Computação Paralela: Fundamentos, Desafios Atuais e Perspectivas Futuras

Parallel Computing: Fundamentals, Current Challenges, and Future Perspectives

Guilherme Rocha Duarte[1]
Robson Ricardo Leite da Silva[1]

Resumo - Este trabalho explora os fundamentos da Computação Paralela, uma área essencial para o processamento de grandes volumes de dados e execução de tarefas complexas em menor tempo. Inicialmente, são apresentados os conceitos básicos, incluindo os principais modelos de programação e arquiteturas paralelas. Em seguida, são discutidas as vantagens e as principais aplicações da Computação Paralela em áreas como simulações científicas e inteligência artificial. O trabalho também aborda os desafios atuais, como a escalabilidade e a otimização do desempenho, além de questões relacionadas à segurança. Por fim, são analisadas as tendências tecnológicas e os desafios futuros, incluindo o impacto da Computação Paralela no mercado e na sociedade.

Palavras-chave - Computação Paralela, Dados Massivos, Computação.

Abstract - This paper explores the fundamentals of Parallel Computing, a crucial field for processing large volumes of data and executing complex tasks in less time. Initially, it presents the basic concepts, including key programming models and parallel architectures. Next, it discusses the advantages and major applications of Parallel Computing in areas such as scientific simulations and artificial intelligence. The paper also addresses current challenges, such as scalability and performance optimization, along with security-related issues. Finally, it analyzes technological trends and future challenges, including the impact of Parallel Computing on the market and society. Practical case studies illustrate the successful use of this technology in various contexts, providing a comprehensive overview of the current state and future prospects of Parallel Computing.

Keywords - Parallel Computing, Massive Data, Computing.

[1] Alunos do 6° período de graduação em Ciência de Dados e Inteligência Artificial no Instituto de Educação Superior de Brasília.

Sumário

1 Introdução

Nos últimos anos, o avanço tecnológico tem transformado de maneira significativa a forma como as operações computacionais são realizadas. Com a crescente demanda por processamento de grandes volumes de dados, simulações complexas e inteligência artificial, as limitações dos processadores tradicionais, baseados em execução sequencial, têm se tornado cada vez mais evidentes. Nesse cenário, a Computação Paralela surge como uma solução essencial para superar essas barreiras e atender às exigências de um mundo digital em constante evolução. Ao permitir que múltiplas operações sejam realizadas simultaneamente, a Computação Paralela não só aumenta a eficiência dos sistemas, mas também possibilita a resolução de problemas que, de outra forma, seriam intratáveis em termos de tempo e recursos computacionais.

A Computação Paralela é, hoje, um dos pilares fundamentais das inovações tecnológicas em diversas áreas, desde a modelagem climática até a exploração espacial, passando pela biomedicina e pela análise de *big data*. Com o advento de arquiteturas mais sofisticadas, como GPUs (Unidades de Processamento Gráfico) e processadores *multicore*, o paralelismo tornou-se uma prática comum tanto em pesquisas acadêmicas quanto em aplicações industriais. A capacidade de processar tarefas em paralelo tem levado a avanços notáveis, especialmente em campos que demandam uma grande quantidade de processamento, como inteligência artificial, onde a eficiência computacional pode determinar o sucesso de algoritmos de aprendizado profundo.

O objetivo deste trabalho é explorar de maneira detalhada os fundamentos da Computação Paralela, destacando suas principais arquiteturas, modelos de programação e aplicações práticas. Além disso, serão discutidos os desafios técnicos que ainda limitam o desempenho e a escalabilidade dos sistemas paralelos, bem como as tendências futuras que podem moldar o desenvolvimento dessa área nos próximos anos. O trabalho está estruturado em cinco partes principais. Inicialmente, serão abordados os conceitos fundamentais e as definições que norteiam o campo da Computação Paralela. Em seguida, discutiremos as vantagens e as aplicações mais relevantes, com exemplos práticos. O terceiro capítulo será dedicado aos desafios e limitações que atualmente afetam a eficiência da Computação Paralela. Por fim, o trabalho se concentrará nas tendências e perspectivas futuras, considerando o impacto que essas inovações podem ter tanto no mercado quanto na sociedade.

O estudo da Computação Paralela é extremamente relevante no cenário atual por diversas razões. Primeiramente, a evolução dos sistemas computacionais está intrinsecamente ligada à capacidade de realizar operações de maneira simultânea, aproveitando ao máximo os recursos disponíveis. Em segundo lugar, com a crescente necessidade de processar grandes volumes de dados em tempo real, como ocorre em aplicações de big data e inteligência artificial, o paralelismo oferece uma solução viável para atender a essas demandas. Além disso, a Computação Paralela desempenha um papel crucial na sustentabilidade da infraestrutura tecnológica, permitindo a otimização de recursos energéticos e a redução de custos operacionais. Com isso, a Computação Paralela não só molda o presente da tecnologia, mas também se apresenta como uma área-chave para o desenvolvimento de inovações futuras, com potencial para transformar radicalmente setores inteiros da economia global.

2 Fundamentos da Computação Paralela

Computação Paralela é uma abordagem de processamento de dados que busca aumentar a eficiência e a velocidade de execução ao dividir tarefas em partes menores que podem ser executadas simultaneamente em múltiplos processadores (HENNESSY & PATTERSON, 2002). Esse conceito contrasta com o modelo tradicional de processamento sequencial, onde as instruções são executadas uma de cada vez em um único núcleo de processamento. Ao permitir que diversas operações sejam realizadas ao mesmo tempo, a Computação Paralela se tornou uma ferramenta essencial para atender às crescentes demandas computacionais em áreas como ciência, engenharia, economia e publicidade, ambientes onde grandes volumes de dados variados precisam ser processados rapidamente.

2.1 Conceito e Definições

A Computação Paralela é uma área da ciência da computação que se dedica ao estudo e à implementação de algoritmos e sistemas capazes de executar múltiplas operações simultaneamente, com o objetivo de aumentar a eficiência e reduzir o tempo de processamento (HENNESSY & PATTERSON, 2002). Essa abordagem contrasta com a computação sequencial tradicional, na qual as instruções são executadas uma após a outra. A necessidade de processar volumes cada vez maiores de dados e realizar cálculos complexos em menor tempo impulsionou o desenvolvimento e a adoção de técnicas de computação paralela em diversas áreas, incluindo simulações científicas, processamento de imagens, inteligência artificial e análise de big data.

Historicamente, a computação paralela surgiu como resposta às limitações físicas e técnicas dos processadores sequenciais. Na medida em que a Lei de Moore[2] começou a enfrentar restrições práticas, como o aquecimento excessivo e o consumo de energia, a indústria da computação buscou alternativas para continuar aumentando o desempenho dos sistemas computacionais. O paralelismo ofereceu uma solução viável, permitindo que múltiplos processadores trabalhassem em conjunto para resolver problemas que seriam intratáveis ou extremamente lentos se tratados de forma sequencial. Com o advento de processadores multicore e a disponibilidade de hardware paralelo mais acessível, a computação paralela tornou-se uma prática comum tanto na pesquisa quanto na indústria.

No contexto da Computação Paralela, é importante distinguir entre os conceitos de *paralelismo* e *concorrência* (FIDELIS, 2023). Enquanto o paralelismo refere-se à execução simultânea de múltiplas operações, a concorrência diz respeito à estruturação do software de forma que múltiplas tarefas possam progredir de maneira independente, mas não necessariamente ao mesmo tempo. Em sistemas de múltiplos processadores ou núcleos, a concorrência pode ser utilizada para alcançar o paralelismo efetivo, mas também pode ser aplicada em sistemas de único processador para melhorar a responsividade e a organização das tarefas.

Dentro da Computação Paralela, destacam-se dois principais tipos de paralelismo: o paralelismo de dados e o paralelismo de tarefas. Cada um desses tipos aborda a execução simultânea de operações de maneiras distintas e é aplicável a diferentes tipos de problemas e cenários computacionais.

Paralelismo de Dados consiste na aplicação simultânea da mesma operação sobre diferentes conjuntos de dados. Esse tipo de paralelismo é especialmente eficaz em situações onde grandes volumes de dados precisam ser processados de forma uniforme. Por exemplo, no processamento de imagens digitais, a mesma operação de filtragem pode ser aplicada simultaneamente a diferentes *pixels* ou regiões da imagem, resultando em um processamento

[2] A chamada Lei de Moore afirma que o número de transistores em um chip de computador dobra aproximadamente a cada dois anos, resultando em um aumento exponencial na capacidade de processamento e uma redução nos custos de fabricação (MOORE, 1965).

mais rápido e eficiente. Outro exemplo comum é encontrado em operações matemáticas sobre vetores e matrizes, onde cálculos idênticos são realizados sobre diferentes elementos de estruturas de dados em paralelo.

As principais vantagens do paralelismo de dados incluem a simplicidade na divisão das tarefas e a eficiência na utilização dos recursos de hardware, especialmente em arquiteturas como GPUs (*Graphics Processing Units*), que são otimizadas para realizar operações idênticas em múltiplos dados simultaneamente. Além disso, esse tipo de paralelismo tende a escalar bem com o aumento dos recursos computacionais, permitindo que sistemas maiores processem volumes maiores de dados com eficiência proporcional.

Por outro lado, Paralelismo de Tarefas envolve a execução simultânea de diferentes operações ou tarefas, que podem ou não estar relacionadas entre si. Esse tipo de paralelismo é adequado para problemas que podem ser divididos em subtarefas distintas, cada uma realizando uma parte específica do processamento necessário. Um exemplo clássico é o pipeline de processamento em CPUs, onde diferentes etapas do processamento de instruções (como busca, decodificação, execução e escrita) são realizadas simultaneamente em diferentes instruções, aumentando a taxa geral de processamento.

No desenvolvimento de aplicações, o paralelismo de tarefas é frequentemente utilizado em sistemas multitarefa ou em aplicações com múltiplos componentes independentes. Por exemplo, em um servidor web, diferentes solicitações de clientes podem ser tratadas simultaneamente por diferentes *threads* ou processos, melhorando a capacidade de resposta e o *throughput* do sistema[3]. Em aplicações de engenharia, por exemplo, diferentes módulos de simulação podem ser executados em paralelo, permitindo a análise simultânea de múltiplos aspectos de um projeto complexo.

A implementação eficaz do paralelismo de tarefas requer uma cuidadosa coordenação e comunicação entre as diferentes tarefas, especialmente quando elas dependem umas das outras ou compartilham recursos comuns. Problemas como sincronização, *deadlocks* e condições de corrida são desafios comuns que precisam ser abordados através de técnicas adequadas de programação e gerenciamento de recursos.

A escolha entre paralelismo de dados e paralelismo de tarefas, ou mesmo a combinação de ambos, depende da natureza específica do problema a ser resolvido e da arquitetura do sistema computacional disponível. Em muitos casos, uma abordagem híbrida pode ser a mais eficaz, aproveitando as vantagens de ambos os tipos de paralelismo para maximizar o desempenho e a eficiência. Por exemplo, em aplicações de aprendizado de máquina, pode-se utilizar paralelismo de dados para processar grandes conjuntos de treinamento simultaneamente, enquanto diferentes fases do processo de aprendizado são gerenciadas através de paralelismo de tarefas.

É importante notar que, apesar dos benefícios significativos, a computação paralela

[3] *Throughput* refere-se à quantidade de dados ou tarefas processadas por um sistema em um determinado período de tempo, medindo a eficiência e a capacidade de desempenho do sistema.

também apresenta desafios consideráveis. A complexidade de desenvolver algoritmos paralelos eficazes, a necessidade de sincronização e comunicação eficiente entre processos, e as limitações impostas pela lei de Amdahl — que descreve como a proporção de uma tarefa que pode ser paralelizada limita os ganhos totais de desempenho (MATTEW & VIJAYAKUMAR, 2011) — são fatores que devem ser cuidadosamente considerados no planejamento e implementação de sistemas paralelos.

2.2 Arquiteturas Paralelas

As arquiteturas paralelas são a base que permite a execução de tarefas simultâneas em um sistema de computação, sendo fundamentais para o funcionamento da Computação Paralela. Essas arquiteturas variam em complexidade e são projetadas para atender a diferentes tipos de aplicações e necessidades computacionais (BODE & DAL CIN, 1993). Entre as mais conhecidas, destacam-se os sistemas multiprocessadores, *clusters* e *grids* (PARK et. al, 2012), cada um com suas características específicas, que influenciam diretamente o desempenho, a escalabilidade e a complexidade do desenvolvimento de software paralelo.

2.2.1 Multiprocessadores

Multiprocessadores são sistemas que possuem dois ou mais processadores, que trabalham em conjunto em um único computador. Esses processadores podem compartilhar a mesma memória (memória compartilhada) ou ter suas memórias independentes (memória distribuída) (PARK et. al, 2012). No caso de sistemas de memória compartilhada, todos os processadores acessam a mesma memória física, o que facilita a comunicação e o compartilhamento de dados entre as tarefas paralelas. No entanto, isso também pode levar a problemas de contenção de memória, onde múltiplos processadores competem pelo acesso aos mesmos dados, causando gargalos de desempenho. Esse tipo de arquitetura é comum em sistemas *multicore*, onde cada núcleo de um processador pode ser visto como um processador separado, trabalhando em paralelo com os outros núcleos.

Por outro lado, sistemas de memória distribuída são compostos por processadores, cada um com sua própria memória local. A comunicação entre os processadores ocorre por meio de mensagens, o que pode ser mais eficiente em termos de evitar contenção de memória, mas ao custo de uma maior complexidade no desenvolvimento de software, pois o programador precisa gerenciar explicitamente o fluxo de dados entre os processadores. Esses sistemas são altamente escaláveis, pois novos processadores podem ser adicionados ao sistema sem a necessidade de compartilhar a mesma memória física, o que torna esse tipo de arquitetura ideal para grandes sistemas de computação, como supercomputadores.

2.2.2 Clusters

Clusters são uma forma de arquitetura paralela que consiste em um conjunto de computadores interconectados, cada um com seu próprio processador e memória. Esses computadores, chamados de nós, trabalham juntos para resolver um problema comum, dividindo a carga de trabalho (PARK et. al, 2012). A comunicação entre os nós em um *cluster* é feita através de uma rede de alta velocidade, o que permite que os dados e tarefas sejam distribuídos eficientemente. *Clusters* são amplamente utilizados em ambientes de supercomputação, onde grandes volumes de dados precisam ser processados rapidamente. Um dos principais benefícios desta arquitetura é a sua escalabilidade, que permite aumentar o poder de processamento adicionando mais nós ao *cluster*. Além disso, *clusters* podem ser construídos utilizando *hardware* relativamente comum, tornando-os uma solução econômica para muitas organizações.

No entanto, *clusters* também apresentam desafios, especialmente no que diz respeito à comunicação entre nós. Como os nós de um *cluster* são sistemas independentes, a sincronização e a troca de dados entre eles podem introduzir latência, afetando o desempenho geral do sistema. Além disso, a programação para *clusters* exige que o desenvolvedor considere cuidadosamente a distribuição de tarefas e dados para minimizar a necessidade de comunicação entre os nós, que pode se tornar um gargalo se não for bem gerenciada.

2.2.3 Grids

Grids são uma extensão do conceito de *clusters*, mas com uma diferença fundamental: enquanto os nós de um cluster estão geralmente localizados no mesmo local físico e pertencem à mesma organização, os nós de um grid podem estar distribuídos geograficamente e pertencer a diferentes organizações. Um *grid* é, essencialmente, uma rede de computadores que trabalham juntos para realizar tarefas que exigem vastos recursos computacionais, muito além das capacidades de um único cluster ou supercomputador (PARK et. al, 2012). *Grids* são frequentemente utilizados em projetos de pesquisa que requerem grandes quantidades de poder de processamento, como na simulação de fenômenos naturais, processamento de grandes volumes de dados genômicos ou na análise de dados de experimentos físicos de alta energia.

A principal vantagem dos *grids* é sua capacidade de agregar recursos computacionais de diferentes fontes, permitindo a criação de uma infraestrutura poderosa e altamente escalável. No entanto, essa distribuição também introduz desafios significativos, especialmente em termos de coordenação e segurança. A heterogeneidade dos recursos, variando desde o hardware até as políticas administrativas de diferentes organizações, exige uma camada complexa de software para gerenciar a distribuição de tarefas e garantir a integridade e a confidencialidade dos dados. Além disso, a latência na

comunicação entre nós geograficamente dispersos pode ser um fator limitante, especialmente em aplicações que exigem uma coordenação estreita entre os diferentes processos.

Em termos de supercomputadores, muitos deles combinam aspectos de *clusters* e *grids*, utilizando milhares ou até milhões de núcleos de processamento distribuídos em sistemas interconectados. Esses sistemas são projetados para alcançar o máximo desempenho possível, sendo utilizados em cálculos científicos de larga escala, como simulações climáticas globais, estudos de cosmologia ou pesquisas em física de partículas. A arquitetura de um supercomputador é geralmente projetada sob medida para o tipo de cálculo que ele irá realizar, e pode incluir uma combinação de CPUs, GPUs, FPGAs e outros tipos de processadores especializados, organizados em uma estrutura paralela altamente otimizada.

A escolha da arquitetura paralela adequada depende de vários fatores, incluindo o tipo de aplicação, a necessidade de escalabilidade, o orçamento disponível e a expertise da equipe de desenvolvimento. Enquanto sistemas de memória compartilhada podem ser mais fáceis de programar, eles são menos escaláveis do que sistemas de memória distribuída ou *clusters*. Por outro lado, *grids* oferecem uma capacidade quase ilimitada de recursos computacionais, mas a um custo de maior complexidade de gerenciamento e comunicação.

2.3 Modelos de Programação Paralela

Os modelos de programação paralela são fundamentais para o desenvolvimento de softwares que tiram proveito das arquiteturas paralelas, permitindo que tarefas sejam distribuídas e executadas simultaneamente em diferentes processadores ou núcleos de processamento. Esses modelos fornecem as abstrações e estruturas necessárias para lidar com a complexidade da execução paralela, facilitando a implementação de algoritmos que aproveitam o paralelismo intrínseco de uma tarefa ou a divisão de grandes problemas em subproblemas menores, que podem ser resolvidos de forma independente e simultânea (MARTINS, 2020). Entre os principais modelos de programação paralela, destacam-se o SIMD (*Single Instruction, Multiple Data*), o MIMD (*Multiple Instruction, Multiple Data*) e o *Pipeline*, cada um com características próprias que os tornam mais adequados para diferentes tipos de aplicações (NONE MINGXIAN JIN et. al, 2002).

2.3.1 Single Instruction, Multiple Data (SIMD)

O SIMD (*Single Instruction, Multiple Data*) é um modelo de programação paralela em que uma única instrução é aplicada simultaneamente a múltiplos dados. Este modelo é particularmente eficaz em operações que envolvem grandes volumes de dados que precisam ser processados de maneira uniforme. Por exemplo, em operações vetoriais ou em processamento de imagens, onde a mesma operação aritmética ou lógica é aplicada a

diferentes elementos de um vetor ou pixels de uma imagem (NONE MINGXIAN JIN et. al, 2002). Arquiteturas SIMD são comuns em GPUs (*Graphics Processing Units*), que são projetadas para realizar cálculos massivamente paralelos, executando a mesma instrução em diferentes dados de forma eficiente. O SIMD é amplamente utilizado em áreas como gráficos, processamento de sinais e aprendizado de máquina, onde grandes quantidades de dados precisam ser processadas em paralelo.

O modelo SIMD oferece várias vantagens, incluindo uma alta eficiência na utilização dos recursos de hardware, já que todas as unidades de processamento estão executando a mesma instrução em diferentes dados ao mesmo tempo. Isso também simplifica a programação em comparação com modelos mais complexos, como o MIMD, pois não é necessário gerenciar a execução de múltiplas instruções distintas (HORD, 1993). No entanto, o SIMD tem limitações significativas, especialmente quando as tarefas não podem ser facilmente divididas em operações idênticas aplicadas a diferentes dados. Em tais casos, o paralelismo não pode ser totalmente explorado, levando a um uso ineficiente dos recursos de hardware.

2.3.2 Multiple Instruction, Multiple Data (MIMD)

Por outro lado, o MIMD (*Multiple Instruction, Multiple Data*) é um modelo de programação paralela em que múltiplos processadores executam diferentes instruções em diferentes conjuntos de dados ao mesmo tempo (NONE MINGXIAN JIN et. al, 2002). Esse modelo oferece uma grande flexibilidade, permitindo que tarefas paralelas sejam completamente independentes, cada uma realizando uma operação distinta em diferentes dados. Arquiteturas MIMD são comuns em sistemas multiprocessadores e clusters, onde cada processador pode estar executando um thread ou processo diferente, colaborando para resolver um problema maior. Este modelo é ideal para aplicações em que as tarefas paralelas são heterogêneas e precisam realizar diferentes tipos de cálculos ou operar em diferentes conjuntos de dados.

O MIMD é particularmente útil em aplicações que envolvem computação distribuída, onde diferentes nós de um cluster ou grid executam diferentes partes de uma tarefa complexa. Um exemplo clássico é encontrado em sistemas de simulação, onde diferentes módulos de uma simulação podem estar modelando aspectos distintos de um fenômeno físico ao mesmo tempo, como a dinâmica de fluidos e a transferência de calor em uma simulação de engenharia. No entanto, a programação em um ambiente MIMD é mais complexa do que em SIMD, pois o desenvolvedor precisa lidar com a coordenação entre as diferentes tarefas, gerenciando a comunicação e a sincronização entre os processos paralelos para evitar conflitos e garantir a consistência dos dados (HORD, 1993).

2.3.2 Pipeline

O *Pipeline* é outro modelo de programação paralela, onde um processo é dividido em várias etapas sequenciais, e cada etapa é executada em paralelo, mas em diferentes partes de um conjunto de dados (LIAO & BERKOVICH, 2013). Este modelo é comparável a uma linha de montagem em uma fábrica, onde cada estação realiza uma operação específica em um produto à medida que ele passa pela linha de produção. No contexto da computação, o *pipeline* é frequentemente utilizado em processadores RISC[4], onde diferentes estágios do ciclo de instrução (busca, decodificação, execução, etc.) são executados simultaneamente em diferentes instruções, aumentando a taxa de execução de instruções.

O *Pipeline* também é amplamente utilizado em fluxos de trabalho de processamento de dados, onde grandes volumes de dados precisam passar por várias etapas de processamento. Por exemplo, em um sistema de processamento de vídeo, o pipeline pode envolver a captura de vídeo, a compressão de dados, a transmissão de dados comprimidos e a descompressão e exibição de vídeo, com cada etapa sendo realizada em paralelo em diferentes quadros de vídeo. Este modelo oferece um aumento significativo na eficiência e no *throughput*, permitindo que múltiplos dados sejam processados ao mesmo tempo, cada um em uma fase diferente do *pipeline*. No entanto, a eficácia do modelo de *pipeline* depende da capacidade de balancear a carga entre as diferentes etapas, evitando que uma etapa se torne um gargalo que limite o desempenho geral do sistema (LIAO & BERKOVICH, 2013).

Além desses modelos, existem também modelos híbridos que combinam aspectos de SIMD, MIMD e pipeline para tirar o máximo proveito das características do hardware subjacente e das necessidades específicas da aplicação. Por exemplo, em arquiteturas modernas de GPUs, pode-se combinar SIMD para o processamento de dados gráficos massivos com MIMD para gerenciar diferentes *threads* de execução que tratam de aspectos diversos da renderização de uma cena complexa.

2.4 Ferramentas e Linguagens de Programação

O desenvolvimento de aplicações paralelas requer o uso de ferramentas e linguagens de programação específicas que permitem a criação, depuração e otimização de código capaz de executar em múltiplos processadores ou núcleos simultaneamente. Essas ferramentas fornecem as abstrações necessárias para programar em um ambiente paralelo, além de oferecerem suporte

[4] Processadores RISC (*Reduced Instruction Set Computer*) utilizam um conjunto reduzido de instruções, visando simplificar a execução de comandos e aumentar a eficiência. Essa abordagem permite que as instruções sejam executadas em um único ciclo de clock, resultando em maior velocidade e desempenho. O design RISC é caracterizado por instruções de tamanho fixo, um número limitado de modos de endereçamento e um foco em operações simples, o que facilita a otimização e o *pipelining* na execução de tarefas.

para a comunicação, sincronização e gerenciamento eficiente dos recursos computacionais. Entre as mais populares e amplamente utilizadas estão o MPI (*Message Passing Interface*), o OpenMP, o CUDA e as linguagens de programação como C/C++ e Python, que têm extensões ou bibliotecas voltadas para o paralelismo.

2.4.1 MPI

O MPI (*Message Passing Interface*) é um padrão amplamente adotado para programação paralela em ambientes com memória distribuída, como *clusters* e *grids*. O MPI permite que processos diferentes se comuniquem entre si por meio de troca de mensagens, possibilitando a execução de tarefas paralelas em sistemas onde cada processador possui sua própria memória local (GAVRILOVSKA, 2016). Uma das principais vantagens do MPI é sua portabilidade e flexibilidade, sendo suportado em uma ampla variedade de arquiteturas, desde computadores pessoais até supercomputadores.

Além disso, o MPI oferece um controle preciso sobre a comunicação entre processos, o que é essencial em aplicações que requerem alta performance e onde a latência de comunicação pode ser um fator crítico. No entanto, a programação com MPI pode ser complexa, exigindo que o desenvolvedor gerencie manualmente a distribuição de tarefas e a comunicação entre os diferentes nós, o que aumenta a complexidade do desenvolvimento e da manutenção do código (GAVRILOVSKA, 2016).

2.4.2 OpenMP

O OpenMP é outra ferramenta amplamente utilizada, mas voltada principalmente para ambientes de memória compartilhada, como sistemas multicore. Consiste em uma API[5] que permite ao programador adicionar diretivas ao código-fonte, geralmente em C, C++ ou Fortran, para especificar quais partes do código devem ser executadas em paralelo (CARLETON UNIVERSITY). Essas diretivas são simples e intuitivas, facilitando a paralelização de programas existentes sem a necessidade de reescrever grandes partes do código. Uma das grandes vantagens do OpenMP é sua simplicidade e facilidade de uso, o que o torna ideal para introduzir o paralelismo em programas sequenciais já existentes.

O OpenMP também permite um controle detalhado sobre o paralelismo, oferecendo suporte para a criação de regiões paralelas, distribuição de tarefas entre threads, sincronização, e gerenciamento de variáveis compartilhadas e privadas. No entanto, o OpenMP é mais adequado para sistemas com memória compartilhada, e sua aplicação em sistemas de memória distribuída é limitada.

[5] API (*Application Programming Interface*) é um conjunto de regras e protocolos que permite que diferentes softwares se comuniquem e interajam entre si, facilitando a integração de funcionalidades e dados.

2.4.3 CUDA

O CUDA (*Compute Unified Device Architecture*), desenvolvido pela NVIDIA, é uma plataforma e modelo de programação que permite o uso de GPUs para a computação geral, além dos gráficos. GPUs são altamente eficientes para executar operações paralelas massivas devido à sua arquitetura SIMD, e o CUDA fornece um ambiente de programação que facilita o uso dessas capacidades para uma ampla gama de aplicações, como processamento de imagens, simulações científicas e aprendizado de máquina. O CUDA permite que os desenvolvedores escrevam código em C/C++ com extensões específicas para programar kernels, que são funções executadas em paralelo em milhares de *threads* na GPU (COOK, 2013).

A utilização do CUDA pode resultar em aumentos significativos de desempenho em tarefas que podem ser massivamente paralelizadas. No entanto, o desenvolvimento em CUDA requer uma compreensão profunda da arquitetura da GPU e das técnicas de otimização específicas, como a gestão da memória compartilhada e a minimização da divergência de *threads*, para aproveitar ao máximo o potencial das GPUs.

Além dessas ferramentas específicas, muitas linguagens de programação têm bibliotecas ou extensões que facilitam a programação paralela. C/C++, por exemplo, continuam sendo as linguagens mais utilizadas em computação de alto desempenho devido à sua eficiência e controle sobre os recursos de hardware. Para esses ambientes, existem bibliotecas como o *Pthreads* (*POSIX threads*), que oferecem um nível baixo de controle sobre a criação e sincronização de *threads*, permitindo a construção de aplicações altamente otimizadas, embora com uma complexidade significativa na programação.

A linguagem Python, por sua vez, tem se tornado cada vez mais popular na programação paralela devido à sua simplicidade e a uma vasta gama de bibliotecas que suportam paralelismo, como o *multiprocessing*, que permite a criação de processos paralelos em memória compartilhada, e o *Dask*, que facilita a paralelização de operações em grandes conjuntos de dados, escalando de um único computador para clusters distribuídos. Embora Python não ofereça o mesmo nível de desempenho bruto que C/C++, seu uso combinado com bibliotecas como *NumPy* e *CuPy* (que permitem o uso de GPUs via CUDA) oferece uma solução poderosa para a prototipagem rápida e execução de tarefas paralelas.

Outra opção importante é o OpenCL (*Open Computing Language*), que permite o desenvolvimento de programas que podem ser executados em diferentes plataformas de hardware, incluindo CPUs, GPUs, FPGAs[6], e outros processadores. OpenCL é um padrão aberto que proporciona um alto grau de portabilidade e flexibilidade, permitindo que o código paralelo seja executado em uma ampla variedade de dispositivos sem a necessidade de grandes alterações (GASTER, 2013). No entanto, assim como CUDA, a programação em OpenCL pode

[6] FPGA (*Field Programmable Gate Array*) é um circuito integrado programável que pode ser configurado após sua fabricação, permitindo implementar diferentes funções lógicas e circuitos digitais de acordo com as necessidades do usuário.

ser complexa e exige uma compreensão detalhada da arquitetura do hardware em que o código será executado.

Por fim, o surgimento de frameworks de alto nível, como *TensorFlow* e *PyTorch* para aprendizado de máquina, também trouxe consigo abstrações que facilitam a programação paralela, especialmente em GPUs. Esses frameworks gerenciam grande parte da complexidade associada à programação paralela e à utilização de hardware especializado, permitindo que pesquisadores e desenvolvedores se concentrem mais no desenvolvimento de modelos e menos na otimização de baixo nível.

3 Vantagens e Aplicações da Computação Paralela

A Computação Paralela oferece uma série de vantagens significativas em relação à computação sequencial tradicional, sendo um pilar essencial para o avanço da tecnologia em diversas áreas. Um dos principais benefícios é o aumento de performance, que permite que grandes volumes de dados sejam processados em uma fração do tempo que levaria em um sistema sequencial. Em vez de realizar tarefas de forma linear, a abordagem paralela divide um problema em subproblemas menores que podem ser resolvidos simultaneamente por múltiplos processadores ou núcleos, resultando em uma redução drástica no tempo total de processamento. Esse aumento de desempenho é crucial em áreas que exigem processamento intensivo, como simulações científicas, modelagem de clima, e análise de grandes volumes de dados.

Outra vantagem importante é a eficiência energética. Embora possa parecer que o uso de múltiplos processadores consumiria mais energia, a Computação Paralela pode, na verdade, ser mais eficiente energeticamente. Isso ocorre porque, ao executar tarefas em paralelo, o tempo total de execução é reduzido, o que permite que o sistema complete a tarefa e entre em um estado de baixo consumo de energia mais rapidamente. Além do mais, processadores modernos são projetados para operar de forma eficiente em termos de energia quando executam várias tarefas simultaneamente. Em supercomputadores e data centers, onde o consumo de energia é uma preocupação crítica, essa abordagem de processamento oferece uma forma de maximizar a performance sem comprometer a sustentabilidade.

A escalabilidade é outra vantagem crucial. Sistemas paralelos podem ser ampliados de forma relativamente simples ao adicionar mais processadores ou nós ao sistema, seja em um supercomputador, *cluster* ou *grid*. Isso permite que as organizações aumentem sua capacidade computacional conforme necessário, sem a necessidade de reformular completamente sua infraestrutura. Essa escalabilidade é especialmente importante em aplicações que envolvem volumes crescentes de dados, como análise de *big data* e inteligência artificial. Além disso, a escalabilidade também se estende à capacidade de lidar com tarefas de diferentes magnitudes, desde pequenas simulações científicas até complexas previsões climáticas globais.

A flexibilidade oferecida pela Computação Paralela é outro aspecto digno de nota. Diversos

modelos de paralelismo, como os citados anteriormente, podem ser aplicados para otimizar diferentes tipos de tarefas, permitindo que os sistemas sejam ajustados para atender às necessidades específicas de cada aplicação. Essa flexibilidade também se reflete na variedade de arquiteturas paralelas disponíveis, desde simples sistemas *multicore* em computadores pessoais até redes complexas de supercomputadores e *grids* distribuídos globalmente. Essa diversidade permite que a Computação Paralela seja adaptada a uma ampla gama de problemas, aumentando a sua aplicabilidade em diferentes setores.

A Computação Paralela é amplamente utilizada em diversas áreas, sendo fundamental para resolver problemas que seriam inviáveis de outra forma. Simulações científicas são uma das áreas mais notórias onde a Computação Paralela desempenha um papel crucial. Em áreas como física, química, biologia e engenharia, a capacidade de simular fenômenos naturais em alta resolução e em grande escala é essencial. Por exemplo, simulações de dinâmica de fluidos computacional (CFD) utilizam a Computação Paralela para modelar o comportamento de líquidos e gases em diferentes condições, o que é essencial em projetos aeroespaciais, engenharia civil e previsão de clima. Da mesma forma, simulações em biologia molecular, como o dobramento de proteínas, beneficiam-se do paralelismo para explorar grandes espaços de configuração molecular em busca de novas drogas e tratamentos médicos.

Na área de inteligência artificial (IA), a Computação Paralela é indispensável para o treinamento de modelos de aprendizado profundo (*deep learning*). Redes neurais profundas, que são a base de muitas aplicações modernas de IA, como reconhecimento de imagem, processamento de linguagem natural e condução autônoma, exigem enormes volumes de dados e poder computacional para serem treinadas. GPUs, com sua capacidade de realizar operações paralelas em massa, são amplamente utilizadas para acelerar o processo de treinamento, permitindo que modelos complexos sejam treinados em horas ou dias, em vez de semanas ou meses. Além disso, *frameworks* de aprendizado profundo, como *TensorFlow* e *PyTorch*, são projetados para tirar proveito da Computação Paralela, facilitando o desenvolvimento de soluções de IA de alta performance.

A análise de *big data* é outra aplicação onde o processamento paralelo é crucial. Com o crescimento exponencial da quantidade de dados gerados por empresas, dispositivos e redes sociais, a capacidade de processar e analisar grandes volumes de dados em tempo hábil tornou-se um diferencial competitivo. Ferramentas de processamento de dados paralelos, como *Apache Hadoop* e *Apache Spark*, permitem a distribuição de tarefas de processamento de dados em *clusters* de computadores, acelerando a análise e permitindo a descoberta de insights valiosos a partir de dados que, de outra forma, seriam impossíveis de analisar em tempo hábil. Esses *insights* podem ser aplicados em diversos setores, desde marketing e finanças até saúde e segurança pública.

Além dessas áreas, a Computação Paralela também é essencial em modelagem financeira, onde é utilizada para simular cenários econômicos complexos e avaliar riscos financeiros com alta precisão. Em engenharia e design assistido por computador (CAD), a capacidade de realizar cálculos intensivos paralelamente permite a criação de modelos detalhados e a execução de simulações de estresse e

otimização de design em tempo real. A indústria de entretenimento também se beneficia enormemente da Computação Paralela, especialmente na criação de efeitos visuais para filmes e jogos, onde a renderização de cenas complexas e a simulação de fenômenos naturais, como fogo e água, são realizadas em ambientes paralelos para garantir alta qualidade e realismo.

4 Desafios e Limitações

Embora a Computação Paralela tenha proporcionado avanços significativos em diversas áreas, ainda existem desafios técnicos substanciais que precisam ser superados para maximizar seu potencial. Um dos principais desafios enfrentados por essa abordagem de processamento é, justamente, uma de suas principais vantagens: a escalabilidade. À medida que o número de processadores ou nós em um sistema paralelo aumenta, garantir que todos esses recursos sejam utilizados de maneira eficiente torna-se cada vez mais complexo. O desafio está em distribuir as tarefas de forma que todos os processadores estejam constantemente ocupados, sem tempos ociosos significativos, e ao mesmo tempo, minimizar a sobrecarga de comunicação entre eles. Em sistemas de larga escala, como supercomputadores e *grids* distribuídos, essa tarefa se torna ainda mais difícil devido à latência de comunicação e à necessidade de coordenação entre milhares de processadores.

A sincronização de processos é outro desafio crítico. Em muitos casos, diferentes tarefas paralelas dependem dos resultados umas das outras, o que exige mecanismos de sincronização para garantir que essas dependências sejam respeitadas. A sincronização inadequada pode levar a problemas como *deadlocks* (situação em que processos ficam presos esperando por recursos uns dos outros), *starvation* (quando um processo é impedido de progredir porque outros processos monopolizam os recursos) e *race conditions* (onde o resultado do processamento depende da ordem de execução dos processos). Esses problemas não apenas complicam o desenvolvimento de sistemas paralelos, mas também podem impactar gravemente o desempenho e a correção dos programas.

O balanceamento de carga é mais um desafio significativo. Em um ambiente paralelo, é essencial que a carga de trabalho seja distribuída de forma equilibrada entre todos os processadores disponíveis para evitar situações onde alguns processadores estão sobrecarregados enquanto outros estão subutilizados. No entanto, alcançar um balanceamento de carga eficiente é uma tarefa complexa, especialmente em aplicações onde a carga de trabalho pode variar dinamicamente ao longo do tempo. Métodos de balanceamento de carga, como divisão estática e balanceamento dinâmico, cada um com suas próprias vantagens e desvantagens, precisam ser cuidadosamente selecionados e ajustados de acordo com a natureza específica da aplicação.

Embora a Computação Paralela ofereça enormes benefícios em termos de desempenho e eficiência, ela também traz consigo uma série de desafios e limitações que devem ser cuidadosamente gerenciados. A superação desses desafios requer um profundo entendimento das questões técnicas envolvidas, bem como o desenvolvimento de técnicas e ferramentas inovadoras que possam mitigar os problemas de escalabilidade, desempenho, segurança e consistência.

4.1 Problemas de Desempenho

Apesar das promessas de maior desempenho oferecidas pela Computação Paralela, a otimização de desempenho em arquiteturas paralelas apresenta vários problemas desafiadores. Um dos principais é a *Lei de Amdahl*, que estabelece um limite teórico para o ganho de desempenho obtido pela paralelização de uma tarefa. De acordo com essa lei, o aumento de desempenho é limitado pela fração do código que deve ser executado sequencialmente. Mesmo que uma parte significativa do programa seja paralelizada, se houver uma pequena fração que não pode ser paralelizada, o ganho de desempenho global será limitado. Essa limitação torna-se ainda mais pronunciada em aplicações que têm um grande componente sequencial, reduzindo o benefício de adicionar mais processadores ao sistema.

Além disso, a localidade de dados e o acesso à memória são fatores cruciais que afetam o desempenho em sistemas paralelos. Em arquiteturas de memória compartilhada, múltiplos processadores competem pelo acesso à memória principal, o que pode criar gargalos e reduzir o desempenho. A eficiência do *cache* também se torna um problema, pois a coerência do *cache* precisa ser mantida, o que pode introduzir sobrecarga adicional. Em arquiteturas de memória distribuída, a latência de comunicação entre diferentes nós pode degradar o desempenho, especialmente se o programa requer um volume significativo de transferência de dados entre nós. Essas questões tornam a otimização do desempenho em sistemas paralelos uma tarefa complexa que exige um profundo entendimento da arquitetura subjacente e do comportamento do programa.

Outro problema de desempenho é a sobrecarga de paralelização, que ocorre quando o custo de gerenciar a execução paralela supera os ganhos obtidos com a paralelização. Isso pode incluir a sobrecarga associada à criação de threads, à sincronização entre processos, à comunicação de dados entre diferentes nós ou processadores, e à manutenção da consistência do cache. Em aplicações com grão fino (onde as tarefas paralelas são muito pequenas), essa sobrecarga pode ser significativa, a ponto de anular qualquer benefício de desempenho que a paralelização poderia trazer.

4.2 Segurança e Consistência

A segurança e a consistência dos dados em ambientes paralelos são desafios críticos que precisam ser cuidadosamente gerenciados para garantir a correção e confiabilidade das aplicações. Em sistemas paralelos, vários processos ou *threads* podem acessar e modificar os mesmos dados simultaneamente, o que pode levar a inconsistências se não houver mecanismos adequados de controle. Condições de corrida (*race conditions*) são um exemplo clássico desse problema, onde o resultado final depende da ordem de execução dos processos, potencialmente resultando em comportamentos indesejados ou incorretos. Para mitigar esses problemas, técnicas como bloqueios (*locks*), semáforos e outras formas de sincronização são

frequentemente usadas para garantir que apenas um processo ou *thread* possa acessar um recurso compartilhado por vez. No entanto, o uso dessas técnicas pode introduzir problemas adicionais, como *deadlocks* e diminuição do desempenho devido ao aumento da latência.

Outro aspecto crítico relacionado à segurança em ambientes paralelos é a gestão de recursos compartilhados, como memória, arquivos e dispositivos de E/S (*I/O*). Sem um gerenciamento adequado, múltiplos processos podem tentar acessar ou modificar os mesmos recursos simultaneamente, resultando em corrupção de dados, falhas de sistema ou comportamentos imprevisíveis. Isso é particularmente preocupante em sistemas de tempo real, onde atrasos ou falhas podem ter consequências graves. Portanto, é essencial implementar mecanismos robustos de controle de acesso e sincronização para garantir que os recursos sejam utilizados de maneira segura e eficiente.

Além disso, a consistência dos dados em sistemas distribuídos é um desafio significativo, especialmente em arquiteturas de memória distribuída, onde diferentes nós podem manter cópias locais de dados que precisam ser sincronizados. A manutenção da consistência em face de falhas de rede, latência de comunicação ou falhas de nó requer protocolos complexos, como algoritmos de consenso e mecanismos de replicação de dados. No entanto, esses protocolos podem introduzir sobrecarga adicional, impactando negativamente o desempenho do sistema. Em ambientes onde a consistência forte é necessária, como em sistemas financeiros ou de controle de missão, a implementação de tais protocolos é essencial, mas deve ser balanceada com a necessidade de desempenho.

5 Futuro da Computação Paralela

Nos últimos anos, a computação paralela tem evoluído de forma acelerada, impulsionada pela demanda crescente por desempenho em aplicações que vão desde simulações científicas até inteligência artificial. Algumas tendências tecnológicas moldarão o futuro dessa área:

- **Aumento do uso de GPUs**: As unidades de processamento gráfico (GPUs) já são amplamente utilizadas para paralelizar tarefas em áreas como aprendizado de máquina, processamento de imagens e simulações. Com a demanda crescente por deep learning e inteligência artificial, o uso de GPUs só tende a aumentar. Empresas como NVIDIA e AMD continuam a inovar em arquiteturas que otimizam o desempenho em ambientes paralelos.
- **Computação Quântica**: A computação quântica está se aproximando de uma fase onde poderá começar a resolver problemas de forma exponencialmente mais rápida do que os computadores clássicos, especialmente em tarefas paralelizadas. Embora a tecnologia ainda esteja em sua infância, empresas como IBM, Google e startups de tecnologia quântica estão investindo pesadamente para viabilizar processadores quânticos escaláveis. No futuro, a computação paralela pode se beneficiar dessa revolução, permitindo resolver problemas complexos que atualmente são impraticáveis, como simulações moleculares e otimizações

complexas.

- **Arquiteturas Heterogêneas**: Em vez de depender exclusivamente de CPUs ou GPUs, há um movimento crescente em direção a arquiteturas heterogêneas, onde diferentes tipos de processadores (CPUs, GPUs, FPGAs, TPUs, NPUs) trabalham em conjunto para resolver tarefas específicas de forma mais eficiente. Essas arquiteturas personalizadas, que incluem aceleradores especializados, permitem que tarefas altamente paralelas e que exigem muita potência de processamento sejam distribuídas entre diferentes tipos de hardware, maximizando o desempenho e a eficiência energética.
- **Memórias e Interconexões mais Rápidas**: O avanço na tecnologia de memória (como HBM e memórias não voláteis) e nas interconexões (como NVLink e PCIe 6.0) será crucial para sustentar o aumento do processamento paralelo. O gargalo de comunicação entre diferentes componentes do sistema precisa ser minimizado para evitar que a latência prejudique o desempenho de sistemas paralelos.

5.1 Impacto no Mercado e na Sociedade

- **Transformação no Mercado de Trabalho**: A evolução da computação paralela irá demandar profissionais altamente qualificados em áreas como ciência de dados, inteligência artificial, e desenvolvimento de software especializado. O mercado verá uma demanda crescente por engenheiros de software capazes de desenvolver sistemas que tirem proveito de GPUs, FPGAs, e outros aceleradores. Ao mesmo tempo, o uso de novas ferramentas de abstração e automação pode facilitar o acesso a essas tecnologias, permitindo que mais desenvolvedores entrem nesse campo.
- **Impacto na Indústria**: Setores como biotecnologia, finanças e entretenimento serão diretamente impactados. Na biotecnologia, a capacidade de simular processos biológicos complexos de forma mais rápida pode acelerar a descoberta de novos medicamentos. No setor financeiro, a computação paralela pode melhorar a precisão de algoritmos de previsão e análise de risco. Já na indústria de entretenimento, o processamento gráfico e de física em tempo real permitirá experiências mais imersivas em jogos e realidade aumentada/virtual.
- **Impacto na Sociedade**: A computação paralela, especialmente através do uso da inteligência artificial, pode transformar várias áreas da vida cotidiana. Desde diagnósticos médicos mais rápidos e precisos até sistemas de transporte inteligentes e personalizados. No entanto, a automação proporcionada por esses avanços também traz preocupações, como o potencial desemprego em setores que se automatizam mais rapidamente do que as pessoas podem se requalificar.
- **Inclusão e Desigualdade Digital**: O acesso à tecnologia de computação paralela pode criar uma nova divisão digital. Enquanto grandes corporações e países desenvolvidos podem tirar proveito dessas inovações, regiões menos desenvolvidas ou empresas menores podem enfrentar dificuldades para competir. Portanto, políticas de inclusão tecnológica e

treinamentos em larga escala serão essenciais para mitigar essa disparidade.

6 Conclusão

Em síntese, este trabalho revisitou os principais aspectos da Computação Paralela, incluindo seus conceitos básicos, arquiteturas e modelos de programação, além de explorar suas aplicações em áreas como simulações científicas e inteligência artificial. Foram discutidas as vantagens, como a redução do tempo de processamento, e os desafios, especialmente no que tange à escalabilidade, otimização do desempenho e segurança.

Fica claro que a Computação Paralela continuará a desempenhar um papel essencial na evolução tecnológica, especialmente com o aumento contínuo da demanda por soluções que processam grandes volumes de dados de forma mais eficiente. No entanto, superar os obstáculos atuais, como a complexidade de paralelização e a necessidade de algoritmos mais eficientes, será crucial para seu avanço.

Por fim, as perspectivas futuras incluem novas áreas de pesquisa, como o desenvolvimento de melhores ferramentas para otimização de desempenho e soluções para questões de segurança. Além disso, a integração da Computação Paralela com tecnologias emergentes, como computação quântica e redes neurais mais sofisticadas, poderá abrir novos horizontes, ampliando ainda mais suas aplicações e impacto.

7 Referências Bibliográficas

BODE, A.; DAL CIN, M., *Parallel Computer Architectures: Theory, Hardware, Software, Applications.* [s.l.] Springer, 1993.

CARLETON UNIVERSITY, *Introduction to OpenMP - Research Computing Services*. Disponível em: <https://carleton.ca/rcs/rcdc/introduction-to-openmp/>. Acesso em 02 set. 2024.

COOK, S. C*UDA programming : a developer's guide to parallel computing with GPUs.* Amsterdam ; Boston: Morgan Kaufmann, 2013.

FIDELIS, M., *Paralelismo, concorrência e multithreading.* Disponível em: https://fidelissauro.dev/concorrencia-paralelismo/. Acesso em: 01 set. 2024.

GASTER, B. et al. *Heterogeneous Computing With OpenCL.* Amsterdam ; Waltham, Ma: Elsevier Morgan Kaufmann, 2013.

GAVRILOVSKA, A. *Attaining High Performance Communications.* [s.l.] CRC Press, (2016).

HENNESSY, J.L.; PATTERSON, D.A., *Computer architecture: a quantitative approach.* (3rd ed.). San Francisco, Calif.: International Thomson, p. 43, (2002).

HORD, R. M., *Parallel Supercomputing in MIMD Architectures*. Boca Raton, CRC Press, (1993).

LIAO, D.; BERKOVICH, S., *A Multi-Core Pipelined Architecture for Parallel Computing.* George Washington University, (2013).

MARTINS, G., *Avaliação do Uso de Desafios no Aprendizado de Programação Paralela*. 2020. Dissertação (Mestrado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, (2020). doi:10.11606/D.55.2020.tde-10092020-160414. Acesso em: 2024-09-01.

MATTEW, J.; VIJAYAKUMAR R., *The Performance of Parallel Algorithms by Amdahl's Law, Gustafson's Trend.* (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (6), ISSN 2796-2799, (2011).

NONE MINGXIAN JIN; BAKER, J. W.; W.C. MEILANDER. *The power of SIMDs vs. MIMDs in real-time scheduling*. CiteSeer X (The Pennsylvania State University), (2002).

PARK, J. J. et al. *Network and Parallel Computing* : 9th IFIP International Conference, NPC 2012, Gwangju, Korea, September 6-8, 2012, Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, (2012).

RAFIQUL, Z.; KHAN; FIROJ, M. *Current Trends in Parallel Computing*. International Journal of Computer Applications, v. 59, n. 2, p. 975–8887, (2012).