UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO DE CIÊNCIAS EXATAS, NATURAIS E DA SAÚDE
DEPARTAMENTO DE COMPUTAÇÃO

GEOVANE DE NOVAIS GOMES

COMPUTAÇÃO PARALELA COM OPENCL: UMA REVISÃO DE LITERATURA

ALEGRE - ES
2024

GEOVANE DE NOVAIS GOMES

COMPUTAÇÃO PARALELA COM OPENCL: UMA REVISÃO DE LITERATURA

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Dayan de Castro Bissoli.

ALEGRE - ES

2024

GEOVANE DE NOVAIS GOMES

COMPUTAÇÃO PARALELA COM OPENCL: UMA REVISÃO DE LITERATURA

Trabalho de conclusão de curso apresentado ao Departamento de Computação do Centro de Ciências Exatas, Naturais e da Saúde da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em ____ de outubro de 2024.

COMISSÃO EXAMINADORA

Documento assinado digitalmente
DAYAN DE CASTRO BISSOLI
Data: 07/10/2024 08:45:16-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Dr. Dayan de Castro Bissoli
Universidade Federal do Espírito Santo
Orientador

Documento assinado digitalmente
JACSON RODRIGUES CORREIA DA SILVA
Data: 07/10/2024 14:30:39-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Dr. Jacson Rodrigues Correia da Silva
Universidade Federal do Espírito Santo

Documento assinado digitalmente
RODRIGO FREITAS SILVA
Data: 07/10/2024 13:19:49-0300
Verifique em https://validar.iti.gov.br

_____

Prof. Dr. Rodrigo Freitas Silva
Universidade Federal do Espírito Santo

DEDICATÓRIA

“Um dia, tudo o que restará de nós serão os nossos ossos. A vida é uma grande piada, sem fim e sem sentido.”

One Piece

AGRADECIMENTOS

Agradeço primeiramente a Deus, pela força e pela sabedoria concedidas ao longo de toda a minha trajetória acadêmica, iluminando meus caminhos e me proporcionando todas as oportunidades que me trouxeram até aqui.

À minha família, por todo o sacrifício, amor e dedicação, que foram fundamentais para que eu pudesse me concentrar nos estudos. A vocês, que sempre acreditaram no meu potencial e me deram suporte emocional e financeiro, sou eternamente grato.

À minha namorada, Daniela Trindade de Abreu, por estar ao meu lado nos momentos mais desafiadores, oferecendo seu carinho, paciência e palavras de encorajamento. Sua presença foi essencial para que eu conseguisse manter o equilíbrio entre os estudos e a vida pessoal.

Ao meu orientador, Dayan de Castro Bissoli, por sua orientação precisa e por sempre estar disponível para guiar-me com paciência, atenção e compromisso. Suas sugestões e críticas construtivas contribuíram imensamente para a qualidade deste trabalho.

À Universidade Federal do Espírito Santo, por ser um espaço de aprendizado, crescimento e desenvolvimento pessoal e acadêmico. Agradeço também a todos os funcionários e professores do Departamento de Computação, que, com seu conhecimento e dedicação, me ajudaram a chegar até aqui.

Aos meus amigos, que ao longo desta jornada acadêmica compartilharam conhecimentos, trocas de ideias e momentos de descontração, tornando esse caminho mais leve e enriquecedor.

Finalmente, a todos aqueles que, direta ou indiretamente, contribuíram para a realização deste trabalho, seja por meio de apoio moral, técnico ou emocional. Cada um de vocês fez parte desta conquista.

RESUMO

Dentro do contexto de computação paralela, este trabalho teve como objetivo analisar o OpenCL (*Open Computing Language)* e compará-la com sua principal alternativa, o CUDA (*Compute Unified Device Architecture*), abordando aspectos técnicos, desempenho e usabilidade de ambas as plataformas. A pesquisa foi realizada através de uma análise crítica da literatura, considerando vinte e nove trabalhos selecionados, incluindo artigos científicos e documentação técnica. O estudo também considerou a tecnologia dOCAL (*Distributed OpenCL/CUDA Abstraction Layer*), que surge como um aliado importante para o OpenCL ao proporcionar uma abstração de alto nível que facilita a programação paralela e distribuída. A combinação de OpenCL com dOCAL permite superar limitações do OpenCL e explorar seu potencial completo em ambientes heterogêneos. Os resultados indicam que a escolha entre OpenCL e CUDA deve ser baseada nas necessidades específicas do projeto. Para aplicações que visam maximizar o desempenho em GPUs (*Graphics Processing Unit*) da NVIDIA, o CUDA é recomendado, enquanto para ambientes que priorizam portabilidade e compatibilidade com diversos dispositivos, o OpenCL é mais adequado. A integração com tecnologias como o dOCAL pode ampliar ainda mais as possibilidades do OpenCL, especialmente em contextos científicos e de alta performance.

Palavras-chave: OpenCL, CUDA, programação paralela, computação heterogênea, dOCAL.

ABSTRACT

In the context of parallel computing, this work aimed to analyze OpenCL (Open Computing Language) and compare it with its main alternative, CUDA (Compute Unified Device Architecture), addressing technical aspects, performance, and usability of both platforms. The research was conducted through a critical literature review, considering twenty-nine selected works, including scientific articles and technical documentation. The study also considered the dOCAL (Distributed OpenCL/CUDA Abstraction Layer) technology, which emerges as an important ally for OpenCL by providing a high-level abstraction that facilitates parallel and distributed programming. The combination of OpenCL with dOCAL allows overcoming the OpenCL limitations and exploring its full potential in heterogeneous environments. The results indicate that the choice between OpenCL and CUDA should be based on the specific needs of the project. For applications aiming to maximize performance on NVIDIA's Graphics Processing Units (GPUs), CUDA is recommended, while for environments that prioritize portability and compatibility with various devices, OpenCL is more suitable. Integration with technologies such as dOCAL can further enhance OpenCL's possibilities, especially in scientific and high-performance contexts.

Keywords: OpenCL, CUDA, parallel programming, heterogeneous computing, dOCAL.

LISTA DE FIGURAS

Figura 1 - Organização dos work-groups em um NDRange 20
Figura 2 - Hierarquia de memória. 24
Figura 3 - O processo de recolha e filtragem da literatura..... 28

LISTA DE TABELAS

Tabela 1 - Resultados da busca na plataforma Google Acadêmico com as palavras-chave em português. 27
Tabela 2 - Resultados da busca na plataforma Google Acadêmico com as palavras-chave em inglês..... 27

SUMÁRIO

1. INTRODUÇÃO 12

1.1. O PROBLEMA E SUA IMPORTÂNCIA 14

1.2. OBJETIVOS 15

1.2.1. Objetivo Geral 15

1.2.2. Objetivos Específicos 15

1.3. ESTRUTURA DO TRABALHO 15

2. REVISÃO DA LITERATURA 17

2.1. OPENCL 18

2.2. PROGRAMAÇÃO PARALELA 20

2.3. COMPUTAÇÃO HETEROGÊNEA 21

2.4. CUDA 23

3. METODOLOGIA 26

3.1. COLETA DE DADOS 28

3.2. ANÁLISE DOS DADOS 28

4. RESULTADOS 30

4.1. ANÁLISE DO OPENCL 30

4.2. COMPARAÇÃO ENTRE OPENCL E CUDA 30

4.3. IMPLICAÇÕES PARA A PRÁTICA 31

4.4. TECNOLOGIA dOCAL COMO ALIADO PARA OPENCL 32

5. CONCLUSÕES 33

5.1. IMPLICAÇÕES PRÁTICAS 33

5.2. CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES 34

REFERÊNCIAS 35

1. INTRODUÇÃO

A Computação de Alto Desempenho (HPC) é uma tecnologia que opera através de *clusters* de processadores potentes, que trabalham em conjunto para processar grandes conjuntos de dados multidimensionais e resolver problemas complexos em velocidades extremamente rápidas. Os sistemas HPC geralmente operam em velocidades que são mais de um milhão de vezes mais rápidas do que os sistemas de *desktop*, *notebook* ou servidor mais rápidos. A HPC é utilizada para sequenciar DNA, automatizar a negociação de ações e executar algoritmos e simulações de inteligência artificial (IA) que analisam terabytes de fluxos de dados de sensores de IoT (*Internet of Things*), radares e sistemas GPS em tempo real para tomar decisões em frações de segundo (IBM, 2024).

Em contextos HPC, a computação paralela é uma alternativa que vem sendo cerceada nos últimos anos. Essa técnica de processamento permite a execução simultânea de várias tarefas computacionais. O princípio central desta metodologia é que problemas de grande escala podem ser fragmentados em partes menores, e assim, poderem ser processadas simultaneamente. O propósito é acelerar o processamento e obter resultados de maneira mais rápida e eficiente (HENNESSY e PATTERSON, 2002; MACIEL JR, DA SILVA, RODRIGUES, 2013).

Dessa forma, o OpenCL (*Open Computing Language*) emerge como uma alternativa. OpenCL oferece uma API (*Application Programming Interface*) comum para a execução em sistemas compostos, permitindo a exploração do poder computacional de unidades de processamento heterogêneas, como CPUs (*Central Processing Unit*) e GPUs (*Graphics Processing Unit*) (STONE, GOHARA, SHI, 2010). Assim como o CUDA (*Compute Unified Device Architecture*), que é uma API proprietária da NVIDIA e otimizada para suas GPUs, o OpenCL se destaca por sua portabilidade, funcionando em dispositivos de diversos fabricantes. Enquanto o CUDA é utilizado por oferecer desempenho superior em GPUs NVIDIA, o OpenCL apresenta a vantagem de suportar uma gama mais ampla de arquiteturas. Neste trabalho, ambos os *frameworks* serão analisados e comparados em termos de desempenho e aplicabilidade, visando entender suas vantagens e limitações em diferentes cenários de computação paralela.

O OpenCL permite uma ampla gama de tarefas paralelas serem descritas explicitamente através de filas de comando fora de ordem e sincronização de

eventos. Além disso, a partir da versão 2.0 do padrão, a extração implícita de paralelismo de tarefas de filas de comando em ordem passou a ser permitida (JÄÄSKELÄINEN et al., 2019).

Segundo Stone, Gohara e Shi (2010), a programação com OpenCL permite que os desenvolvedores utilizem a localidade de acesso de memória concorrente para os dados, o que é especialmente útil em cenários onde os dados são acessados por vários itens de trabalho ao mesmo tempo. Ao aproveitar a memória local rápida presente em cada unidade de computação OpenCL, os dados podem ser preparados na memória local e, em seguida, efetivamente transmitidos para todos os itens de trabalho dentro do mesmo grupo de trabalho. Isso amplifica significativamente a largura de banda de memória efetiva disponível para o algoritmo, melhorando o desempenho.

Além disso, de acordo com Ahmed (2021), o OpenCL permite que os programadores mapeiem aplicativos para dispositivos específicos em um ambiente de *cluster* heterogêneo. A decisão sobre o mapeamento de trabalhos para uma variedade de dispositivos de computação heterogêneos pode ser muito difícil para um programador. A programação de tarefas de *kernel* OpenCL em um ambiente de *cluster* integrado permite um equilíbrio de carga eficaz de um lote de trabalhos, aproveitando efetivamente a heterogeneidade inerente aos dispositivos de computação heterogêneos.

O objetivo deste trabalho de conclusão de curso é investigar e analisar as potencialidades do OpenCL no contexto da computação paralela. Com a crescente complexidade das aplicações computacionais e a necessidade de processamento eficiente, é essencial explorar abordagens que permitam a execução simultânea de tarefas. Neste contexto, o OpenCL é uma solução viável, fornecendo um ambiente de programação aberto e portátil para diferentes arquiteturas de *hardware*.

Neste estudo será realizada uma análise crítica do OpenCL, explorando suas vantagens, desafios e potenciais aplicações. A pesquisa visa não apenas compreender o funcionamento dessa tecnologia, mas também oferecer perspectivas valiosas para profissionais, pesquisadores e estudantes que buscam otimizar o desempenho de suas aplicações por meio da computação paralela.

1.1. O PROBLEMA E SUA IMPORTÂNCIA

OpenCL é um *framework* de computação paralela que inclui uma linguagem de programação para *kernels* e uma API de gerenciamento que permite a execução de código em CPUs e GPUs. Ele foi projetado para tirar proveito do poder computacional de unidades de processamento heterogêneas, como CPUs e GPUs, permitindo que os desenvolvedores escrevam código que pode ser executado em qualquer dispositivo que suporte OpenCL (PEREIRA et al., 2016). Isso torna o OpenCL uma ferramenta considerável para otimização de desempenho e aceleração de algoritmos, pois permite explorar a paralelização em diferentes dispositivos de *hardware*.

A eficiência de um código OpenCL pode ser amplamente influenciada pela plataforma de *hardware* em que é executado. Por exemplo, um código que foi aprimorado para uma GPU de alta performance pode apresentar um desempenho notavelmente inferior quando rodado em uma CPU *multicore*, ou seja, quando executado em um processador que possui dois ou mais núcleos de processamento de tarefas em paralelo no interior de um único circuito integrado (MINHAS, WOODS, KARAKONSTANTIS, 2018). De acordo com Minhas, Woods e Karakonstantis (2018), isso pode ser atribuído a diferenças nos recursos de *hardware*, como a arquitetura do processador, a quantidade de memória disponível e a velocidade do processador. Tais variações podem impactar a eficácia do código, resultando em uma variação no desempenho do código em diferentes plataformas de *hardware*.

Ainda segundo Pereira (2016), apesar de OpenCL visar a portabilidade no nível do código, diferentes plataformas de *hardware* exigem abordagens distintas para extrair o melhor desempenho para o código baseado em OpenCL. Cada arquitetura possui características específicas que influenciam a forma como os recursos são utilizados, e, por isso, a otimização de código deve ser feita com base na plataforma de destino.

Por exemplo, Pereira (2016) em seu estudo mostrou que, embora a versão OpenCL-GPU seja a mais rápida quando executada em uma GPU *high-end*, ou seja, quando executada em uma GPU mais robusta, com alto desempenho, muitos núcleos, grande quantidade de memória (VRAM) e alta frequência de memória, o que garante maior largura de banda, seu desempenho cai em aproximadamente 65% quando executada sem alterações em uma máquina de CPU *multicore*. Nas

versões otimizadas para CPU, o código OpenCL-CPU levou 805 segundos para codificar a imagem Lenna. Já a abordagem que utiliza vetorização executou a mesma operação em 672 segundos. Esses resultados indicam que é possível obter ganhos de desempenho significativos ao adaptar o código OpenCL para a CPU. Além disso, o uso de instruções de vetorização por meio do OpenCL se mostrou tanto uma abordagem simples quanto eficiente, oferecendo melhorias de desempenho consideráveis.

1.2. OBJETIVOS

1.2.1. Objetivo Geral

Diagnosticar contextos favoráveis à aplicabilidade do OpenCL como ferramenta para implementar programação paralela.

1.2.2. Objetivos Específicos

a) Analisar a importância do OpenCL, destacando sua capacidade de tirar proveito do poder computacional de unidades de processamento heterogêneas, como CPUs e GPUs.
b) Avaliar, através de estudos na área, a eficiência do OpenCL em diferentes cenários de computação, comparando-o com outros métodos de programação paralela.
c) Fornecer recomendações para a utilização do OpenCL, com base na análise realizada. Isso envolverá a síntese das descobertas dos objetivos específicos e a apresentação de recomendações para a utilização do OpenCL em diferentes cenários de computação.

1.3. ESTRUTURA DO TRABALHO

O restante deste trabalho está organizado da seguinte forma. No Capítulo 2 são apresentadas diferentes abordagens sobre o OpenCL e seu uso, além de discorrer e comparar com outras tecnologias similares, como CUDA (*Compute Unified Device Architecture*), de forma a ser realizado comparativo entre ambos. O Capítulo 3 define a metodologia utilizada para realização deste trabalho. Os

resultados obtidos estão descritos no Capítulo 4 e, por fim, no Capítulo 5 estão as conclusões.

2. REVISÃO DA LITERATURA

Nos últimos anos, a computação paralela tornou-se crucial para enfrentar a crescente demanda por desempenho em sistemas computacionais. A capacidade de dividir tarefas em partes menores e processá-las simultaneamente por múltiplos núcleos, aceleradores ou processadores revolucionou várias áreas, como simulações científicas, aprendizado de máquina e processamento de dados em larga escala. Diversas tecnologias têm sido desenvolvidas para explorar esse potencial, com destaque para o OpenCL e CUDA, além de outras alternativas reconhecidas, como OpenACC, MPI, OpenMP, SYCL e LibWater.

Uma dessas alternativas é o OpenACC, uma API projetada para facilitar a programação de sistemas paralelos, especialmente no uso de GPUs. O OpenACC permite que programadores insiram diretivas no código em linguagens como C, C++ e Fortran, simplificando o processo de paralelização. A grande vantagem do OpenACC está na facilidade de uso, uma vez que o programador não precisa gerenciar diretamente a alocação de memória ou a execução em dispositivos. No entanto, sua portabilidade é limitada, sendo otimizado principalmente para GPUs, o que pode restringir sua aplicação em sistemas heterogêneos que utilizem múltiplos tipos de aceleradores (WIENKE et al., 2012).

Outra abordagem utilizada é o MPI (*Message Passing Interface*), ele é amplamente utilizado em sistemas distribuídos e de larga escala, permitindo a comunicação entre processos que rodam em diferentes nós de um *cluster*. Esta abordagem é ideal para aplicações em ambientes distribuídos e simulações científicas complexas. Gropp (1999) mostra como o MPI foi desenvolvido para ser portátil e escalável, tornando-se a base de muitos sistemas de computação paralela de grande porte. No entanto, o MPI é mais voltado para computação distribuída e não é tão flexível para lidar com ambientes heterogêneos que envolvem GPUs ou FPGAs, onde soluções como OpenCL podem ser mais adequadas.

OpenMP trata-se de outra alternativa, sendo um padrão adotado para a programação paralela em sistemas com memória compartilhada. O OpenMP permite que os programadores especifiquem facilmente as partes do código que devem ser paralelizadas por meio de diretivas, simplificando a criação de aplicações *multithreaded*. Segundo Dagum e Menon (1998), o OpenMP é uma ferramenta poderosa para sistemas baseados em CPUs *multicore*, mas suas limitações surgem

quando é necessário suportar GPUs ou dispositivos heterogêneos, áreas onde o OpenCL oferece maior flexibilidade.

SYCL é uma extensão do C++ que tem como objetivo facilitar a programação paralela em ambientes heterogêneos, proporcionando uma interface de alto nível. Alpay e Heuveline (2020) destacam o hipSYCL como uma implementação que vai além do OpenCL, permitindo maior facilidade de uso para programadores de C++ e integração com dispositivos variados, como GPUs e CPUs. SYCL, no entanto, ainda é uma tecnologia emergente em comparação ao OpenCL, que já possui maior maturidade e suporte em uma vasta gama de dispositivos e arquiteturas.

Por fim, LibWater é uma biblioteca que visa facilitar a programação paralela distribuída em sistemas heterogêneos. Grasso (2013) demonstrou como o LibWater permite que desenvolvedores criem sistemas paralelos dinâmicos com alta portabilidade, oferecendo uma interface que simplifica a execução de tarefas distribuídas em múltiplos dispositivos. Embora promissora, sua adoção ainda é limitada em comparação com o OpenCL, que já se consolidou como uma solução de uso amplo em diferentes plataformas.

A escolha do OpenCL para este trabalho foi motivada por sua flexibilidade e ampla adoção em cenários de computação heterogênea. O OpenCL oferece suporte para uma grande variedade de dispositivos, como CPUs, GPUs, FPGAs e outros aceleradores, permitindo que desenvolvedores criem código portátil e escalável. Ao contrário de alternativas como OpenMP e MPI, que são mais limitadas a ambientes específicos (memória compartilhada ou distribuída), o OpenCL se destaca por sua capacidade de operar em múltiplas arquiteturas de *hardware* com o mesmo código base. Além disso, tecnologias como CUDA, embora poderosas em dispositivos NVIDIA, restringem-se a um único fabricante, enquanto o OpenCL é suportado por uma vasta gama de fornecedores, tornando-se uma escolha mais versátil para este estudo (STONE, GOHARA, SHI, 2010).

2.1. OPENCL

De acordo com o Khronos Group (2024), o OpenCL oferece à indústria a camada de execução mais próxima do *hardware* para processadores flexíveis, permitindo acelerar aplicações, bibliotecas e motores, além de fornecer um alvo para a geração de código por compiladores. Ao contrário de APIs que são exclusivas para GPUs, como Vulkan, Ahmed (2021) cita que OpenCL é amplamente usado devido à

sua portabilidade e ao grande número de dispositivos de computação suportados, o que podemos confirmar de acordo com Khronos Group (2024), pois ele demonstra que o OpenCL permite o uso de uma variedade de aceleradores, incluindo CPUs *multicore*, GPUs, DSPs (*Digital Signal Processor*), FPGAs (*Field Programmable Gate Array*) e *hardware* dedicado, como motores de inferência. Podemos afirmar ainda, de acordo com Stone, Gohara e Shi (2010), que o OpenCL é uma API de programação paralela que fornece uma linguagem comum, interfaces de programação e abstrações de *hardware*, permitindo que os desenvolvedores acelerem aplicações com cálculos paralelos de tarefas ou dados em um ambiente de computação heterogêneo que consiste no CPU do host e em qualquer dispositivo OpenCL anexado.

Um programa em OpenCL é composto por *kernels*, que são executados em um ou mais dispositivos, e pelo host, responsável por gerenciar a execução desses *kernels*. Os *kernels* são processados por *work-items*, que, por sua vez, são organizados em *work-groups*. Esses *work-groups* estão dispostos dentro de um NDRange (DE PAULA, 2014). No OpenCL, um *kernel* é a função que define o código a ser executado em paralelo pelos *work-items*, que são as menores unidades de execução, responsáveis por processar dados de forma independente. Esses *work-items* são organizados em *work-groups*, permitindo comunicação entre si e compartilhamento de memória local. A distribuição dos *work-items* ocorre em um espaço de execução chamado NDRange, que pode ser unidimensional, bidimensional ou tridimensional, especificando como o *kernel* será paralelamente aplicado aos dados. A Figura 1 demonstra a forma como ocorre a organização dos *work-items* e *work-groups* em um NDRange.

De acordo com Jääskeläinen (2019), o OpenCL possui uma grande capacidade de explorar o paralelismo de tarefas, oferecendo diversas formas de expressar explicitamente o paralelismo em várias tarefas, através do uso de filas de comando que não seguem uma ordem específica e sincronização de eventos. A partir da versão 2.0 do padrão, é permitido extrair implicitamente o paralelismo de tarefas a partir de filas de comando que seguem uma ordem específica.

Entretanto, Pereira (2016) e Minhas, Woods e Karakonstantis (2018) demonstram em seus respectivos estudos, apesar da praticidade dessa tecnologia devido ao fato da sua facilidade de programabilidade e de visar alcançar a portabilidade no nível de código, ainda há dúvidas sobre a portabilidade de

desempenho, pois diferentes plataformas de *hardware* exigem abordagens diferentes para extrair o melhor desempenho para o código baseado em OpenCL. Ou seja, é necessário buscar uma otimização específica para cada fim ao qual se é utilizada, pois o que funciona bem em uma plataforma pode ser ineficiente em outra, devido a diferenças na arquitetura, na memória, na largura de banda, entre outros fatores.

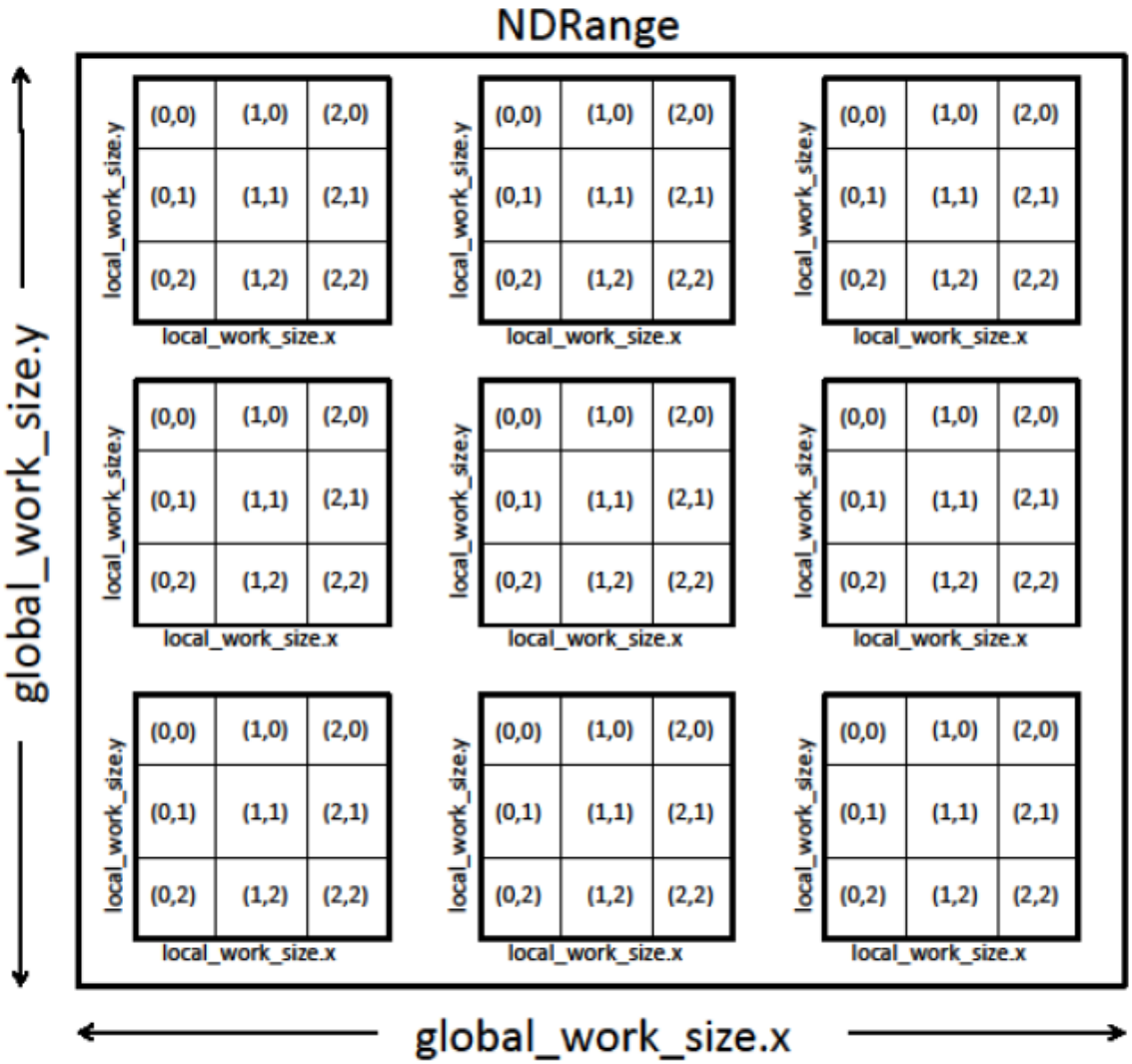

Figura 1 - Organização dos *work-groups* em um NDRange

Fonte: DE PAULA (2014).

2.2. PROGRAMAÇÃO PARALELA

A programação paralela é um modelo de computação em que muitos processos são executados simultaneamente. De acordo com Maciel Jr, Da Silva e Rodrigues (2013) o princípio central desta técnica é que problemas de grande escala

podem ser fragmentados em partes menores, que podem ser processadas simultaneamente. Portanto, para situações que requerem um alto poder de processamento, uma abordagem de computação paralela pode ser utilizada. O objetivo é acelerar o processamento e obter resultados de maneira mais rápida e eficiente (HENNESSY e PATTERSON, 2002).

A programação paralela designa tarefas computacionais para serem executadas simultaneamente em vários elementos de processamento (SCARPINO, 2011). No contexto do OpenCL, essas tarefas são conhecidas como *kernels*. Um *kernel* é uma função especificamente codificada que tem como objetivo ser executada por um ou mais dispositivos compatíveis com OpenCL.

A programação paralela pode ser realizada em uma variedade de dispositivos, incluindo CPUs, GPUs e outros aceleradores. Isso permite que os desenvolvedores aproveitem os benefícios de diferentes tipos de processadores, como a alta velocidade de processamento da GPU e a grande quantidade de memória da CPU (STONE, GOHARA, SHI, 2010).

Embora o padrão OpenCL ofereça uma plataforma de programação abrangente para programação paralela heterogênea portátil, a versão 1.2 do padrão é bastante de baixo nível, revelando muitos detalhes da plataforma para o programador. Assim, é possível usar essas consultas de plataforma para adaptar o programa para cada plataforma (JÄÄSKELÄINEN et al., 2015).

No OpenCL, a programação paralela é realizada através de uma API que permite aos programas de *software* realizar processamento paralelo utilizando vários processadores simultaneamente. Isso aumenta a eficiência do processamento pela distribuição da carga entre vários processadores (STONE, GOHARA, SHI, 2010).

2.3. COMPUTAÇÃO HETEROGÊNEA

O termo computação heterogênea é usado para indicar o uso de diferentes tipos de arquiteturas em um único nó computacional, visando melhorar o desempenho das aplicações que serão executadas nesse nó. Entre as arquiteturas mais destacadas estão os conhecidos como aceleradores, GPUs e FPGAs (STRINGHINI, GONÇALVES, GOLDMAN, 2012).

A computação heterogênea é um tipo de computação onde diferentes tipos de processadores são usados para realizar tarefas. Isso inclui CPUs, GPUs e outros aceleradores, que são dispositivos de *hardware* especializados projetados para

aumentar a velocidade de processamento de tarefas específicas. Kim e Choi (2013) explicam que para tirar proveito dos múltiplos núcleos, os programas poderiam ser escritos de forma que executam suas tarefas utilizando várias threads paralelas. Dessa forma é realizada então a distribuição de tarefas entre os recursos disponíveis, permitindo, por exemplo, que um código em execução na CPU possa delegar trabalho para um acelerador. A computação heterogênea permite que os desenvolvedores aproveitem os benefícios de diferentes tipos de processadores, como a alta velocidade de processamento da GPU e a grande quantidade de memória da CPU. Nesse tipo de computação, a heterogeneidade está no desenvolvimento de aplicações que podem ser executadas em dispositivos específicos da máquina (STONE, GOHARA, SHI, 2010; SANTOS, KOERICH, 2013).

A computação heterogênea envolve tanto o processamento sequencial quanto o paralelo. De acordo com Kaeli (2015), na computação heterogênea, as tarefas que compõem um aplicativo são direcionadas ao melhor dispositivo de processamento disponível no sistema. A existência de vários dispositivos em um sistema oferece a oportunidade para que os programas utilizem concorrência e paralelismo, melhorando assim o desempenho e a eficiência energética. O Open Computing Language (OpenCL) é um *framework* criado especificamente para dar suporte a ambientes de computação heterogênea, permitindo que programas sejam executados em diferentes dispositivos, como CPUs e GPUs.

O OpenCL define tanto uma linguagem para a programação de *kernels* no lado do dispositivo quanto uma camada de gerenciamento para o host que controla esses dispositivos. De acordo com Kaeli (2015), a linguagem do lado do dispositivo (usada para escrever *kernels*) é projetada para mapear eficientemente uma ampla gama de sistemas de memória e modelos de execução. O lado do *host*, por sua vez, é responsável por orquestrar a execução dos *kernels* e gerenciar a transferência de dados entre o host e o dispositivo, suportando a execução eficiente de programas concorrentes com baixo overhead. Juntos, esses elementos oferecem ao desenvolvedor uma maneira eficiente de mover-se do design do algoritmo para sua implementação.

Embora o OpenCL seja uma das principais tecnologias que facilitam a programação heterogênea, não é a única opção disponível. Outras plataformas, como o CUDA da NVIDIA, também permitem o desenvolvimento de aplicações que aproveitam a capacidade de processamento paralelo de diferentes dispositivos.

2.4. CUDA

Lançada pela NVIDIA em 2006, a API CUDA foi uma das primeiras a possibilitar o uso de GPUs para uma ampla gama de aplicações além de gráficos (NVIDIA CUDA, 2024), sendo compatível apenas com as placas gráficas da empresa. No modelo de programação CUDA, o processamento é dividido entre a CPU (*host*) e a GPU (*device*). Um programa em CUDA contém partes que são executadas no host e outras que rodam no *device*. Essa divisão é realizada automaticamente durante a compilação pelo compilador nvcc da NVIDIA. O código CUDA expande a linguagem C (CUDA-C), adicionando palavras-chave específicas para marcar funções paralelas (*kernels*) e suas respectivas estruturas de dados (NVIDIA CUDA, 2024).

O modelo de memória do CUDA é hierárquico, incluindo memória global, compartilhada, local, constante e de textura. A memória global, acessível a todas as *threads*, apresenta alta latência, enquanto a memória compartilhada tem menor latência e é acessível apenas às threads dentro de um bloco. Esse gerenciamento de memória é fundamental para otimizar o desempenho das aplicações paralelas (NVIDIA CUDA, 2024). As *threads* do CUDA têm a capacidade de acessar dados provenientes de vários espaços de memória durante sua execução. Cada *thread* conta com uma memória local privada. Dentro de cada bloco de threads, existe uma memória compartilhada que é acessível a todas as *threads* do bloco e que possui a mesma duração do bloco em si. Os blocos de *threads* em um *cluster* podem realizar operações de leitura e escrita na memória compartilhada entre si. Além disso, todas as *threads* têm acesso à memória global.

Adicionalmente, existem dois espaços de memória somente para leitura que podem ser acessados por todas as *threads*: os espaços de memória constante e de textura. Os espaços de memória global, constante e de textura são otimizados para diferentes tipos de uso de memória. A memória de textura oferece ainda modos de endereçamento variados e recursos de filtragem de dados para certos formatos específicos (NVIDIA CUDA, 2024). Os espaços de memória global, constante e de textura permanecem disponíveis entre os lançamentos de kernels pela mesma aplicação. Na Figura 2 podemos esse modelo de hierarquia de memória do CUDA.

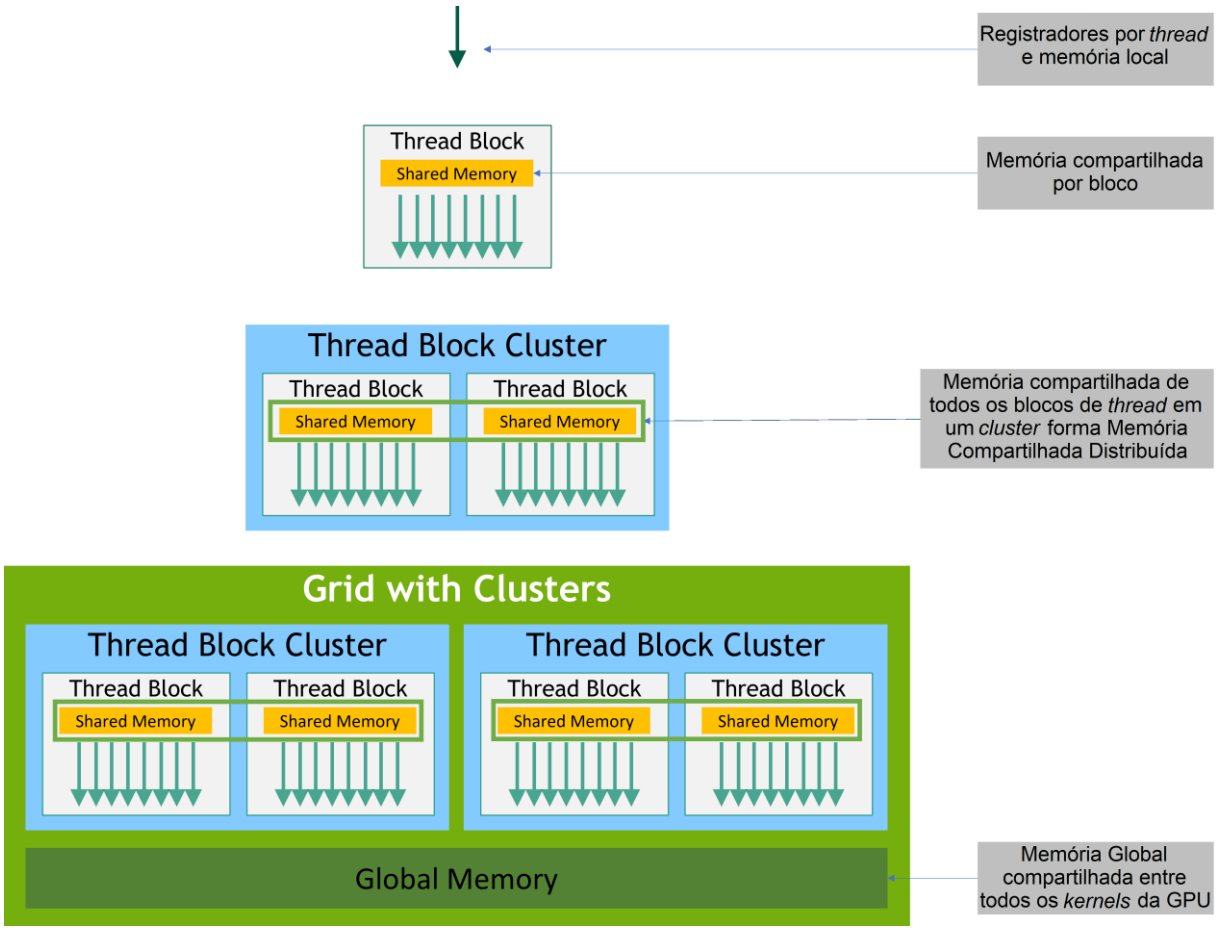

Figura 2 - Hierarquia de memória.

Fonte: NVIDIA CUDA (2024).

Estudos demonstram que o CUDA oferece ganhos significativos de desempenho em comparação com as CPUs em uma ampla gama de aplicações que exigem grande poder computacional, como simulações físicas, processamento de imagens e aprendizado profundo (CHE et al., 2008). No campo do aprendizado profundo, por exemplo, GPUs com suporte ao CUDA aceleram de maneira significativa o treinamento de Redes Neurais Convolucionais (CNNs), um processo que pode levar dias em CPUs (RAINA, MADHAVAN, NG, 2009).

Embora o CUDA proporcione um desempenho elevado em GPUs da NVIDIA, sua principal limitação é a dependência desse *hardware* específico, o que limita a portabilidade dos códigos para outras plataformas. Tecnologias como o OpenCL oferecem maior flexibilidade, permitindo a execução de programas em diversos tipos de dispositivos, como CPUs, GPUs e FPGAs. No entanto, em termos de desempenho em GPUs da NVIDIA, o CUDA se destaca em relação ao OpenCL, pois

é otimizado para a arquitetura específica desses dispositivos (KARIMI, DICKSON, HAMZE, 2010).

Outro desafio é a curva de aprendizado associada à programação paralela e ao gerenciamento de memória da GPU. No entanto, a NVIDIA disponibiliza diversas bibliotecas otimizadas, como cuBLAS (para operações de álgebra linear) e cuFFT (para transformadas rápidas de Fourier), que facilitam a implementação de aplicações de alto desempenho (NVIDIA CUBLAS, 2024; NVIDIA CUFFT, 2024).

O CUDA se estabeleceu como uma das principais plataformas para computação paralela em GPUs, proporcionando uma interface robusta e acessível para pesquisadores e desenvolvedores que buscam explorar o poder de processamento das GPUs da NVIDIA. Suas aplicações vão desde simulações físicas e processamento de imagens até aprendizado profundo. Contudo, sua dependência de *hardware* específico e a exigência de um conhecimento técnico avançado sobre a arquitetura da GPU são desafios a serem levados em conta (MITTAL, VETTER, 2014).

3. METODOLOGIA

Realizou-se uma pesquisa no repositório Google Acadêmico[1] por trabalhos relacionados ao OpenCL, Programação Paralela, Computação Heterogênea e OpenCL vs CUDA, utilizando o filtro de busca pela relevância e as palavras-chave em português: “OpenCL”, “Programação Paralela”, “Computação Heterogênea” e “OpenCL vs CUDA”, e em inglês: “*OpenCL*”, “*Parallel Programming*”, “*Heterogeneous computing*”, “*OpenCL vs CUDA”* e *“languages for heterogeneous computing”*.

Para cada palavra-chave consultada, realizou-se a leitura do título de todos os artigos e livros listados, reservando os que tinham relação com o tema deste trabalho. Em seguida foi realizada a leitura do resumo de cada artigo separado e mantido apenas os que continuavam em relação com esta pesquisa. As consultas resultaram em mais de 1000 documentos, mas o Google Acadêmico tem a limitação de permitir apenas o acesso a 1000 artigos por busca, logo, não foi possível analisar todos os documentos das consultas que ultrapassaram esse limite.

Nas buscas sobre a Programação Paralela e Computação Heterogênea, encontrou-se estudos relacionados com o uso de diversas ferramentas e tecnologias. Logo, para as palavras-chave “Programação Paralela”, “Computação Heterogênea”, “*Parallel Programming*” e “*Heterogeneous Computing*” foram considerados apenas os trabalhos relacionados ao OpenCL.

Nas buscas sobre OpenCL vs CUDA, foram considerados trabalhos que abrangessem aplicações mais comuns do uso de ambas tecnologias, de forma a fazer um comparativo mais eficiente.

Nas buscas sobre outras tecnologias para computação heterogênea, buscou-se pelo termo “*languages for heterogeneous computing”* de forma a reduzir a quantia de resultados encontrados, para direcionar melhor a busca.

Assim como é mostrado na Tabela 1, foram encontrados cinquenta e nove mil e seiscentos resultados para a palavra-chave “OpenCL” e selecionados sete trabalhos, e também foram encontrados trinta e um mil resultados para a palavra-chave “OpenCL vs CUDA” e selecionados oito trabalhos, porém, como será demonstrado na Tabela 2, pelo fato de tratarem-se das mesmas palavras em inglês,

1 https://scholar.google.com.br

os resultados encontrados e número de trabalhos selecionados mantiveram-se os mesmos, pois foram selecionados os mesmos trabalhos.

As Tabelas 1 e 2 apresentam o resultado das buscas realizadas no Google Acadêmico.

Tabela 1 - Resultados da busca na plataforma Google Acadêmico com as palavras-chave em português.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
OpenCL	59600	7
Programação Paralela	51100	1
Computação Heterogênea	23200	2
OpenCL vs CUDA	31000	8
Total	164900	18

Fonte: o autor.

Tabela 2 - Resultados da busca na plataforma Google Acadêmico com as palavras-chave em inglês.

Palavra-chave	**Resultados**	**Número de trabalhos selecionados**
OpenCL	59600	7
Parallel Programming	3740000	4
Heterogeneous Computing	4900000	2
OpenCL vs CUDA	31000	8
Languages for Heterogeneous Computing	888000	5
Total	9618600	26

Fonte: o autor.

A maior quantidade de resultados encontra-se na Tabela 2, com nove milhões seiscentos e dezoito mil e seiscentos documentos. É mostrado que quatro trabalhos foram selecionados a partir da palavra-chave "*Parallel Programming*". A terceira palavra-chave "*Heterogeneous Computing"* resultou em dois trabalhos. A quinta palavra-chave "*Languages for Heterogeneous Computing"* resultou em cinco trabalhos. No total, vinte e nove trabalhos foram escolhidos e serão discorridos a seguir.

Este estudo se fundamenta em uma análise crítica da literatura, com o objetivo de abordar o OpenCL, seu uso e suas capacidades, além de compará-la com sua principal concorrente, o CUDA. A intenção é destacar os principais aspectos de cada tecnologia e, a partir disso, avaliar se o OpenCL ainda é uma opção viável para utilização.

3.1. COLETA DE DADOS

Após a agregação da literatura e remoção de duplicatas foram selecionados sessenta e cinco itens únicos, depois foi realizada uma filtragem a partir da leitura do título, resumo e das seções de conclusão, retirando assim vinte e cinco artigos que não se encaixavam com a proposta deste estudo. Continuando o processo de filtragem, foi feito uma análise das seções de modelação do problema proposto e a sua solução, removendo consequentemente mais onze artigos que não seriam de grande uso para os objetivos desse trabalho. Logo chegamos ao número final de vinte e nove itens únicos analisados, entre artigos e livros, para a realização desse estudo.

A Figura 3 apresenta o processo de recolha e filtragem da literatura.

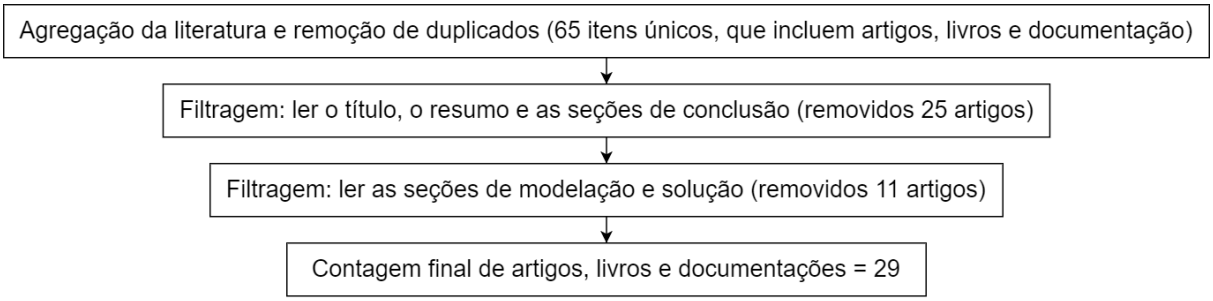

Figura 3 - O processo de recolha e filtragem da literatura.

Fonte: o autor.

3.2. ANÁLISE DOS DADOS

Após a seleção, foram efetuados estudos minuciosos nos artigos, livros e documentações as quais foram selecionadas para o uso no trabalho em questão. Foi encontrado então a possibilidade de realizar um estudo sobre a computação heterogênea, programação paralela, OpenCL, CUDA e como a computação heterogênea e a programação paralela são utilizados com o OpenCL, e em quais situações se torna vantajoso o uso do OpenCL, principalmente quando comparado com o CUDA. Encontrando assim uma oportunidade de importância do OpenCL,

destacando sua capacidade de tirar proveito do poder computacional de unidades de processamento heterogêneas, como CPUs e GPUs, sendo assim capaz de fornecer recomendações sobre quando e como utilizar o OpenCL, destacando as situações em que seu uso é mais vantajoso.

A partir da análise descrita acima, foi possível realizar de forma eficaz um estudo sobre computação heterogênea, programação paralela e uma comparação entre OpenCL e CUDA. Podendo demonstrar quais as principais desvantagens e vantagens de cada uma dessas tecnologias e chegar a uma conclusão sobre essas tecnologias.

4. RESULTADOS

Este capítulo apresenta os resultados da análise da literatura sobre o OpenCL e sua comparação com o CUDA. A análise é baseada em vinte e nove trabalhos selecionados que abordam aspectos técnicos, desempenho e usabilidade de ambas as tecnologias. A seguir, são discutidos os principais resultados encontrados.

4.1. ANÁLISE DO OPENCL

A análise da literatura revelou que o OpenCL é uma plataforma de programação paralela desenvolvida para computação heterogênea, permitindo a execução de programas em diversos tipos de dispositivos, como CPUs, GPUs, DSPs e FPGAs (KHRONOS GROUP, 2024). De acordo com Stone, Gohara e Shi (2010), o OpenCL proporciona uma abstração comum para programar diferentes tipos de *hardware*, facilitando o desenvolvimento de aplicações paralelas.

A revisão de Jääskeläinen (2019) mostrou que o OpenCL permite a exploração de paralelismo de tarefas, utilizando filas de comandos e sincronização de eventos para gerenciar a execução dos *kernels*. A versão 2.0 do padrão introduziu a capacidade de extrair paralelismo implicitamente, melhorando a eficiência da programação (JÄÄSKELÄINEN et al., 2019).

No entanto, Pereira (2016) e Minhas, Woods e Karakonstantis (2018) destacam que a portabilidade de desempenho ainda é uma preocupação, já que diferentes plataformas de *hardware* podem exigir otimizações específicas para alcançar o máximo desempenho. Essa limitação sugere que, embora o OpenCL ofereça flexibilidade, a eficiência pode variar dependendo do dispositivo utilizado.

4.2. COMPARAÇÃO ENTRE OPENCL E CUDA

CUDA e OpenCL são tecnologias utilizadas para programação em GPUs, permitindo a realização de cálculos paralelos em tarefas de propósito geral. Enquanto CUDA, desenvolvida pela NVIDIA, é uma tecnologia proprietária que só pode ser utilizada com suas próprias placas gráficas, OpenCL é um padrão aberto que é compatível com *hardware* de diferentes fabricantes (DE PAULA, 2014).

A análise comparativa entre OpenCL e CUDA revelou diversas diferenças significativas entre as duas tecnologias. CUDA, desenvolvida pela NVIDIA, é uma plataforma amplamente utilizada para programação paralela em GPUs, oferecendo

um desempenho superior em *hardware* da NVIDIA devido à sua otimização específica (NVIDIA CUDA, 2024). Estudos como o de Karimi, Dickson e Hamze (2010) mostram que o CUDA apresenta vantagens de desempenho em comparação com o OpenCL em GPUs da NVIDIA, principalmente em tarefas que exigem grande poder computacional.

Vários estudos acadêmicos têm comparado o desempenho entre CUDA e OpenCL. Karimi, Dickson e Hamze (2010) realizaram uma análise comparativa entre essas duas tecnologias, revelando que, apesar da portabilidade oferecida pelo OpenCL, que permite a execução em diversos tipos de GPUs, essa flexibilidade pode acarretar em um desempenho reduzido. De forma mais abrangente, Fang, Varbanescu e Sips (2011) conduziram uma comparação detalhada e encontraram que, para a maioria dos casos analisados, o desempenho do CUDA foi, no máximo, 30% superior ao do OpenCL.

Por outro lado, OpenCL se destaca pela sua portabilidade e capacidade de executar código em uma ampla gama de dispositivos, não se limitando a *hardware* específico (KHRONOS GROUP, 2024). Isso é corroborado por Ahmed (2021), que destaca a flexibilidade do OpenCL em suportar diversos tipos de aceleradores.

O estudo de Fang, Varbanescu e Sips (2011) oferece uma visão detalhada das diferenças de desempenho entre OpenCL e CUDA, indicando que, apesar da vantagem do CUDA em *hardware* NVIDIA, o OpenCL pode ser mais vantajoso em ambientes heterogêneos onde a portabilidade e a flexibilidade são essenciais. Além disso, a análise dos resultados de Raina, Madhavan e Ng (2009) e Mittal e Vetter (2014) sugere que, enquanto o CUDA é otimizado para GPUs da NVIDIA, o OpenCL continua sendo uma escolha válida para sistemas heterogêneos e para desenvolvedores que necessitam de portabilidade entre diferentes plataformas.

4.3. IMPLICAÇÕES PARA A PRÁTICA

Os resultados sugerem que a escolha entre OpenCL e CUDA deve ser baseada nas necessidades específicas do projeto. Para aplicações que visam maximizar o desempenho em GPUs da NVIDIA, o CUDA pode ser a melhor opção devido às suas otimizações e suporte avançado (NVIDIA CUDA, 2024). Em contraste, para ambientes onde a portabilidade e a compatibilidade com uma variedade de dispositivos são prioritárias, o OpenCL oferece uma solução mais flexível (KHRONOS GROUP, 2024).

4.4. TECNOLOGIA dOCAL COMO ALIADO PARA OPENCL

A revisão de Noack (2017) revela que o OpenCL tem se mostrado uma ferramenta poderosa, mas com desafios significativos em termos de desempenho e portabilidade, principalmente em contextos científicos de alta performance. Noack (2017) destaca tanto os pontos positivos quanto os negativos da tecnologia, sugerindo que o OpenCL pode não ser a melhor solução para todos os cenários, especialmente onde o desempenho é crítico.

Nesse contexto, a tecnologia dOCAL (*Distributed OpenCL/CUDA Abstraction Layer*), descrita por Rasch (2020), emerge como um importante aliado para o OpenCL. O dOCAL é uma biblioteca de abstração de alto nível implementada em C++ para facilitar o desenvolvimento de código de host em sistemas distribuídos heterogêneos que utilizam dispositivos OpenCL (como CPUs de múltiplos núcleos e GPUs) e CUDA (para GPUs NVIDIA), oferecendo uma solução que pode mitigar algumas das limitações do OpenCL ao permitir um desenvolvimento mais eficiente e otimizado para ambientes distribuídos.

O dOCAL facilita a programação paralela e distribuída, permitindo que desenvolvedores aproveitem o melhor dos dois mundos: a flexibilidade do OpenCL e a eficiência do CUDA. Com dOCAL, é possível escrever código que pode ser executado em diferentes tipos de *hardware* de forma mais coesa, melhorando o desempenho e a portabilidade das aplicações.

Portanto, a integração de dOCAL com OpenCL representa um avanço significativo, proporcionando uma maneira eficaz de superar as limitações inerentes do OpenCL e explorar seu potencial completo em ambientes heterogêneos. Esta combinação pode ser especialmente valiosa para aplicações científicas e de alta performance que necessitam de uma abordagem flexível e poderosa para a programação paralela.

5. CONCLUSÕES

Este estudo investigou o OpenCL e sua comparação com CUDA, com base em uma análise crítica de vinte e nove trabalhos relevantes. A revisão da literatura revelou que OpenCL é uma plataforma robusta para computação heterogênea, oferecendo flexibilidade ao suportar uma ampla gama de dispositivos, incluindo CPUs, GPUs, DSPs e FPGAs. No entanto, a portabilidade de desempenho do OpenCL ainda apresenta desafios, dada a grande diversidade de arquiteturas disponíveis, especialmente quando comparado ao CUDA, que se destaca por suas otimizações específicas para *hardware* da NVIDIA.

A análise comparativa entre OpenCL e CUDA indicou que, enquanto o CUDA proporciona um desempenho superior em GPUs NVIDIA devido a otimizações específicas, o OpenCL se destaca pela sua portabilidade e flexibilidade em ambientes heterogêneos. Estudos como os de Karimi, Dickson e Hamze (2010) e Fang, Varbanescu e Sips (2011) confirmaram que, embora o CUDA seja mais eficiente em *hardware* NVIDIA, o OpenCL é uma escolha válida para aplicações que necessitam de suporte a múltiplas plataformas.

5.1. IMPLICAÇÕES PRÁTICAS

Os resultados sugerem que a escolha entre OpenCL e CUDA deve ser orientada pelas necessidades específicas do projeto. O CUDA é recomendado para aplicações que buscam otimizar o desempenho em GPUs NVIDIA, enquanto o OpenCL é preferível para cenários que requerem portabilidade e compatibilidade com uma diversidade de dispositivos. A flexibilidade do OpenCL é um ponto forte em ambientes heterogêneos, mas a necessidade de otimização específica para diferentes *hardwares* pode limitar seu desempenho.

A análise também destacou a importância da tecnologia dOCAL, que combina OpenCL e CUDA para fornecer uma solução poderosa em contextos distribuídos. O dOCAL permite superar as limitações do OpenCL ao oferecer uma interface de programação de alto nível que facilita a integração de diferentes tipos de *hardware*. Essa abordagem é especialmente relevante para aplicações científicas e de alta performance, onde a flexibilidade e a eficiência são cruciais.

5.2. CONSIDERAÇÕES FINAIS E RECOMENDAÇÕES

Em conclusão, tanto OpenCL quanto CUDA têm seus pontos fortes e fracos. O OpenCL é valioso para a portabilidade e compatibilidade entre diferentes plataformas, enquanto o CUDA é superior em termos de desempenho para GPUs NVIDIA. A integração do dOCAL com OpenCL oferece uma solução inovadora que pode ajudar a maximizar o desempenho em ambientes heterogêneos, tornando o OpenCL uma escolha mais viável para projetos que necessitam de flexibilidade e eficiência.

Para futuras pesquisas, recomenda-se a realização de estudos adicionais que explorem a combinação de OpenCL com outras tecnologias emergentes e a avaliação de novas abordagens para melhorar a portabilidade e o desempenho. A continuidade da evolução do OpenCL e o desenvolvimento de ferramentas como o dOCAL serão cruciais para atender às demandas crescentes da computação paralela e distribuída.

REFERÊNCIAS

AHMED, Usman et al. A load balance multi-scheduling model for OpenCL kernel tasks in an integrated cluster. **Soft Computing**, v. 25, n. 1, p. 407-420, 2021.

ALPAY, Aksel; HEUVELINE, Vincent. SYCL beyond OpenCL: The architecture, current state and future direction of hipSYCL. In: **Proceedings of the International Workshop on OpenCL**. 2020. p. 1-1.

DAGUM, Leonardo; MENON, Ramesh. OpenMP: an industry standard API for shared-memory programming. **IEEE computational science and engineering**, v. 5, n. 1, p. 46-55, 1998.

DE PAULA, Lauro Cássio Martins. CUDA vs. OpenCL: uma comparação teórica e tecnológica. **ForScience**, v. 2, n. 1, p. 31-46, 2014.

FANG, Jianbin; VARBANESCU, Ana Lucia; SIPS, Henk. A comprehensive performance comparison of CUDA and OpenCL. In: **2011 International Conference on Parallel Processing**. IEEE, 2011. p. 216-225.

GRASSO, Ivan et al. LibWater: heterogeneous distributed computing made easy. In: **Proceedings of the 27th international ACM conference on International conference on supercomputing**. 2013. p. 161-172.

GROPP, William. **Using MPI: Portable Parallel Programming with the Message Passing Interface**. MIT Press, 1999.

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

IBM. O que é HPC? Disponível em: <https://www.ibm.com/br-pt/topics/hpc>. Acesso em: 14 mar. 2024.

JÄÄSKELÄINEN, Pekka et al. Exploiting task parallelism with OpenCL: a case study. **Journal of Signal Processing Systems**, v. 91, p. 33-46, 2019.

JÄÄSKELÄINEN, Pekka et al. pocl: A performance-portable OpenCL implementation. **International Journal of Parallel Programming**, v. 43, p. 752-785, 2015.

KAELI, David R. et al. **Heterogeneous computing with OpenCL 2.0**. Morgan Kaufmann, 2015.

KARIMI, Kamran; DICKSON, Neil G.; HAMZE, Firas. A performance comparison of CUDA and OpenCL. **arXiv preprint arXiv:1005.2581**, 2010.

KHRONOS GROUP. OpenCL. Disponível em: <https://www.khronos.org/opencl/>. Acesso em: 4 mai. 2024.

KIM, Cheong Ghil; CHOI, Yong Soo. A high performance parallel DCT with OpenCL on heterogeneous computing environment. **Multimedia Tools and Applications**, v. 64, p. 475-489, 2013.

MACIEL JR, Paulo Ditarso; DA SILVA, Thiago Gouveia; RODRIGUES, Thomas Ribeiro. Utilizando a computação paralela para alavancar a pesquisa científica no IFPB. **Revista Principia-Divulgação Científica e Tecnológica do IFPB**, n. 23, p. 112-118, 2013.

MINHAS, Umar Ibrahim; WOODS, Roger; KARAKONSTANTIS, Georgios. Exploring functional acceleration of OpenCL on FPGAs and GPUs through platform-independent optimizations. In: **International symposium on applied reconfigurable computing**. Cham: Springer International Publishing, 2018. p. 551-563.

MITTAL, Sparsh; VETTER, Jeffrey S. A survey of methods for analyzing and improving GPU energy efficiency. **ACM Computing Surveys (CSUR)**, v. 47, n. 2, p. 1-23, 2014.

NOACK, Matthias. OpenCL in Scientific High Performance Computing: The Good, the Bad, and the Ugly. In: **Proceedings of the 5th International Workshop on OpenCL**. 2017. p. 1-3.

NVIDIA. cuBLAS Library Documentation. NVIDIA Developer, 2024. Disponível em: <https://docs.nvidia.com/cuda/cublas/index.html>. Acesso em: 20 jul. 2024.

NVIDIA. CUDA Programming Guide. Versão 12.5. NVIDIA Developer, 2024. Disponível em: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/#driver-api>. Acesso em: 20 jul. 2024.

NVIDIA. cuFFT Library Documentation. NVIDIA Developer, 2024. Disponível em: <https://docs.nvidia.com/cuda/cufft/index.html>. Acesso em: 20 jul. 2024.

PEREIRA, Pedro MM et al. Optimizing GPU Code for CPU Execution Using OpenCL and Vectorization: A Case Study on Image Coding. In: **Algorithms and Architectures for Parallel Processing: 16th International Conference, ICA3PP 2016, Granada, Spain, December 14-16, 2016, Proceedings 15**. Springer International Publishing, 2016. p. 537-545.

RAINA, Rajat; MADHAVAN, Anand; NG, Andrew Y. Large-scale deep unsupervised learning using graphics processors. In: **Proceedings of the 26th annual international conference on machine learning**. 2009. p. 873-880.

RASCH, Ari et al. dOCAL: high-level distributed programming with OpenCL and CUDA. **The Journal of Supercomputing**, v. 76, p. 5117-5138, 2020.

SANTOS, Fernando Araújo dos; KOERICH, Kaio Cesar. Computação heterogênea: roteiro de desenvolvimento de uma aplicação utilizando unidade de processamento gráfico (GPU). **Ciência da Computação-Pedra Branca**, 2013.

SCARPINO, Matthew. **OpenCL in action: how to accelerate graphics and computations**. Simon and Schuster, 2011.

STONE, John E.; GOHARA, David; SHI, Guochun. OpenCL: A parallel programming standard for heterogeneous computing systems. **Computing in science & engineering**, v. 12, n. 3, p. 66, 2010.

STRINGHINI, Denise; GONÇALVES, Rogério A.; GOLDMAN, Alfredo. Introdução à computação heterogênea. In: **Anais da XXXI Jornada de Atualização em Informática do XXXII Congresso da Sociedade Brasileira de Computação [Internet]**. 2012. p. 16-19.

WIENKE, Sandra et al. OpenACC—first experiences with real-world applications. In: **Euro-Par 2012 Parallel Processing: 18th International Conference, Euro-Par 2012, Rhodes Island, Greece, August 27-31, 2012. Proceedings 18**. Springer Berlin Heidelberg, 2012. p. 859-870.