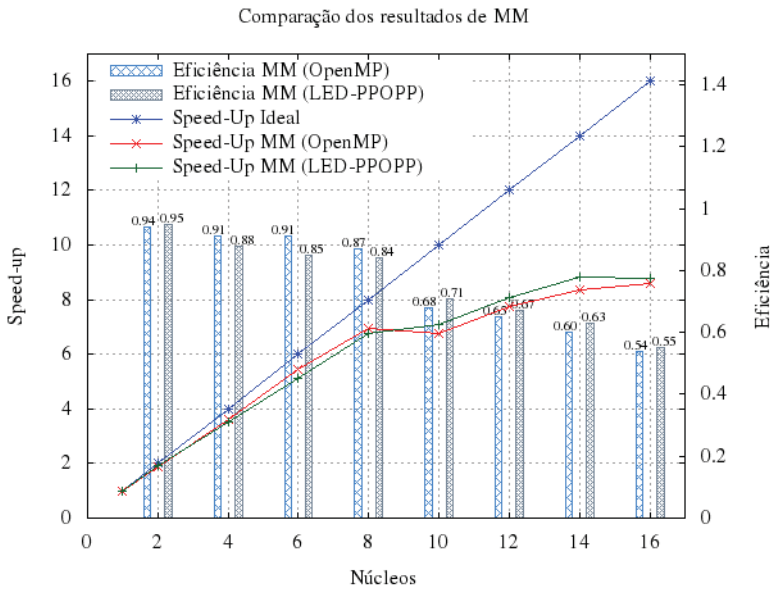


Figura 7.9: Gráfico do *Speed-Up* e Eficiência com MM.

O OpenMP se mostrou mais favorável com 4, 6 e 8 núcleos. No restante, a LED-PPOPP obteve maior vantagem sobre a arquitetura. Novamente, como aconteceu nos demais algoritmos, o desempenho foi prejudicado pela utilização de núcleos lógicos a partir de 10 núcleos. A Tabela 7.5 apresenta em mais detalhes os resultados obtidos.

Tabela 7.5: *Resultados com benchmark MM.*

Núcleos	μTempo (s)	N	σ	E	Confiança de 95% (s)	*Speed-Up*	Eficiência	n
MM - OpenMP								
1	160.01	70	0.78	0.19	$159.81 < \mu < 160.20$	1.00	1.00	65
2	85.37	210	3.16	0.39	$84.97 < \mu < 85.76$	1.87	0.94	176
4	44.11	80	0.89	0.21	$43.91 < \mu < 44.32$	3.63	0.91	73
6	29.36	40	0.30	0.12	$29.24 < \mu < 29.48$	5.45	0.91	24
8	23.05	210	2.18	0.33	$22.72 < \mu < 23.38$	6.94	0.87	210
10	23.60	40	0.42	0.14	$23.46 < \mu < 23.74$	6.78	0.68	33
12	20.65	40	0.32	0.12	$20.53 < \mu < 20.78$	7.75	0.65	25
14	19.13	50	0.65	0.18	$18.95 < \mu < 19.31$	8.36	0.60	49
16	18.64	60	0.77	0.19	$18.45 < \mu < 18.84$	8.58	0.54	60
MM - LED-PPOPP								
1	133.58	60	46.49	1.51	$132.07 < \mu < 135.09$	1.00	1.00	46
2	69.96	190	42.11	1.44	$68.53 < \mu < 71.40$	1.91	0.95	184
4	38.14	150	40.04	1.40	$36.74 < \mu < 39.54$	3.50	0.88	150
6	26.15	90	31.07	1.23	$24.92 < \mu < 27.38$	5.11	0.85	85
8	19.79	100	28.63	1.18	$18.60 < \mu < 20.97$	6.75	0.84	93
10	18.88	110	28.95	1.19	$17.69 < \mu < 20.07$	7.07	0.71	104
12	16.54	130	28.63	1.18	$15.35 < \mu < 17.72$	8.08	0.67	125
14	15.17	190	29.33	1.20	$13.97 < \mu < 16.37$	8.81	0.63	179
16	15.21	150	28.74	1.19	$14.02 < \mu < 16.39$	8.78	0.55	145

Com a LED-PPOPP, a implementação do MM foi realizada compartilhando os trabalhos escravos dentro do bloco mestre (disponível em [53]). Em teoria, se o trabalho não for balanceado corretamente o desempenho pode ser prejudicado. Mesmo sem possuir um balanceamento otimizado, a LED-PPOPP se mostrou melhor do que o OpenMP no algoritmos MM, obtendo o desempenho e a eficiência melhor na maioria dos conjuntos de núcleos testados. Porém, a execução da versão paralelizada com LED-PPOPP se mostrou mais irregular, sendo necessário uma amostra maior para garantir 95% de confiança.

7.2.6 Comparação dos Resultados do *Benchmark* PN (*Prime Numbers*)

A maior parte do código da aplicação de PN é sequencial, onde a exploração do paralelismo se torna mais difícil. Na paralelização com a LED-PPOPP utilizou-se uma construção simples compartilhando a carga dos escravos com o Mestre (código fonte está disponível em [53]). Embora os resultados obtidos tenha sido favoráveis para a versão paralela com OpenMP, acredita-se que a implementação com os escravos, compartilhando a carga com o mestre não seja um fator prejudicial na obtenção do desempenho. Isso porque o desempenho e a eficiência obtidos com as duas abordagens ficaram bem próximos uns dos outros, como pode ser visto na representação gráfica da Figura 7.10.

Na Tabela 7.6 é possível constatar claramente uma pequena vantagem no desempenho da versão OpenMP em relação a LED-PPOPP a partir de 4 núcleos, bem como a eficiência na utilização dos núcleos.

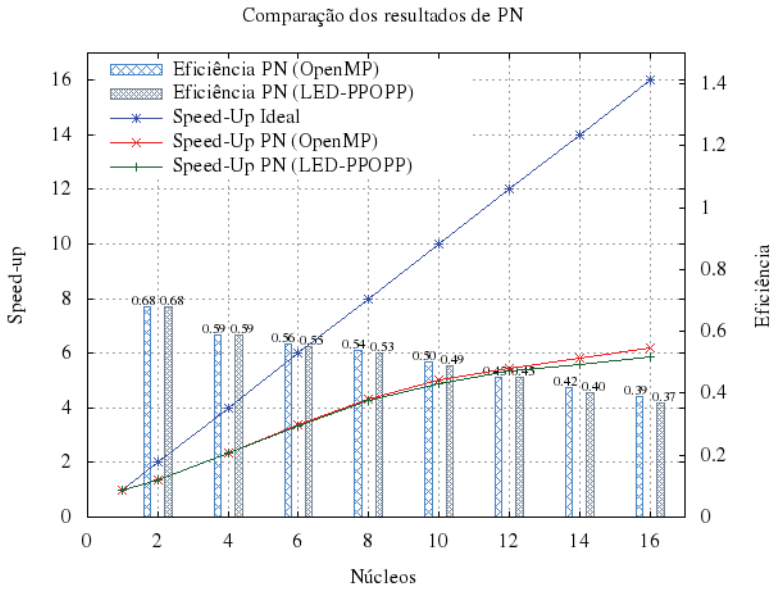


Figura 7.10: Gráfico do *Speed-Up* e Eficiência com PN.

Tabela 7.6: *Resultados com benchmark PN.*

Núcleos	μTempo (s)	N	σ	E	Confiança de 95% (s)	*Speed-Up*	Eficiência	n
PN - OpenMP								
1	50.95	40	0.01	0.02	$50.93 < \mu < 50.97$	1.00	1.00	1
2	37.47	40	0.02	0.03	$37.44 < \mu < 37.50$	1.36	0.68	2
4	21.66	40	0.02	0.03	$21.63 < \mu < 21.68$	2.35	0.59	2
6	15.11	40	0.03	0.04	$15.07 < \mu < 15.14$	3.37	0.56	2
8	11.81	80	0.78	0.20	$11.61 < \mu < 12.00$	4.31	0.54	55
10	10.13	80	1.02	0.22	$9.91 < \mu < 10.36$	5.03	0.50	72
12	9.35	80	1.23	0.25	$9.11 < \mu < 9.60$	5.45	0.45	79
14	8.73	80	1.04	0.23	$8.50 < \mu < 8.95$	5.84	0.42	46
16	8.19	40	0.39	0.14	$8.05 < \mu < 8.32$	6.22	0.39	31
PN - LED-PPOPP								
1	50.97	40	0.01	0.02	$50.95 < \mu < 50.98$	1.00	1.00	1
2	37.50	40	0.02	0.03	$37.46 < \mu < 37.53$	1.36	0.68	2
4	21.78	40	0.04	0.04	$21.73 < \mu < 21.82$	2.34	0.59	4
6	15.33	40	0.06	0.05	$15.28 < \mu < 15.38$	3.32	0.55	5
8	12.00	40	0.20	0.10	$11.90 < \mu < 12.10$	4.25	0.53	17
10	10.45	40	0.29	0.12	$10.33 < \mu < 10.57$	4.88	0.49	23
12	9.52	40	0.51	0.16	$9.36 < \mu < 9.68$	5.35	0.45	40
14	9.08	40	0.43	0.15	$8.94 < \mu < 9.23$	5.61	0.40	34
16	8.68	40	0.44	0.15	$8.54 < \mu < 8.83$	5.87	0.37	35

7.2.7 Conclusão

A diversidade das aplicações possibilitou a utilização das funcionalidades fornecidas de núcleo e subnúcleo do padrão Mestre/Escravo. Deste modo, as construções utilizando blocos escravos dentro ou fora do bloco mestre também foram possíveis. No entanto, a homogeneidade das aplicações em isolar a seção crítica evitou o uso da primitiva de proteção de dados.

Este experimento avaliou e comparou o desempenho e a eficiência das aplicações de *benchmark*

(EI_2D, FFT, MD, MM e PN) paralelizadas com a LED-PPOPP. O objetivo da medição foi analisar se abstração criada consegue explorar o paralelismo sem grandes perdas de desempenho comparado a uma solução otimizada com esta finalidade. Então, calculou-se os ganhos de desempenho (*Speed-Up*) obtidos em relação ao OpenMP. A porcentagem de ganho (G) em relação ao OpenMP é calculada com a seguinte formula:

$$G = \frac{(S_pLED\_PPOPP - S_pOpenMP) \times 100}{S_pOpenMP},$$

onde S_pLED_PPOPP é o *Speed-Up* obtido com a paralelização usando LED-PPOPP e $S_pOpenMP$ é o *Speed-Up* obtido com a versão OpenMP.

O resultado da diferença de desempenho em relação ao OpenMP é apresentado na Tabela 7.7, apresentando um cenário favorável para o OpenMP na maioria dos casos de testes. Somente no *benchmark* MM a LED-PPOPP obteve alguns resultados favoráveis. O FFT é um caso isolado, obtendo maior quantidade de situações em que a diferença é significativa entre as abordagens. Isso acontece porque as medidas de *Speed-Up* obtidas são baixas, então, qualquer diferença no desempenho representa bastante no cálculo da porcentagem. Nota-se que as diferenças não foram significativas (em torno de 10%) na maioria dos testes realizados. Logo, é possível validar a hipótese **HP2**, pois os resultados demostraram que o desempenho da LED-PPOPP não foi significativamente diferente em relação ao obtido com OpenMP na maior parte (cerca de 80%) das aplicações experimentadas.

Tabela 7.7: *Diferença de desempenho da LED-PPOPP em relação ao OpenMP*

Núcleos	EI_2D		FTT		MD		MM		PN	
2	-0.05	-2.70%	0.11	6.71%	0.00	0.00%	0.03	1.86%	0.00	0.00%
4	-0.38	-9.53%	-0.17	-7.83%	-0.02	-0.38%	-0.12	-3.43%	-0.01	-0.52%
6	-0.62	-10.51%	-0.28	-12.84%	-0.07	-1.18%	-0.34	-6.28%	-0.05	-1.41%
8	-0.23	-3.10%	-0.23	-10.89%	-0.13	-1.68%	-0.19	-2.75%	-0.07	-1.60%
10	-0.06	-0.79%	-0.30	-14.62%	-0.16	-1.99%	0.29	4.34%	-0.15	-2.97%
12	-0.02	-0.24%	-0.38	-17.92%	-0.52	-5.47%	0.33	4.26%	-0.09	-1.70%
14	0.25	2.51%	-0.44	-20.28%	-0.74	-6.94%	0.44	5.27%	-0.23	-3.88%
16	0.53	4.97%	-0.44	-20.30%	-0.79	-6.76%	0.20	2.34%	-0.35	-5.69%

Legenda

—	Favorável para LED-PPOPP (LED-PPOPP $>=$ OpenMP)
—	Favorável para OpenMP (OpenMP $>$ LED-PPOPP)

8. CONCLUSÃO

Este trabalho apresentou uma proposta de uma Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP), realizando um estudo de caso com o padrão Mestre/Escravo para arquiteturas *multi-core*. O primeiro passo foi a construção da interface de programação (interface da linguagem). Desenvolveu-se então um analisador para detectar erros na declaração das funcionalidades, bem como introduzir os conceitos de núcleo e subnúcleo para o padrão Mestre/Escravo baseando-se no modelo PPOPP proposto. Posteriormente, um gerador de código paralelo implementando o comportamento do padrão em linguagem C usando Pthread POSIX fora criado, envolvendo questões de divisão do trabalho e exploração do paralelismo.

O segundo passo foi a validação do estudo de caso para a LED-PPOPP. Utilizou-se uma metodologia de experimentos para a medição do esforço de programação paralela e para a medição de desempenho, cujo objetivo era reduzir o esforço de programação sem que ocorressem grandes perdas de desempenho. Em virtude disso, o esforço foi medido com estudantes, coletando o tempo necessário para paralelizar uma aplicação com as abordagens Pthread e LED-PPOPP. Por outro lado, na medição de desempenho foram paralelizadas 5 aplicações de *benchmark* com a LED-PPOPP, para que então, fossem comparadas com as mesmas aplicações utilizando OpenMP.

Desta maneira, as seguintes questões de pesquisa foram respondidas:

1. **Qual é o impacto no desenvolvimento de *software* paralelo, utilizando a LED-PPOPP em relação a uma das soluções disponíveis atualmente na programação paralela?**

   Diante do impacto no desenvolvimento de *software*, o objetivo foi identificar se a LED-PPOPP consegue reduzir o esforço de programação paralela em relação a biblioteca Pthread. Os resultados mostraram que o esforço com Pthread é, em média, maior que 50% para paralelizar o algoritmo de multiplicação de matrizes. Com isso, a tendência é que desenvolver com LED-PPOPP requer menos esforço do programador do que utilizar Pthread.

2. **Qual é o impacto no desempenho de uma aplicação que utiliza a LED-PPOPP comparado a soluções disponíveis atualmente na exploração de paralelismo?**

   Para o impacto no desempenho de uma aplicação, buscou-se verificar se a LED-PPOPP possui uma diferença significativa em relação ao uso do OpenMP. Os resultados mostraram que nas 5 aplicações de *benchmark*, na maioria dos casos o desempenho ficou favorável para o OpenMP, mas o desempenho não foi significativamente diferente na maioria dos programas. Portanto, a tendência é que algoritmos paralelizados com a LED-PPOPP forneçam um desempenho próximo ao esperado com OpenMP.

Mesmo que o desempenho na maior parte das aplicações experimentadas tenha sido favorável ao OpenMP, os objetivos foram alcançados. O trabalho demonstrou que é possível reduzir o esforço de programação paralela sem comprometer o desempenho das aplicações que utilizam a LED-PPOPP. A Seção 8.1 descreve as contribuições desta pesquisa e a Seção 8.2 descreve os trabalhos futuros.

8.1 Contribuições

As principais contribuições deste trabalho são:

- **PPOPP**. Nesta dissertação criou-se um modelo para programação paralela orientada a padrões paralelos. A definição conceitual permite que outras interfaces possam utilizar esta abordagem.

- **LED-PPOPP**. Interface de programação disponível para programação paralela, possibilitando a paralelização de algoritmos em arquiteturas *multi-core* usando o padrão Mestre/Escravo.
- **A hipótese afirma que programar com a LED-PPOPP reduz o esforço de programação em relação à biblioteca Pthread.** A hipótese foi comprovada.
- **A hipótese afirma que o desempenho da LED-PPOPP não é significativamente diferente em relação à biblioteca OpenMP**. A hipótese foi comprovada.

8.2 Trabalhos Futuros

Considerando os benefícios proporcionados neste trabalho, utilizando o modelo de programação PPOPP, identificam-se os seguintes trabalhos futuros:

- Otimizar a exploração do paralelismo da LED-PPOPP, implementando opções de balanceamento de carga e escalonamento;
- Otimizar o analisador de código do compilador da LED-PPOPP, utilizando técnicas de análise de código;
- Comparar o esforço de programação da LED-PPOPP com outras abordagens de programação paralela;
- Comparar o desempenho da LED-PPOPP em outras aplicações e com outras abordagens;
- Medir a complexidade e o desempenho em função da utilização de subnúcleos de padrões paralelos;
- Implementar outros padrões paralelos na LED-PPOPP (por exemplo, *pipeline* e divisão e conquista);
- Realizar um estudo para a LED-PPOPP explorando o paralelismo em arquiteturas híbridas.

Referências Bibliográficas

[1] MATTSON G. T., SANDERS A. B., and MASSINGILL L. B. *Patterns for Parallel Programming*. Addison-Wesley, Boston, MA, 2005.

[2] FOWLER M. *Domain Spacific Languages*. Addison-Wesley Professional, Boston, MA, USA, 2010.

[3] KURBEL K. E. *The Making of Information Systems: Software Engineering and Management in a Globalized World*. Springer, Berlin, Heidelberg, German, 2008.

[4] GAMMA E., HELM R., JONHSON R., and VLISSIDES J. *Design Patterns: Elements os Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, 2002.

[5] INTEL and MCCOOL D. M. Structured Parallel Programming with Deterministic Patterns. In *HotPar-2nd USENIX Workshop on Hot Topics in Parallelism*, pages 1–6, Berkeley, CA, June 2010.

[6] GRAMA A., GUPTA A., KARYPIS G., and KUMAR V. *Introduction to Parallel Computing*. Pearson (Addison-Wesley), Boston, MA, 2003.

[7] CATANZARO R. and KEUTZER K. Parallel Computing with Patterns and Frameworks. *XRDS: Crossroads, The ACM Magazine for Students*, 17(1):22–27, 2010.

[8] GHOSH D. *DSLs in Action*. Manning, Stamford, CT, USA, 2011.

[9] JURISTO N. and MORENO A. M. *Basics of Software Engineering Experimentation*. Kluwer Academic Publishers, Boston, MA, USA, 2001.

[10] JOHNSON E. R. Frameworks = (Components + Patterns). *Commun. ACM*, 40(10):39–42, 1997.

[11] YANG-MING Z. and COCHOFF S.M. Medical Image Viewing on Multicore Platforms Using Parallel Computing Patterns. *IT Professional*, 12(2):33–41, 2010.

[12] RAEDER M., GRIEBLER D., BALDO L., and FERNANDES L. G. Performance Prediction of Parallel Applications with Parallel Patterns Using Stochastic Methods. In *Sistemas Computacionais (WSCAD-SSC), XII Simpósio em Sistemas Computacionais de Alto Desempenho*, pages 1–13, Vitória, Espírito Santo, Brasil, October 2011.

[13] BALDO L., BRENNER L., FERNANDES L. G., FERNANDES P., and SALES A. Performance Models For Master/Slave Parallel Programs. *Electron. Notes Theor. Comput. Sci.*, 128(4):101–121, 2005.

[14] RAEDER M., GRIEBLER D., RIBEIRO N., FERNANDES L. G., and CASTRO M. A Hybrid Parallel Version of ICTM for Cluster of NUMA Machines. In *IADIS International Conference Applied Computing*, pages 1–8, Rio de Janeiro, Brazil, November 2011.

[15] MASSINGILL B. L., MATTSON T. G., and SANDERS B. A. Reengineering for Parallelism: an Entry Point into PLPP (Pattern Language for Parallel Programming) for Legacy Applications. *Concurr. Comput. : Pract. Exper.*, 19(4):503–529, 2007.

[16] QUINN J. M. *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, New York, 2004.

[17] LEE E. A. The Problem with Threads. *IEEE Computer*, 39(5):33–42, 2006.

[18] SCHMIDT D., STAL M., ROHNERT H., and BUSCHMANN F. *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*. Willey and Sons, New York, 2000.

[19] RAUBER T. and RÜNGER G. *Parallel Programming for Multicore and Cluster Systems*. Springer, New York, 2010.

[20] MASSINGILL B. L., MATTSON T. G., and SANDERS B. A. Patterns for Parallel Application Programs. In *Pattern Languages of Programs'99*, pages 1–29, Monticello, Chicago, August 1999.

[21] BROMLING S., MACDONALD S., ANVIK J., SCHAEFFER J., SZAFRON D., and TAN K. Pattern-based Parallel Programming. In *Parallel Processing, 2002. Proceedings. International Conference on*, pages 257–265, Vancouver, Canada, August 2002.

[22] SCOTT L. R., CLARK T., and BAGHERI B. *Scientific Parallel Computing*. Princeton, New Jersey, 2005.

[23] NICBOLS B., BUTTLAR D., and FARRELL P. J. *Pthread Programming*. O'Reilly, Sebastopol, CA, 1996.

[24] FOSTER I. *Desining and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*. Addison-Wesley, Boston, MA, USA, 1995.

[25] REDDY M. *API Design for C++*. Elsevier, Burlington, MA, USA, 2011.

[26] FAYAD E. M., SCHMIDT C. D., and JOHNSON E. R. *Building Application Frameworks*. Wiley, New York, NY, USA, 1999.

[27] CHAPMAN B., JOST G., and PAS R. van der. *Using OpenMP*. Massachusetts Institute of Technology, London, England, 2008.

[28] REINDERS J. *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*. O'Reilly and Associates, Sebastopol, USA, 2007.

[29] C. E. LEISERSON. The Cilk++ Concurrency Platform. In *Proceedings of the 46th Annual Design Automation Conference*, pages 522–527, New York, NY, USA, July 2009.

[30] BADER D. A., KANADE V., and MADDURI K. SWARM: A Parallel Programming Framework for Multicore Processors. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8, Long Beach, California USA, March 2007.

[31] INTEL. Intel Software Network (Official page of Intel), Extracted from <http://software.intel.com/en-us/articles/intel-parallel-studio-home>. Access in November, 2011.

[32] KULKARNI M., CARRIBOULT P., PINGALI K., RAMANARAYANAN G., WALTER B., BALA K., and CHEW L. P. Scheduling Strategies for Optimistic Parallel Execution of Irregular Programs. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 217–228, New York, NY, USA, June 2008.

[33] ALDINUCCI M., RUGGIERI S., and TORQUATI M. Porting Decision Tree Algorithms to Multicore Using FastFlow. In *Proc. of European Conference in Machine Learning and Knowledge Discovery in Databases (ECML PKDD)*, pages 7–23, Barcelona, Spain, September 2010.

[34] LEWIS B. and BERG D. J. *Multithreaded Programming with Pthreads*. Sun Microsystems, California, USA, 1998.

[35] KIM W. and VOSS M. Multicore Desktop Programming with Intel® Threading Building Blocks. *Software, IEEE*, 28(1):23–31, 2011.

[36] WILLHALM T. and POPOVICI N. Putting Intel® Threading Building Blocks to Work. In *Proceedings of the 1st international workshop on Multicore software engineering*, page 2, New York, NY, USA, May 2008.

[37] LEISERSON C. E. and MIRMAN I. B. *How to Survive the Multicore Software Revolution*. Cilk Arts, e-Book, 2008.

[38] BADER D. A., KANADE V., and MADDURI K. Software and Algorithms for Running on Multicore (Home Page), Extracted from <http://multicore-swarm.sourceforge.net>. Access in November, 2011.

[39] MÉNDEZ-LOJO M., NGUYEN D., PROUNTZOS D., SUI X., HASSAAN M. A., KULKARNI M., BURTSCHER M., and PINGALI K. Structure-Driven Optimizations for Amorphous Data-Parallel Programs. In *Proceedings of the 15th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 3–14, New York, NY, USA, January 2010.

[40] GALOIS. Documentation - Getting Started (Official page of Galois), Extracted from <http://iss.ices.utexas.edu/?p=projects/galois/doc/current/getting_started>. Access in December, 2011.

[41] ALDINUCCI M. and TORQUATI M. Fastflow Framework Parallel Programming (Home Page), Extracted from <http://calvados.di.unipi.it/dokuwiki/doku.php?id=ffnamespace:about>. Access in November, 2011.

[42] AHO A. V., SETHI R., and ULLMAN L. D. *Compiladores Princípios, Técnicas e Ferramentas*. LTC, Rio de Janeiro, RJ, Brazil, 1995.

[43] FIELD A. *Discovering Statistics Using SPSS*. SAGE, Dubai, EAU, 2009.

[44] TRIOLA F. M. *Introdução à Estatística*. LTC, Rio de Janeiro, RJ, BR, 2005.

[45] BURKARDT J. John Burkardt (Personal Page), Extracted from <http://people.sc.fsu.edu/ jburkardt/>. Access in December, 2011.

[46] BURKARDT J. Estimate an Integral Over a 2D Domain Using OpenMP (Source Code), Extracted from <http://people.sc.fsu.edu/ jburkardt/c_src/quad2d_openmp/quad2d_openmp.c>. Access in December, 2011.

[47] BURKARDT J. Fast Fourier Transform Using OpenMP (Source Code), Extracted from <http://people.sc.fsu.edu/ jburkardt/c_src/fft_openmp/fft_openmp.c>. Access in December, 2011.

[48] BURKARDT J. Molecular Dynamics Using OpenMP (Source Code), Extracted from <http://people.sc.fsu.edu/ jburkardt/c_src/md_openmp/md_openmp.c>. Access in December, 2011.

[49] BURKARDT J. Matrix Multiplication with OpenMP (Source Code), Extracted from <http://people.sc.fsu.edu/ jburkardt/c_src/mxm_openmp/mxm_openmp.c>. Access in December, 2011.

[50] BURKARDT J. Count Primes Using OpenMP (Source Code), Extracted from <http://people.sc.fsu.edu/ jburkardt/c_src/prime_openmp/prime_openmp.c>. Access in December, 2011.

[51] CASTRO M., FERNANDES L. G., POUSA C., MEHAUT J. F., and AGUIAR M. S. NUMA-ICTM: A Parallel Version of ICTM Exploiting Memory Placement Strategies for NUMA Machines. In *Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pages 1–8, Washington, DC, USA, May 2009.

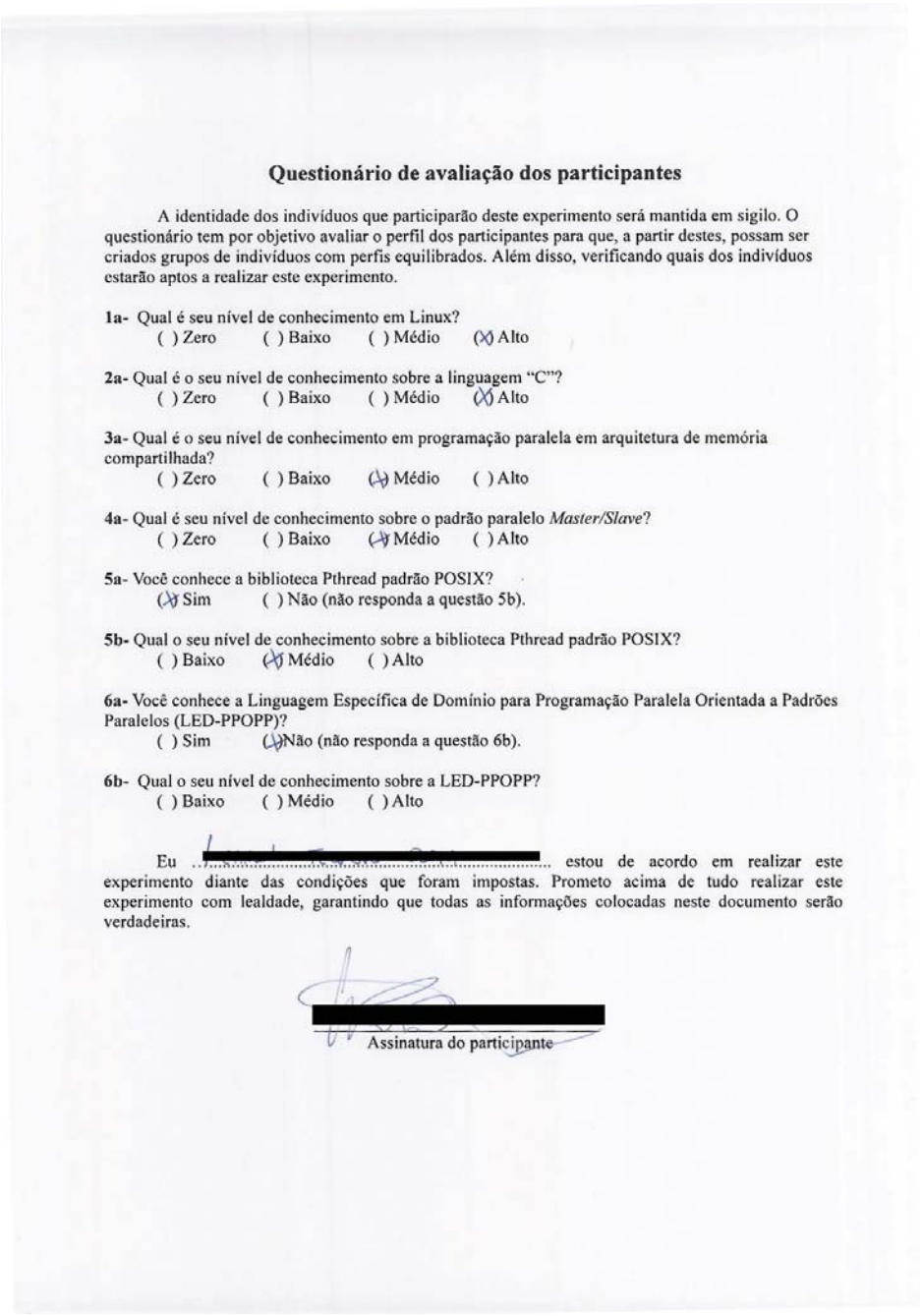
[52] HANAWA T., SATA M., LEE J., IMADA T., KIMURA H., and BOKU T. Evaluation of Multicore Processors for Embedded Systems by Parallel Benchmark Program Using OpenMP. In *Proceedings of the 5th International Workshop on OpenMP: Evolving OpenMP in an Age of Extreme Parallelism*, pages 15–27, Berlin, Heidelberg, May 2009.

[53] GRIEBLER D. Benchmarks LED-PPOPP, Avaliable in <http://www.inf.pucrs.br/gmap/ledppopp.html>. Access in Janeiro, 2012.

A. APÊNDICE

A.1 Questionários

Esta seção apresenta os questionários utilizados no experimento para medição do esforço.



Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo (X) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
( ) Sim (X) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
( ) Sim (X) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

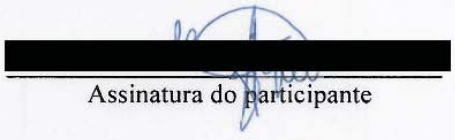
5a- Você conhece a biblioteca Pthread padrão POSIX?
( ) Sim (X) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu HE[redacted]HO..... estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.



Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo ( ) Médio (X) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] ..... estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo (X) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo (X) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo (X) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo (X) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
( ) Baixo (X) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X)Não (não responda a questão 6b).

6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted] estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

Questionário de avaliação dos participantes

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. O questionário tem por objetivo avaliar o perfil dos participantes para que, a partir destes, possam ser criados grupos de indivíduos com perfis equilibrados. Além disso, verificando quais dos indivíduos estarão aptos a realizar este experimento.

1a- Qual é seu nível de conhecimento em Linux?
( ) Zero ( ) Baixo (X) Médio ( ) Alto

2a- Qual é o seu nível de conhecimento sobre a linguagem "C"?
( ) Zero ( ) Baixo ( ) Médio (X) Alto

3a- Qual é o seu nível de conhecimento em programação paralela em arquitetura de memória compartilhada?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

4a- Qual é seu nível de conhecimento sobre o padrão paralelo *Master/Slave*?
( ) Zero (X) Baixo ( ) Médio ( ) Alto

5a- Você conhece a biblioteca Pthread padrão POSIX?
(X) Sim ( ) Não (não responda a questão 5b).

5b- Qual o seu nível de conhecimento sobre a biblioteca Pthread padrão POSIX?
(X) Baixo ( ) Médio ( ) Alto

6a- Você conhece a Linguagem Específica de Domínio para Programação Paralela Orientada a Padrões Paralelos (LED-PPOPP)?
( ) Sim (X) Não (não responda a questão 6b).

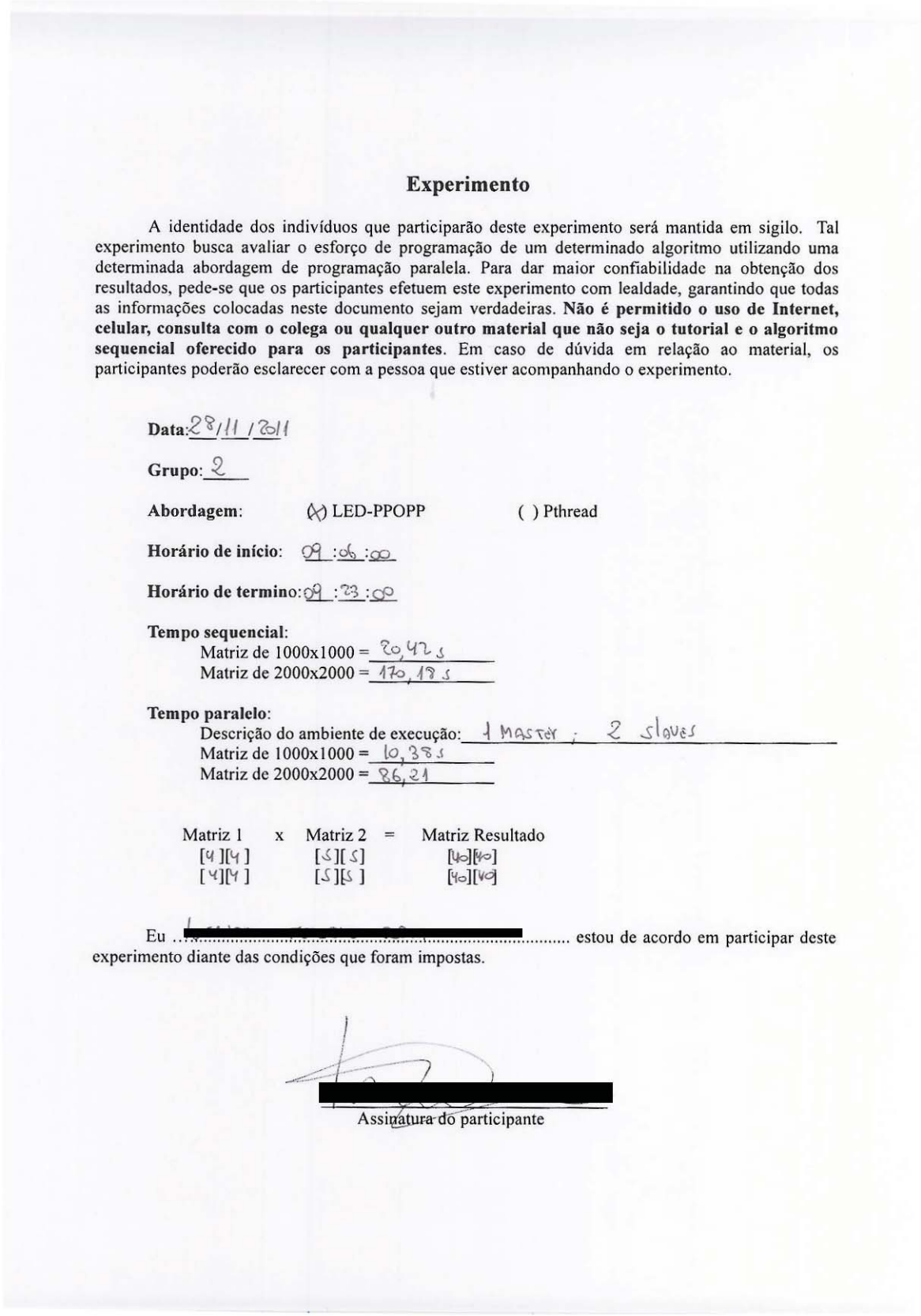
6b- Qual o seu nível de conhecimento sobre a LED-PPOPP?
( ) Baixo ( ) Médio ( ) Alto

Eu [redacted]..... estou de acordo em realizar este experimento diante das condições que foram impostas. Prometo acima de tudo realizar este experimento com lealdade, garantindo que todas as informações colocadas neste documento serão verdadeiras.

[redacted]
Assinatura do participante

A.2 Formulários

Esta seção apresenta os formulários utilizados no experimento para medição do esforço.



Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 28/11/2011

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 09:06:00

Horário de termino: 09:23:00

Tempo sequencial:
Matriz de 1000x1000 = 20,42 s
Matriz de 2000x2000 = 170,18 s

Tempo paralelo:
Descrição do ambiente de execução: 1 master ; 2 slaves
Matriz de 1000x1000 = 10,38 s
Matriz de 2000x2000 = 86,21

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/11

Grupo: 01

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 18:13:__

Horário de termino: 19:31:__

Tempo sequencial:
Matriz de 1000x1000 = 20.3872 segundos.
Matriz de 2000x2000 = ~~20.3726~~ 169.6947 segundos.

Tempo paralelo:
Descrição do ambiente de execução: 2 Thread
Matriz de 1000x1000 = 10.57 segundos
Matriz de 2000x2000 = 87.5232 segundos

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 21 / 11 / 2024

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 07 : 31 : ___

Horário de termino: 07 : 43 : ___

Tempo sequencial:
Matriz de 1000x1000 = 20.45
Matriz de 2000x2000 = 169.12

Tempo paralelo:
Descrição do ambiente de execução: 1 MASTER AND 2 SLAVES
Matriz de 1000x1000 = 10.42
Matriz de 2000x2000 = 87.10

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25 / 11 / 2011

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 12 : 16 : 30

Horário de termino: 13 : 10 : 00

Tempo sequencial:
Matriz de 1000x1000 = 20,43 s
Matriz de 2000x2000 = 169,57

Tempo paralelo:
Descrição do ambiente de execução: _____
Matriz de 1000x1000 = 10,60 s
Matriz de 2000x2000 = 88,23 s

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/11

Grupo: 2

Abordagem: (x) LED-PPOPP ( ) Pthread

Horário de início: 10:16:00

Horário de termino: 10:31:00

Tempo sequencial:
Matriz de 1000x1000 = 20.963176
Matriz de 2000x2000 = 173.017124

Tempo paralelo:
Descrição do ambiente de execução: 17 THREAD Master - 2 slave
Matriz de 1000x1000 = 10.653770
Matriz de 2000x2000 = 89.269131

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[4][4]
[4][4]		[5][5]		[4][4]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/2021

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10:17:__

Horário de termino: 10:32:__

Tempo sequencial:
Matriz de 1000x1000 = 20,88 6623
Matriz de 2000x2000 = 174,09 7214

Tempo paralelo:
Descrição do ambiente de execução: 2 thread
Matriz de 1000x1000 = 10,7066 17
Matriz de 2000x2000 = 88,44 3370

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[20][40]
[4][4]		[5][5]		[40][80]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/2011

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10 : 17

Horário de termino: 10 : 32 :

Tempo sequencial:
Matriz de 1000x1000 = 20.86
Matriz de 2000x2000 = 173.34

Tempo paralelo:
Descrição do ambiente de execução: 2 threads
Matriz de 1000x1000 = 10.15
Matriz de 2000x2000 = 33.12

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[3][3]		[60][60]
[4][4]		[5][5]		[60][60]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/11

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10:16:00

Horário de termino: 10:33:00

Tempo sequencial:
Matriz de 1000x1000 = 20,95 seg.
Matriz de 2000x2000 = 173,88 seg.

Tempo paralelo:
Descrição do ambiente de execução: 1 master e 2 slaves
Matriz de 1000x1000 = 10,66 seg.
Matriz de 2000x2000 = 88,88 seg.

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/2011

Grupo: 2

Abordagem: (x) LED-PPOPP ( ) Pthread

Horário de início: 10:16:30

Horário de termino: 10:40:10

Tempo sequencial:
Matriz de 1000x1000 = 20.87
Matriz de 2000x2000 = 173.67

Tempo paralelo:
Descrição do ambiente de execução: 01 thread master 02 slaves
Matriz de 1000x1000 = 10.32
Matriz de 2000x2000 = 89.18

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/11

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10:19:54

Horário de termino: 10:31:58

Tempo sequencial:
Matriz de 1000x1000 = 20.865373
Matriz de 2000x2000 = ~~219.131273~~, digo, 173.108099

Tempo paralelo:
Descrição do ambiente de execução: 1 thread Master e 2 Threads Slaves
Matriz de 1000x1000 = 10.671923
Matriz de 2000x2000 = ~~176.568100~~, digo, 88.414647

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[10][10]
[4][4]		[5][5]		[10][10]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/11

Grupo: 02

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10:13:50

Horário de termino: 10:40:40

Tempo sequencial:
Matriz de 1000x1000 = 20.201281
Matriz de 2000x2000 = 173.445228

Tempo paralelo:
Descrição do ambiente de execução: 1 thread master e 2 threads Slave
Matriz de 1000x1000 = 10.277000.3
Matriz de 2000x2000 = 89.831133

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/11

Grupo: 2

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10 : 18 : __

Horário de termino: 10 : 40 : __

Tempo sequencial:
Matriz de 1000x1000 = 20.848569
Matriz de 2000x2000 = 173.360639

Tempo paralelo:
Descrição do ambiente de execução: 1 thread master 2 slave
Matriz de 1000x1000 = 10.645485
Matriz de 2000x2000 = 88.432274

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25 / 11 / 21

Grupo: 01

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10 :08:___

Horário de termino: 10 :47:___

Tempo sequencial:
Matriz de 1000x1000 = 20,945434
Matriz de 2000x2000 = 173,713544

Tempo paralelo:
Descrição do ambiente de execução: 2 THREADS
Matriz de 1000x1000 = 10,627872
Matriz de 2000x2000 = 89,345479

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25 / 11 / 21

Grupo: 1

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10 : 07 :

Horário de termino: 10 : 52 :

Tempo sequencial:
Matriz de 1000x1000 = 20.865859
Matriz de 2000x2000 = 774.054774

Tempo paralelo:
Descrição do ambiente de execução: 2 cores
Matriz de 1000x1000 = 10.618363
Matriz de 2000x2000 = 88.225184

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[20][44]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25 / 11 / 11

Grupo: 1

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 16 : 10 : 00

Horário de termino: 10 : 20 : 00

Tempo sequencial:
Matriz de 1000x1000 = 21,07 segundos
Matriz de 2000x2000 = 174,29 segundos

Tempo paralelo:
Descrição do ambiente de execução: 2 THREADS (DUAL CORE)
Matriz de 1000x1000 = 11,48
Matriz de 2000x2000 = 91,54

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes.** Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/2011

Grupo: 01

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10:10:00

Horário de termino: 10:40:00

Tempo sequencial:
Matriz de 1000x1000 = 21.1728
Matriz de 2000x2000 = 174.0391

Tempo paralelo:
Descrição do ambiente de execução: 1 processador, duas threads.
Matriz de 1000x1000 = 11.4753
Matriz de 2000x2000 = 88.1833

Matriz 1	x	Matriz 2	=	Matriz Resultado
[2][2]		[2][2]		[10][10]
[2][2]		[2][2]		[10][10]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25 / 11 / 2011

Grupo: 1

Abordagem: ( ) LED-PPOPP (x) Pthread

Horário de início: 10 : 07 :

Horário de termino: 10 : 34 :

Tempo sequencial:
Matriz de 1000x1000 = 20.85
Matriz de 2000x2000 = 173,50

Tempo paralelo:
Descrição do ambiente de execução: 2 THREADS
Matriz de 1000x1000 = 10.66
Matriz de 2000x2000 = 88.39

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes.** Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 25/11/2011

Grupo: 1

Abordagem: ( ) LED-PPOPP (x) Pthread

Horário de início: 10:08:__

Horário de termino: 11:07:__

Tempo sequencial:
Matriz de 1000x1000 = 20.861384
Matriz de 2000x2000 = 173.333866

Tempo paralelo:
Descrição do ambiente de execução: 2 Threads
Matriz de 1000x1000 = 10.650791
Matriz de 2000x2000 = 89.040973

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/2011

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 07:04:___

Horário de termino: 08:05:___

Tempo sequencial:
Matriz de 1000x1000 = 20.405
Matriz de 2000x2000 = 169.485

Tempo paralelo:
Descrição do ambiente de execução: 2 THREADS
Matriz de 1000x1000 = 9,81
Matriz de 2000x2000 = 82,24

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[20][20]
[4][4]		[5][5]		[20][20]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/11

Grupo: 1

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 18:14:00

Horário de termino: 19:19:00

Tempo sequencial:
Matriz de 1000x1000 = 20.37
Matriz de 2000x2000 = 169.07

Tempo paralelo:
Descrição do ambiente de execução: 2 threads slaves
Matriz de 1000x1000 = 10.57
Matriz de 2000x2000 = 87.89

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/2011

Grupo: 1

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 18:14:___

Horário de termino: 18:47:___

Tempo sequencial:
Matriz de 1000x1000 = 20.492554
Matriz de 2000x2000 = 170.246234

Tempo paralelo:
Descrição do ambiente de execução: 2 threads
Matriz de 1000x1000 = 10.380807
Matriz de 2000x2000 = 86.308232

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23 / 11 / 11

Grupo: 1

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 18 : 12 : 00

Horário de termino: 18 : 41 : 00

Tempo sequencial:
Matriz de 1000x1000 = 23.10
Matriz de 2000x2000 = 169.22

Tempo paralelo:
Descrição do ambiente de execução: 2 threads da lib pthread
Matriz de 1000x1000 = 10.57
Matriz de 2000x2000 = 87.62

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[20][20]
[4][4]		[5][5]		[20][20]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 22/11/11

Grupo: 01

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 18:35:00

Horário de termino: 19:06:00

Tempo sequencial:
Matriz de 1000x1000 = 20.3855
Matriz de 2000x2000 = 162.4626

Tempo paralelo:
Descrição do ambiente de execução: 1- master, 3- slave
Matriz de 1000x1000 = 6.9423
Matriz de 2000x2000 = 59.40

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][5]		[4][5]		[40][40]
[4][5]		[4][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 21 / 11 / 11

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 08 : 59 : 00

Horário de termino: 09 : 28 : 00

Tempo sequencial:
Matriz de 1000x1000 = 20,37 seg
Matriz de 2000x2000 = 169,80 seg

Tempo paralelo:
Descrição do ambiente de execução: 2 THREADS SLAVE E 1 MASTER
Matriz de 1000x1000 = 10,36 seg
Matriz de 2000x2000 = 86,22 seg

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 21/11/11

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 8:59:__

Horário de termino: 9:27:__

Tempo sequencial:
Matriz de 1000x1000 = 20.506236
Matriz de 2000x2000 = 170.397181

Tempo paralelo:
Descrição do ambiente de execução: 3 threads → 1 mestre e 2 escravos
Matriz de 1000x1000 = 10.436393
Matriz de 2000x2000 = 86.912092

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][60]
[4][4]		[5][5]		[40][60]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 22/11/2011

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 18:32:__

Horário de termino: 18:55:__

Tempo sequencial:
Matriz de 1000x1000 = 20.398205
Matriz de 2000x2000 = 169,475925

Tempo paralelo:
Descrição do ambiente de execução: Executei com 1 Master e 2 slaves
Matriz de 1000x1000 = 10.380944
Matriz de 2000x2000 = 86.089125 (86.089125)

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 22 / 11 / 12

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 18 : 31 : 00

Horário de termino: 18 : 56 : 00

Tempo sequencial:
Matriz de 1000x1000 = 20.4465
Matriz de 2000x2000 = 174.6959

Tempo paralelo:
Descrição do ambiente de execução: 1 master e 2 slaves
Matriz de 1000x1000 = 10.423
Matriz de 2000x2000 = 87.056

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 22/11/11

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 18:33:00

Horário de termino: 19:06:00

Tempo sequencial:
Matriz de 1000x1000 = 20.37
Matriz de 2000x2000 = 169.16

Tempo paralelo:
Descrição do ambiente de execução: MASTER + 2 SLAVES, ALOCAÇÃO ESTÁTICA
Matriz de 1000x1000 = 6.35
Matriz de 2000x2000 = 49.18

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/11

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10:24:00

Horário de termino: 11:24:00

Tempo sequencial:
Matriz de 1000x1000 = 20,92 segundos
Matriz de 2000x2000 = 173,34 segundos

Tempo paralelo:
Descrição do ambiente de execução: duas threads
Matriz de 1000x1000 = 10,65
Matriz de 2000x2000 = 88,31

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23 / 11 / 2011

Grupo: 2

Abordagem: ( ) LED-PPOPP (x) Pthread

Horário de início: 10 : 24 : 17

Horário de termino: 11 : 25 : 20

Tempo sequencial:
Matriz de 1000x1000 = 20.87
Matriz de 2000x2000 = 173.77

Tempo paralelo:
Descrição do ambiente de execução: 2 threads
Matriz de 1000x1000 = 10.67
Matriz de 2000x2000 = 89.14

Matriz 1	x	Matriz 2	=	Matriz Resultado
[1][4]		[6][5]		[10][10]
[1][4]		[5][5]		[10][10]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/11

Grupo: 02

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10:27:19

Horário de termino: 11:30:54

Tempo sequencial:
Matriz de 1000x1000 = 20.991740
Matriz de 2000x2000 = 173.760258

Tempo paralelo:
Descrição do ambiente de execução: 2 threads
Matriz de 1000x1000 = 10.618466
Matriz de 2000x2000 = 89.342385

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/11

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10:27:19

Horário de termino: 11:32:17

Tempo sequencial:
Matriz de 1000x1000 = 20.885995
Matriz de 2000x2000 = 173.561523

Tempo paralelo:
Descrição do ambiente de execução: 2 Thread
Matriz de 1000x1000 = 10.658233
Matriz de 2000x2000 = 88.627963

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[46][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23 / 11 / 11

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10 : 24 : ___

Horário de termino: 11 : 30 : ___

Tempo sequencial:
Matriz de 1000x1000 = 20.889658 segundos
Matriz de 2000x2000 = 173.189757 segundos

Tempo paralelo:
Descrição do ambiente de execução: 2 thread
Matriz de 1000x1000 = 10.609559 segundos
Matriz de 2000x2000 = 85.147147 segundos

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[5][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23 / 11 / 2013

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10 : 23

Horário de termino: 11 : 33 :

Tempo sequencial:
Matriz de 1000x1000 = 30,908648
Matriz de 2000x2000 = 173,895509

Tempo paralelo:
Descrição do ambiente de execução: 2 threads
Matriz de 1000x1000 = 10,672309
Matriz de 2000x2000 = 89,062137

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/11

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10:23:00

Horário de termino: 11:34:00

Tempo sequencial:
Matriz de 1000x1000 = 20.846755
Matriz de 2000x2000 = 273.966764

Tempo paralelo:
Descrição do ambiente de execução: 2 7 Hreads
Matriz de 1000x1000 = 10.5532
Matriz de 2000x2000 = 87.1621

Matriz 1	x	Matriz 2	=	Matriz Resultado
[M][9]		[S][S]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/2012

Grupo: 2

Abordagem: ( ) LED-PPOPP (X) Pthread

Horário de início: 10:20:10

Horário de termino: 10:55:20

Tempo sequencial:
Matriz de 1000x1000 = 20.8
Matriz de 2000x2000 = 173.5

Tempo paralelo:
Descrição do ambiente de execução: 2 THREADS
Matriz de 1000x1000 = 10.6
Matriz de 2000x2000 = 88.3

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][8]		[8][5]		[40][40]
[4][4]		[8][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/2011

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10:22:__

Horário de termino: 10:50:__

Tempo sequencial:
Matriz de 1000x1000 = 20.862464 seg
Matriz de 2000x2000 = 173,663857

Tempo paralelo:
Descrição do ambiente de execução: Mestre/Escravo
Matriz de 1000x1000 = 12.028763
Matriz de 2000x2000 = 88.566652

Matriz 1	x	Matriz 2	=	Matriz Resultado
[2][2]		[3][3]		[2][2]
[2][2]		[3][3]		[2][2]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23 / 11 / 18

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10 : 21 : ___

Horário de termino: 10 : 46 : ___

Tempo sequencial:
Matriz de 1000x1000 = ~~21 segundos~~ 20,803126
Matriz de 2000x2000 = ~~173 segundos~~ 172,850791

Tempo paralelo:
Descrição do ambiente de execução: 1 MESTRE e 2 ESCRAVOS
Matriz de 1000x1000 = 10,677153
Matriz de 2000x2000 = 88,505785

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23 / 11 / 2011

Grupo: 1

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10 : 20 : ___

Horário de termino: 10 : 49 : ___

Tempo sequencial:
Matriz de 1000x1000 = 20,98
Matriz de 2000x2000 = 173,54

Tempo paralelo:
Descrição do ambiente de execução: 1 MÁQUINA + 2 THREADS.
Matriz de 1000x1000 = 10,77
Matriz de 2000x2000 = 89,57

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

Experimento

A identidade dos indivíduos que participarão deste experimento será mantida em sigilo. Tal experimento busca avaliar o esforço de programação de um determinado algoritmo utilizando uma determinada abordagem de programação paralela. Para dar maior confiabilidade na obtenção dos resultados, pede-se que os participantes efetuem este experimento com lealdade, garantindo que todas as informações colocadas neste documento sejam verdadeiras. **Não é permitido o uso de Internet, celular, consulta com o colega ou qualquer outro material que não seja o tutorial e o algoritmo sequencial oferecido para os participantes**. Em caso de dúvida em relação ao material, os participantes poderão esclarecer com a pessoa que estiver acompanhando o experimento.

Data: 23/11/2011

Grupo: 01

Abordagem: (X) LED-PPOPP ( ) Pthread

Horário de início: 10:22:00

Horário de termino: 10:45:00

Tempo sequencial:
Matriz de 1000x1000 = 24.0056
Matriz de 2000x2000 = 173.7979

Tempo paralelo:
Descrição do ambiente de execução: 1 Mestre / 2 Escravos
Matriz de 1000x1000 = 10.8681
Matriz de 2000x2000 = 89.3443

Matriz 1	x	Matriz 2	=	Matriz Resultado
[4][4]		[5][5]		[40][40]
[4][4]		[5][5]		[40][40]

Eu [redacted] estou de acordo em participar deste experimento diante das condições que foram impostas.

[redacted]
Assinatura do participante

A.3 Manual LED-PPOPP

Autor: Dalvan Jair Griebler (GRIEBLER D. J.)

Programação Paralela Orientada a Padrões Paralelos (PPOPP)

1. INTRODUÇÃO

Padrão de projeto é uma solução genérica para resolver problemas encontrados frequentemente na computação para um domínio específico. Inicialmente introduzido na Engenharia de Software no cenário de programação orientada a objetos, uma de suas necessidades era estruturar classes em estruturas de dados heterogêneas. Esta descrição em alto nível possibilita que soluções inteligentes sejam utilizadas por outros desenvolvedores de software [3], [4].

Partindo desta premissa de reutilização de soluções inteligentes, padrões de linguagem foram propostos no contexto de programação paralela. Estes estão divididos em quatro espaços de projeto (encontrando concorrência, estrutura de algoritmo, estrutura de apoio e mecanismos de implementação), a fim de tornar mais fácil o entendimento do problema a ser resolvido. Assim, guiando-se através de um padrão de linguagem é possível obter uma avaliação qualitativa de diferentes padrões de programação paralela [4], [5], [6] .

Padrões paralelos definem a estrutura de um programa paralelo em um nível mais alto de abstração. No cenário atual, eles são utilizados para modelar algoritmos, os quais têm a finalidade de explorar o paralelismo disponível das arquiteturas paralelas. A abordagem de paralelismo é implementada por meio de padrões que exploram a decomposição de dados e tarefas. Para tanto, a definição genérica permite que os padrões paralelos sejam implementados em um vasto conjunto de aplicações [2], [7], [8], [4], [9], [10], [11] . Existem vários padrões de programação paralela definidos na literatura (i.e., Mestre/Escravo, Divisão e Conquista, *Pipeline*), destes, inicialmente, a Linguagem Específica de Domínio (LED) [1], foi implementada apenas com o padrão Mestre/Escravo.

1.1 Mestre/Escravo

O padrão Mestre/Escravo é uma solução baseada no paralelismo de tarefas, onde uma tarefa mestre possui instâncias de uma ou mais tarefas escravas [4]. Conforme o exemplo da Figura A.1, a tarefa mestre (M) inicializa a computação dividindo-a em tarefas independentes. Isso irá resultar em um saco de tarefas, as quais são enviadas aos escravos (S) para serem computadas. Ao resolverem a computação os escravos retornam o resultado para o mestre, o qual une os resultados e processa a solução final do problema.

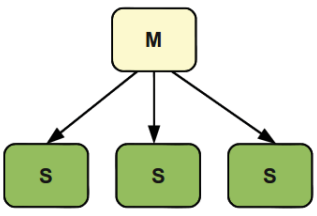


Figura A.1: Padrão *Mestre/Escravo*.

Soluções que possibilitam modelar a computação em tarefas independentes (e.g., paralelismo de laço) podem apresentar bom desempenho na exploração de paralelismo com este padrão paralelo.

No entanto, para que a implementação apresente bons resultados é preciso dividir o problema adequadamente. No padrão Mestre/Escravo existe um comunicador global para resolver dependências entre as tarefas e distribuí-las. Em alguns casos, quando a comunicação é bastante requisitada é comum acontecer uma sobrecarga de comunicação, o que afeta o desempenho do sistema. A solução para este tipo de problema é aumentar o tamanho das tarefas escravas para diminuir o número de acessos ao saco de tarefas.

LED-PPOPP (Linguagem Específica de Domínio de Programação Paralela Orientada a Padrões Paralelos)

LED é uma linguagem de programação direcionada para um domínio específico [13] . As linguagens normalmente utilizadas são de propósito geral, como por exemplo C, C++ e Java. Uma LED contém sintaxe e semântica no mesmo nível de abstração que o domínio do problema oferece. Um exemplo de domínio é a programação paralela, para o qual é fornecida uma linguagem orientada a padrões paralelos, dos quais o desenvolvedor tem entendimento do funcionamento. Neste sentido, a LED é responsável pela abstração dos detalhes de implementação dos padrões.

Além de fornecer uma linguagem específica para o domínio de programação paralela para arquiteturas multi-core, a LED-PPOPP também é aliada a uma linguagem de programação de propósito geral, a linguagem C. Assim, o desenvolvimento dos algoritmos é realizado utilizando a linguagem C através da interface LED-PPOPP.

2. INTERFACE LED-PPOPP

A interface LED-PPOPP é implementada orientada ao padrão *Mestre/Escravo*. Para tanto, é fornecida uma linguagem coerente a este padrão. A proposta é que o programador trabalhe com núcleos e subnúcleos de padrões paralelos, os quais são definidos através de uma função e nela são fornecidos blocos (para descrever e modelar o código) correspondentes ao padrão paralelo da função. Desta forma, o programador pode modelar sua aplicação paralela orientada a um padrão.

A Figura A.2 ilustra o modelo de programação proposto no LED-PPOPP. O núcleo principal refere-se ao padrão paralelo que a aplicação será implementada. Sabe-se que podem haver diferentes tipos de computação dentro de um programa. Nem sempre é possível obter bom desempenho com um único padrão. Padrões normalmente resolvem uma computação específica. Com a implementação de subnúcleos é possível obter maior flexibilidade para decompor tarefas paralelas. Entretanto, nem sempre é possível obter bons resultados de desempenho utilizando este tipo de construção. O ideal é que construções de subnúcleo sejam usadas somente quando o grão da tarefa é muito grande para ser paralelizado, a fim de evitar sobrecarga de trabalho.

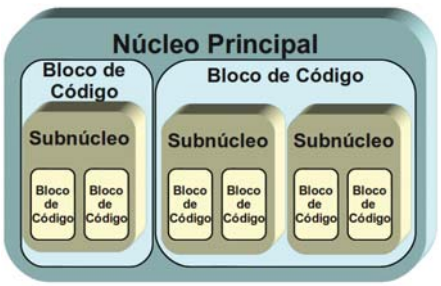


Figura A.2: Proposta de modelo de programação com PPOPP.

A linguagem LED-PPOPP é reconhecida através da declaração do cabeçalho "ppoppLinux.h" (por exemplo, *#include <ppoppLinux.h>*), operando em conjunto com a linguagem C . Para diferenciar as linguagens a interface utiliza caracteres especiais para facilitar a operação com núcleos, blocos e primitivas. Sendo assim, os núcleos são inicializado com "$", os blocos são inicializam com "@" e as primitivas são inicializadas com "_&", as definições utilizadas são as seguintes:

$**mainMasterSlavePattern(**$int\ num_threads$**)**{} -> Núcleo principal

O núcleo principal é a função principal da linguagem LED-PPOPP e este pode ser declarado apenas uma vez no programa, onde é possível configurar a quantidade de *threads* que este núcleo irá trabalhar (sempre considerando que uma *thread* é destinada ao bloco *Master*). O código e os algoritmos são modelados dentro dos respectivos blocos, pertencentes a este núcleo:

@**Master_**{} -> Bloco de código Mestre do núcleo principal.

Este bloco obrigatoriamente deve ser declarado dentro do núcleo principal e antes dos blocos Slave_{}. O bloco também é responsável pela coordenação e sincronização dos blocos Slave_{}. O bloco pode ser chamado uma única vez, e as declarações permitidas dentro do bloco Master_{} são: blocos Slave_{}, subnúcleos e primitivas de sincronização e proteção de dados.

@**Slave_**{} -> Bloco de código Escravo do núcleo principal.

A computação contida no bloco é replicada conforme a quantidade de *threads* declaradas no núcleo principal. Assim, todas as declarações dentro do bloco **Slave_**{} serão executadas independentemente em cada *thread*. Somente declarações globais podem ser compartilhadas entre as *threads* escravas. Se existirem laços "**For**", estes são paralelizados automaticamente. No entanto, se existirem operações de escrita em declarações globais, o programador deve assegurar que não ocorram conflitos no acesso a memória, utilizando a primitiva de proteção de dados. Quando as declarações globais forem do tipo vetores e matrizes e estiverem dentro de laços "**For**" não é necessário a utilização de primitiva de proteção de dados desde que não seja forçada a escrita de dados no mesmo índice do vetor. Computações dentro de laços "**For**" com declarações globais que não trabalham com índices não são passíveis de serem separadas em tarefas independentes. Sendo assim, o programador deve se precaver da confiabilidade da execução daquela operação.

A declaração deste bloco pode ser realizada várias vezes dentro do núcleo principal, desde que a chamada esteja após ou dentro do bloco **Master_**{}. Quando um bloco **Slave_**{} é declarado dentro do bloco **Master_**{} o que estiver declarado naquele bloco **Slave_**{} será compartilhado com o **Master_**{}. Já as declarações externas não serão compartilhadas com o **Master_**{}, quando estas existirem, para que possam começar o seu trabalho, precisarão de um sinal de sincronização que é realizado no bloco **Master_**{}, mediante a primitiva de sincronização de escravos. Além disso, a execução dos blocos **Slave_**{} é realizada conforme a ordem que eles estão declarados no núcleo principal. Subnúcleos também podem ser declarados dentro de blocos **Slave_**{}.

Para melhorar o entendimento da LED-PPOPP na resolução de uma determinada computação, a Figura A.3 exemplifica como ocorrer o paralelismo e a execução do padrão *Mestre/Escravo*. Neste caso, o núcleo está recebendo por parâmetro 4 *threads*, como descrito anteriormente, uma destas 4 é o process mestre. Sendo assim, a process **Master** é o responsável pela criação e a junção dos resultados das outras 3 *threads* derivadas do bloco **Slave**. O código contido no bloco Slave é replicado para todas as *threads* escravas, porém, se o bloco **Slave** estiver dentro do bloco **Master**, o código deste bloco também é replicado para o mestre para ajudar a resolver a computação (isso pode

ser uma boa alternativa para não deixar o mestre ocioso enquanto os escravos estão trabalhando).

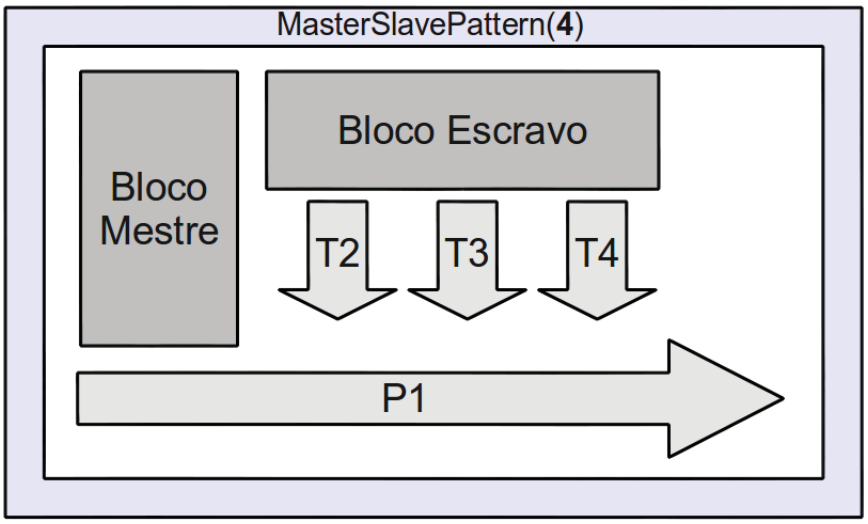


Figura A.3: Resolução da computação com o padrão Mestre/Escravo na LED-PPOPP.

O padrão *Mestre/Escravo* é comumente implementado em ambientes distribuído de troca de mensagens. Mas nesta ocasião, está se trabalhando com ambiente com memória compartilhada, sendo assim, este é o meio de comunicação dos escravos com o mestre.

$\$$**MasterSlavePattern(**$int\ num\_threads$**)**{} -> Subnúcleo

A funcionalidade deste subnúcleo é a mesma do núcleo principal, porém, um subnúcleo pode ser declarado várias vezes em um programa, desde que ele pertença ao núcleo principal. Um subnúcleo não pode ser declarado dentro de outro, pois este é o maior nível atingível na modelagem da LED-PPOPP. A estruturação é realizada a partir dos seguintes blocos:

@**Master**{} -> Bloco de código Mestre do subnúcleo

Este bloco pertence ao subnúcleo, sendo unicamente declarado dentro do subnúcleo e antes dos blocos **Slave**{}. O bloco também é responsável pela coordenação e sincronização dos blocos **Slave**{}. O bloco pode ser chamado uma única vez e as declarações permitidas dentro do bloco **Master**{} são: blocos **Slave**{} e primitivas de sincronização e proteção de dados.

@**Slave**{} -> Bloco de código Escravo do subnúcleo

A funcionalidade deste bloco é semelhante ao bloco **Slave_**{} que pertence ao núcleo principal. Entretanto, a restrição é que o subnúcleo não pode ser declarado dentro deste bloco, mas este bloco deve ser declarado em um subnúcleo.

&SynchronizeSlaves; -> Primitiva de Sincronização dos blocos **Slave_**{}

Uma primitiva de sincronização é responsável por sinalizar e executar os blocos escravos declarados fora do bloco mestre. Esta declaração é realizada dentro do bloco mestre, que determina como a computação e o trabalho é realizado. A primitiva apenas se faz necessária quando existem blocos **Slave_**{} fora do bloco **Master_**{}.Neste caso, esta primitiva serve unicamente para sincronizar os blocos **Slave_**{} do núcleo principal.

_&SyncronizeSlaves; -> Primitiva de Sincronização dos blocos **Slave**{}

Esta primitiva de sincronização de blocos escravos serve para sincronizar os blocos **Slave**{} declarados fora do bloco Master{}, os quais são pertencentes ao subnúcleo.

_&ProtectData($CONSTANTE$, var_name**)**{} -> Primitiva de proteção da memória compartilhada.

O objetivo é garantir o acesso seguro ao recurso que está sendo usado dentro desta primitiva. Quando o recurso está sendo usado, um bloqueio é realizado a fim de que outras *threads* que disputam do mesmo recurso não acessem simultaneamente este. Assim que o recurso é utilizado a região é desbloqueada e a próxima *thread* pode usar o recurso em segurança. A primitiva de proteção de dados fornece três alternativas para realizar a proteção dos dados por meio da constante, podendo ser:

- **MUTEX**: mais conhecido como exclusão mútua, sendo o objetivo evitar que dois processos ou *threads* acessem simultaneamente um recurso compartilhado, denominado de seção crítica. Sua funcionalidade é baseada em bloquear e desbloquear a região compartilhada.

A **var_name** é o campo onde é atribuído um nome para a variável que será utilizada pela constante (tipo de proteção), ser vindo de controle da proteção dos dados. Esta não pode ser declarada como um tipo qualquer da linguagem C, pois será do tipo da constante declarada na primitiva de proteção de dados.

2.1 Exemplo de código

Um exemplo de código utilizando a LED-PPOPP com o padrão *Mestre/Escravo* é apresentado no trecho de código a seguir na Figura A.4:

```
1  #include <ppoppLinux.h> //biblioteca universal para PPOPP e C.
2
3  //núcleo principal do Mestre/Escravo com 2 threads escravas.
4  $mainMasterSlavePattern(3){
5          //Bloco do Mestre.
6          @Master_{
7                  //sincroniza os blocos Slaves.
8                  _&SynchronizeSlaves_;
9                  printf("Fim do Master\n");
10         }
11         //bloco do Escravo.
12         @Slave_{
13                 printf("Slave\n");
14         }
15 }
```

Figura A.4: Exemplo de código LED-PPOPP.

2.2 Organização dos dados

Para o contexto de organização dos dados usando a LED-PPOPP, os dados globais são os dados declarados antes do núcleo principal, podendo assim serem usados dentro do núcleo e subnúcleos. Dados que estão declarados no **Master_**{}, **Slave_**{}, **Master**{} e **Slave**{}, são unicamente deles. Entretanto, quando um **Slave_**{} é declarado dentro do **Master_**{}, os dados de **Slave_**{} são compartilhados com o **Master_**{}, mas não significa que os dados de **Master_**{} sejam acessíveis nos **Slave_**{} que estão dentro dele. Acontece que, uma cópia do **Slave_**{} é realizada

no **Master_**{} para ajudar a resolver a computação. Exemplificando como é a organização dos dados utilizando a LED-PPOPP, uma demonstração prática é apresentada na Figura A.5.

Figura A.5: Organização dos dados na LED-PPOPP.

3. COMPILAR E EXECUTAR UM PROGRAMA

Como se trata de uma versão compacta da LED-PPOPP, algumas restrições são impostas, como por exemplo, os programas devem ser escritos em apenas um "arquivo.c". Para realizar a compilação basta executar o seguinte comando no terminal:

ppopp programa.c programa

A execução dos programas é realizado da seguinte forma:

./programa

Referências

[1] FOWLER M. Domain Spacific Languages. Addison-Wesley Professional, Boston, MA, USA, 2010.

[2] INTEL and MCCOOL D. M. Structured Parallel Programming with Deterministic Patterns. In HotPar-2nd USENIX Workshop on Hot Topics in Parallelism, pages 1–6, Berkeley, CA, June 2010.

[3] GAMMA E., HELM R., JONHSON R., and VLISSIDES J. Design Patterns: Elements os Reusable Object-Oriented Software. Addison-Wesley, Boston, MA, 2002.

[4] MATTSON G. T., SANDERS A. B., and MASSINGILL L. B. Patterns for Parallel Programming. Addison-Wesley, Boston, MA, 2005.

[5] YANG-MING Z. and COCHOFF S.M. Medical Image Viewing on Multicore Platforms Using Parallel Computing Patterns. IT Professional, 12(2):33–41, 2010.

[6] CATANZARO R. and KEUTZER K. Parallel Computing with Patterns and Frameworks. XRDS: Crossroads, The ACM Magazine for Students, 17(1):22–27, 2010.

[7] MASSINGILL B. L., MATTSON T. G., and SANDERS B. A. Patterns for Parallel Application Programs. In Pattern Languages of Programs'99, pages 1–29, Monticello, Chicago, August 1999.

[8] BROMLING S., MACDONALD S., ANVIK J., SCHAEFFER J., SZAFRON D., and TAN K. Pattern-based Parallel Programming. In Parallel Processing, 2002. Proceedings. International Conference on, pages 257–265, Vancouver Canada, August 2002.

[9] QUINN J. M. Parallel Programming in C with MPI and OpenMP. McGraw-Hill, New York, 2004.

[10] RAUBER T. and RÜNGER G. Parallel Programming. Springer, New York, 2010.

[11] SKILLICORN D. B. and TALIA D. Models and Languages for Parallel Computation. ACM Comput. Surv., 30(2):123–169, 1998.

[13] GHOSH D. DSLs in Action. Manning, Stamford, CT, USA, 2011.

A.4 Manual Pthread

Autor: Dalvan Jair Griebler (GRIEBLER D. J.)

Pthread

1. INTRODUÇÃO

A biblioteca de *threads* POSIX é um padrão baseado na **API Pthread**. Esta permite criar um novo fluxo do processo concorrente, sendo mais eficiente quando utilizado em sistemas multi-core, onde o fluxo deste pode ser escalonado para que seja executado em outro processador. De tal modo, passa a obter maior desempenho através do processamento paralelo ou distribuído. Assim, a *Thread* gera menos sobrecarga no sistema do que criar um novo processo a partir de outro ou realizar a troca de contexto de um já existente, pois o sistema não inicializará um novo espaço de memória virtual e um novo ambiente para o processo. Contudo, a utilização de pthread é limitada em um computador de memória compartilhada. Todas as *threads* dentro de um processo compartilham o mesmo espaço de endereçamento. A geração da *thread* dá-se a partir de uma função e dos argumentos a serem processados na mesma. O objetivo desta biblioteca é executar o software de uma maneira mais rápida [2] [1].

2. INTERFACE DE PROGRAMAÇÃO PTHREAD

As operações com threads são basicamente: criação, finalização, sincronização, escalonamento, gerenciamento de dados e interação de processo. É importante ressaltar que uma thread não mantem uma lista de threads criadas, pois ela não conhece quem é o seu criador. Além disso, todas as threads dentro de um processo compartilham do mesmo espaço de endereçamento [3].

As threads de um mesmo processo compartilham:

- Instruções de processo.
- Dados.
- Operações com arquivos.
- Diretório de trabalho atual.

- Sinais.
- ID de usuário e grupo.

Cada Thread tem um único:

- ID da thread.
- Conjunto de registradores e ponteiros de pilha.
- Pilha de variáveis locais e retorno de endereçamento.
- Máscara de sinal.
- Prioridade.
- Valor de retorno.

OBS: As funções pthread retornam "0" se a execução deu certo.

2.1 Criando *threads* com Pthread

A criação de uma *thread* é realizada através da função "pthread_create" da estrutura "pthread_t", que identifica cada thread das demais [3]. O protótipo da função é a seguinte:

- **int pthread_create**(*pthread_t* * thread, *pthread_attr_t* *attr, **void** * (*start_routine)(**void** *), **void** *arg).
- **thread**: contém os dados da *thread*, sendo utilizada para identificá-los.
- **attr**: atributos para a inicialização da *thread*, como por exemplo, as prioridades e o algoritmo da *thread*.
- **start_routine**: utilizada no endereço da função a ser executada, a qual deverá retornar e ter como parâmetro ponteiros genéricos (void *).
- **arg**: é um ponteiro ao parâmetro passado para a função.

Depois de criar as **threads**, duas opções podem ser escolhidas: esperar que elas terminem, caso nos interesse recuperar algum resultado, ou simplesmente deixarmos a biblioteca de pthreads decidir quando a execução da função da thread deve terminar, eliminando todos os dados de suas tabelas internas [4]. A biblioteca pthread dispõem de diversas funções, as quais estão explanadas abaixo:

- **int pthread_join**(*pthread_t* th, **void** **thread_return); Função responsável por suspender a *thread* que a chamou até que termine a execução da *thread* que fora indicada. Além disso, em seu termino é retornado em "thread_return" o resultado da função .
- **th**: indicador da thread.
- **thread_return**: ponteiro de ponteiro que aponta para o resultado devolvido pela *thread*.
- **int pthread_detach**(*pthread_t* th); Esta função indica que não deve ser recebido o resultado da *thread* referenciada por "**th**".
- **th**: identificador da *thread*.

Para terminar a execução da **thread**, utiliza-se a função "**void** pthread_exit(void *retval)". E **retval** é um ponteiro genérico para os dados que quisermos retornar.

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

void *print_message_function( void *ptr );

main()
{
     pthread_t thread1, thread2;
     char *message1 = "Thread 1";
     char *message2 = "Thread 2";
     int  iret1, iret2;

    /* Create independent threads each of which will execute function */

     iret1 = pthread_create( &thread1, NULL, print_message_function, (void*) message1);
     iret2 = pthread_create( &thread2, NULL, print_message_function, (void*) message2);

     /* Wait till threads are complete before main continues. Unless we  */
     /* wait we run the risk of executing an exit which will terminate   */
     /* the process and all threads before the threads have completed.   */

     pthread_join( thread1, NULL);
     pthread_join( thread2, NULL);

     printf("Thread 1 returns: %d\n",iret1);
     printf("Thread 2 returns: %d\n",iret2);
     exit(0);
}

void *print_message_function( void *ptr )
{
     char *message;
     message = (char *) ptr;
     printf("%s \n", message);
}
```

Figura A.6: Exemplo de criação de thread com pthread. Adaptado de [3].

Logo abaixo na Figura A.6 está ilustrado um exemplo de utilização da função de criação e sincronização de *threads*.

2.2. Sincronização de Threads

A biblioteca pthread fornece alguns mecanismos de sincronização, os quais são detalhados a seguir:

Mutex: exclusão mútua, bloqueio de acesso a variáveis por outras *threads*. A biblioteca pthread dispõem das seguintes funções para utilização de mutex:

- **pthread_mutex_lock();** Executa o bloqueio em uma variável *mutex* específica. Se o *mutex* já estiver bloqueado por outra *thread*, esta chamada será bloqueada até o *mutex* ser desbloqueado.
- **pthread_mutex_unlock();** desbloqueia a variável *mutex*.
- **pthread_mutex_trylock();** tenta bloquear um mutex ou retorna um erro se estiver ocupado. Bastante útil para prevenir *deadlock*. Um exemplo de código utilizando *mutex* é apresentado a seguir na Figura A.7:

Código Sem Mutex

```c
int counter=0;

/* Function C */
void functionC()
{

    counter++

}
```

Código com Mutex

```c
/* Note scope of variable and mutex are the same */
pthread_mutex_t mutex1 = PTHREAD_MUTEX_INITIALIZER;
int counter=0;

/* Function C */
void functionC()
{
    pthread_mutex_lock( &mutex1 );
    counter++
    pthread_mutex_unlock( &mutex1 );
}
```

Figura A.7: Exemplo de mutex com pthread. Adaptado de [3].

Join: Faz uma *thread* esperar o termino das demais. Sua utilização já foi demonstrada na seção 2.1, onde foi tratado da criação de *threads*.

Variáveis condicionais: consiste em uma função que espera e depois continua seu processamento. As variáveis condicionais permitem que as *threads* suspendam a execução e abandonem o processador até que uma condição seja satisfeita. Uma variável condicional deve ser associada com um *mutex* para evitar condições de corrida entre uma *thread* que está se preparando para esperar e outra *thread* que pede um sinal a condição antes da primeira, neste caso, a *thread* ficará esperando por um sinal que nunca receberá. Qualquer *mutex* poderá ser utilizado, não existe um link explícito entre *mutex* e varável condicional.

As seguintes funções são usadas para criação de variáveis condicionais:

- **int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t *attr);** inicializa a variável condicional.

- **int pthread_cond_destroy(pthread_cond_t *cond);** destrói a variável condicional.

As seguintes funções são usadas para esperar uma condição:

- **int pthread_cond_wait(pthread_cond_t *cond, pthread_mutex_t *mutex);** desbloqueia um *mutex* e coloca a chamada da *thread* em um estado bloqueado. Quando uma variável condicional específica é sinalizada, esta função bloqueia o *mutex* e retorna para o que a chamou.

- **int pthread_cond_timedwait(pthread_cond_t *cond, pthread_mutex_t *mutex, struct timespec *timeout);** desbloqueia um *mutex* e coloca a chamada da thread em um estado bloqueado. Quando uma variável condicional específica é sinalizada ou se o tempo é maior que ' **i** ' ou igual ao '*timeout*', esta função bloqueia o mutex e retorna para o que a chamou.

As seguintes funções são usadas para disparar um sinal quando satisfeita uma condição:

- **int pthread_cond_signal(pthread_cond_t *cond);** desbloqueia pelo menos uma *thread* esperando uma variável condicional, enquanto que a prioridade do escalonamento determina qual *thread* será executada.

- **int pthread_cond_broadcast(pthread_cond_t *cond);** desbloqueia todas as *threads* que estão esperando em uma dada variável condicional. Um exemplo de código utilizando variáveis condicionais é apresentado na Figura A.8.

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

pthread_mutex_t count_mutex     = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t  condition_var   = PTHREAD_COND_INITIALIZER;

void *functionCount1();
void *functionCount2();
int  count = 0;
#define COUNT_DONE  10
#define COUNT_HALT1  3
#define COUNT_HALT2  6

main()
{
   pthread_t thread1, thread2;

   pthread_create( &thread1, NULL, &functionCount1, NULL);
   pthread_create( &thread2, NULL, &functionCount2, NULL);

   pthread_join( thread1, NULL);
   pthread_join( thread2, NULL);

   printf("Final count: %d\n",count);

   exit(0);
}

// Write numbers 1-3 and 8-10 as permitted by functionCount2()

void *functionCount1()
{
   for(;;)
   {
      // Lock mutex and then wait for signal to relase mutex
      pthread_mutex_lock( &count_mutex );

      // Wait while functionCount2() operates on count
      // mutex unlocked if condition varialbe in functionCount2() signaled.
      pthread_cond_wait( &condition_var, &count_mutex );
      count++;
      printf("Counter value functionCount1: %d\n",count);

      pthread_mutex_unlock( &count_mutex );

      if(count >= COUNT_DONE) return(NULL);
   }
}

// Write numbers 4-7

void *functionCount2()
{
   for(;;)
   {
      pthread_mutex_lock( &count_mutex );

      if( count < COUNT_HALT1 || count > COUNT_HALT2 )
      {
         // Condition of if statement has been met.
         // Signal to free waiting thread by freeing the mutex.
         // Note: functionCount1() is now permitted to modify "count".
         pthread_cond_signal( &condition_var );
      }
      else
      {
         count++;
         printf("Counter value functionCount2: %d\n",count);
      }

      pthread_mutex_unlock( &count_mutex );

      if(count >= COUNT_DONE) return(NULL);
   }
}
```

Figura A.8: Exemplo de variavel condicional com pthread. Adaptado de [3].

3. COMPILAR E EXECUTAR UM PROGRAMA

Para compilar um programa utilizando a biblioteca pthread é preciso indicá-la durante a compilação do programa através de "-lpthread".

gcc progama.c -o programa -lpthread

A execução de um programa é realizada da mesma forma que qualquer programa desenvolvido em linguagem C, por exemplo:

./programa

Referências

[1] LEWIS B. and BERG D. J. Multithreaded Programming with Pthreads. Sun, California, USA, 1998.

[2] LEWIS B. and BERG D. J. A Guide to Multithreaded Programming Threads Primer. Sun, California, USA, 1996.

[3] IPPOLITO G. POSIX thread (pthread) library. Extracted from <http://www.yolinux.com/TUTORIALS/LinuxTutorialPosixThreads.html>, in November, 2011.

[4] FILHO A. E. F., BARBOSA F. R., GREHS G. H. and CARRARO M. S.Tutorial de Pthreads. Extracted from. <http://www.inf.ufrgs.br/gppd/disc/inf01151/trabalhos/sem2000-2/tutorial_ pthreads/page1.html>, in November, 2011.

A.5 Código Fonte do Algoritmo Sequencial de Multiplicação de Matrizes

```c
#include <stdio.h>
#include <stdlib.h>
//tamanho da matrix...
#define MX      1000
long int **matrix1, **matrix2, **matrix;
double tempo(){
        struct timeval tv;
        gettimeofday(&tv,0);
        return tv.tv_sec + tv.tv_usec/1e6;
}
void atribui_val(){
        long int i, j;
        for(i=0; i<MX; i++){
                for(j=0; j<MX; j++){
                        matrix1[i][j] = 4;
                        matrix2[i][j] = 5;
                        matrix[i][j] = 0;
                }
        }
}
void printMatrix(long int **matrix){
        int i, j;
        for(i=0; i<MX; i++){
                printf("\n");
                for(j=0; j<MX; j++){
                        printf("%ld ", matrix[i][j]);
                }
        }
        printf("\n");
}
int main(){
        double t_start, t_end;
        long int i, j, k;
        //alocação de memória de colunas da matriz...
        matrix = (long int**)malloc(sizeof(long int) * MX);
        matrix1 = (long int**)malloc(sizeof(long int) * MX);
        matrix2 = (long int**)malloc(sizeof(long int) * MX);
        //alocação de memória de linhas da matriz...
        for (i=0; i < MX; i++){
            matrix[i] = (long int*)malloc(sizeof(long int) * MX);
            matrix1[i] = (long int*)malloc(sizeof(long int) * MX);
            matrix2[i] = (long int*)malloc(sizeof(long int) * MX);
        }
        t_start = tempo();
        atribui_val();
        for(i=0; i<MX; i++){
                for(j=0; j<MX; j++){
                        for(k=0; k<MX; k++){
                                matrix[i][j] += (matrix1[i][k] * matrix2[k][j]);
                        }
                }
        }
        t_end = tempo();
        printf("TEMPO DE EXECUÇÃO: %lf segundos\n", t_end-t_start);
        printMatrix(matrix);
}
```

Figura A.9: Código fonte algoritmo sequencial de multiplicação de matrizes.

A.6 Benchmark paralelizados com a LED-PPOPP

Figura A.10: Código fonte do algoritmo EI_2D com LED-PPOPP.

```
#include <ppoppLinux.h>

double tempo(){
        struct timeval tv;
        gettimeofday(&tv,0);
        return tv.tv_sec + tv.tv_usec/1e6;
}
double f(double x, double y){
        double value;
        value = 1.0 / ( 1.0 - x * y );
        return value;
}

double total;
double a, b;
int n=0, ny, nx;
double *result_total;

$mainMasterSlavePattern(17){
        @Master_{
                double t_start, t_end;
                double error;
                double exact;
                int i, j;
                double pi;
                int proc=16, th=16;
                FILE *arq;
                a = 0.0;
                b = 1.0;
                nx = 91768; //81768;    //orinal data 32768
                ny = 91768; //81768;    //orinal data 32768
                n = nx * ny;
                pi = 3.141592653589793;
                exact = pi * pi / 6.0;
                result_total = (double*)malloc(sizeof(double) * nx);



                printf ( "\n" );
                printf ( "QUAD2D_PPOPP:\n" );
                printf ( "  C/PPOPP version\n" );
                printf ( "  Estimate the integral of f(x,y) over [0,1]x[0,1].\n" );
                printf ( "  f(x,y) = 1 / ( 1 - x * y ).\n" );
                printf ( "\n" );
                printf ( "  Number of processors available = %d\n", proc);
                printf ( "  Number of threads =              %d\n", th);
                printf ( "  A        = %f\n", a );
                printf ( "  B        = %f\n", b );
                printf ( "  NX       = %d\n", nx );
                printf ( "  NY       = %d\n", ny );
                printf ( "  N        = %d\n", n );
                printf ( "  Exact    = %24.16f\n", exact );
                t_start = tempo();
                total=0.0;
                nx++;
                ny++;
                for(i=1; i<nx; i++){
                        result_total[i]=0.0;
                }
                //Sincronizando o bloco slave...
                _&SynchronizeSlaves_;
                nx--;
                ny--;
                for(i=1; i<=nx; i++){
                        total = total + result_total[i];
                }
                t_end = tempo();
                total = ( b - a ) * ( b - a ) * total / ( double ) ( nx ) / ( double ) ( ny );
                error = fabs ( total - exact );
                printf ( "\n" );
                printf ( "  Estimate = %24.16f\n", total );
                printf ( "  Error    = %e\n", error );
                printf("TEMPO DE EXECUÃ‡ÃƒO: %lf segundos\n", t_end-t_start);

                printf ( "\n" );
                printf ( "QUAD_PPOPP:\n" );
                printf ( "  Normal end of execution.\n" );
```

```
        }
        @Slave_{
                int i, j, h;
                double x, y;
                for(i=1; i<nx; i++){
                        x = ( ( 2 * (nx-1) - 2 * i + 1 ) * a + ( 2 * i - 1 ) * b ) / ( 2 * (nx-1) );
                        for(j=1; j<ny; j++){
                                y = ( ( 2 * (ny-1) - 2 * j + 1 ) * a + ( 2 * j - 1 ) * b ) / ( 2 * (ny-1) );
                                result_total[j] += f ( x, y );
                        }
                }
        }
}
```

Figura A.11: Código fonte do algoritmo FFT com LED-PPOPP.

```
#include <ppoppLinux.h>

//Global variables
int n;
int mj;
double *x, *z, *y, *w;
double sgn;
double *a, *b, *c, *d;

double tempo(){
        struct timeval tv;
        gettimeofday(&tv,0);
        return tv.tv_sec + tv.tv_usec/1e6;
}

void ccopy(int n, double *x, double  *y){
        int i;
        for ( i = 0; i < n; i++ ){
                y[i*2+0] = x[i*2+0];
                y[i*2+1] = x[i*2+1];
        }
        return;
}


void cfft2(double *x, double *y, double *w){
        int j;
        int m;
        int tgle;

        m = ( int ) ( log ( ( double ) n ) / log ( 1.99 ) );
        mj   = 1;


/*
  Toggling switch for work array.
*/
        tgle = 1;
        //void step(int n, int mj, double *a, double *b, double *c, double *d, double *w, double sgn){

        a = &x[0*2+0];
        b = &x[(n/2)*2+0];
        c = &y[0*2+0];
        d = &y[mj*2+0];

        step();

        if(n == 2){
                return;
        }

        for ( j = 0; j < m - 2; j++ ){
                mj = mj * 2;
                if ( tgle ){
                        a = &y[0*2+0];
                        b = &y[(n/2)*2+0];
                        c = &x[0*2+0];
                        d = &x[mj*2+0];
                        step ();
                        tgle = 0;
```

```
                }else{
                        a = &x[0*2+0];
                        b = &x[(n/2)*2+0];
                        c = &y[0*2+0];
                        d = &y[mj*2+0];
                        step ();
                        tgle = 1;
                }
        }

/*
  Last pass through data: move Y to X
*/
        if(tgle){
                ccopy ( n, y, x );
        }

        mj = n / 2;
        a = &x[0*2+0];
        b = &x[(n/2)*2+0];
        c = &y[0*2+0];
        d = &y[mj*2+0];
        step ();

        return;
}



void cffti(){

$MasterSlavePattern(17){

        @Master{

                _&SynchronizeSlaves;

                return;
        }
        @Slave{



                double arg;
                double aw;
                int i;
                int n2;
                const double pi = 3.141592653589793;

                n2 = n / 2;
                aw = 2.0 * pi / ( ( double ) n );


                for ( i = 0; i < n2; i++ ){
                        arg = aw * ( ( double ) i );
                        w[i*2+0] = cos ( arg );
                        w[i*2+1] = sin ( arg );
                }
        }
}

}


double ggl(double *seed){
        double d2 = 0.2147483647e10;
        double t;
        double value;

        t = ( double ) *seed;
        t = fmod ( 16807.0 * t, d2 );
        *seed = ( double ) t;
        value = ( double ) ( ( t - 1.0 ) / ( d2 - 1.0 ) );

        return value;
}
```

```
void step(){
$MasterSlavePattern(17){
        @Master{

                _&SynchronizeSlaves;
        }
        @Slave{
                double ambr;
                double ambu;
                int j;
                int ja;
                int jb;
                int jc;
                int jd;
                int jw;
                int k;
                int lj;
                int mj2;
                double wjw[2];
                mj2 = 2 * mj;
                lj  = n / mj2;
                for ( j = 0; j < lj; j++ ){
                        jw = j * mj;
                        ja  = jw;
                        jb  = ja;
                        jc  = j * mj2;
                        jd  = jc;
                        wjw[0] = w[jw*2+0];
                        wjw[1] = w[jw*2+1];
                        if ( sgn < 0.0 ){
                                wjw[1] = - wjw[1];
                        }
                        for ( k = 0; k < mj; k++ ){
                                c[(jc+k)*2+0] = a[(ja+k)*2+0] + b[(jb+k)*2+0];
                                c[(jc+k)*2+1] = a[(ja+k)*2+1] + b[(jb+k)*2+1];

                                ambr = a[(ja+k)*2+0] - b[(jb+k)*2+0];
                                ambu = a[(ja+k)*2+1] - b[(jb+k)*2+1];









                                d[(jd+k)*2+0] = wjw[0] * ambr - wjw[1] * ambu;
                                d[(jd+k)*2+1] = wjw[1] * ambr + wjw[0] * ambu;
                        }
                }
        }

}

}
```

```
$mainMasterSlavePattern(17){

        @Master_{

                int proc=16, th=16;
                double t_start, t_end;
                double wtime;
                double error;
                int first;
                double flops;
                double fnm1;
                int i;
                int icase;
                int it;
                int ln2;
                int ln2_max = 25;       //original is 25
                double mflops;

                int nits = 1000; //original is 10000
                int proc_num;
                static double seed;



                double z0;
                double z1;

                t_start = tempo();


                printf ( "\n" );
                printf ( "FFT_PPOPP\n" );
                printf ( "  PPOPP-Master/Slave version\n" );
                printf ( "\n" );
                printf ( "  Demonstrate an implementation of the Fast Fourier Transform\n" );
                printf ( "  of a complex data vector, using OpenMP for parallel execution.\n" );




                 printf ( "  Number of processors available = %d\n", proc );
                 printf ( "  Number of threads =              %d\n", th );
                 /*
                   Prepare for tests.
                 */
                 printf ( "\n" );
                 printf ( "  Accuracy check:\n" );
                 printf ( "\n" );
                 printf ( "    FFT ( FFT ( X(1:N) ) ) == N * X(1:N)\n" );
                 printf ( "\n" );
                 printf ( "             N      NITS    Error         Time          Time/Call     MFLOPS\n" );
                 printf ( "\n" );

                 seed  = 331.0;
                 n = 1;
                 /*
                   LN2 is the log base 2 of N.  Each increase of LN2 doubles N.
                 */
                 for ( ln2 = 1; ln2 <= ln2_max; ln2++ ){
                         n = 2 * n;

                         w = ( double * ) malloc (     n * sizeof ( double ) );
                         x = ( double * ) malloc ( 2 * n * sizeof ( double ) );
                         y = ( double * ) malloc ( 2 * n * sizeof ( double ) );
                         z = ( double * ) malloc ( 2 * n * sizeof ( double ) );

                         first = 1;

                         for (icase = 0; icase < 2; icase++){
                                 if (first){
                                         for ( i = 0; i < 2 * n; i = i + 2 ){
                                                 z0 = ggl ( &seed );
                                                 z1 = ggl ( &seed );
                                                 x[i] = z0;
                                                 z[i] = z0;
                                                 x[i+1] = z1;
                                                 z[i+1] = z1;
                                         }
```

```
                }else{

                        _&SynchronizeSlaves_;
                }
                /*
                  Initialize the sine and cosine tables.
                */
                cffti ();
                /*
                  Transform forward, back
                */
                if (first){
                        sgn = + 1.0;
                        cfft2 ( x, y, w);
                        sgn = - 1.0;
                        cfft2 ( y, x, w);
                        /*
                          Results should be same as the initial data multiplied by N.
                        */
                        fnm1 = 1.0 / ( double ) n;
                        error = 0.0;
                        for ( i = 0; i < 2 * n; i = i + 2 ){
                                error = error + pow ( z[i]   - fnm1 * x[i], 2 ) + pow ( z[i+1] - fnm1 * x[i+1], 2 );
                        }

                        error = sqrt ( fnm1 * error );
                        printf ( "  %12d  %8d  %12e", n, nits, error );
                        first = 0;
                }else{
                        wtime = tempo();

                        for(it = 0; it < nits; it++){
                                sgn = + 1.0;
                                cfft2( x, y, w);
                                sgn = - 1.0;
                                cfft2 ( y, x, w);
                        }
                        wtime = tempo() - wtime;




                         flops = 2.0 * ( double ) nits * ( 5.0 * ( double ) n * ( double ) ln2 );

                         mflops = flops / 1.0E+06 / wtime;

                         printf ( "  %12e  %12e  %12f\n", wtime, wtime / ( double ) ( 2 * nits ), mflops );
                 }
        }
        if ( ( ln2 % 4 ) == 0 ){
                nits = nits / 10;
        }
        if ( nits < 1 ){
                nits = 1;
        }
        free ( w );
        free ( x );
        free ( y );
        free ( z );
}
/*
  Terminate.
*/
printf ( "\n" );
printf ( "FFT_OPENMP:\n" );
printf ( "  Normal end of execution.\n" );
printf ( "\n" );

t_end = tempo();

printf("TEMPO DE EXECUÃ‡ÃƒO: %lf segundos\n", t_end-t_start);

}
```

```
        @Slave_{
                int i, nc;
                double z0, z1;
                nc = 2 * n;
                for ( i = 0; i < nc; i++){
                        z0 = 0.0;               /* real part of array */
                        z1 = 0.0;               /* imaginary part of array */
                        x[i] = z0;
                        z[i] = z0;           /* copy of initial real data */
                        x[i+1] = z1;
                        z[i+1] = z1;         /* copy of initial imag. data */
                        i++;
                }

        }
}
```

Figura A.12: Código fonte do algoritmo MD com LED-PPOPP.

```
#include <ppoppLinux.h>

///global variables ...

int nd, np;
double *pos, *vel, *froce, *acc, *box;
double mass, dt, kinetic, potential;
double *ke, *pe;

double tempo(){
        struct timeval tv;
        gettimeofday(&tv,0);
        return tv.tv_sec + tv.tv_usec/1e6;
}

double dist(double r1[], double r2[], double dr[]){

        double d;
        int i;

        d = 0.0;
        for ( i = 0; i < nd; i++ ){
                dr[i] = r1[i] - r2[i];
                d = d + dr[i] * dr[i];
        }
        d = sqrt ( d );

        return d;
}
```

```
void compute(){

$MasterSlavePattern(15){

        @Master{
                double ke_result=0.0, pe_result=0.0;
                int i;

                ke = (double*)malloc(sizeof(double) * np);
                pe = (double*)malloc(sizeof(double) * np);

                for(i=0; i<np; i++){
                        ke[i] = 0.0;
                        pe[i] = 0.0;
                }
                _&SynchronizeSlaves;

                for(i=0; i<np; i++){
                        pe_result = pe_result + pe[i];
                        ke_result = ke_result + ke[i];
                }

                ke_result = ke_result * 0.5 * mass;

                potential = pe_result;
                kinetic = ke_result;

                free(ke);
                free(pe);

                return;
        }



        @Slave{
                int i, j, k, l;
                double d, d2;
                double PI2 = 3.141592653589793 / 2.0;
                double rij[3];


                for(k=0; k<np; k++){
                /*
                  Compute the potential energy and froces.
                */
                        for(i=0; i<nd; i++){
                                froce[i+k*nd] = 0.0;
                        }

                        for(j=0; j<np; j++){
                                if ( k != j ){
                                        d = dist (pos+k*nd, pos+j*nd, rij );
                                        /*
                                          Attribute half of the potential energy to particle J.
                                        */
                                        if ( d < PI2 ){
                                                d2 = d;
                                        }else{
                                                d2 = PI2;
                                        }

                                        pe[k] = pe[k] + 0.5 * pow ( sin ( d2 ), 2 );

                                        for(i= 0; i<nd; i++) {
                                                froce[i+k*nd] = froce[i+k*nd] - rij[i] * sin ( 2.0 * d2 ) / d;
                                        }
                                }
                        }
                        /*
                          Compute the kinetic energy.
                        */
```

```
                        for(i=0; i<nd; i++){
                                ke[k] = ke[k] + vel[i+k*nd] * vel[i+k*nd];
                        }
                }

          }
}

}
```

```
double r8_uniform_01( int *seed){

        int k;
        double r;

        k = *seed / 127773;

        *seed = 16807 * ( *seed - k * 127773 ) - k * 2836;

        if ( *seed < 0 ){
                *seed = *seed + 2147483647;
        }

        r = ( double ) ( *seed ) * 4.656612875E-10;

        return r;
}
```

```
 void initialize(int np, int nd, double *box, int *seed, double *pos, double *vel, double *acc){

         int i;
         int j;
         /*
           Give the particles random positions within the box.
         */
         for ( i = 0; i < nd; i++ ){
                 for ( j = 0; j < np; j++ ){
                         pos[i+j*nd] = box[i] * r8_uniform_01 ( seed );
                 }
         }

         for ( j = 0; j < np; j++ ){

                 for ( i = 0; i < nd; i++ ){
                         vel[i+j*nd] = 0.0;
                 }
         }

         for ( j = 0; j < np; j++ ){
                 for ( i = 0; i < nd; i++ ){
                         acc[i+j*nd] = 0.0;
                 }
         }

         return;
 }
```

```
$mainMasterSlavePattern(15){

        @Master_{
                int proc=16, th=14;
                double t_start, t_end;
                double e0;
                int i;
                int seed = 123456789;
                int step;
                int step_num = 4; //400
                int step_print;
                int step_print_index;
                int step_print_num;

                nd = 3;
                np = 4000;      //1000
                mass = 1.0;
                dt = 0.0001;


                acc = ( double * ) malloc ( nd * np * sizeof ( double ) );
                box = ( double * ) malloc ( nd * sizeof ( double ) );
                froce = ( double * ) malloc ( nd * np * sizeof ( double ) );
                pos = ( double * ) malloc ( nd * np * sizeof ( double ) );
                vel = ( double * ) malloc ( nd * np * sizeof ( double ) );

                printf ( "\n" );
                printf ( "MD_PPOPP\n" );
                printf ( "  PPOPP-Master/Slave version\n" );
                printf ( "\n" );
                printf ( "  A molecular dynamics program.\n" );

                printf ( "\n" );
                printf ( "  NP, the number of particles in the simulation is %d\n", np );
                printf ( "  STEP_NUM, the number of time steps, is %d\n", step_num );
                printf ( "  DT, the size of each time step, is %f\n", dt );



               printf ( "\n" );
               printf ( "  Number of processors available = %d\n", proc );
               printf ( "  Number of threads =              %d\n", th );

               /*
                 Set the dimensions of the box.
               */
               for ( i = 0; i < nd; i++ ){
                       box[i] = 10.0;
               }

               printf ( "\n" );
               printf ( "  Initializing positions, velocities, and accelerations.\n" );


               /*
                 Set initial positions, velocities, and accelerations.
               */
               initialize ( np, nd, box, &seed, pos, vel, acc );
               /*
                 Compute the froces and energies.
               */
               printf ( "\n" );
               printf ( "  Computing initial froces and energies.\n" );

               compute ();

               e0 = potential + kinetic;
               /*
                 This is the main time stepping loop:
                   Compute froces and energies,
                   Update positions, velocities, accelerations.
               */
               printf ( "\n" );
               printf ( "  At each step, we report the potential and kinetic energies.\n" );
               printf ( "  The sum of these energies should be a constant.\n" );
               printf ( "  As an accuracy check, we also print the relative error\n" );
               printf ( "  in the total energy.\n" );
               printf ( "\n" );
```

```
        printf ( "      Step      Potential       Kinetic        (P+K-E0)/E0\n" );
        printf ( "                Energy P        Energy K       Relative Energy Error\n" );
        printf ( "\n" );

        step_print = 0;
        step_print_index = 0;
        step_print_num =4;

        step = 0;
        printf ( "  %8d  %14f  %14f  %14e\n", step, potential, kinetic, ( potential + kinetic - e0 ) / e0 );
        step_print_index = step_print_index + 1;
        step_print = ( step_print_index * step_num ) / step_print_num;


        t_start = tempo();
        for ( step = 1; step <= step_num; step++ ){
                compute();

                if( step == step_print ){
                        printf ( "  %8d  %14f  %14f  %14e\n", step, potential, kinetic, ( potential + kinetic - e0 ) / e0 );
                        step_print_index = step_print_index + 1;
                        step_print = ( step_print_index * step_num ) / step_print_num;
                }

                //UPDATE updates positions, velocities and accelerations.
                _&SynchronizeSlaves_;

        }

        t_end = tempo();

        printf("TEMPO DE EXECUÃ‡ÃƒO: %lf segundos\n", t_end-t_start);

        free ( acc );
        free ( box );
        free ( froce );
        free ( pos );
        free ( vel );







          /*
            Terminate.
          */
          printf ( "\n" );
          printf ( "MD_PPOPP\n" );
          printf ( "  Normal end of execution.\n" );

          printf ( "\n" );


    }
    @Slave_{

          int i;
          int j;
          double rmass;

          rmass = 1.0 / mass;


          for(j=0; j<np; j++){
                  for( i=0; i<nd; i++){
                          pos[i+j*nd] = pos[i+j*nd] + vel[i+j*nd] * dt + 0.5 * acc[i+j*nd] * dt * dt;
                          vel[i+j*nd] = vel[i+j*nd] + 0.5 * dt * ( froce[i+j*nd] * rmass + acc[i+j*nd] );
                          acc[i+j*nd] = froce[i+j*nd] * rmass;
                  }
          }

    }


}
```

Figura A.13: Código fonte do algoritmo MM com LED-PPOPP.

```
#include <ppoppLinux.h>

double tempo(){
        struct timeval tv;
        gettimeofday(&tv,0);
        return tv.tv_sec + tv.tv_usec/1e6;
}

void printMatrix(double **matrix, int tam){

        int i, j;
        for(i=0; i<tam; i++){
                printf("\n");
                for(j=0; j<tam; j++){

                        printf("%lf ", matrix[i][j]);

                }
        }
        printf("\n");
}

//global variables...
double **a, **b, **c;
int n;
double s, pi;



$mainMasterSlavePattern(16){

        @Master_{


                int proc=16, th=16;
                double t_start, t_end;
                int i, j;

                n = 1800;       // original 500
                pi = 3.141592653589793;

                a = (double**)malloc(sizeof(double) * n);
                b = (double**)malloc(sizeof(double) * n);
                c = (double**)malloc(sizeof(double) * n);

                //alocaÃ§Ã£o de memÃ³ria de linhas da matriz...
                for (i=0; i < n; i++){
                        a[i] = (double*)malloc(sizeof(double) * n);
                        b[i] = (double*)malloc(sizeof(double) * n);
                        c[i] = (double*)malloc(sizeof(double) * n);
                }

                printf ( "\n" );
                printf ( "MXM_PPOPP:\n" );
                printf ( "  C/PPOPP-Master/Slave version\n" );
                printf ( "  Compute matrix product C = A * B.\n" );
                printf ( "\n" );
                printf ( "  The number of processors available = %d\n", proc );
                printf ( "  The number of threads available    = %d\n", th );
                printf ( "  The matrix order N                 = %d\n", n );

                s = 1.0 / sqrt ( ( double ) ( n ) );


                t_start = tempo();
```

```
		@Slave_{
			double angle;
			int im, jm;
			/*
			  Loop 1: Evaluate A.
			*/

			for ( im = 0; im < n; im++ ){
				for ( jm = 0; jm < n; jm++ ){
					angle = 2.0 * pi * im * jm / ( double ) n;
					a[im][jm] = s * ( sin ( angle ) + cos ( angle ) );
				}
			}
		}

		/*
		  Loop 2: Copy A into B.
		*/

		for ( i = 0; i < n; i++ ){
			for ( j = 0; j < n; j++ ){
				b[i][j] = a[i][j];

			}
		}









		@Slave_{

			int in, jn, kn;
			/*
			  Loop 3: Compute C = A * B.
			*/

			for ( in = 0; in < n; in++ ) {
				for ( jn = 0; jn < n; jn++ ){
					c[in][jn] = 0.0;
					for ( kn = 0; kn < n; kn++ ){
						c[in][jn] = c[in][jn] + a[in][kn] * b[kn][jn];
					}
				}
			}
		}

		t_end = tempo();

		//printf ( "  Elapsed seconds = %g\n", wtime );
		printf ( "  C(100,100)  = %g\n", c[99][99] );
		/*
		  Terminate.
		*/
		printf ( "\n" );
		printf ( "MXM_PPOPP:\n" );
		printf ( "  Normal end of execution.\n" );
		printf ( "\n" );

		printf("TEMPO DE EXECUÃ‡ÃƒO: %lf segundos\n", t_end-t_start);

		//printing the matrix ....
		//printMatrix(c, 4);

	}

}
```

Figura A.14: Código fonte do algoritmo PN com LED-PPOPP.

```
#include <ppoppLinux.h>

double tempo(){
        struct timeval tv;
        gettimeofday(&tv,0);
        return tv.tv_sec + tv.tv_usec/1e6;
}

int n;
int *primo;

$mainMasterSlavePattern(16){

        @Master_{
                int proc=16, th=16;
                int total, i, n_factor, n_hi, n_lo;
                double wtime=0.0;
                double t_start, t_end;



                printf ( "\n" );
                printf ( "PRIME_PPOPP\n" );
                printf ( "  C/PPOPP-Master/Slave version\n" );

                printf ( "\n" );
                printf ( "  Number of processors available = %d\n", proc );
                printf ( "  Number of threads =              %d\n", th );

                t_start = tempo();

                n_lo = 1;
                n_hi = 131072;
                n_factor = 2;

                /*
                        Start primes.....
                */
```

```
printf ( "\n" );
printf ( "TEST01\n" );
printf ( "  Call PRIME_NUMBER to count the primes from 1 to N.\n" );
printf ( "\n" );
printf ( "         N        Pi          Time\n" );
printf ( "\n" );

n = n_lo;
n++;


while ( n <= n_hi+1 ){
        total = 0;
        primo = (int*)malloc(sizeof(int) * n);
        //start slaves...

        @Slave_{
                int im, jm;

                for ( im = 2; im < n; im++ ){
                        primo[im] = 1;

                        for ( jm = 2; jm < im; jm++ ){
                                if ( im % jm == 0 ){
                                        primo[im] = 0;
                                        break;
                                }
                        }
                }
        }

        for(i=2; i<n; i++){
                total = total + primo[i];
        }
        n--;
        printf ( "  %8d  %8d  %14f\n", n, total, wtime );
        n = n * n_factor;
        n++;


        free(primo);
}
/*
        End primes.....
*/

n_lo = 5;
n_hi = 500000;
n_factor = 10;

/*
        Start primes.....
*/
printf ( "\n" );
printf ( "TEST01\n" );
printf ( "  Call PRIME_NUMBER to count the primes from 1 to N.\n" );
printf ( "\n" );
printf ( "         N        Pi          Time\n" );
printf ( "\n" );

n = n_lo;
n++;
while ( n <= n_hi+1 ){
        total = 0;
        primo = (int*)malloc(sizeof(int) * n);
        //start slaves...
        @Slave_{

                int in, jn;
                for ( in = 2; in < n; in++ ){
                        primo[in] = 1;

                        for ( jn = 2; jn < in; jn++ ){
                                if ( in % jn == 0 ){
                                        primo[in] = 0;
                                        break;
                                }
                        }
                }
        }
```

```
                for(i=2; i<n; i++){
                        total = total + primo[i];
                }
                n--;
                printf ( "  %8d  %8d  %14f\n", n, total, wtime );
                n = n * n_factor;
                n++;
                free(primo);
        }
        /*
                End primes.....
        */

        /*
          Terminate.
        */
        t_end = tempo();
        printf ( "\n" );
        printf ( "PRIME_PPOPP\n" );
        printf ( "  Normal end of execution.\n" );

        printf("\nTEMPO DE EXECUÃ‡ÃƒO: %lf segundos\n", t_end-t_start);

    }


}
```