UNIVERSIDADE DE TAUBATÉ

Shie Chen Fang

RESOLUÇÃO COMPUTACIONAL DE IMAGEM DIGITAL USANDO GRAPHICS PROCESSING UNIT - GPU

Taubaté – SP

2021

Shie Chen Fang

RESOLUÇÃO COMPUTACIONAL DE IMAGEM DIGITAL USANDO GRAPHICS PROCESSING UNIT - GPU

Dissertação apresentada para obtenção do Título de Mestre pelo curso de Mestrado em Engenharia Mecânica (Profissional) do Departamento de Engenharia da Universidade de Taubaté.
Área de Concentração: Gestão da Produção

Orientadora: Profa. Dra. Miroslava Hamzagic
Co-orientador: Prof. Me. Antonio Ricardo Mendrot

Taubaté – SP

2021

Shie Chen Fang

RESOLUÇÃO COMPUTACIONAL DE IMAGEM DIGITAL USANDO GRAPHICS PROCESSING UNIT - GPU

Dissertação apresentada para obtenção do Título de Mestre pelo curso de Mestrado em Engenharia Mecânica (Profissional) do Departamento de Engenharia da Universidade de Taubaté.
Área de Concentração: Gestão da Produção

Orientadora: Profa. Dra. Miroslava Hamzagic
Co-orientador: Prof. Me. Antonio Ricardo Mendrot

Data: _____

Resultado: _____

BANCA EXAMINADORA

Profa. Dra. Miroslava Hamzagic Universidade de Taubaté

Assinatura: _____

Profa. Dra. Valesca Alves Corrêa Universidade de Taubaté

Assinatura: _____

Profa. Dra. Divani Barbosa Gavinier Faculdade de Tecnologia – FATEC Taubaté

Assinatura: _____

Grupo Especial de Tratamento da Informação - GETI
Sistema Integrado de Bibliotecas – SIBi
Universidade de Taubaté – Unitau

F211r Fang, Shie Chen
Resolução computacional de imagem digital usando Graphics Processing Unit - GPU / Shie Chen Fang. -- 2021.
83 f. : il.

Dissertação (mestrado) – Universidade de Taubaté, Pró-reitoria de Pesquisa e Pós-graduação, Taubaté, 2021.
Orientação: Profa. Dra. Miroslava Hamzagic, Departamento de Engenharia Mecânica.
Coorientação: Prof. Me. Antonio Ricardo Mendrot, Departamento de Engenharia Mecânica.

1. Fluxo ótico. 2. Matriz de ordem elevada. 3. GPU. I. Universidade de Taubaté. Departamento de Engenharia Mecânica. Mestrado em Engenharia Mecânica. II. Titulo.

CDD – 621.367

Ficha catalográfica elaborada **por Shirlei Righeti – CRB-8/6995**

RESUMO

A representação visual foi sempre um recurso adotado para documentar resultados de experimentos científicos, e era necessário a interpretação humana e subjetiva da representação visual para a extração de alguma informação adicional. Com a evolução das tecnologias de hadware e software, o desenvolvimento de novos equipamentos de multimídias tem possibilitado a automatização de tratamento de imagens, sequencias de imagens e até o manuseio de imagens em 3D. As técnicas de processamento de imagens são aplicadas em diversas áreas da ciência, como por exemplo para processsamento de imagens médicas. Existem alguns modelos para processamento de imagens, por exemplo o modelo do fluxo ótico, que é uma técnica que descreve o movimento ocorrido entre dois quadros consecutivos de uma sequência de imagens usando padrões de luminosidade, e poderia ser usado para a análise do movimento cardíaco. A identificação de irregularidade no movimento cardíaco pode ser interpretada como deficiência circulatória das artérias coronarianas. O objetivo desse trabalho é a resolução computacional do sistema linear envolvido na imagem digital do método do fluxo ótico, e é uma tarefa computacionalmente custosa, porque para cada ponto da imagem 3D a ser analisada, são geradas três equações que compõem o sistema linear. Para representar as equações é usado uma matriz esparsa de ordem elevada, e essa ordem será mais elevada quanto melhor o grau de resolução da imagem. A evolução tecnológica dos equipamentos de captura de imagens, tem melhorado significativamente o grau de resolução da imagem, e consequentemente aumentado o tempo de processamento em equipamento computacional tradicional do tipo CPU. Com o surgimento da arquitetura computacional GPU é possível propor a abordagem paralela para a resolução das matrizes esparsas de ordem elevada.

Palavras-chave: Fluxo ótico. Matriz de Ordem Elevada. GPU.

ABSTRACT

The visual representation is always a resource adopted to document results of scientific experiments, requiring human and subjective interpretation of the representation of visual representation to extract some additional information. With the evolution of hardware and software technologies, the development of new multimedia equipment has allowed the automation of image processing, image sequences and even the handling of 3D images. Image processing techniques are applied in several areas of science, such as medical image processing. There are some models of image processing, for example the optical flow model, which is a technique that requires the movement between two consecutive frames of a sequence of images using the patterns of luminosity, and can be used for the analysis of cardiac movement. The identification of irregularities in the cardiac movement can be interpreted as circulatory deficiency of the coronary arteries. The objective of this work is the computational resolution of the linear system in the digital image of the optical flow method, and a computationally costly task, since for each point of the 3D image to be analyzed, are generate three equations that make up the linear system. The equations are are represent a sparse high-order matrix, and this order will be greater the better the degree of resolution of the image. The technological evolution of the equipment for capturing images, has improved the degree of image resolution, and consequently increased the processing time in traditional computational equipment of the CPU type. With the emergence of the GPU's computational architecture, it is possible to propose a parallel approach to the resolution of high-order arrays.

Keywords: Optical Flow, High Order Matrix, GPU

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS

ABM	Agent-Based Model
AC	Acumulador
ALU	ArithmeticLogic Unit
AmI	Ambient intelligence
ANN	Artificial Neural Network
AVC	Acidente Vascular Cerebral
B2C	Business-to-consumer
BPS	Bayesian Program Synthesis
CPU	Cental Processing Unit
CT	Computerized Tomography
CUDA	Computer Unified Device Architecture
CV	Visão computacional
DAI	Distributed Artificial Intelligence
DRAM	Dynamic Random Access Memory
E / S	Entrada / Saída
EA	Evolutionary Algorithm
FLOPS	Floating point operations per second
Gbps	GigaBits or GigaBytes per SECond
GDDR5X	SGRAM Double Data Rate Type 5X
Gigaflops	Onebillion (1,000,000,000) of FLOPS
GPGPU	General-Purpose GPU
GPU	Graphics Processing Unit
I / O	Input / Output
IA	Inteligencia Artificial
ILP	Inductive logic programming
IR	Instruction Register
MAS	Multi-agent system
MIMD	Multiple Instruction Multiple Data
MISD	Multiple Instruction Single Data
ML	Machine learning
MRI	Magnetic Resonance Imaging

NN	Neural network
OpenCV	Open Source Computer Vision Library
PC	Program Counter
PC	Personal Computer
PCIe	Peripheral Component Interconnect express
PET	Positron Emission Tomography
PNL	Programação neurolinguística
RAM	Random Access Memory
RDBMS	Relacional Database Management System
RPA	Robotic Process Automation
SGRAM	Synchronous Graphic Random Access Memory
SIMD	Multiple Instruction Multiple Data
SIMT	Single instruction, multiple threads
SISD	Single Instruction Single Data
SM	Streaming multiprocessadores
SMM	Maxwell SM
SMX	Streaming multiprocessadores extremely
SP	Stack Pointer
SPECT	Single Photon Emission Computed Tomography
TC	Tomografia computadorizada
Teraflop	1000 gigaflops
TI	Tecnologia da Informação
UC	Unidade de Controle
US	Ultrassom
VE	Ventrículo Esquerdo

Sumário

1 INTRODUÇÃO

Ao observar-se uma situação relevante e complexa como o de uma doença crônica em um órgão vital, para qualquer organismo principalmente o humano, cientistas são desafiados a gerar e conceber o melhor uso de tecnologias.

Em se tratando, por exemplo, de doenças do aparelho circulatório, verifica-se que foi a causadora de maior número de óbitos no Brasil em 2018 (ALVES, 2020). Como a maioria dos diagnósticos neste tipo de patologia, são realizados por meio de exame de imagens, a precisão e veracidade da conclusão obtida resulta em menores impactos na saúde pública de qualquer país. Devido à complexidade de todos estes casos, a visualização e a quantificação do movimento cardíaco em imagens médicas de diversas modalidades, é uma ferramenta importante para o diagnóstico e acompanhamento, dando indicação do progresso da doença e/ou terapia (TAI et al., 2009).

A análise do movimento cardíaco é uma tarefa complexa, realizada por especialistas altamente qualificados e que está relacionada principalmente à interpretação de movimentação patológica, causada por deficiências na circulação das artérias coronarianas, anormalidades no músculo cardíaco ou distúrbios de condução elétrica intraventricular (PUENTES et al., 2000, MURAMATSU et al., 2005). Sistemas de processamento e análise dessas imagens devem atuar como auxiliares efetivos aos clínicos para avaliar objetivamente as variações em relação aos padrões normais.

A interpretação ou quantificação de movimento vai além das técnicas usuais empregadas em análise de imagens bidimensionais (2D) e tridimensionais (3D), que incluem tratamento de ruído, segmentação e identificação de estruturas. É necessário, também, quantificar o movimento e criar representações de dados, que permitam a sua avaliação quantitativa, através da criação de índices para classificação de padrões de normalidade e patológicos, e garantir a visualização eficiente das informações pelos clínicos.

O conhecimento da dinâmica e estrutura do movimento cardíaco é muito importante para o estabelecimento de diagnósticos eficientes e tratamentos efetivos de distúrbios cardíacos (LEE et al., 2006). O desenvolvimento de

técnicas de aquisição de imagens médicas passou a ser o ferramental básico que possibilita a estimativa de movimento cardíaco e os parâmetros relevantes para diagnóstico. A pesquisa em sistemas baseados em imagens, que auxiliem os clínicos na determinação precisa de anomalias, representa um passo importante para a prevenção e tratamento, além de eventual diminuição de custos em todo o processo. O estudo e quantificação do movimento do músculo cardíaco por métodos automáticos podem permitir a identificação e caracterização de diversas dessas patologias, auxiliando na decisão da melhor conduta a ser seguida.

Em se tratando de um trabalho com imagens, onde qualquer decisão a ser tomada tem que ser embasada em dados extremamente claros, precisos e rápidos, a flexibilização de tecnologias avançadas só pode ser obtida por pesquisadores que entendam não só os requisitos técnicos, mas estejam envolvidos na sua real aplicabilidade.

1.1 Abordagens Existentes de Imagens de Alta Complexidade

A contração regional do ventrículo esquerdo (VE) tem sido modelada de diversas maneiras conforme afirmam Tamaki et al. (1984), Garcia et al. (1985), Faber et al. (1991), Cauvin et al. (1993), Germano et al., (1995), Bardinet et al. (1996), Faber et al. (1999), Garcia et al. (2001), Meyering (2002), Gutierrez et al. (2003). A grande variedade de técnicas quantitativas é resultado da enorme dificuldade em se descrever, através de modelos, o complexo movimento do ventrículo esquerdo. Atualmente, não existe um método que seja universalmente aceito para análise do movimento do VE e os métodos existentes apresentam limitações que influenciam na quantificação precisa da função regional do VE. Alguns dos métodos propostos, de acordo com Tamaki et al. (1984) e Meyering (2002), utilizam, para quantificação, apenas as imagens obtidas no final dos movimentos cardíacos: sístole e da diástole, enquanto o VE descreve uma onda temporal complexa de contração, balanço e expansão durante o ciclo cardíaco. Essas limitações estão diretamente relacionadas com o fato de o ventrículo constituir um objeto não-rígido, como outros existentes, que não estão ligados à formação humana, que rotaciona, translada e apresenta deformação no espaço 3D. A identificação de

movimentos assíncronas das superfícies, especificamente do VE e o encurtamento das paredes em cada instante de tempo e por regiões de interesse, podem indicar a presença de isquemia do músculo cardíaco.

1.2 Abordagem do fluxo ótico

Uma abordagem para quantificar o movimento do VE no espaço 3D é o método baseado na técnica de fluxo ótico, que foi definido por Horn e Schunck em 1981, como o movimento aparente dos padrões de luminosidade de um objeto, observado quando a câmera e o objeto apresentam movimento relativo (HORN et al.,1981). O fluxo ótico pode ser representado por um vetor bidimensional de velocidade, associado a cada pixel no plano da imagem. Pixel é definido como um elemento de imagem e cada pixel contém informações sobre as cores vermelha, verde e azul (SILVA, 2020). O campo de vetores de velocidade especifica o movimento da luminosidade em cada ponto da imagem, à medida que a câmera e o objeto apresentem movimento relativo (GUTIERREZ et al., 1996).

1.3 Formulação do problema

Imagens SPECT - *Single Photon Emission Computed Tomography* (ou Tomografia Computadorizada de Emissão de Fóton Único, em português) é um procedimento de medicina nuclear em que um equipamento denominado câmera gama, gira ao redor do paciente, tirando fotos em vários ângulos. Isto é o que um computador usa para formar uma imagem tomográfica transversal (SHIEL JR, 2021). As imagens spect realizado no coração, é um procedimento não invasivo de imagem nuclear, que usa radiofármacos (substâncias químicas com alguma propriedade radioativa), que podem indicar a presença de isquemia no músculo cardíaco.

O sistema linear envolvido no método do fluxo ótico, para imagens spect, é uma tarefa computacionalmente onerosa, porque para cada voxel (neologisma entre volume e pixel) da imagem a ser analisada, são geradas três equações que compõem o sistema linear. Portanto, quanto maior for a resolução da imagem, maior será o tamanho da memória alocada, e mais

operações computacionais deverão ser realizadas. Para imagens spect de quadros de 64x64x64 voxels o sistema linear possui mais de 750 mil equações e desta forma, o processamento de uma imagem como esta pela CPU tem alto custo computacional. Com a evolução para tecnologia de GPU, *Graphics Processing Unit* (ou Unidade de Processamento Gráfico, em português), oferecendo novas interfaces de programação das placas, como por exemplo, o *Computer Unified Device Architecture* (*CUDA*) e/ou *Open Source Computer Vision Library (openCV*) com interface para *CUDA*, abre-se a possibilidade de utilizar a GPU para processar o sistema linear envolvido no método do fluxo ótico, com baixo tempo de resposta. As interfaces de programação das placas como *CUDA* e *openCV* são necessárias porque permitem que as placas gráficas possam ser usadas para outras finalidades, além de jogos. *CUDA* e *openCV* disponibilizam dezenas de rotinas para manipulação de placas gráficas, para que o analista possa desenvolver soluções para GPUs usando linguagens de programações tradicionais como C++ e Fortran.

Em resumo, o problema está na identificação da viabilidade de usar GPU para resolução de equações lineares, representados por matrizes esparsas de grandes dimensões, com tempo significativamente menor, possibilitando um ferramental adicional, por exemplo, para apoiar os médicos na interpretação das imagens e diagnósticas.

1.4 Hipótese

A Arquitetura CPU, conforme mencionado anteriormente, executa o processamento das informações de forma lenta, inadequada para viabilizar o diagnóstico de doenças do aparelho circulatório. Ocorre que a resolução de um processo em arquitetura CPU é baseada no modelo de execução sequencial, em que a próxima instrução do processo somente será executada quando a instrução atual finalizar.

Com o surgimento da arquitetura GPU, é possível executar as instruções em paralelo, diminuindo sensivelmente o tempo de processamento. A diminuição do tempo será maior, quanto maior o paralelismo do programa. Por exemplo, a soma dos elementos de um vetor, representada pela Fórmula 1:

$S = \sum_{i=0}^{n} V(i)$ (1)

terá um tempo maior do que a soma de dois vetores, representada pela Fórmula 2:

S(i) = Va(i) + Vb(i) (2)
para i = 0 até ‘n’.

Pela GPU poderemos associar cada ‘i’ a um *thread* - um *thread* é uma unidade básica de utilização da CPU, um processo tradicional tem um único *thread* e se um processo tem vários *threads (multithread),* ele pode realizar mais de uma tarefa por vez *(*SILBERSCHATZ et la., 2018). Dessa forma todas as ‘n’ adições, ‘Va(0) + Vb(0), Va(1) + Vb(1),.., Va(n) + Vb(n)’, serão executadas em um único ciclo de máquina, paralelamente, com tempo de resolução muito menor do que a soma dos elementos de um vetor serialmente. Com base no que foi apresentado, tem-se as seguintes questões de pesquisa:

(a) O tempo necessário para processamento das equações lineares, representadas por matrizes esparsas de ordem elevada na GPU é menor que na CPU?
(b) É possível a resolução de equações lineares, representadas por matrizes esparsa, de ordem elevada usando a GPU?

As questões de pesquisa levam então às Hipóteses:

(a) O paralelismo identificado na resolução das equações lineares de ordem elevada, representada por matrizes esparsas é ideal para o processamento pelo GPU porque poderá usar milhares de *threads* para realizar as resoluções em um tempo significativamente menor em comparação com o processamento pelo GPU.
(b) Usando subconjuntos de dados, por causa da limitação física de tamanho da memória GPU, é possível a resolução parcial usando esses subconjuntos e, ao término de toda resolução parcial, a solução deve ser o mesmo de uma resolução não parcial.

1.5 Justificativa

A resolução de equações de fluxo ótico em arquitetura CPU tem custo computacional muito alto, porque as equações geram matrizes de ordem muito elevadas e a sua execução é bastante demorada porque são milhares de instruções sendo executados serialmente. Por exemplo, o sistema linear envolvido no método do fluxo ótico é uma tarefa computacionalmente custosa, porque para cada voxel da imagem a ser analisada, são geradas três equações que compõem o sistema linear. Tomando como exemplo as imagens spect onde são gerados quadros de 64x64x64 voxels o sistema linear possui mais de 750 mil equações e desta forma, o processamento de uma imagem como esta pela CPU tem alto custo computacional. Com a evolução da tecnologia de GPU oferecendo novas interfaces de programação das placas como o *Computer Unified Device Architecture* (CUDA) e/ou *Open Source Computer Vision Library* (openCV) com interface para CUDA, abre-se a possibilidade de utilizar a GPU para processar o sistema linear envolvido no método do fluxo ótico com tempo de resposta significativamente menor.

1.6 Objetivos

1.6.1 Objetivo Geral

A proposta desta dissertação é apresentar a resolução computacional de equações lineares representadas por matrizes de ordem elevada, usando arquitetura GPU para comparação entre as *performances* de tempo com a arquitetura CPU.

1.6.2 Objetivos Específicos

Para atingir o objetivo geral, têm-se como objetivos específicos:

- Demonstrar a viabilidade da resolução parcial de equações lineares de ordem elevada, como uma alternativa para limitação do tamanho físico da memória principal da GPU

- Explorar as características de processamento paralelo das arquiteturas GPU usando *multithreads*;
- Efetuar estudos da multiplicação de matrizes por ser um modelo matemáticos com características de resolução paralela.

1.7 Delimitação

Este trabalho irá apresentar a resolução de equações lineares de ordem elevadas usando GPU e como alternativa para limitação da memória principal, será usado um algoritmo que fará a resolução parcial. Os valores das equações lineares serão gerados aleatoriamente e gravado em uma mídia não volátil, tipo USB, para serem lidos e enviados parcialmente para a memória principal para o processamento.

Não será abordado o detalhamento das caracteristicas internas da arquitetura GPU.

1.8 Organização do Trabalho

Este trabalho é composto por seis capítulos e referências:

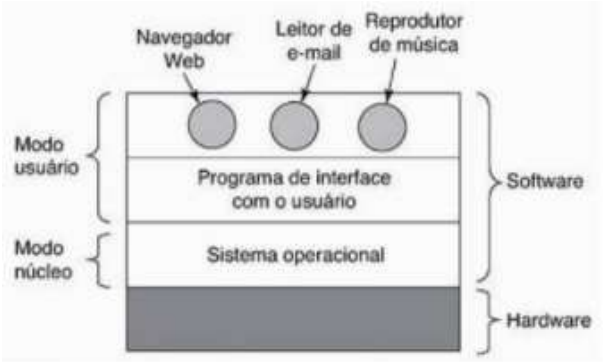
- Primeiro Capítulo tem-se a introdução, com a descrição do problema, hipótese, objetivos, delimitação e conteúdo.
- Segundo Capítulo trata da Revisão de Literatura, onde serão apresentados os pontos relevantes tratados pelos autores da área.
- Terceiro Capítulo apresenta a metodologia utilizada para comprovar a hipótese.
- Quarto Capítulo trata do desenvolvimento propriamente dito.
- Quinto Capítulo apresenta os Resultados e Discussão.
- Sexto Capítulo apresenta as Considerações Finais
- Por fim, as Referências utilizadas.

2 REVISÃO DE LITERATURA

2.1 Sistema Operacional

Segundo Tanenbaum et la. (2015), um computador moderno básico é composto de processadores, memória principal, discos, teclado, mouse, monitor de tela, interface de rede e dispositivos de entrada / saída. Para gerenciar os componentes e usar-las de forma otimizada, os computadores são equipados com uma camada de software chamada sistema operacional, como mostra a Figura 1.

Figura 1 – Localização do sistema operacional



Fonte: Tanenbaum et la. (2015), adaptado pelo autor (2021)

Conforme Silberschatz et la. (2018) um sistema operacional é um software que gerencia o hardware de um computador, e estão presentes em toda parte, carros, celulares, computadores e eletrodoméstico em geral, que fazem parte dos dispositivos de ‘Internet das Coisas’.

A parte de sistema operacional que faz a execução dos programas é o gerenciamento dos processos. De acordo com Tanenbaum et la. (2015), um processo é apenas uma instância de um programa em execução.

De acordo com Silberschatz et la. (2018), conforme um processo é executado, ele muda de estado. O estado de um processo é definido em parte pela atividade atual desse processo. Um processo pode estar em um dos seguintes estados, conforme mostrado pela Figura 2:

- Novo – o processo está sendo criado;
- Pronto – o processo está aguardando algum processador;

- Em Execução – as instruções (do processo) estão sendo executadas;
- Encerrado – término do processo;
- Em Espera - o processo está esperando que algum evento ocorra (como um I / O / conclusão ou recepção de um sinal).

Figura 2 – Diagrama dos estados do processo

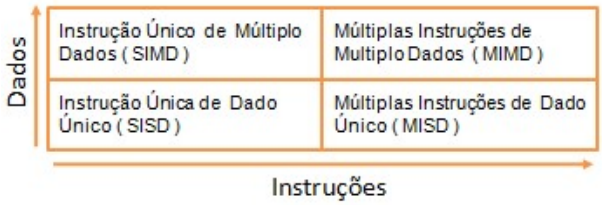
Fonte: Silberschatz et la. (2018)

2.2 Arquitetura de Computadores

Conforme Cheng et la. (2014), existem várias maneiras diferentes de classificar arquiteturas de computador. Uma classificação de esquema amplamente utilizada é a taxonomia de Flynn, que classifica as arquiteturas em quatro tipos diferentes, conforme as instruções e os dados fluem através dos núcleos, conforme mostra a Figura 3.

- *Instrução Única de Dado Único (Single Instruction Single Data - SISD)* refere-se ao computador tradicional, com um núcleo;
- *Instrução Única de Dados Múltiplos (Multiple Instruction Multiple Data - SIMD)* referem-se a um tipo de arquitetura paralela, com vários núcleos no computador e todos executam o mesmo fluxo de instrução;
- *Múltiplas Instruções de Dado Único (Multiple Instruction Single Data – MISD)* referem-se a uma arquitetura incomum, onde cada núcleo opera o mesmo fluxo de dados por meio de fluxos de instruções separados.
- *Multiplas Instruções de Multiplo Dados (Multiple Instruction Multiple Data - MIMD)* refere-se a um tipo de arquitetura paralela em que múltiplos núcleos operar em vários fluxos de dados, cada um executando instruções independentes.

Figura 3 – Classificação de arquitetura

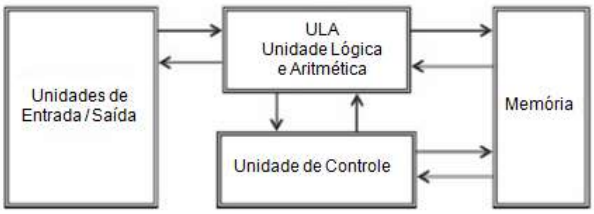


Fonte: Cheng et la. (2014)

2.3. Arquitetura CPU

Conforme Yadin (2016), Von Neumann projetou uma arquitetura de computador que é usada até os dias atuais para o desenvolvimento de computadores modernos do tipo CPU. Um princípio importante que se originou da arquitetura Von Neumann é a modularidade, mostrado na Figura 4.

Figura 4 - O modelo de Von Neumann



Fonte: Yadin (2016)

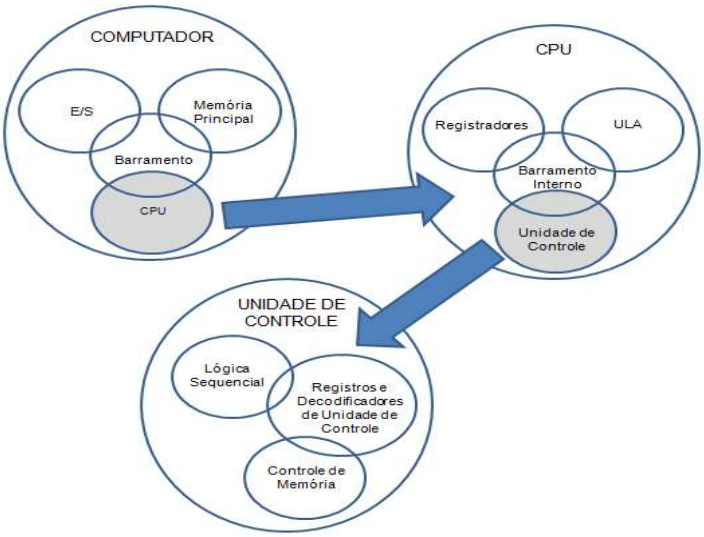
De acordo com Stallings (2016) a arquitetura CPU é composta de quatro componentes: Unidade Central de Processamento (Central Processing Unit - CPU), Memória Principal, Unidades de Entrada e Saída e Interconexão de Sistemas, conforme mostra a Figura 5.

- A CPU é responsável pelo controle e processamento das instruções do computador, é o coração de um computador. A CPU é composta de:
    - Unidade Lógica Aritmética (ULA) – responsável pelas operações Lógicas e Aritméticas;
    - Registradores – memória com finalidade específica, e a quantidade de registradores é dependente da arquitetura da CPU. Segue abaixo alguns registradores:
        - ✓ *Program Counter* (PC) - registrador que indica qual é a próxima instrução a ser executada;

- ✓ *Stack Pointer* (SP) – registrador que indica qual é o endereço de retorno de uma chamada de subrotina e/ou usado para passagem de parametros de variáveis de sub-rotinas;
- ✓ *Instruction Register* (IR) – registrador que recebe a instrução a ser executada;
- ✓ Acumulador (AC) – registrador de uso geral, normalmente usado para apoiar algumas instruções;

- A memória principal é um componente muito importante porque um processo para ser executado pela CPU precisa obrigatóriamente está na memória principal, e quanto maior a capacidade da memória principal, mais processos serão executados.
- Unidade de entrada e saída são os componentes que permite a troca de dados entre o computador e os meus externos, é através das unidades de entrada e saída que o sistema operacional consegue ler e armazenar dados em mídias não volátil;
- Interconexão de Sistemas é o mecanismo necessário para que os dados trafeguem entre os componentes de um computador.

Figura 5 - Estrutura interna de um computador de arquitetura CPU

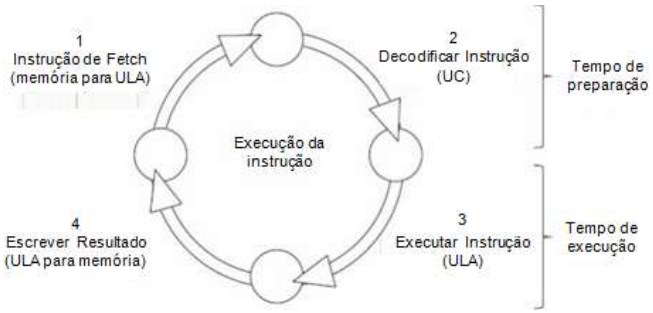


Fonte: Stallings (2016), adaptado pelo autor

Yadin (2016) descreve os quatro estágios da execução de uma instrução. Os dois estágios iniciais compreendem a iniciação pela UC (Unidade de Controle) e tem como objetivo preparar as instruções para execução. A segunda fase, que consiste nas duas últimas etapas, é a execução, porque

contém as execuções (realizadas pela ULA) e o armazenamento do resultado (realizado pela UC). O processador nunca para, e mesmo que não tenha nada a fazer, ele executa um loop ocioso. Por esse motivo, a execução de instruções é um processo sem fim (desde que o sistema esteja ativo), como mostra a Figura 6.

Figura 6 – Estágios de execução de instrução da CPU

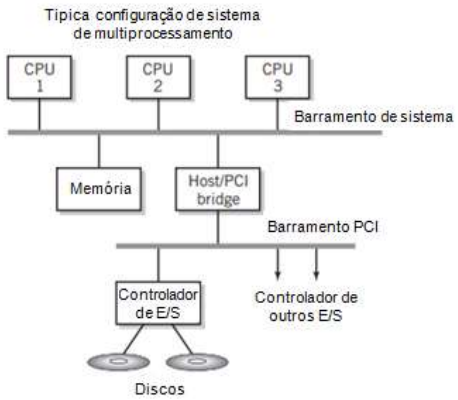


Fonte: Yadin (2016), adaptado pelo autor

Uma forma de aumentar o poder de processamento das CPU é colocando várias CPUs compartilhando recursos computacionais como memória principal, recursos de entrada e saída, entre outros.

Conforme Englander (2014) equipamentos que possuem várias CPUs em um único computador, compartilhando a memória principal e recursos de entrada e saídas, são chamados de ‘sistemas multiprocessadores’ ou ‘sistemas fortemente acoplados’. Quando vários processadores de CPU são fornecidos em um único circuito integrado, eles são mais comumente chamados de ‘processadores multicore’, conforme mostra a Figura 7.

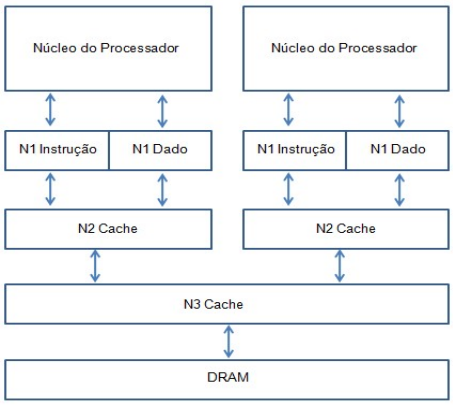
Figura 7 - Configuração típica do sistema de multiprocessamento



Fonte: Englander (2014), adaptado pelo autor

Conforme Cook (2013) a maioria dos aplicativos, CPU e GPU, são frequentemente limitados à memória e não ao ciclo do processador ou ao clock / carga do processador. Os fornecedores de CPU tentam resolver esse problema usando memória cache, conforme a Figura 8.

Figura 8 - Organização típica de cache de CPU moderna



Fonte: Cook (2013), adaptado pelo autor

De acordo com Cheng et la. (2014), o sistema operacional de uma arquitetura CPU gerencia a execução dos processos como multitarefas, seguindo o ciclo de vida de um processo. É comum dividir o problema em uma série de tarefas e cada parte sendo executada sequencialmente, representada pela Figura 9.

Figura 9 - O problema é dividido em pequenas partes

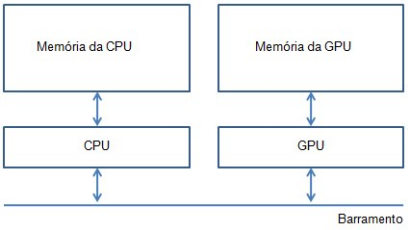
Fonte: Cheng et la. (2014), adaptado pelo autor

2.4 Arquitetura GPU

Conforme Aamodt et la. (2018) muitas vezes surgem questionamento sobre a possibilidade de substituir internamente as CPUs pelas GPUs, mas isso ainda parece improvável. Nos sistemas atuais, as GPUs não são dispositivos autônoma de computação. Em vez disso, eles são combinados com uma CPU, como mostrado na Figura 10, em um único chip ou inserindo

um cartão contendo apenas uma GPU em um sistema contendo uma CPU. A CPU é responsável por iniciar a computação na GPU e transferir dados de e para a GPU. Uma razão para esta divisão de trabalho entre CPU e GPU é que o início e o fim do cálculo normalmente requerem acesso a dispositivos de entrada / saída (E / S).

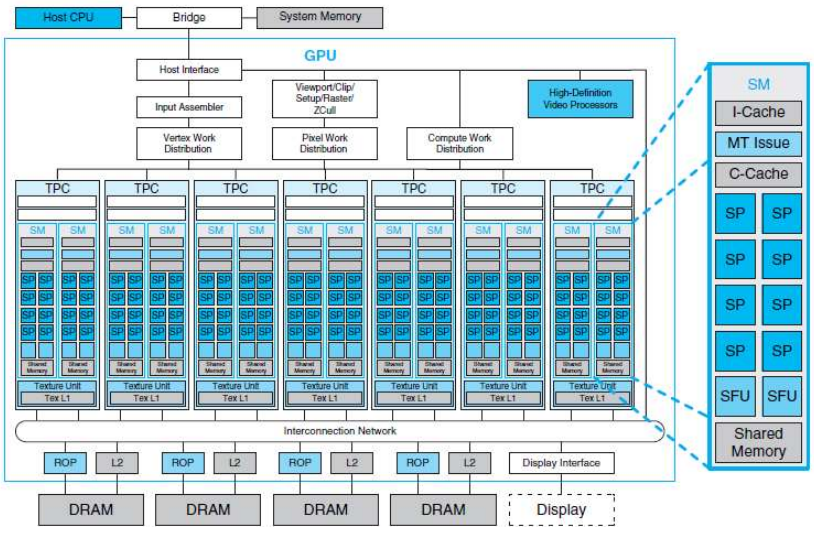
Figura 10 – Combinação entre CPU e GPU



Fonte: Aamodt et la. (2018), adaptado pelo autor

De acordo com o Stallings (2016), a GPU está sendo usado como alternativa para processamento paralelo para outros tipos de aplicações além dos games, como bioinformática, finanças computacionais, modelagem estatística, visão computacional e imagens médicas, originando o termo de computação de uso geral usando GPGPU (*General-Purpose GPU*). A amplitude das possibilidades do uso da GPU também é decorrente do surgimento do CUDA - *Compute Unified Device Architecture*. A Figura 11 mostra uma arquitetura GPU básica.

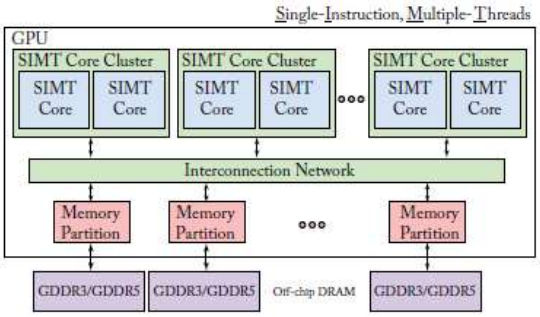
Figura 11 - Arquitetura GPU básica



Fonte: Nickolls et la. (2020)

De acordo com Aamodt et la. (2018), uma GPU moderna é composta de muitos núcleos, conforme mostrado na Figura 12. A NVIDIA chama isso de núcleos multiprocessadores e a AMD os chama de unidades de computação.

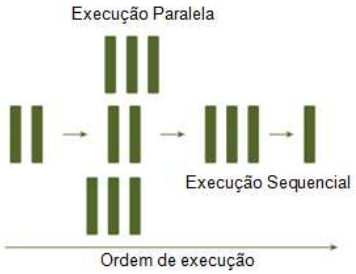
Figura 12 - Arquitetura de GPU moderna



Fonte: Aamodt et la. (2018)

Conforme Cheng et la. (2014), existem duas maneiras de classificar a relação entre duas peças de computação, algumas são relacionadas por uma restrição de precedência e, portanto, deve ser processada sequencialmente e outros não têm tal restrições e podem ser processadas simultaneamente. Qualquer programa que contém tarefas que são realizadas simultaneamente é um programa paralelo, conforme representado na Figura 13.

Figura 13 – Execução paralela



Fonte: Cheng et la. (2014), adaptado pelo autor (2021)

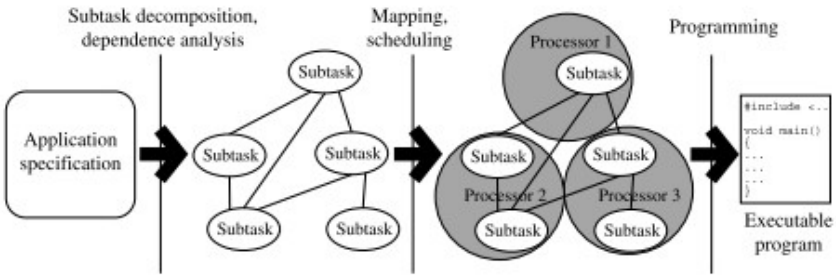
2.5 Palalelismo

Conforme Cheng et la. (2014), o paralelismo está se tornando onipresente, e a programação paralela está se tornando dominante no mundo da programação. O paralelismo em vários níveis é a força motriz do design de

arquitetura. Existe dois tipos fundamentais de paralelismo em aplicativos: paralelismo de tarefas e paralelismo de dados.

O paralelismo de tarefas se concentra na distribuição de funções em vários núcleos. De acordo com Sinnen (2007), programar um sistema paralelo para a execução de um único aplicativo é extremamente desafiador. A Figura 14 mostra o processo de programação paralela.

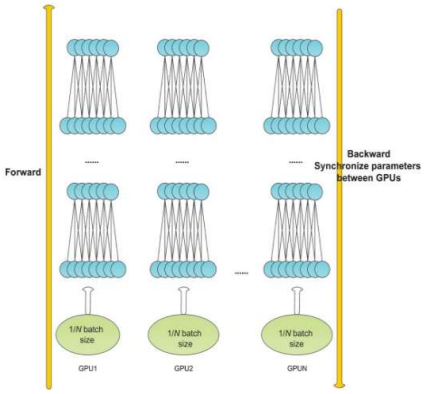
Figura 14 – Processo de programação paralela



Fonte: Sinnen (2007)

De acordo com Li et la. (2016), paralelismo de dados significa que cada GPU usa o mesmo processo para diferentes subconjuntos de dados, e não há sincronização entre GPUs, mostrado na Figura 15.

Figura 15 - A ilustração do modo de paralelismo de dados



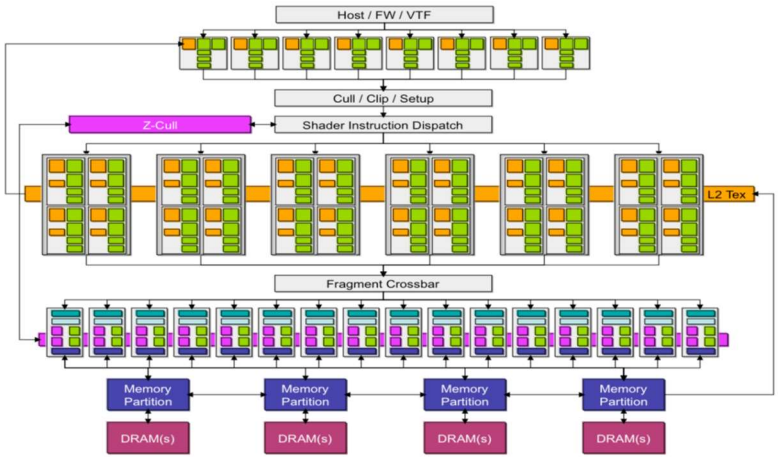
Fonte: Li et la. (2016)

2.6 Evolução da GPU da NVIDIA

De acordo com o Glaskowsky (2009), os chips progrediram das funções de desenho de pixels mais simples para implementar o pipeline 3D completo.

Os produtos GeForce tornaram-se mais flexíveis e rápidos com o GeForce 7800, conforme mostrado na Figura 16.

Figura 16 – GeForce 7800



Fonte: Glaskowsky (2009)

Conforme Glaskowsky (2009), com o Tesla, os programadores não precisam se preocupar em fazer as tarefas parecerem operações gráficas, a GPU pode ser tratada como um multi-coreprocessador.

De acordo com Mills (2020), com a arquitetura Tesla, a NVIDIA também introduziu a linguagem de programação *CUDA (Compute Unified Devide Architecture*) em C, permitindo o uso de uma linguagem de programação de alto nível para programar as placas GPU. A próxima geração após o Tesla é a arquitetura Fermi da NVIDIA, que consiste em múltiplos *streaming* multiprocessadores (*SMs*) e os *SMs* são apoiados por um segundo nível cache.

Em 2012 surgiu a arquitetura NVIDIA Kepler, com a qual a eficiência energética de sua matriz foi drasticamente melhorada ao diminuir sua velocidade de clock e unificar o *clock* central com a placa, resolvendo assim o problema de GTX 480 da geração anterior (havia superaquecimento).

Conforme Mills (2020), em 2014 surgiu a arquitetura NVIDIA Maxwell, sua 10ª geração de GPUs com extrema eficiência energética e desempenho excepcional por watt consumido, ideal para uso em mini PCs e laptops, e o SMM apresenta menos núcleos do que seu antecessor, com apenas 128 núcleos. De acordo com Mills (2020), em 2016, foi lançado o NVIDIA Pascal, que melhorou o desempenho ao ser capaz de colocar mais SMs no mesmo

chip. Outras melhorias foram o sistema de memória GDDR5X e velocidades de transferência de até 10 Gbps.

Conforme Mills (2020), em 2018 foi lançado a arquitetura de Turing, em que é introduzido pela primeira vez o hardware Ray Tracing dedicado com núcleos Tensor e RayTracing.

Na Tabela 1 é apresentado um resumo da evolução das arquiteturas GPU.

Tabela 1 – Evolução da arquitetura GPU

Ano	Arquitetura	Série	Modelo	Processo litográfico
2006	Tesla	GeForce 8	G80	90nn
2010	Fermi	GeForce 400	GF100	40nn
2012	Kepler	GeForece 600	GK104	28nn
2014	Maxwell	Geforce 900	GM204	28nn
2016	Pascal	GeForce 10	GP102	16nn
2018	Turing	GeForce 20	TU102	12nn

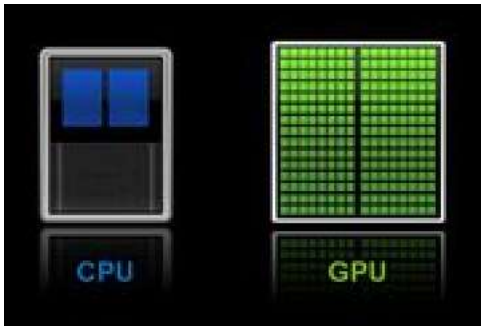
Fonte: Mills (2020)

2.7 Diferença entre CPU e GPU

Conforme Cook (2013), CPUs e GPUs são arquitetonicamente muito diferentes. CPUs são projetadas para executar um pequeno número de tarefas complexas e a GPUs são projetadas para executar grande número de tarefas simples. O *design* da GPU é voltado para problemas que podem ser divididos em milhares de pequenos fragmentos que trabalham individualmente.

Conforme Caulfield (2009) a arquitetura CPU foi construída para computação sequencial e a GPU foi construída para computação paralela, conforme mostrado na Figura 17.

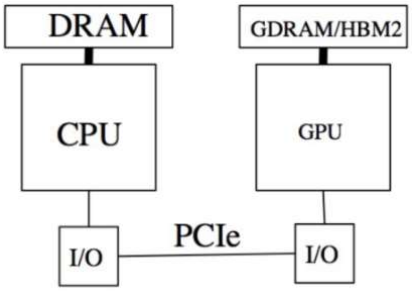
Figura 17 - Diferença entre arquitetura CPU e GPU



Fonte: Caulfield (2009)

De acordo com Allalen et la. (2017) a GPU, na sua essência, é uma placa de vídeo que contém memórias do tipo RAM separado da memória principal da CPU e não possui unidades de entrada e saída como uma CPU. Dessa forma é necessário que a CPU envie os dados para GPU usando barramento PCIe, representado na Figura 18.

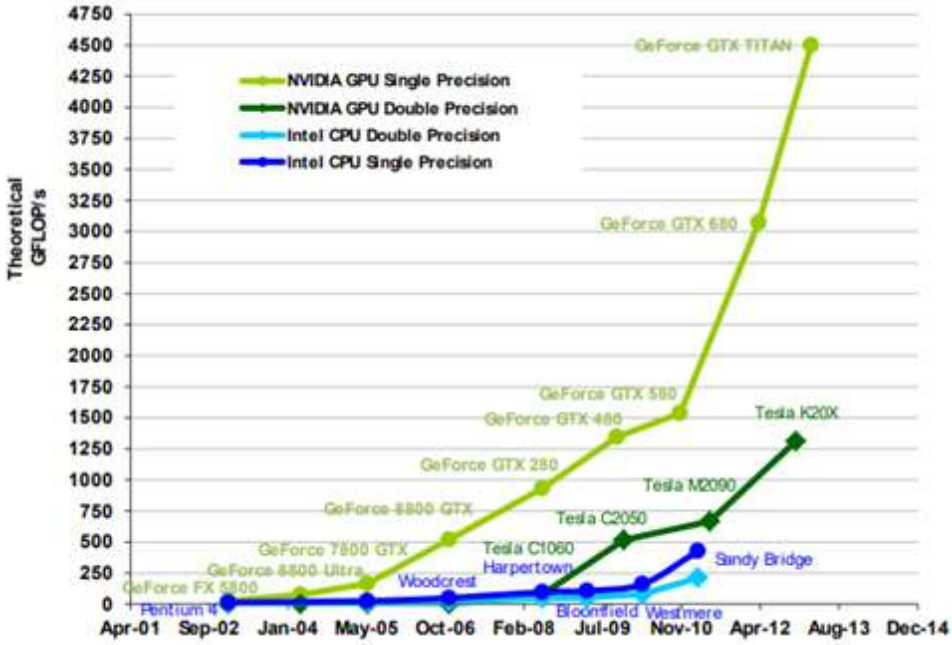
Figura 18 - Comunicação entre memórias da CPU e GPU



Fonte: Allalen et al. (2017)

De acordo com Nvidia (2013), impulsionado pela grande demanda do mercado por gráficos 3D de alta definição em tempo real, a unidade do processador gráfico programável ou GPU, evoluiu para um equipamento paralelo, com processador *multithread* e *multicore* e com grande capacidade computacional, mostrado no Gráfico 1.

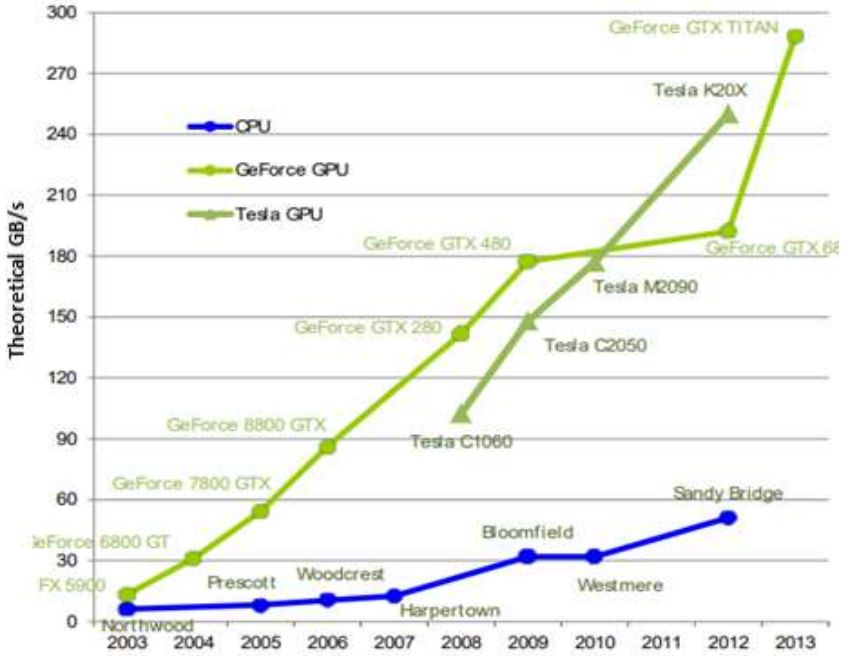
Gráfico 1 - Operações de ponto flutuante por segundo para a CPU e GPU



Fonte: Nvidia (2013)

Também houve uma evolução na capacidade de tranferência de dados entre as memórias CPU e GPU, mostrado no Gráfico 2.

Gráfico 2 - Largura de banda da memória para CPU e GPU



Fonte: Nvidia (2013)

Conforme Caulfield (2009), as CPUs continuam essenciais para tarefas que exigem muita interatividade: acessar informações de um disco rígido em resposta aos pressionamentos de tecla do usuário, por exemplo. As GPUs dividem problemas complexos em milhares ou milhões de tarefas separadas, sendo executados de uma vez. No Quadro 1 é apresentada uma comparação entre a CPU e GPU.

Quadro 1 – Comparação entre CPU e GPU

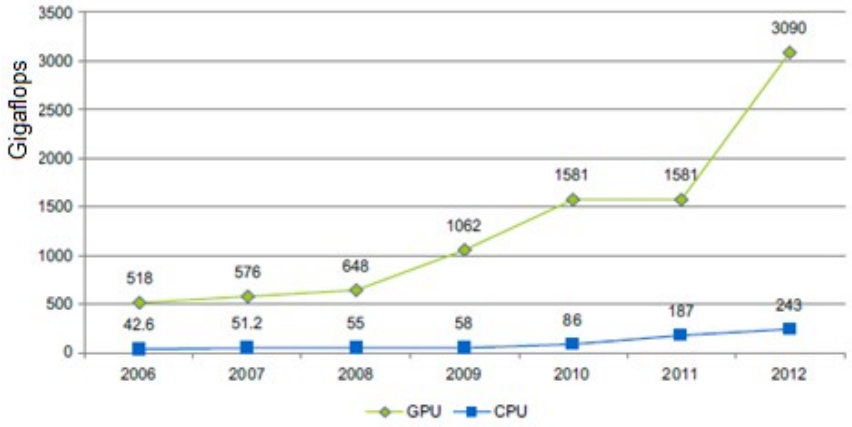
CPU (Central Processing Unit)	GPU (Graphics Processing Unit)
Vários núcleos	Muitos núcleos
Baixa latência	Alta produtividade
Bom para processamento serial	Bom para processamento paralelo
Pode executar várias operações de uma vez	Pode executar milhares de operações de uma vez

Fonte: Caulfield (2009), adaptado pelo autor (2021)

Dos estudos de Cook (2013), é possível apresentar uma comparação do poder computacional entre a GPUs e CPUs, e a divergência do poder computacional da CPU e GPU é até 2009, quando a GPU finalmente

ultrapassou a barreira de 1000 gigaflops ou 1 teraflop. Isto é ilustrado no Gráfico 3.

Gráfico 3 - Desempenho da CPU e GPU em gigaflops

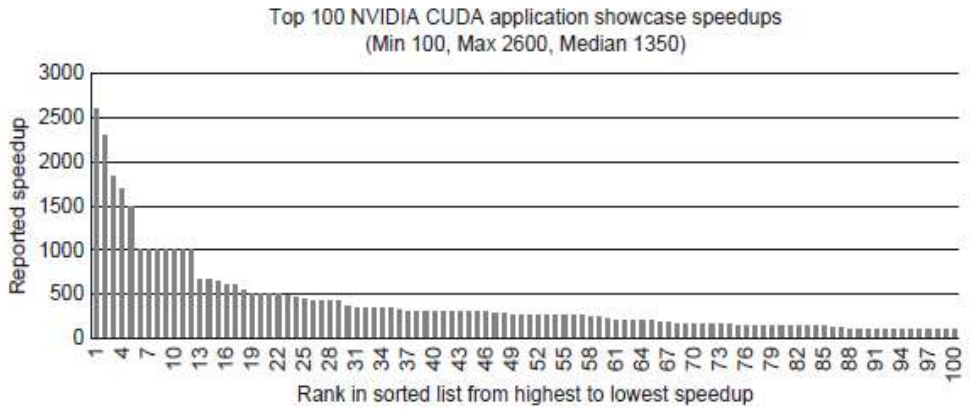


Fonte: Cook (2013)

2.8 CUDA

De acordo com Farber (2011), o desempenho é a principal razão para usar a tecnologia GPU, conforme o Gráfico 4, que ilustra o desempenho dos 100 principais aplicativos. Eles demonstram a ampla variedade de aplicações que a tecnologia GPU pode acelerar em dois ou mais ordens de magnitude (100 vezes) sobre processadores *multicore*. A tecnologia GPU ocorre como oportunidade, onde os processadores *multicore* tradicionais não podem mais atingir acelerações significativas.

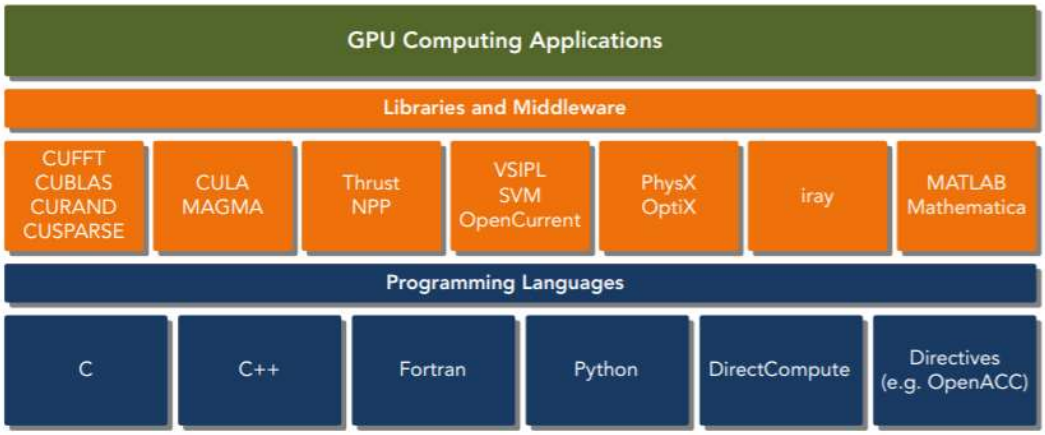
Gráfico 4 – Desempenho dos 100 melhores aplicativos da NVIDIA



Fonte: Farber (2011)

Conforme Cheng et la. (2014), CUDA é um conjunto de tecnologias para programação de placas gráficas NVIDIA e hardware de computador. CUDA C é uma extensão para linguagem C ou C++ e é a arquitetura oficial da GPGPU desenvolvida pela NVIDIA. É atualizada regularmente e há uma abundância de documentação e bibliotecas disponíveis (ROSE, 2014). A Figura 19 mostra a plataforma CUDA.

Figura 19 - Plataforma CUDA



Fonte: Cheng et la. (2014)

Conforme Cheng et la. (2014), os programas em CUDA C podem ser desenvolvidas usando *IDE (Integrated Development Environment*) instalados em CPU, como por exemplo *Visual Studio* ou usando linhas de comandos. Na plataforma CUDA existem dois ambientes que se interagem, especialmente em se tratando de troca de dados entre as memórias da CPU e GPU: um ambiente é chamado de *host* que é a CPU e o outro que é chamado de *device*, que é a GPU. Na figura 20 é apresentado o programa *Hello World* em linguagem de programação 'C'.

Figura 20 - Programa *Hello World* em linguagem C

```
1  #include <stdio.h>
2
3  int main(void)
4  {
5    // hello from cpu
6    printf("Hello World from CPU!\n");
7    return 0;
8  }
```

Fonte: Cheng et la. (2014)

Na Figura 21, é apresentado o programa *Hello World* em linguagem de programação CUDA C.

Figura 21 - Programa *Hello World* em linguagem CUDA C

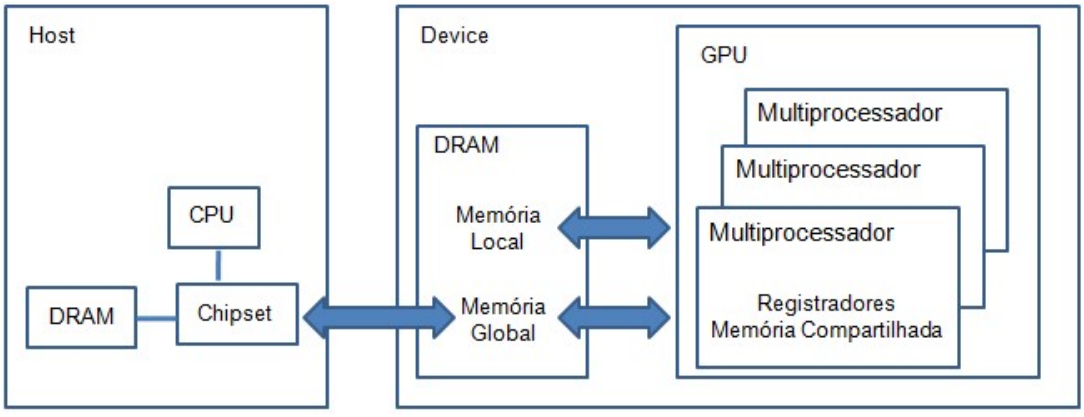
```
1   #include <stdio.h>
2
3   _global_ void helloFromGPU (void)
4   {
5      printf("Hello World from GPU!\n");
6   }
7
8   int main(void)
9   {
10    // hello from gpu
11    helloFromGPU <<<1, 10>>>();
12    cudaDeviceReset();
13    return 0;
14  }
```

Fonte: Cheng et la. (2014)

Conforme Cheng et la. (2014), o modelo de programação CUDA permite que sejam executados aplicativos em computação de sistema heterogênea, simplesmente anotando o código com um pequeno conjunto de extensões para a linguagem de programação ‘C’. Um ambiente heterogêneo consiste em CPUs (*host*) complementadas por GPUs (*device)*, cada uma com sua própria memória, separados por um barramento PCI-Express, mostrado pela Figura 22, da seguinte forma:

- Host: a CPU e sua memória (memória host);
- Device: a GPU e sua memória (memória do dispositivo).

Figura 22 – Ambiente heterogênio

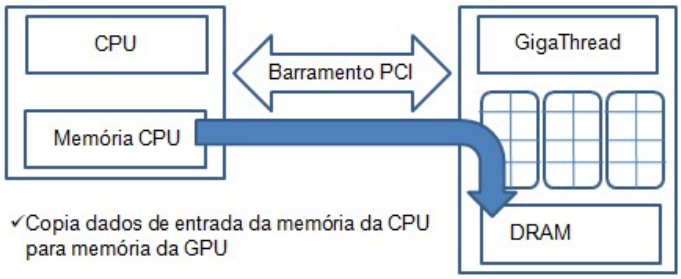


Fonte: Nvidia (2009), adaptado pelo autor (2021)

A execução de um programa CUDA tem basicamente as seguintes etapas:

1. Copiar os dados da memória da CPU para a memória da GPU, Figura 23.

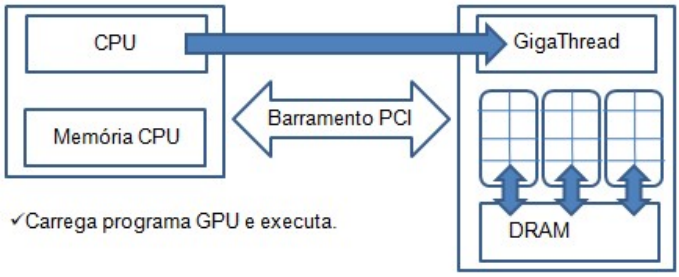
Figura 23 - Copiar os dados da memória host (CPU) para device (GPU)



Fonte: Southard (2013), adaptado pelo autor (2021)

2. Executar os 'kernels'usando os dados da memória da GPU, Figura 24.

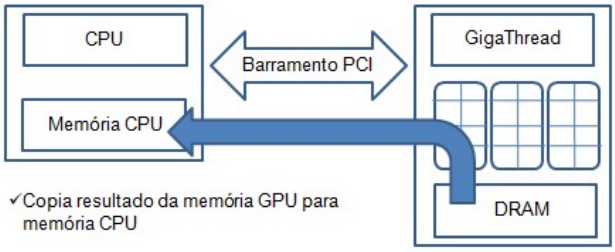
Figura 24 – Executar programa na device



Fonte: Southard (2013), adaptado pelo autor (2021)

3. Copiar os dados de volta da memória da GPU para a memória da CPU, Figura 25.

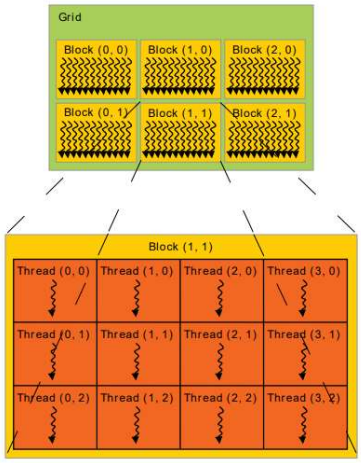
Figura 25 – Copiar dados do device (GPU) para host (CPU)



Fonte: Southard (2013), adaptado pelo autor (2021)

Conforme Wilt (2013), os *kernels* do GPU são lançados como '*grids*' de *'block'* de '*threads*'. *Threads* podem ainda ser dividido em warps de 32 *threads,* mostrado na Figura 26.

Figura 26 – Modelo de memória de CUDA

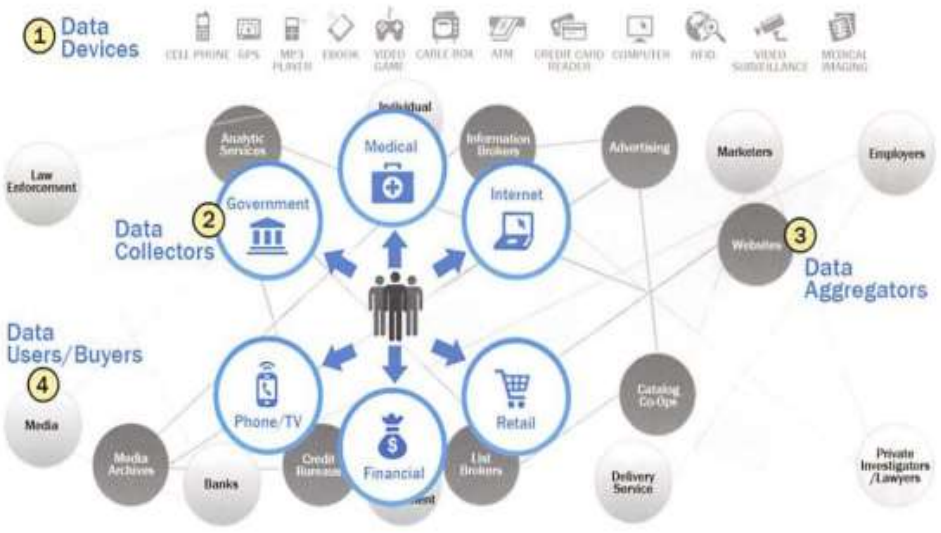


Fonte: Nvidia (2013)

2.9 Big Data

De acordo com Glass et la. (2015), dados, informações, fatos - qualquer termo que você deseja usar, coletar e análise de dados desempenham um papel crucial na capacidade da humanidade de sobreviver e prosperar desde o início da consciência. De acordo com Jewell (2014), são criados 2,5 quintilhões de bytes de dados, todos os dias, o que significa que 90% dos dados do mundo foram criados nos últimos dois anos. A grande quantidade de dados e a capacidade crescente de processo levou à criação do termo 'Big Data'. A Figura 27 mostra um ecosistema de Big Data.

Figura 27 – Tipico ecosistema de Big Data



Fonte: Dietrich et la. (2015)

As tecnologias de Big Data estão sendo cada vez mais usadas para fins biomédicos e pesquisas de informática em saúde. Grandes quantidades de dados são geradas e coletadas em velocidade e escala excepcionais para biológicos e clínicos. Novas tecnologias fizeram a movimentação de centenas de terabytes / petabytes de dados possíveis, que estão sendo disponibilizados para a comunidade médica e científica (SUGANYA et la., 2018).

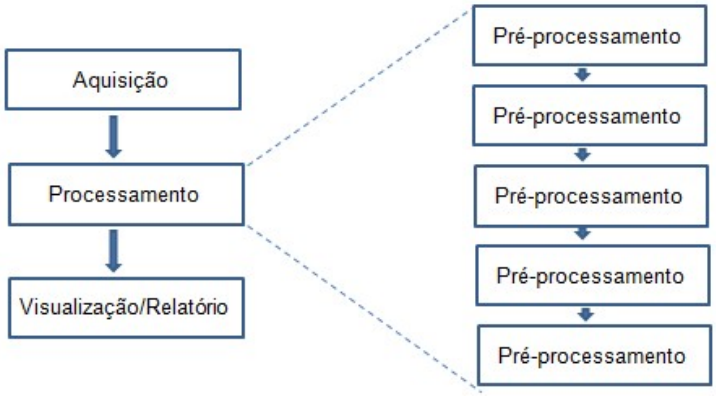
Ainda segundo Suganya et la. (2018), técnicas de imagem anatômica como ultrassom (US), tomografia computadorizada (TC) e imagens magnéticas de ressonância (MRI) são usadas diariamente em todo o mundo para exames não invasivos.

Conforme Maes et la. (2019), devido aos contínuos avanços tecnológicos em aquisição de imagens médicas, novas modalidades de imagens estão sendo introduzidas nas práticas médicas, como multi-slice (volumétrico) e multienergia CT, multi-paramétrico e multi-frame (dinâmico) MRI, multidimensional (3D + tempo) US, imagem intervencionista multi-planar ou PET / CT e PET / MRI multimodal (híbrido). A análise de grandes quantidades de dados de imagem tornou-se um grande desafio e é um gargalo para diagnósticos, terapias, planejamentos e acompanhamentos, além da pesquisa biomédica.

O objetivo do processamento de imagem é dividido em cinco grupos, conforme mostra a Figura 28.

- Grupo 1: Visualização - Observa as diferentes modalidades de regiões médicas que não são visíveis;
- Grupo 2: Nitidez e restauração de imagem - Cria uma imagem melhor para diagnosticar doenças;
- Grupo 3: Recuperação de imagem - Procura a região de interesse pelos médicos para tratamento futuro ou planejamento cirúrgico;
- Grupo 4: Medição do padrão - Medir várias regiões na imagem para veracidade;
- Grupo 5: Reconhecimento de imagem - Distingue a região doente / infectada em uma imagem.

Figura 28 - Cinco fases do processamento de imagem



Fonte: Suganya et la. (2018), adaptado pelo autor (2021)

É comum classificar a Big Data em dados estruturados, como por exemplo, idade, estado civil e ano de fabricação e dados não estruturas como por exemplo chat, email e ligação telefonica. A Figura 29 mostra quatro tipos de estrutura de dados, com 80-90% do crescimento futuro de dados oriundo de não estruturados. Embora diferentes, os quatros tipo são normalmente usados em conjuntos.

Figura 29 - O Big Data está cada vez mais desestruturado

Fonte: Dietrich et la. (2015), adaptado pelo autor (2021)

A IBM foi uma das grandes indústrias de tecnologia que investiu fortemente no modelo de Big Data porque percebeu que o volume de dados estava crescendo rapidamente e que os modelos tradicionais de tratamento de dados seriam insuficientes. A IBM frequentemente descreve a Big Data usando os quatroVs (Gentsch, 2019), mostrado no Quadro 2.

Quadro 2 – Descrição de big data segundo a IBM

Volume	Isto descreve a quantidade de dados recebidos que devem ser armazenados e analisado. O ponto em que uma quantidade de dados é realmente declarada como o big data descrito acima depende dos sistemas disponíveis. Empresas ainda enfrentam o desafio de armazenar e analisar os valores recebidos de dados de forma eficiente e eficaz. Nos últimos anos, várias tecnologias, como sistemas distribuídos, foram estabelecidas para esses finalidades.
Velocidade	Isto descreve dois aspectos: Por um lado, os dados são gerados em uma velocidade muito alta e, por outro lado, os sistemas devem ser capazes de armazene, processe e analise essas quantidades de dados imediatamente.
Variedade	A grande variedade de dados do mundo do big data confronta os sistemas com a tarefa de não mais processar apenas com dados estruturados de tabelas, mas também com dados semi e não estruturados de textos contínuos, imagens ou vídeos, que representam até 85% das quantidades de dados. Especialmente no campo das mídias sociais, uma infinidade de não estruturados dados são acumulados, cuja semântica pode ser coletada com a ajuda de IA tecnologias.
Veracidade	Considerando que as três dimensões descritas aqui podem ser dominadas pelas empresas hoje com a ajuda de tecnologias, métodos e o uso de meios suficientes, há um desafio que ainda não foi resolvido na mesma medida. Veracidade significa os termos de confiabilidade, veracidade e significado do big data. Portanto, não é uma questão de todos os dados armazenados são confiáveis e não devem ser analisados.

Fonte: Gentsch (2019), adaptado pelo Autor (2021)

2.10 Inteligência Artificial

De acordo com o Corea (2019), Inteligência Artificial (IA) representa hoje em dia uma mudança de paradigma para o progresso científico e a evolução da indústria. Os investidores estão mobilizando grandes quantidades de capital, mas ainda não têm uma visão clara das vantagens competitivas que a IA poderá agregar. IA é um sistema que possibilita que computadores possam escrever seus próprios algoritmos, sem a necessidade de intervenção de lógica computacional de um analista e/ou programador. Pode-se classificar IA em:

- Automação de processos robóticos (Robotic Process Automation - RPA)
- Sistemas especialistas
- Visão computacional (CV)
- Processamento de Linguagem Natural (PNL)
- Redes Neurais (NNs ou ANNs)
- Sistemas Autônomos
- Inteligencia Artificial Distribuído (DAI
- Computação Afetiva

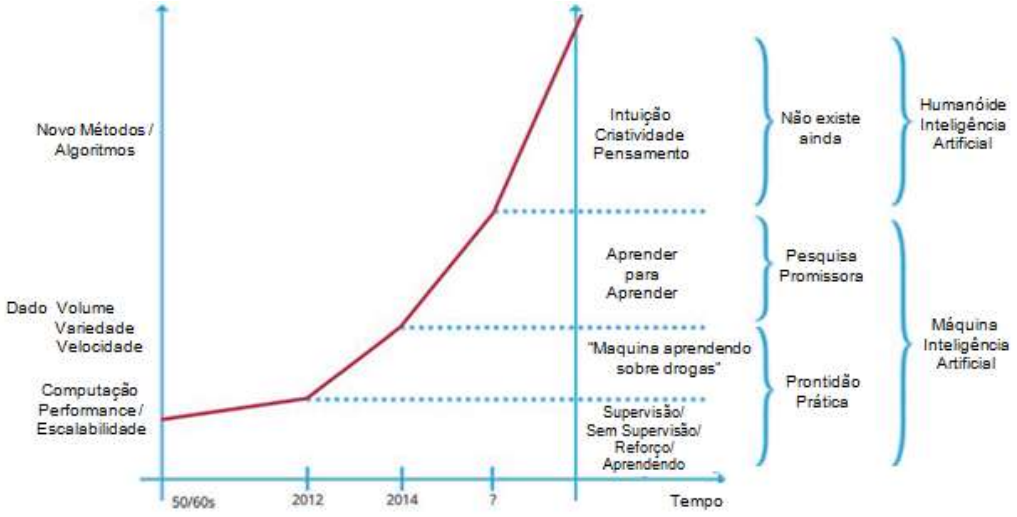
- Algoritmos Evolucionários (EA
- Programação em Lógica Indutiva (ILP)
- Redes de decisão
- Programação Probabilística
- Ambiente inteligente (AmI)

Conforme Stephenson (2018), IA é um termo amplo para definir quando uma máquina pode responder de forma inteligente ao seu ambiente. O ser humano interage com IA no Siri da Apple, Amazon Eco, carros autônomos, bots de bate-papo online e adversários de jogos. IA está filtrando spam das caixas de entrada, corrigindo ortografia, decidindo quais postagens aparecem no topo dos feeds de mídia social. IA tem uma ampla gama de aplicações, incluindo reconhecimento de imagem, linguagem natural processamento, diagnóstico médico, movimentos robóticos, detecção de fraude e muito mais.

De acordo com Ranschaert et la. (2019), as principais oportunidades de desenvolvimento na saúde podem ser atribuídas a três fatores: uma integração adicional de tecnologias de informação (TI) e sistemas de saúde, a conexão de provedores de saúde em redes capazes de compartilhar informação digital e a padronização de procedimentos médicos e seus formatos digitais. Os três fatores irão incentivar e facilitar o desenvolvimento e implantação de ferramentas baseadas em IA para otimização do fluxo de trabalho e valor em saúde. Além disso, aplicativos para telemedicina e telerradiologia, podem fornecer remotamente cuidados e diagnósticos. Nos próximos anos, provavelmente também haja a implantação de métodos analíticos e preditivos, ferramentas médicas que são fornecidas no modelo B2C para o consumidor médio.

Conforme Gentsch (2019), a IA ainda está no início do que se pode esperar de seu desenvolvimento. O Gráfico 5 mostra quais métodos subjacentes e tecnologias não mudaram fundamentalmente desde 1950/1960 até hoje. No entanto, devido ao aumento da quantidade de dados e capacidades de computador, os métodos poderiam ser aplicados com mais eficiência e sucesso. Os sistemas ainda estão aprendendo de acordo com certas regras e configurações, padrões e características distintivas.

Gráfico 5 - Etapas de evolução em direção à inteligência artificial



Fonte: Gentsch (2019), adaptado pelo autor (2021)

2.11 Fluxo Ótico

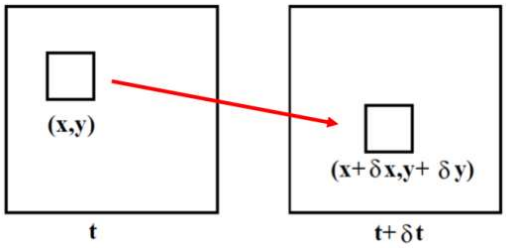
Conforme Moussu (2010), muitos campos científicos, como medicina, biologia ou computacional, exigem o rastreamento de objetos em sequências de vídeo. O problema de fluxo ótico visa encontrar o deslocamento de algumas características ao longo de quadros sucessivos de uma sequência de vídeo. A computação do fluxo ótico usando os melhores métodos é demorada e onerosa. Alguns mercados não suportam altos custos e usam métodos e algoritmos básicos, obtendo resultados ruins. No modelo do fluxo ótico, o deslocamento de alguns pixels deve ser calculado para cada par de imagens, basicamente assumindo que valores de cinza na primeira imagem (posição atual) será semelhante aos valores de cinza na segunda imagem (nova posição). A Figura 30 dá um exemplo de computação de fluxo ótico entre duas imagens.

Figura 30 – Exemplo de fluxo ótico

Fonte: Moussu, 2010

Fluxo ótico é a distribuição de velocidades aparentes de movimento dos padrões de brilho em uma imagem (HORN et la., 1981), em 2D específica quanto cada pixel da imagem se move entre imagens adjacentes, como mostra a Figura 31.

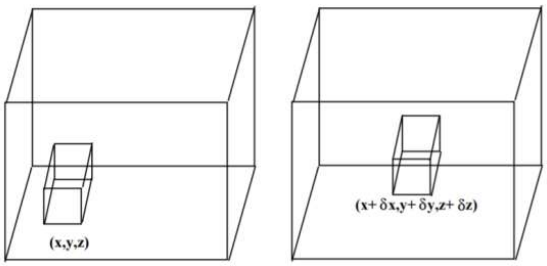
Figura 31 - Representação do movimento do ponto (x, y, t)



Fonte: Barron et la. (2005), adaptado pelo autor (2020)

De acordo com Barron et la. (2005), '2D' especifíca quanto cada pixel da imagem se move entre imagens adjacentes, enquanto em '3D' o quanto cada voxel de volume se move entre volumes adjacentes. Estender o modelo de fluxo ótico de 2D para 3D, possibilitará que análise das imagens em 3D sejam mais detalhadas. Por exemplo, fazer a análise do fluxo ótico do ventrículo esquerdo usando GPU possibilitará que a sua resolução forneça, aos médicos, informações adicionais sobre o estado clínico do paciente. A Figura 32 representa o movimento do objeto em '3D'.

Figura 32 - Movimento 3D do ponto (x, y, z, t)



Fonte: Barron et la. (2005)

De acordo com Gutierrez (*et la*, 1996), seja *E(x(t), y(t), z(t))* a intensidade do voxel na posição (x, y, z) no instante t. Considerando a função estacionária com respeito ao tempo, tem-se e Equação (3):

$$\frac{dE}{dt} = 0 \qquad (3)$$

Se um dado voxel mover-se para a posição $(\delta x, \delta y, \delta z)$ no instante de tempo δt, então a partir da hipótese proposta por Horn e Schunck (1981), pode-se escrever a Equação (4):

$$E(x+\Delta x, y+\Delta y, z+\Delta z, t+\Delta t) = E(x,y,z) + dx\frac{\partial E}{\partial x} + dy\frac{\partial E}{\partial y} + dz\frac{\partial E}{\partial z} + dt\frac{\partial E}{\partial t} + \epsilon \quad (4)$$

onde ϵ contém os termos de ordem superior. Após subtração $E(x,y,z,t)$ em ambos os lados da equação (4) e divisão por δt, tem se a Equaçao (5):

$$\frac{dx\partial E}{dt\partial x} + \frac{dy\partial E}{dt\partial y} + \frac{dz\partial E}{dt\partial z} + \frac{\partial E}{\partial t} + \xi(\delta t) = 0 \quad (5)$$

onde $\xi(\delta t)$ contém os termos de ordem superior.

O limite da Equação (6), quando δt tende a zero é:

$$\frac{dx\partial E}{dt\partial x} + \frac{dy\partial E}{dt\partial y} + \frac{dz\partial E}{dt\partial z} + \frac{\partial E}{\partial t} = 0 \quad (6)$$

Fazendo $v_x = \frac{dx}{dt}$, $v_y = \frac{dy}{dt}$, $v_z = \frac{dz}{dt}$, substituindo na Equação (6) e colocando em uma forma mais compacta, tem-se a Equação (7):

$$\Delta\bar{E} * v = -E_t \quad (7)$$

onde $\Delta\bar{E} = \left[\frac{\partial E}{\partial x}, \frac{\partial E}{\partial y}, \frac{\partial E}{\partial z}\right]$, é o gradiente espacial da imagem 3D, $E_t = \frac{\partial E}{\partial t}$ o

gradiente temporal da imagem 3D e $v = \left[v_x(x,y,z), v_y(x,y,z), v_z(x,y,z)\right]$ é o vetor que define o movimento do voxel.

O conjunto de todos os vetores de velocidade, obtidos por meio das componentes de velocidade nas direções x, y e z, constitui um campo de vetores de velocidade que caracteriza o fluxo ótico para a sequência de imagens.

Em cada voxel, resolvendo-se as equações diferenciais de Euler-Lagrange com variáveis independentes v_x, v_y, v_z, obtém-se a Figura 33.

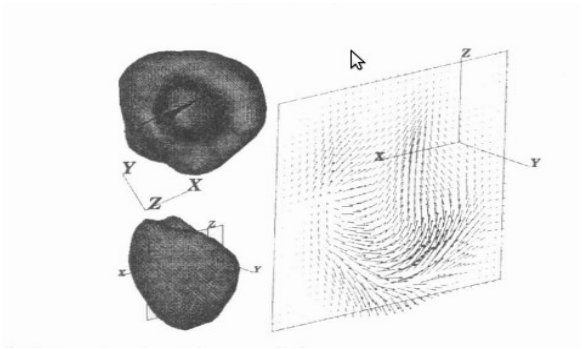
Figura 33 – Equações diferencias de Euler-Lagrange

$$\begin{bmatrix} v_x = \bar{v}_x - \frac{E_x\,(E_x\bar{v}_x + E_y\bar{v}_y + E_z\bar{v}_z + E_t)}{\alpha_m^2 + E_x^2 + E_y^2 + E_z^2} \\ v_y = \bar{v}_y - \frac{E_x\,(E_x\bar{v}_x + E_y\bar{v}_y + E_z\bar{v}_z + E_t)}{\alpha_m^2 + E_x^2 + E_y^2 + E_z^2} \\ v_z = \bar{v}_z - \frac{E_x\,(E_x\bar{v}_x + E_y\bar{v}_y + E_z\bar{v}_z + E_t)}{\alpha_m^2 + E_x^2 + E_y^2 + E_z^2} \end{bmatrix}$$

Fonte: Gutierrez (et la, 1996)

A matriz correspondente a esse sistema de equações é esparsa e de ordem elevada, uma vez que o número de linhas e colunas é três vezes o número de voxels no volume (GUTIERREZ, *et la.*, 1996). Para a visualização do campo de vetores de velocidade, apresenta-se um vetor sobre cada voxel. A magnitude de cada vetor é proporcional à velocidade do voxel e a direção e o sentido indicam o movimento do voxelno espaço, mostrado na Figura 34.

Figura 34 - Corte coronal em um coração normal obtido durante a sístole



Fonte: Gutierrez (et.al., 1996)

Conforme Lacroix (2015) a resolução de uma matriz esparsa usando equações lineares, representada pela matriz ‘M’, é mostrado na Figura 35

Figura 35 – Matriz esparsa

$$\left[\begin{pmatrix} M_{\mathbf{x}_1} & & & \\ & M_{\mathbf{x}_2} & \cdots & 0 \\ & 0 & & \\ & & \cdots & M_{\mathbf{x}_{Card(\Omega)}} \end{pmatrix} - \alpha^2 \mathcal{L}\right] \begin{pmatrix} \mathbf{v}(\mathbf{x}_1) \\ \mathbf{v}(\mathbf{x}_2) \\ \cdots \\ \cdots \\ \mathbf{v}(\mathbf{x}_{Card(\Omega)}) \end{pmatrix} = \begin{pmatrix} b_{\mathbf{x}_1} \\ b_{\mathbf{x}_2} \\ \cdots \\ \cdots \\ b_{\mathbf{x}_{Card(\Omega)}} \end{pmatrix}$$

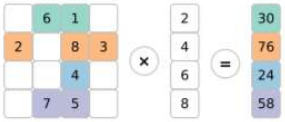
Fonte: Lacroix (2015)

2.12 Multiplicação

2.12.1 Multiplicação de Matriz Esparsa-Vetor

Matriz esparsa existe em várias disciplinas computacionais e, como resultado, métodos para processá-las eficientemente geralmente são críticos para o desempenho de muitos aplicativos. Elas representam o custo dominante em muitos métodos iterativos para resolver sistemas lineares de larga escala e problemas de autovalor que surgem em uma ampla variedade de aplicações científicas e de engenharia (BELL et la., 2009). A Figura 36 é a representação de uma simples matriz esparsa de tamanho 4x4.

Figura 36 – Multiplicação de uma matriz 4x4 por um vetor



Fonte: Chen et la. (2016)

A multiplicação matricial é sem dúvida a operação mais importante em cálculos de matriz esparsa. É um método iterativo para resolver grandes sistemas lineares (Ax = b) e problemas de autovalores (Ax = λx) geralmente requerem centenas, senão milhares de produtos matriz vetor para alcançar convergência (BELL et la., 2009).

2.12.2 Multiplicação de duas matrizes

Conforme Thomas (2017), para a multiplicação de duas matrizes usando um computador de arquitetura CPU, se a dimensão da matriz A é m x k, e matriz B é k x n, então a matriz C será uma matriz de m x n, mostrado na Figura 37.

Figura 37 – Multiplicação de duas matrizes usando CPU

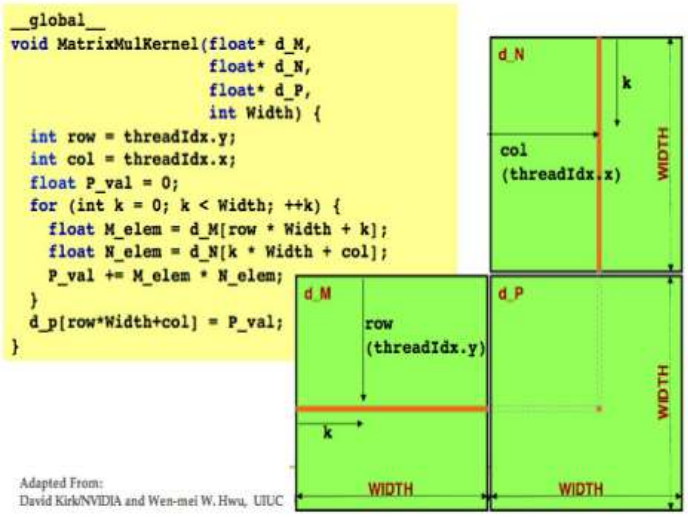
$$c_{ij} = \sum_{t=1}^{k} a_{it} b_{tj} = a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{k1} b_{kj}$$

$$\begin{bmatrix} a_{00} \cdots a_{0j} \cdots a_{0,k-1} \\ \cdots \\ a_{i0} \cdots a_{ij} \cdots a_{i,k-1} \\ \cdots \\ a_{m-1,0} \cdots a_{m-1,j} \cdots a_{m-1,k-1} \end{bmatrix} \bullet \begin{bmatrix} b_{00} \cdots b_{0j} \cdots b_{0,n-1} \\ \cdots \\ b_{i0} \cdots b_{ij} \cdots b_{i,n-1} \\ \cdots \\ b_{k-1,1} \cdots b_{kj} \cdots b_{k-1,n-1} \end{bmatrix} = \begin{bmatrix} c_{00} \cdots c_{1j} \cdots c_{1,n-1} \\ \cdots \\ c_{i0} \cdots c_{ij} \cdots c_{i,n-1} \\ \cdots \\ c_{m-1,0} \cdots c_{mj} \cdots c_{m-1,n-1} \end{bmatrix}$$

Fonte: Thomas (2017)

A multiplicação de duas matrizes é um modelo matemático com propriedades de paralelismo, e dessa forma é possível realizar a mesma multiplicação numa arquitetura GPU, mostrado na Figura 38.

Figura 38 – Multiplicação de duas matrizes usando GPU



Fonte: Thomas (2017)

2.13 Complexidade algoritmica

De acordo com Neapolitan (2015), muitas vezes são usadas abordagens diferentes para resolver o mesmo problema objetivando encontrar algoritmos cada vez mais eficientes.

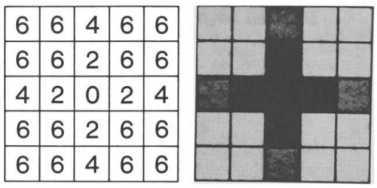
Conforme Roughgarden (2017), a notação Big-O diz respeito às funções T (n) definidas nos inteiros positivos n = 1, 2, ... Ou seja: T (n) quase sempre denotará um limite no tempo de execução de pior caso de um algoritmo, em função do tamanho ‘n’ de entrada. O que significa dizer que T (n) = O (f (n)), para alguns função “canônica” f (n), como n, n log n ou n^2. De acordo com Rich (2008), algoritmos de multiplicação de matrizes simples levam um tempo que é 0 (n^3).

2.14 Imagens Digital

Conforme Sprawls (1995), as imagens devem estar em formato digital para serem processadas por um computador. Uma imagem digital consiste em

uma matriz em que cada elemento, ou pixel, é representado por um valor numérico valor, conforme mostrado na Figura 39.

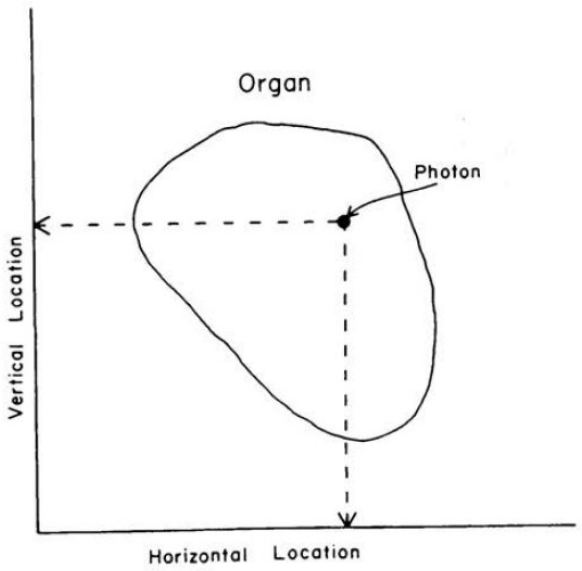
Figura 39 – Representação de uma imagem em matriz



Fonte: Sprawls (1995)

Cada pixel (2D), no modo de matriz, é representado por um local específico (endereço) em uma posição de memória do computador. De acordo com Kaplan et la. (1989), em geral, spect e pet fornecem imagens de alto contraste e baixa resolução de significado funcional, enquanto a radiografia, tomografia computadorizada e ressonância magnética frequentemente retratam anatomia requintada. Cada ponto da matriz terá um valor em função do tipo de imagem, por exemplo, podemos atribuir os valores de pixel de ct entre 0 a 255 e para pixel spect valores entre 256 a 512. Na Figura 40 é mostrado a representação de um órgão.

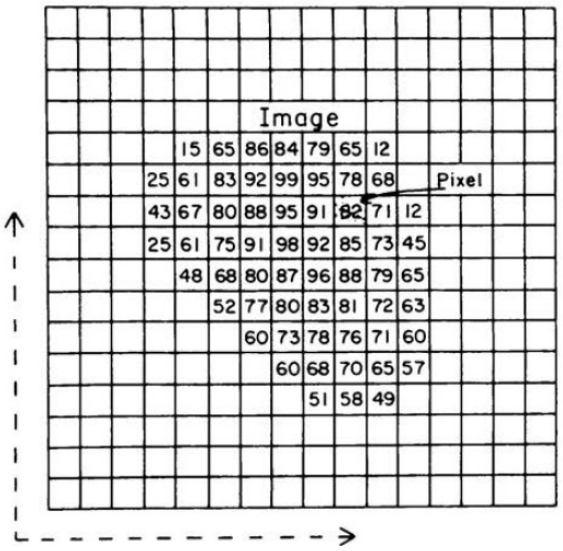
Figura 40 – Representação de um órgão



Fonte: Sprawls (1995)

Na Figura 41 é mostrado a representação matriz da Figura 43.

Figura 41 – Representação matriz da imagem do órgão



Fonte: Sprawls (1995)

3 METODOLOGIADE PESQUISA

De acordo com Gupta et al. (2006), os tipos de pesquisa mais utilizados em Engenharia de Produção, por ordem de utilização e importância, nas publicações nacionais e internacionais são: (1) Pesquisa empírica, (2) Modelagem e Métodos Analíticos, (3) Artigos Conceituais, (4) Ensino em Gestão de Operações e (5) Revisão de Literatura. Na Pesquisa Empírica, as publicações contêm dados de pesquisa de campo. Na modelagem trabalha-se com dados numéricos e simulações matemáticas. Já na análise de Artigos Conceituais os autores realizam uma discussão teórica e na Revisão de Literatura (pesquisa teórica e bibliográfica), a análise para embasamento teórico da pesquisa. Finalmente os artigos de Gestão de Operações apresentam histórico, evolução e modernização do ensino nesta área.

Filippini (1997), Berto e Nakano (2000), classifica as pesquisas, considerando as informações mencionadas na publicação de Gupta et al. (2006), realizando um ajuste simples, da seguinte forma:

- Modelagem e métodos analíticos: Modelagem e Simulaçao;
- Pesquisa Empírica: Survey, Estudo de Caso, Estudo de Campo e Experimento;
- Revisão de Literatura e Artigos Conceituais: Teórico Conceitual.

Esta dissertação utilizou os métodos de pesquisa mais comuns na área de engenharia de produção, respeitando a clareza e objetividade de cada um deles na apresentação dos resultados e compreensão da sequência de elaboração, definida pelo pesquisador.

3.1 Métodos de Pesquisa Utilizados

3.1.1 Pesquisa Bibliográfica

De acordo com Gil (2002), a pesquisa bibliográfica é constituída por livros e artigos científicos, e para a elaboração da pesquisa bibliográfica deste trabalho, buscou-se por materiais publicados por autores relevantes na área, principalmente, da tecnologia da informação.

A pesquisa bibliográfica é aquela que se realiza, segundo Severino (2014):

> *[..] a partir do registro disponível, decorrente de pesquisas anteriores, em documentos impressos, como livros, artigos, teses etc. Utiliza-se de dados ou de categorias teóricas já trabalhados por outros pesquisadores e devidamente registrados. Os textos tornam-se fontes dos temas a serem pesquisados. O pesquisador trabalha a partir das contribuições dos autores dos estudos analíticos constantes dos textos (SEVERINO, 2014, p.76)*

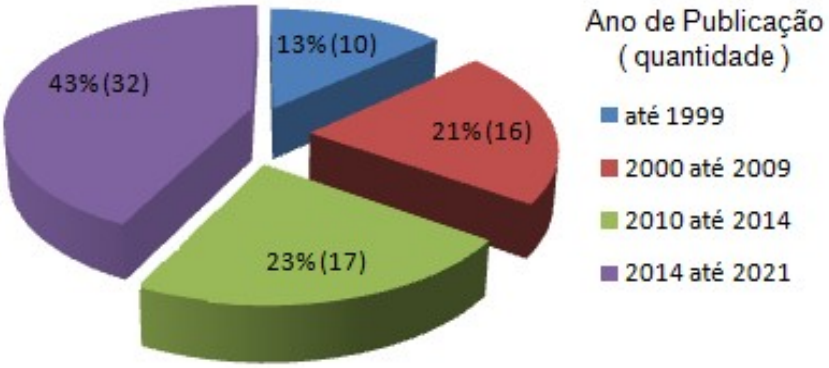
Os principais conceitos tecnológicos usados nesta pesquisa foram embasados em publicações sobre arquitetura de computadores, como *Computer Organization and Architecture: Designing for Performance* (STALLINGS, 2016), sobre sistemas operacionais como Modern Operating Systems (TANENBAUM et la., 2015), estrutura de dados como *Data Structures Using C* (THAREJA, 2014) e CUDA como *CUDA Programming A Developer's Guide to Parallel Computing with GPUs* (COOK, 2013).

Existem muitas publicações com temas relacionados à essa pesquisa, sendo publicadas anualmente. No site de pesquisa da NVIDIA (2021) constam mais de 85 artigos publicados no ano de 2020, que usam a GPU para solucionar questões das áreas de ciência humanas, de saúde e exatas.

Apesar das placas gráficas serem uma tecnologia conhecida desde 1995, o seu uso para outras finalidades, somente foi possível em 2006 com a entrada no mercado comercial das placas gráficas com arquitetura chamada TESLA da NVIDIA e juntamente com a arquitetura TESLA, a NVIDIA apresentou a plataforma CUDA para desenvolvimento de programas em linguagem 'C'.

Dessa forma, o embasamento teórico e científico para a elaboração dessa pesquisa, foram concentrados em publicações mais recentes. Foram usadas 75 publicações entre artigos, sites, livros e dissertações. O Gráfico 6 mostra a distribuição das fontes por ano de publicação.

Gráfico 6 – Distribuição das publicações



Fonte: Autor (2021)

3.1.2 Pesquisa Quantitativa

Segundo Wainer (2007), a pesquisa quantitativa:

> *[..]* vem da tradição das ciências naturais, onde as variáveis observadas são poucas, objetivas e medidas em escalas numéricas. Filosoficamente, a pesquisa quantitativa baseia-se numa visão dita positivista onde:
>
> - as variáveis a serem observadas são consideradas objetivas, isto é, diferentes observadores obterão os mesmos resultados em observações distintas.
> - não há desacordo do que é melhor e o que é pior para os valores dessas variáveis objetivas.
> - medições numéricas são consideradas mais ricas que descrições verbais, pois elas se adequam à manipulação estatística.
>
> (WAINER, 2007, p.6)

3.1.3 Simulação

De acordo com Madachy et la. (2018), a simulação é usada para exercitar modelos com dados de entrada para ver como o sistema funciona, e pode ser usada para explicar o comportamento do sistema, melhorar os sistemas existentes ou para projetar novos sistemas muito complexos para serem analisados por planilhas ou fluxogramas. O software de simulação fornece um meio de especificar a estrutura de um modelo por meio de fórmulas

e executa a tarefa de atualizar automaticamente os valores das variáveis. Os sistemas projetados são frequentemente muito complexos para representar em uma solução de forma fechada, uma equação que resolve um determinado problema em termos de funções e operações matemáticas. Os modelos de simulação são usados para entender como um processo ou sistema funciona, particularmente quando as entradas podem variar e a estrutura de um modelo contém ciclos de feedback e atrasos. Simuladores têm sido usados para ensinar habilidades de tomada de decisão em gerenciamento de projetos, como o quanto um cronograma pode ser reduzido antes que a qualidade do produto seja prejudicada.

Para essa dissertação, foi usada a estrutura de dados matriz para simular digitalmente a imagem spect de menor ordem, e totalmente populada aleatóriamente com números de simples precisão. É objetivo da simulação verificar a viabilidade do uso da arquitetura GPU para o processamento do fluxo ótico e estimar o tempo necessário do processamento. Para simular o tempo de processamento, será usado o algoritmo de multiplicação de duas matrizes, que tem maior complexidade algorítmica do que resolução de equação linear.

3.1.4 Estudo de Caso

Segundo Fia (2020), estudos de caso são métodos de pesquisa ampla sobre um assunto específico, permitindo aprofundar o conhecimento sobre ele, e assim oferecer subsídios para novas investigações sobre a mesma temática. De acordo Yin (2001), estudo de caso:

> *[..]* é a estratégia preferida quando se colocam questões do tipo "como" e "por que", quando o pesquisador tem pouco controle sobre os eventos e quando o foco se encontra em fenômenos contemporâneos inseridos em algum contexto da vida real. Pode-se complementar esses estudos de casos "explanatónos" com dois outros tipos - estudos "exploratórios" e "descritivos". Independentemente do tipo de estudo de caso, os pesquisadores devem ter muito cuidado ao projetar e realizar estudos de casos a fim de superar as tradicionais críticas que se faz ao método. (YIN, 2001, p.17)

Esta dissertação fez uso da resolução de multiplicação de duas matrizes como estudo de caso para aprofundar conhecimento na programação ‘CUDA C’. Também foi usado para verificar a *performance* do estudo de caso para processamento parcial, quando a quantidade de dados for muito maior do que a quantidade de memória principal disponível na GPU.

4 DESENVOLVIMENTO

O desenvolvimento dessa dissertação foi realizado em fases como mostrado no Quadro 3. A dissertação foi desenvolvida para avaliar o tempo de processamento entre uma arquitetura CPU com uma arquitetura GPU, usando como estudo de caso, a multiplicação de duas matrizes. Este Capítulo tem o objetivo de apresentar como foi desenvolvida a pesquisa quantitativa utilizando a programação, a verificação da viabilidade de processamento de grande quantidade de números pela GPU.

Quadro 3 – Fases do desenvolvimento da pesquisa

#	Assunto	Abordagem
1	Entendimento da questão e motivação para a pesquisa.	É apresentado a questão e a abordagem com novas tecnologias.
2	Conhecimento das arquiteturas computacional da pesquisa.	Compreender as arquiteturas CPU e GPU e os seus recursos de hadware e software.
3	Estudo das linguagens de programação C e CUDA C.	Estudar as características das linguagem de programação C e CUDA C.
4	Desenvolvimento e execução do algoritmo para geração de números de precisão simple.	Geração de números aletorios de precisão simple para os processamentos.
5	Desenvolvimento do algoritmo de multiplicação de duas matrizes para CPU.	Usado o resultado da multiplicação como base para comparação
6	Desenvolvimento do algoritmo de multiplicação de duas matrizes para GPU.	Usado o resultado da multiplicação pela GPU para comparação com o resultado da CPU.
7	Desenvolvimento do algoritmo de multiplicação de submatrizes para GPU.	Usado para processamentos de grande quantidade de dados pela GPU.
8	Execução da multiplicação pela CPU.	Geração de metricas da CPU.
9	Execução da multiplicação pela GPU.	Geração de metricas da GPU.
10	Execução da multiplicação pela GPU usando submatrizes.	Geração de metricas da GPU.
11	Desenvolvimento e execução do algoritmo de comparação de resultados	Comparação dos resultados da CPU, GPU e GPU por processamento parcial
12	Formatação dos resultados	Planilhar os resultados
13	Apresentação dos resultados	Apresentar os resultados e realizar as considerações.

Fonte: Autor (2021).

4.1 Pesquisa Quantitativa

Para o desenvolvimento da pesquisa quantitativa, foi necessária a criação de um conjunto de algoritmos que gerassem as métricas quantitativas para o estudo dos resultados do processamento na arquitetura CPU e na arquitetura GPU. Os algoritmos são:

- Algoritmo de geração dos números aleatórios de precisão simples, desenvolvido para criar dois conjuntos de números, e cada conjunto com mais de 67 milhões de números;

- Algoritmo de multiplicação de duas matrizes – para geração das métricas que foram usadas para interpretação da pesquisa. Houve necessidade do desenvolvimento de mais três algoritmos:
  - ✓ multiplicação de duas matrizes pela CPU
  - ✓ multiplicação de duas matrizes pela GPU
  - ✓ multiplicação de submatrizes pela GPU
- Algoritmo de comparação dos resultados – para apoiar a validação e interpretação dos resultados.

De acordo com Miguel et la. (2012), quando o pesquisador constrói seus argumentos baseados em dados estatísticos, usa a pesquisa quantitativa. É uma modalidade (ou um tipo de abordagem), que atua sobre um problema, apoiando-se no teste de uma teoria e baseando-se em variáveis quantificadas em números ou símbolos.

Neste trabalho foi utilizada a simulação como uma forma de Pesquisa Quantitativa, tendo como auxílio a tecnologia dos computadores.

Foram realizadas 5 fases de processamento para cada um dos 3 algoritmos de multiplicação, totalizando 15 processamentos. Os 3 algoritmos usaram o mesmo conjunto de dados para a realização dos processamentos. Na Figura 42 são mostrados os primeiros números das duas matrizes.

Figura 42 – Números de precisão simples

Primeira matriz

0.00125	0.56359	0.19330	0.80874	0.58501
0.09140	0.36445	0.14731	0.16590	0.98853
0.45079	0.35212	0.05704	0.60768	0.78332
0.86224	0.20960	0.77966	0.84365	0.99680
0.00879	0.91879	0.27589	0.27290	0.58791

Segunda matriz

0.28251	0.71984	0.72457	0.21497	0.36625
0.37266	0.52071	0.67858	0.57216	0.76977
0.73205	0.50838	0.58831	0.01392	0.56456
0.14347	0.36586	0.31089	0.24989	0.77804
0.29005	0.07501	0.80715	0.67916	0.36915

Fonte: Autor (2021)

As rodadas foram para multiplicação de duas matrizes de ordem 512 x 512, 1024 x 1024, 2048 x 2048, 4096 x 4096 e 8192 x 8192. A proposta de usar os mesmos conjuntos de números para todos os processamentos para geração das métricas, foi para evitar distorções de comportamento decorrente de processamento de números de precisão diferentes. Outro motivo, foi para facilitar a interpretação dos resultados dos processamentos.

Como a questão de pesquisa deste trabalho foi a mensuração da capacidade e do tempo necessário para o processamento de grande quantidade de números, isso porque existem casos, em diversas áreas da ciência, com necessidades dessas características: alto volume de dados e necessidade de rapidez nos resultados. Um desses caso é o processamento do fluxo ótico por meio de imagens spect, porque cada imagem spect gera mais de 750 mil equações lineares, representadas por matrizes esparsas de ordem bastante elevada.

4.2 Pesquisa das arquiteturas CPU e GPU

A pesquisa sobre as arquiteturas CPU e GPU foi necessária para realizar os experimentos e conseguir as métricas para o estudo de caso.

A arquitetura GPU foi desenvolvida essencialmente para processamento de dados, e quando a GPU precisava realizar atividades de entrada e saída de dados, essas atividades foram realizadas pela arquitetura CPU.

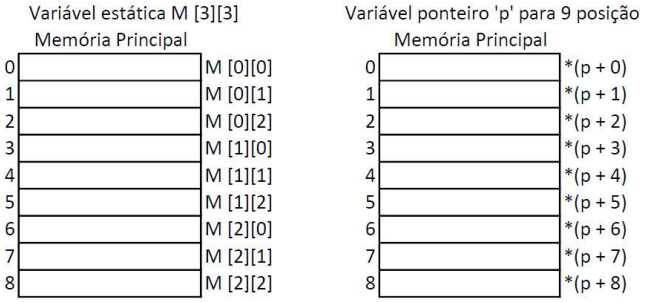
O tamanho da memória principal da GPU é menor do que o tamanho da memória principal da CPU, e quando a quantidade de dados para ser processado é maior que o tamanho da memória principal disponível na GPU, é necessário a transferência parcial dos dados para o processamento pela GPU.

A troca de dados da memória principal da CPU para a memória principal da GPU e vice-versa, são realizadas por meio de barramento de dados, e os tempos de transferência dos dados pelo barramento, podem ser relevantes e devem fazer parte do tempo total para o processamento pela GPU, principalmente para uso do barramento para transferência de alto volume de dados ou também para muitas transferências de dados necessários nos processamentos de submatrizes pela GPU.

Para usar as placas GPU, é possível usar (a) a GPU como monitor de tela da CPU ou (b) adicionar a placa GPU numa CPU que já dispõem de monitor de tela, e em qualquer um dos casos, os procedimentos de entrada e saída de dados com a memória da GPU é realizada pelas unidades de entrada e saída da CPU.

Pelas características da memória principal da GPU e para facilitar a troca de dados entre a memória principal da GPU e da CPU, as variáveis matrizes foram definidas como ponteiros em vez de definir variáveis matrizes estáticas de duas dimensões. Uma outra vantagem de usar ponteiro para representar as variáveis matrizes é que as memorias ponteiros foram alocadas numa outra área da memória principal, o que possibilitou representar matrizes de ordem muito elevadas. Na Figura 43 é mostrada a representação na memória principal de uma variável ‘M’ de ordem ‘3 x 3’, e a representação de uma variável ponteiro ‘p’ com ‘9’ posições de memória principal.

Figura 43 – Mapeamento da variável estática e variável ponteiro



Fonte: Autor (2021)

4.3 Estudo de Caso

O estudo de caso considerado nesta dissertação foi o estudo de caso único. A unicidade estaria na informação base do processamento computacional de um único algoritmo complexo, em dois ambientes diferentes: CPU e GPU. Como idéia final, o algorítmo produziu dois resultados. A análise do tempo e *performance* deste processamento foi o que levou ao resultado do trabalho.

Para verificar se é possível o processamento de grande volume de dados, com baixo tempo de resposta, usando a GPU, foi realizada a multiplicação de duas matrizes de complexidade algoritmo $O(n^{3'})$, como estudo

de caso, em vez de usar a resolução de equação linear que tem complexidade algoritmo $O(n^2)$. A proposta de escolher um estudo de caso com maior complexidade é para avaliar o comportamento do resultado do processamento de algoritmo que exige maior tempo de processamento.

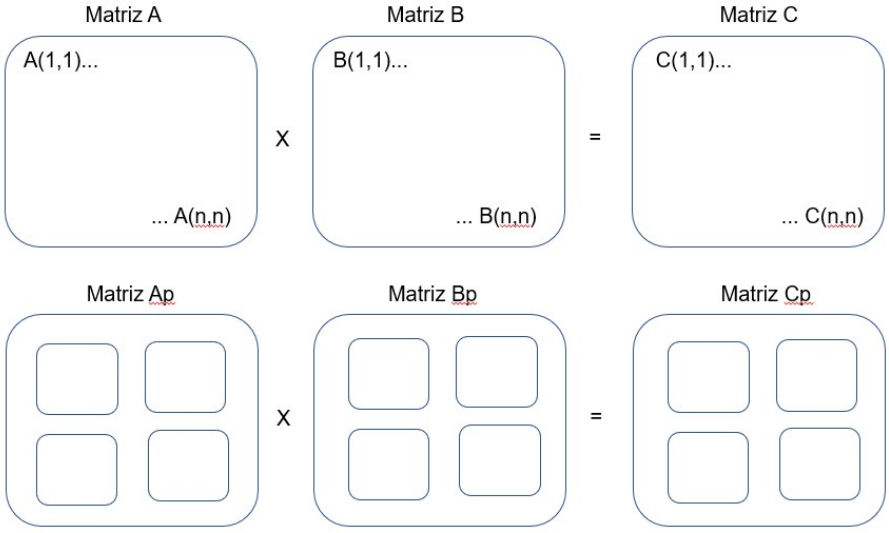
Foram realizados ensaios com multiplicação de duas matrizes na arquitetura GPU e os mesmos ensaios em arquitetura CPU.

O processamento pela CPU é o modelo tradicional de execução e os resultados das multiplicações de duas matrizes por ela, são considerados valores corretos e que foram usados como comparativos com os valores conseguidos nos processamentos pela GPU.

Outra observação do estudo de caso, foi a capacidade de processamento de alto volume de dados pela GPU, porque a memória da GPU é limitada. A sugestão para análise da capacidade de processamento de alto volume foi propor o processamento parcial pela GPU.

Para a avaliação da viabilidade do processamento parcial, foram realizados os mesmos ensaios do processamento da multiplicação de duas matrizes, usando submatrizes e, posteriormente realizando a comparação dos resultados dos ensaios entre processamento parcial e não parcial. Na Figura 44 é mostrada a representação gráfica dos ensaios ‘não parcial’ e ‘parcial’ e a comparação dos resultados.

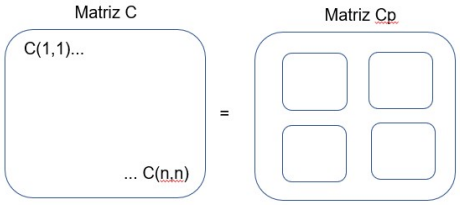
Figura 44 – Representação gráfica dos ensaios



Fonte: Autor (2021)

Na Figura 45 é mostrada que se as matrizes resultantes do processamento parcial e não parcial forem iguais, pode se concluir a viabilidade do processamento parcial.

Figura 45 – Comparação das matrizes resultantes



Fonte: Autor (2021)

4.4 Detalhamento do Desenvolvimento

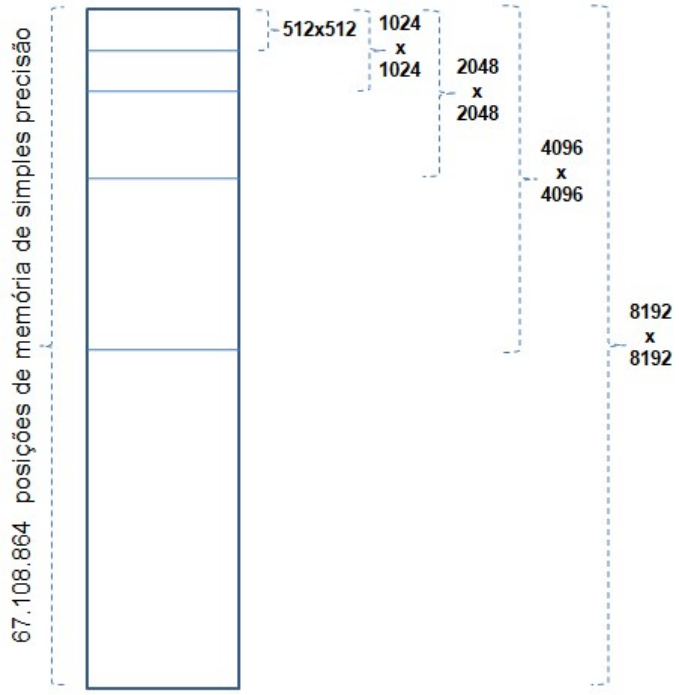
Para a realização dessa pesquisa, foi usando um computador com processador intel(R) Pentium(R) Gold G5400 CPU @ 3.70GHz, memória principal de 8 GB, sistema operacional Windows 10 Pro de 64 bits e processador com base de x64. A placa de vídeo é o NVIDIA GeForce GTX 1050 com 2GB de memória.

Foi usado o software Dev-C++ 5.11, linguagem de programação C, para o desenvolvido de algoritmo para a geração de dois conjuntos de números aleatórios de precisão simples. Para cada conjunto foi gerado 67.108.864 números e representa a estrutura de dados matriz de ordem de 8192 x 8192. Os dois conjuntos de números foram usados como dados de entrada para a multiplicação de duas matrizes usando arquitetura CPU e arquitetura GPU, evitando que conjuntos de números diferentes possam afetar o tempo de resolução.

Para realização dos cincos ensaios com matrizes de ordem 512, 1024, 2048 e 4098, foram usados subconjuntos dos dados de entradas, e para o ensaio com ordem 8192, foi usado o conjunto completo, conforme mostra graficamente a Figura 46.

Os algoritmos para multiplicação de duas matrizes foram desenvolvidos em linguagem de programação CUDA C (para GPU) e linguagem de programação C (para CPU) usando IDE Visual Studio 2015.

Figura 46 – Composição das matrizes



Fonte: Autor (2021)

A Figura 47 mostra os primeiros números da matriz A.

Figura 47 – Primeiros números da matriz A

```
0.00125  0.56359  0.19330  0.80874  0.58501
0.09140  0.36445  0.14731  0.16590  0.98853
0.45079  0.35212  0.05704  0.60768  0.78332
0.86224  0.20960  0.77966  0.84365  0.99680
0.00879  0.91879  0.27589  0.27290  0.58791
```

Fonte: Autor (2021)

A Figura 48 mostra os primeiros números da matriz B.

Figura 48 – Primeiros números da matriz B

```
0.28251  0.71984  0.72457  0.21497  0.36625
0.37266  0.52071  0.67858  0.57216  0.76977
0.73205  0.50838  0.58831  0.01392  0.56456
0.14347  0.36586  0.31089  0.24989  0.77804
0.29005  0.07501  0.80715  0.67916  0.36915
```

Fonte: Autor (2021

Para analisar o comportamento das arquiteturas, CPU e GPU, foram usados os resultados do processamento da multiplicação de duas matrizes de cinco ensaios. Inicialmente foi realizado os cinco ensaios usando a arquitetura CPU com matrizes de ordem 512, 1025, 2048, 4096 e 8192 e depois foi realizado os mesmo cinco ensaios usando a arquitetura GPU.

Na Figura 49 é mostrado o código da rotina em linguagem de programação ‘C’ da multiplicação de duas matrizes numa arquitetura CPU.

Figura 49 – Código em C da multiplicação de duas matrizes

```
11  void matrixMultiplication(float *A, float *B, int N)
12  {
13      int i, j, k;
14
15      for(i=0; i<N; i++){
16          for(j=0;j<N; j++){
17              *(c + i*N + j) = 0.0;
18              for(k=0;k<N;k++){
19                  *(c + i*N + j) += *(A + i*N + k) * *(B + k*N + j);
20              }
21          }
22      }
23  }
```

Fonte: Autor (2020)

De acordo com Rich (2008), para o processo de multiplicação de duas matrizes na CPU é necessário o uso de 3 comandos de repetição (‘for’), e isso representa que esse algoritmo tem complexidade O(n^3), onde ‘n ‘é a dimensão da matriz.

Na Figura 50 é mostrado o código em linguagem de programação CUDA C, da multiplicação de duas matrizes na arquitetura GPU.

Figura 50 - Código em CUDA para multiplicação de duas matrizes

```
11
12  __global__ void matrixMultiplicationKernel(float* A, float* B, float* C, int N) {
13
14      int ROW = blockIdx.y*blockDim.y + threadIdx.y;
15      int COL = blockIdx.x*blockDim.x + threadIdx.x;
16
17      if (ROW < N && COL < N) {
18          float tmpSum = 0;
19          for (int i = 0; i < N; i++) {
20              tmpSum += A[ROW * N + i] * B[i * N + COL];
21          }
22          C[ROW * N + COL] = tmpSum;
23      }
24  }
25
```

Fonte: Autor (2020)

No cálculo do tempo, tanto na CPU como GPU, foi usado somente o tempo do processamento, retirando o tempo de leitura dos números, o tempo de alocação de memória dinâmica na memória principal da CPU e memória principal da GPU, e o tempo de gravação do resultado do processamento.

Dessa forma o tempo do processamento foi efetivamente o tempo da CPU e GPU, eliminando as possíveis ocilações nos processos de leituras e gravações na unidade de saída e entrada.

No caso da GPU, foi adicionado no tempo total, o tempo de transferência de dados de entrada entre a memória principal da CPU para memória principal da GPU e o tempo de transferência dos dados resultante do processamento, da memória GPU para memória CPU, conforme mostra a Figura 51.

Figura 51 – Trecho do programa em CUDA para cálculo do tempo

```
127     printf("Start..\n\n");
128     clock_t begin = clock();
129
130     cudaMemcpy(d_a, a, SIZE * SIZE * sizeof(float), cudaMemcpyHostToDevice);
131     cudaMemcpy(d_b, b, SIZE * SIZE * sizeof(float), cudaMemcpyHostToDevice);
132     cudaMemcpy(d_c, c, SIZE * SIZE * sizeof(float), cudaMemcpyHostToDevice);
133
134     matrixMultiplication(d_a, d_b, d_c, SIZE);
135     cudaDeviceSynchronize();
136
137     cudaMemcpy(c, d_c, SIZE * SIZE * sizeof(float), cudaMemcpyDeviceToHost);
138
139     clock_t end = clock();
```

Fonte: Autor (2020)

Essa adição no tempo da GPU foi necessária, porque a GPU usa as unidades de entrada e saída da CPU. Por causa da limitação física da memória principal da GPU, foi desenvolvido algoritmo para processamento parcial, usando submatrizes. O processamento parcial é importante para avaliar o comportamento das transferências de dados entre a memória da CPU e a memória da GPU. As matrizes foram divididas em quatro submatrizes, que foram multiplicados pela GPU, seguindo as seguintes etapas:

1. Para informar o SO que uma rotina é para ser executada na GPU é necessário que na definição da rotina tenha a palavra ‘_global_’. Na Figura 52 é mostrado a rotina de multiplicação de duas matrizes pela GPU.

Figura 52 – Rotina de multiplicação de duas matrizes usando a GPU

```
18  __global__ void matrixMultiplicationKernel(float* A, float* B, double* C, int N) {
19
20      int ROW = blockIdx.y*blockDim.y + threadIdx.y;
21      int COL = blockIdx.x*blockDim.x + threadIdx.x;
22
23      if (ROW < N && COL < N) {
24          double tmpSum = 0;
25          // each thread computes one element of the block sub-matrix
26          for (int i = 0; i < N; i++) {
27              tmpSum += A[ROW * N + i] * B[i * N + COL];
28          }
29          C[ROW * N + COL] = tmpSum;
30      }
31  }
```

Fonte: Autor (2021)

2. Antes de chamar a rotina para processamento pela GPU, é necessário fazer a definição da quantidade de threads. A Figura 53 mostra a definição de threads antes da execução pela GPU.

Figura 53 – Definições de threads, chamada da execução pela GPU

```
33  void matrixMultiplication(float *A, float *B, double *C, int N) {
34
35      dim3 threadsPerBlock(N, N);
36      dim3 blocksPerGrid(1, 1);
37      if (N*N > 512) {
38          threadsPerBlock.x = 512;
39          threadsPerBlock.y = 512;
40
41          blocksPerGrid.x = BLK;
42          blocksPerGrid.y = BLK;
43      }
44
45      matrixMultiplicationKernel << < blocksPerGrid, 256 >> >(A, B, C, N);
46  }
```

Fonte: Autor (2021)

3. A Figura 54 mostra como é realizado a transferência dos dados das submatrizes da CPU para uma matriz GPU, o processamento pela GPU e a transferência dos dados do resultado do processamento pela GPU para a submatriz da CPU.

Figura 54 – Transferencia de dados entre submatriz CPU e matriz GPU

```
67  void loadmatrix(float *ax, float *bx, double *cx, int ai, int aj, int bi, int bj)
68  {
69      int i, j;
70
71      for (i = 0; i<SIZE2; i++)
72          for (j = 0; j<SIZE2; j++) {
73              *(ax + i*SIZE2 + j) = *(a + (i + ai)*SIZE + j + aj);
74              *(bx + i*SIZE2 + j) = *(b + (i + bi)*SIZE + j + bj);
75          }
76
77      cudaMemcpy(d_a, ax, SIZE2 * SIZE2 * sizeof(float), cudaMemcpyHostToDevice);
78      cudaMemcpy(d_b, bx, SIZE2 * SIZE2 * sizeof(float), cudaMemcpyHostToDevice);
79
80      matrixMultiplication(d_a, d_b, d_c, SIZE2);
81
82      cudaMemcpy(cx, d_c, SIZE2 * SIZE2 * sizeof(double), cudaMemcpyDeviceToHost);
83
84      for (i = 0; i<SIZE2; i++)
85          for (j = 0; j<SIZE2; j++) {
86              *(c + (i+ai)*SIZE + (j+bj)) += *(cx + i*SIZE2 + j);
87          }
88  }
```

Fonte: Autor (2021)

A Figura 55 mostra o trecho programa principal que faz as chamadas para a multiplicação das quatros submatrizes.

Figura 55 – Trecho da multiplicação de submatrizes

```
119     loadmatrix(ax, bx, cx, 0, 0, 0, 0);
120
121     loadmatrix(ax, bx, cx, 0, SIZE2, SIZE2, 0);
122
123     loadmatrix(ax, bx, cx, 0, 0, 0, SIZE2);
124
125     loadmatrix(ax, bx, cx, 0, SIZE2, SIZE2, SIZE2);
126
127     loadmatrix(ax, bx, cx, SIZE2, 0, 0, 0);
128
129     loadmatrix(ax, bx, cx, SIZE2, SIZE2, SIZE2, 0);
130
131     loadmatrix(ax, bx, cx, SIZE2, 0, 0, SIZE2);
132
133     loadmatrix(ax, bx, cx, SIZE2, SIZE2, SIZE2, SIZE2);
```

Fonte: Autor (2021)

5 RESULTADOS E DISCUSSÕES

Os primeiros resultados obtidos foram positivos, demonstrando que a GPU pode ser uma alternativa relevante, para resolução computacional de equações lineares do fluxo ótico, principalmente de ordem elevada, como por exemplo para processamento de images spect.

A operação de multiplicação de duas matrizes tem complexidade algoritmo $O(n^3)$, enquanto a resolução de uma equação linear, representada por matriz, tem complexidade algoritmo $O(n^2)$.

Para o estudo de caso desse trabalho, foi adotada a operação de multiplicação de duas matrizes, porque existe o interesse de explorar a capacidade da GPU usando um algoritmo de maior complexidade.

Foram gerados dois conjuntos de 64.108.864 números de precisão simples, e cada conjunto representou a estrutura de dados de uma variável de 2 dimensão, matriz, de 8192 x 8192 de números de precisão simples.

Pelas características de acesso à memória principal da GPU, a representação das matrizes nos programas CUDA C de multiplicação de duas matrizes pela GPU, foram representadas por ponteiros, ao invés da estrutura de dados do tipo matrizes estáticas.

Para avaliar o comportamento do resultado da multiplicação pela GPU, foi necessário o desenvolvimento do algoritmo de multiplicação de duas matrizes usando a CPU. O resultado da multiplicação da CPU foi usado para validar o resultado do processamento da multiplicação de duas matrizes pela GPU.

Foram realizados cinco ensaios de multiplicação das duas matrizes, usando a arquitetura CPU. No primeiro ensaio ocorreu a multiplicação de matrizes de ordem 512x512, seguido de 1024x1024, 2048x2048, 4096x4096 e o quinto ensaio foi com matrizes de ordem de 8192x8192. Também foram realizados os mesmos cinco ensaios usando a arquitetura GPU.

Além da questão do tempo de processamento da arquitetura GPU, houve a preocupação com a confiabilidade dos resultados das multiplicações realizadas pela GPU.

Como o conjunto de dados usado como entrada nos ensaios realizados pela CPU foi o mesmo usado nos ensaios realizados pela GPU, e como os

resultados das multiplicações pela CPU foram identicas aos resultados das multiplicações pela GPU, é possível concluir que o algoritmo de multiplicação da GPU e o algoritmo da CPU produzem os mesmos resultados.

Na formatação dos resultados, foi verificado que a GPU obteve os resultados com o tempo de processamento significativamente menor do que os resultados conseguidos pela CPU. A Tabela 2 mostra o tempo necessário para os processamentos da multiplicação de duas matrizes pela CPU e GPU.

Tabela 2 - Tempo para resolução dos ensaios

Dimensão	CPU (segundos)	GPU (segundos)
512 x 512	1,381	0,077
1024 x 1024	37,110	0,466
2048 x 2048	351,972 (6 minutos)	2,814
4096 x 4096	2.950,833 (49 minutos)	20,879
8192 x 8192	25.357,016 (7 horas)	162,283 (3 minutos)
16384 x 16384	60 horas - 2 dias e 12 horas (*)	21 minutos(*)

(*) Estimativa do tempo de processamento usando percentual de crescimento do tempo de processamento da ordem de 4096 para 8192

Fonte: Autor (2021)

Os resultados apresentados na Tabela 2, demonstram que quanto maior a quantidade de números a serem processados, maior a diferença de tempo entre a CPU e a GPU. Deve-se então, viabilizar o processamento de grande quantidade de números sempre por meio da GPU, quando isto se tornar possível técnica e cientificamente.

Nos ensaios, ficou envidenciado que os processamentos pela GPU são muito mais eficientes do que os processamentos realizados pela CPU.

Deve-se considerar a possibilidade de análise de como seria o comportamento do processamento pela GPU com grande quantidade de dados.

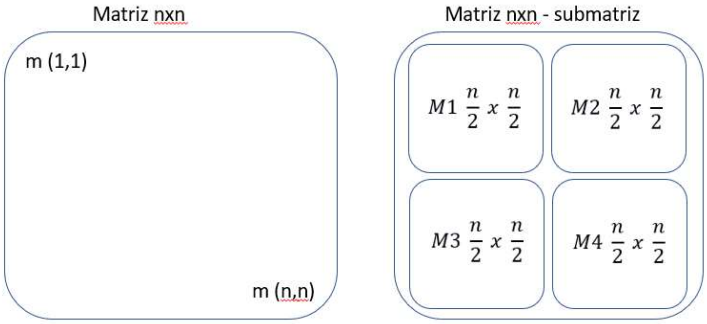
A memória principal da GPU tem limitação de tamanho e isso impede que ela processe grande quantidade de dados em somente uma execução, pois não seria possível carregar todos os dados na memória principal da GPU.

Para solucionar a questão da limitação de memória da GPU, foi usado o processamento parcial, e pôde-se realizar a multiplicação parcial de duas

matrizes, usando submatrizes de menor ordem e consequentemente alocando menos memória principal da GPU. O resultado de todos os processamentos parciais das submatrizes deu origem a matriz resultante.

Para facilitar a análise dos resultados dos processamentos parciais, as matrizes foram subdivididas em quatro submatrizes, conforme mostra a Figura 56.

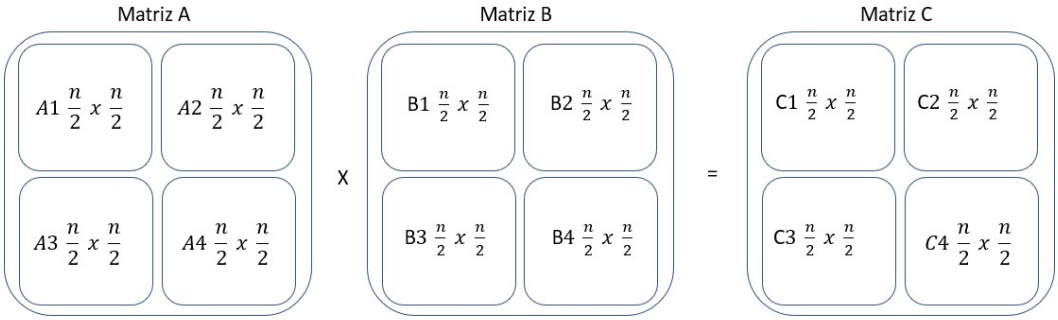
Figura 56 – Representação da matriz e submatrizes



Fonte: Autor (2021)

Para validar o resultado do processamento parcial, foram usados os mesmos conjuntso de números das matrizes de entrada as matrizes da multiplicação de duas matrizes pela GPU. Dessa forma, foram realizados cinco ensaios para a matriz de ordem de 512, 1024, 2048, 4096 e 8192, usando submatrizes de ordem 256, 512, 1024, 2048 e 4096 respectivamente. Na Figura 57 é mostrada a representação gráfica da multiplicação das submatrizes.

Figura 57 – Representação gráfica da multiplicação de submatrizes



Fonte: Autor (2021)

A Figura 58 mostra a resolução matemática para calcular a matriz ‘C’, resultante da multiplicação de duas matrizes usando submatrizes.

Figura 58 – Resolução matemática usando submatrizes

$$\begin{bmatrix} C1 = A1 \times B1 + A2 \times B3 \\ C2 = A1 \times B2 + A2 \times B4 \\ C3 = A3 \times B1 + A4 \times B3 \\ C4 = A3 \times B2 + A4 \times B4 \end{bmatrix}$$

Fonte: Autor (2021)

A arquitetura GPU foi desenhada para processamento de dados e todas as necessidades de entrada e saída de dados são realizadas pela arquitetura CPU, e a transferência dos dados entre a memória principal da CPU e a memória principal da GPU dá-se por meio de barramento de transferência de dados.

De acordo com Nvidia (2013), o modelo de programação CUDA também assume que o *host* e o dispositivo, mantêm seus próprios espaços de memória separados em DRAM, conhecidos como memória *host*, localizada na CPU e memória *device*, localizada na GPU. Dessa forma a execução de um programa CUDA segue conforme mostrado na Tabela 3.

Tabela 3 – Algoritmo para processamento na GPU

#	Algoritmo
1	Inicio do programa
2	Leitura dos dados para a memória da CPU
3	Alocação de memória na GPU
4	Transferencia dos dados da CPU para GPU
5	Chamada do processamento na GPU
6	Transferencia do resultado da GPU para CPU
7	Desalocação da memória da GPU
8	Finalização do programa

Fonte: Autor (2020)

Foram então realizados os cinco ensaios para multiplicação de duas matrizes usando submatrizes. A Tabela 4 mostra o tempo necessário para os processamentos da multiplicação de duas matrizes pela GPU e pelo processamento parcial de submatrizes pela GPU.

Tabela 4 - Tempo em segundos para resolução dos ensaios GPU e GPU (submatrizes)

Dimensão	GPU (segundos)	GPU - Submatrizes (segundos)
512 x 512	0,077	0,112
1024 x 1024	0,466	0,557
2048 x 2048	2,814	3,244
4096 x 4096	20,879	22,559
8192 x 8192	162,283 (3 minutos)	168,915 (3 minutos)
16384 x 16384	21 minutos(*)	21 minutos(*)

(*) Estimativa do tempo de processamento usando percentual de crescimento do tempo de processamento da ordem de 4096 para 8192

Fonte: Autor (2021)

Para o processamento parcial, houve necessidade de usar mais vezes o barramento de dados entre a CPU e GPU, e mesmo assim, conforme mostra a Tabela 4, o tempo adicional para as transferências de dados entre as memórias principal foi percebido somente para tansferencia de poucos dados.

Baseado nessa observação, mesmo usando mais vezes o barramento de dados entre as memórias da CPU e GPU, o tempo de transferência de dados do barramento comprometeu, muito pouco, o tempo total do processamento.

Com a GPU é possível realizar o processamento do fluxo ótico com um tempo aceitável já que resolução de equações lineares são de menor complexidade em relação a resolução de multiplicação de matrizes de ordem elevadas.

6 CONSIDERAÇÕES FINAIS

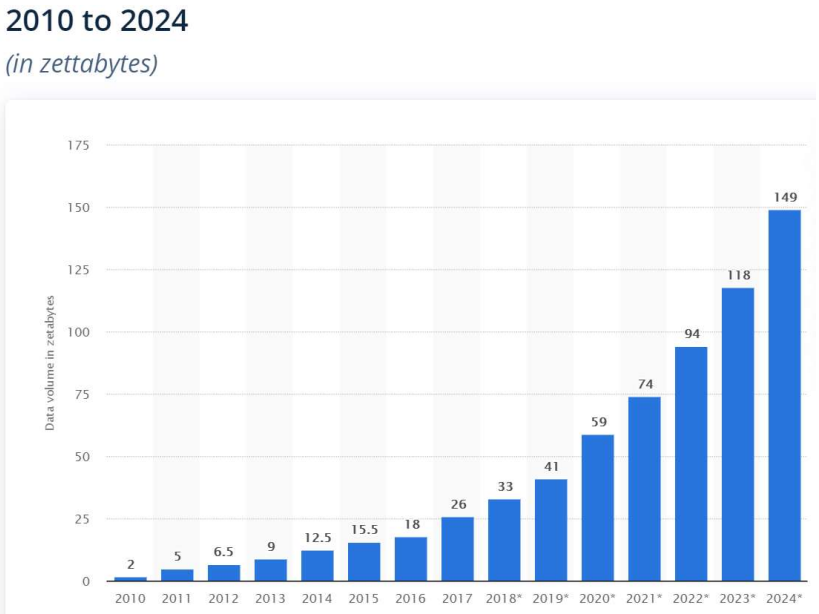
Os primeiros resultados são bastante interessantes e relevantes, porque mostra a capacidade de processamento da GPU e poderá alavancar novas linhas de pesquisas em hardware e software.

Essa pesquisa demonstrou que os problemas que podem ser solucionados com execuções paralelas poderão conseguir soluções em tempo incrivelmente menor usando a GPU em vez de usar a CPU.

Será possível realizar processamento de grande quantidade de dados, com baixo tempo de resolução usando estrutura computacional simples, de pouco custo financeira e que antes da GPU, esse processamento seria somente possível com estrutura computacional de custo financeiro bastante elevado.

De acordo com Holst (2020), a quantidade total de dados criados, capturados, copiados e consumidos no mundo deve aumentar rapidamente, atingindo 59 zetabytes em 2020, conforme mostra o Gráfico 7.

Gráfico 7 – Quantidade de dados no mundo.



Fonte: Holst (2020).

As imagens spect fazem parte dessa grande quantidade de dados, e o uso da GPU possibilita uma nova opção tecnológica para processamento das imagens, de baixo custo financeiro.

Os ensaios das pesquisas também mostraram que o uso frequente de barramento de dados, por causa da necessidade de processamento parcial de grande quantidade de números, não gerou nenhum tempo adicional significativo no processamento.

A demonstração de que o processamento pela GPU é significativamente mais rapidas do que o processamento pela CPU abre várias novas pesquisas nas áreas de ciências que exige rápidas tomadas de decisões.

Na área médica, a rápida identificação de que o fluxo sanguíneo do ventrículo esquerdo do coração está com comportamento irregular, usando o processo de fluxo ótico, poderá ajudar o médico a receitar procedimentos mais adequado para o paciente evitando, por exemplo, que o paciente tenha um AVC. Seria bastante custoso montar uma estrutura computacional usando arquitetura CPU que identificasse automaticamente e rapidamente irregularidade no fluxo sanguíneo. O uso do fluxo ótico para imagem spect gera mais de 750 mil equações lineares, e essas equações podem ser representada por uma matriz esparsa de ordem bastante elevada. Nos ensaios dessa pesquisa, o processamento da multiplicação de duas matrizes de ordem 16.384 x 16.384 demorará mais de dois dias se a execução usar a arquitetura CPU, e para a mesma multiplicação realizada pela arquitetura GPU a resolução estará disponível em somente 21 minutos.

Outro exemplo na área médica, está relacionado as câncer de mama, que conforme Sanders (2010), o número de pessoas que foram afetadas pelo câncer cresceu ao longo dos últimos 20 anos. A mamografia, uma das melhores técnicas para a detecção precoce decâncer de mama, tem várias limitações. Para análise da mamografia, duas ou mais imagens precisam ser tiradas, e o filme precisa ser revelado e lido por um médico habilidoso para identificar potenciais tumores. A proposta para desenvolver um modelo tridimensional para imagem de ultrassom, exige uma computação considerada proibitivamente demorado e onerosa para uso prático. Com a adoção da GPU / ‘CUDA C’, já é possível, por exemplo, processar 35 GB de dados, gerados por uma varredura de 15 minutos de ultrasom, em uma imagem tridimensional, em apenas 20 minutos.

Um dos trabalhos na área de engenharia, diz respeito ao objetivo de melhoria na previsão da dispersão atmosférica de radionuclídeos (DAR), com o

desenvolvimento de um novo algoritmo baseado em GPU, que está sendo implementado usando CUDA e a linguagem de programação'C' (PINHEIRO, 2017).

Ainda é um grande desafio processar dados em ambiente computacional Big Data, porque o processamento pode envolver milhões de dados estruturados e não estruturados, com custos financeiros bastante elevados. A possibilidade de usar GPU para processamento de Big Data deverá oferecer uma alternativa para processamento de Big Data com tempo de processamento semelhante aos usados pelos grandes modelos computacionais, mas com custo financeiro significativamente menor, permitindo que mais empresas, pequenas e médias, usem o Big Data.

Uma outra área que a GPU deverá alavancar muitas pesquisas é em relação a área de Inteligência Artificial – IA.

Sistemas especialistas usando inteligencia artificial acessando o Big Data, poderão, por exemplo, analisar várias imagens spectde fluxo ótico, e ensinar futuros equipamentos inteligentes como por exemplo o desenvolvimento de equipamento de tomografia inteligentes para apoiar as decisões dos médicos.

O uso da GPU para reconhecimento facial irá possibilitar que câmeras inteligentes possam ser usadasparacontrole de acesso físico e/ou sistêmico com baixo custo financeiro e baixo tempo de resposta.

Os bons resultados conseguidos nessa pesquisacom a GPU, foram alcançados usando um desktop, com a GPU sendo a placa de vídeo usado para gerenciar as atividades de telas da CPU e ao mesmo tempo servindo como equipamento para processamento de dados, no sistema operacional Windows desktop. Essa concorrência de atividades deve ter causado o aumentado nostemposde respostas dos processamentos dos ensaios.

Uma futura linha de pesquisa é o estudo do comportamento no processamento de dados em que a arquitetura CPU possui uma placa de vídeo dedicado para as atividades da CPU, e um ou mais arquitetura GPU dedicada para processamento de dados, usando sistemas operacionais de servidor, como Linux.

A GPU poderá ser usada para processamento de algoritmos paralelos que envolvam milhões de números, e um ponto de atenção para

processamento de milhões de dados pela GPU é o barramento de dados que é usado para a transferência dos dados da memória principal da CPU para memória principal da GPU, para o processamento pela GPU, e a transferência dos dados para a memória da CPU dos resultados do processamento. Nessa pesquisa, o tempo gasto para as transferências de dados entre as memórias principais foi significativamente baixo.

A evolução do equipamento computacional pode gerar outras pesquisas, como por exemplo, o estudo do comportamento dos resultados do processamento de dados de grande quantidade de números usando arquitetura totalmente GPU, desenvolvimento de equipamento GPU dedicado,como tomógrafos, equipamentos de identificação facial, simuladores, equipamentos de previsão de tempo, estruturas para Big Data, e muitos outros equipamentos que trabalham com alto volume de dados.

REFERÊNCIAS

AAMODT, T.; FUNG, W. W.; ROGERS, T. **General-Purpose Graphics Processor Architecture**. Morgan & Claypool, 2018.

ALLALEN, M.; CODREANU, V.; CODREANU A.; LLIEVA-LITOVA, N.; GRAY, A.; SJÖSTRÖM, A.; WEINBERG, V. **Best Practice Guide – GPGPU**. Disponível no link https://www.researchgate.net/publication/314118681_Best_Practice_Guide_-_GPGPU, 2017, acessado em maio/2020.

ALVES, L. **Saiba as principais causas de mortes no Brasil e nos EUA.** Disponível em https://www.poder360.com.br/internacional/saiba-as-principais-causas-de-mortes-no-brasil-e-nos-eua/, 2020, acessado em março/2021.

BARDINET, E.; COHEN, L. D.; AYACHE, N. **Tracking and Motion Analysis of the Left Ventricle with Deformable Superquadrics.** Medical Image Analysis, 1996; 1(2): p.129 – 149.

BARRON, J. L.; THACKER, N.A. **Tutorial: Computing 2D and 3D Optical Flow.** Imaging Science and Biomedical Engineering Division, 2005.

BELL, N.; GARLAND, M. **Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors.**ACM, 2009.

BERTO, R.M.V.S.; NAKANO, D.N. **A Produção Cientifica nos Anais do Encontro Nacional de Engenharia de Produção: Um Levantamento de Métodos e Tipos de Pesquisa.** Produção, 2000.

CAULFIELD, B. **What's the Difference Between a CPU and a GPU?**. Disponível no https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/, 2009, acessadoemoutubro/2020.

CAUVIN, J. C.; BOIRE, J. Y.; ZANCA, M.; BONNY, J. M.; MAUBLAN, J.; VEYRE, A. **3D Modeling in myocardial 201TL SPECT. Computerized Medical Imaging and Graphics**, 1993;17 (4):p.345–350.

CHEN, S.; FANG, J.; CHEN, D.; XU, C.; WANG, Z.**Optimizing Sparse Matrix-Vector Multiplication on Emerging Many-Core Architectures.** arXiv:1805.11938v1 [cs.MS], 2016.

CHENG, J.;GROSSMAN, M.; MCKERCHER, T. **Professional CUDA® C Programming**. John Wiley & Sons, Inc, 2014.

COOK, S. **CUDA Programming A Developer's Guide to Parallel Computing with GPUs**. Elsevier Inc, 2013.

COREA, F. **An Introduction to Data**, © Springer Nature Switzerland AG 2019.

DIETRICH, D.; HELLER, B.; YANG, B.**Data Science & Big Data Analytics.** John Wiley & Sons, Inc., 2015.

ENGLANDER, I.**THE ARCHITECTURE OF COMPUTER HARDWARE, SYSTEMS SOFTWARE, &NETWORKING**.John Wiley& Sons, Inc, 2014.

FABER, T. L.; COOKE, C. D.; FOLKS, R. D.; VANSANT, J. P.; NICHOLS, K. J.; DEPUEY, E. G.; PETTIGREW, R. I.; GARCIA, E. V. **Left ventricular function and perfusion from gated SPECT perfusion images: an integrated method.** Journal of Nuclear Medicine, 1999; 40 (4): 650-659.

FABER, T. L.; STOKELY, E. M.; PESHOCK, R. M.; CORBETT, J. R. **A model-based four-dimensional left ventricular surface detector.** IEEE Transaction on Medical Imaging, 1991; 10 (3): 321 – 329.

FARBER, R. CUDA **Application Design and Development.** Elsevier Inc, 2011.

FIA – Fundação Instituto de Administração. **Estudos de Caso: O que são, Exemplos e Como Fazer para TCC.** Disponível no link: https://fia.com.br/blog/ estudos-de-caso/, 2020, acessado em jan/2021.

FILIPPINI, R. **Operations Management Research: Some Refletions On Evolution, Models and Empirical Studies in OM**. Internacional Journal of Operactions and Production Management, 1997.

GARCIA, E. V.; VAN TRAIN, K. F.; MADDAHI, J.; PRIGENT, F.; FRIEDMAN, J.; AREEDA, J.; WAXMAN, A.; BERMAN, D.S. **Quantification of rotational thallium-201 myocardial tomography.** Journal of Nuclear Medicine, 1985; 26 (1): 17-26.

GARCIA, E. V.; COOKE, C. D.; FOLKS, R. D.; SANTANA, C. A.; KRAWCZYNSKA, E. G.; DE BRAAL, L.; EZQUERRA, N. F. **Diagnostic Performance of an Expert System for the Interpretation of Myocardial Perfusion SPECT Studies.** J Nucl Med, 2001; 42 (8):1185–1191.

GENTSCH, P. **AI in MARKETING, SALES and SERVICE**, Springer Nature Switzerland AG, 2019.

GERMANO, G.; KAVANAGH, P. B.; CHEN, J.; WAECHTER, P.; SU, H.T.; KIAT, H.; BERMAN, D. S. **Operator-less processing of myocardial perfusion SPECT studies.**Journalof Nuclear Medicine, 1995; 36 (11): 2127 – 2132.

GIL, A. C. **Como Elaborar Projetos de Pesquisa.**EDITORAATLAS S.A. 2002.

GLASKOWSKY, P. **NVIDIA's Fermi: The First Complete GPU Computing Architecture.** A white paper of NVIDIA Corporation, Creative Commons, Copyright ©. 2009.

GLASS, R.; CALLAHAN, S.**THE BIG DATA - DRIVEN BUSINESS**, Wiley, 2015

GUPTA, S. et la. **Empirical Research Published in Production and Operations Management (1992-2005): Trends and Future Research Directions.** Production and Operations Management, 2006

GUTIERREZ, M. A.; FURUIE, S. S.; MOURA, L.; MENEGHETTI, J. C.; ALENS N. **Quantificação do Movimento 3D do Miocárdio em Imagens de Medicina Nuclear.**RBE – Caderno de EngenhariaBiomédica, v12, p3, p.155-167, out 1996.

GUTIERREZ, M. A.; REBELO, M. S.; FURUIE, S. S.; MENEGHETTI, J. C. **Automatic quantification of three-dimensional kinetic energy in gated myocardial perfusion single-photon-emission computerized tomography improved by amultiresolution technique.** Journal of Electronic Imaging, 2003; 12(1), p. 118 – 124.

HOLST, A. **Amount of information globally 2010-2024.**Disponível no link: https://www.statista.com/statistics/871513/worldwide-data-created/, acessadoemjan/2021.

HORN, B.K.P.; SCHUNCK, B. G. **Determining Optical Flow Artificial Intelligence 17.**Article in Artificial Intelligence(1981)

JEWELL, D. et al. **Performance and Capacity Implications for Big Data.** Copyright IBM Corp. 2014.

KAPLAN, I. L.; SWAYNE, L. C. **Composite SPECT-CT Images: Technique and Potential Applications in Chest and Abdominal Imaging.** American Roentgen Ray Society, 1989.

LACROIX, R. **3D Optical flow analysis of a pulsed contrast agent in the bloodstream. Application to virtual angiography and Magnetic Particle Imaging.** Medical Imaging. Télécom Bretagne; Université, de Bretagne Occidentale, 2015

LEE, B.; SON, B.H.; CHOI,H.J.; HWANG, H.G.;KIM, H.Y.; CHOI,H.K. **Modeling of myocardial contractility using parameterized super-quadric SPECT images.** Computerized Medical Imaging and Graphics, 2006; 30(1): 43-51.

LI, X.; ZHANG, G.; LI, K.; ZHENG W. **Big Data Principles and Paradigms, Chapter 4 - Deep Learning and Its Parallelization.** Elsevier Inc., 2016.

MADACHY, R. J.; HOUSTON, D. X. **What Every Engineer Should Know About Modeling and Simulation.** Taylor & Francis Group, LLC, 2018

MAES, F.; ROBBEN, D.; VANDERMEULEN, D.; SUETENS, P. **Artificial Intelligence in Medical Imaging - The Role of Medical Image Computing and Machine Learning in Healthcare**. Springer, 2019.

MEYERING, W. I. **Quantificação de Movimento Cardíaco em Imagens Dinâmicas [Tese].** São Paulo: Universidade de São Paulo, ESUSP, 2002.

MIGUEL, P. A. C. et al**. Metodologia de Pesquisa em Engenharia de Produção e Gestão de Operações.** Elsevier, 2012.

MILLS, M. **How the NVIDIA Architecture Has Evolvedf rom Tesla to Turing.** Disponível o link https://itigic.com/how-nvidia-architecture-evolved-from-tesla-to-turing/ july/2020, acessado em outubro/2020.

MOUSSU, C., **GPU based real-time optical ow computation.** Imperial College London, 2010.

MURAMATSU, T.; MATSUMOTO, K.; NISHIMURA, S. **Efficacy of the phase images in Fourier analysis using gated cardiac pool-SPECT for determining the indication for cardiac resynchronization therapy.** Circulation Journal, 2005; 69 (12): 1521-1526.

NEAPOLITAN, R.E.**Foundations of Algorithms.** Jones & Bartlett Learning LLC, 2015.

NICKOLLS, J.; KIRK, D.**Graphics and Computing GPUs.** Disponível em https://www.comp.nus.edu.sg/~cs2100/2_resources/AppendixA_Graphics_and_Computing_GPUs.pdf, acessado em maio/2020.

NVIDIA. **CUDA C PROGRAMMING GUIDE.** NVIDIA Corporation, 2013.

NVIDIA. **High Performance Computing with CUDA**. User Group Conference, 2009.

NVIDIA. **Research**. Disponível em https://research.nvidia.com/publications, acessado em janeiro/2021.

PINHEIRO, A. L. S. **Modelo Computacional Paralelo Baseado em GPU para Cálculo do Campo de Vento de um Sistema de Dispersão Atmosférica de Radionuclídeos.**Tese (DoutoradoemEngenharia Nuclear) - COPPE/UFRJ, 2017

PUENTES, J.; GARREAU, M.; ROUX, C.; COATRIEUX, J.L. **Towards dynamic cardiac scenes interpretation based on spatial-temporal knowledge.** Artificial Intelligence in Medicine, 2000; 19 (2): 155-183.

RANSCHAERT, E. R.; MOROZOV, S.; ALGRA, P. R. **Artificial Intelligence in Medical Imaging - Opportunities, Applications and Risks.** Springer Nature Switzerland AG, 2019.

RICH, E., **Automata, Computability and Complexity Theory and Applications.** PearsonEducation , Inc, 2008.

ROSE, C. **CUDA Succinctly.**Syncfusion, Inic, 2014.

ROUGHGARDEN, T. **Algorithms Illuminated Part 1: The Basic**. Soundlikeyourself Publishing, LLC, 2017.

SANDERS, J.; KANDROT, E. **CUDA by Example An Introduction to General-Purpose GPU Programming.**Addison-Wesley, 2010.

SEVERINO, A. J. **Metodologia de Trabalho Científico**. Cortez Editora, 2014

SHIEL JR, W.**Medical Definition of SPECT.**Disponível no link https://www.medicinenet.com/spect/definition.htm, acessado em jan/2021.

SILBERSCHATZ, A.; GALVIN, P. B.; GAGNE, G. **Operation System Concepts - Tenth Edition.** John Wiley & Sons, Inc, 2018.

SILVA, R. **What Pixels Are and What They Mean for TV Viewing.** Disponível no link https://www.lifewire.com/what-is-a-pixel-1846929, 2020, acessado em jan/2021.

SINNEN, O. **Task Scheduling For Parallel Systems.** John Wiley & Sons, Inc, 2007.

SOUTHARD, D. **Really Fast Introduction to CUDA and CUDA C.** NVIDIA Corporation, 2013.

SPRAWLS, P. **PHYSICAL PRINCIPLES of MEDICAL IMAGING, Second Edition.** Copyright 1995 by Perry Sprawls and Associates, Inc

STALLINGS W. **Computer organization and architecture : designing for performance — Tenth edition**. Person, 2016.

STEPHENSON, D. **BIG DATA DEMUSTIFIED.** Pearson Education Limited, 2018.

SUGANYA, R.; RAJARAM, S.; ABDUILLAH, A. S. **Big Data in Medical Image Processing.**CRC Press is an imprint of Taylor &Franci, 2018.

TAI, X.C.; MORKEN, K.; LYSAKER, M.; LIE, K.A. **(Eds.) Scale Space and Variational Methods in Computer Vision**. Second International Conference, Voss, Norway, June 2009.

TAMAKI, N.; YONEKURA, Y.; MUKAI, T.; KODAMA, S.; KADOTA, K.; KAMBARA, H.; KAWAI, C.; TORIZUKA, K. **Stress thalluim-201 transaxial emission computed tomography: quantitative versus qualitative analysis for evaluation of coronary artery disease.** Journal Am. Coll. Cardiology, 1984; 4 (6): 1213-1221.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems - Fourth Edition.** Pearson Education, Inc, 2015.

THAREJA, R. **Data Structures Using C, Second Edition**. Oxford University Press, 2014.

THOMAS, M. **CUDA Matrix-Matrix Multiplication.**San Diego StateUniversity, 2017.

WAINER, J. **Métodos de pesquisa quantitativa e qualitativa para a Ciência da Computação.**Atualização em informática 2007. Sociedade Brasileira de

Computação e Editora PUC-Rio, 2007

WILT, N. **The CUDA Handbook**. Pearson Education, Inc, 2013.

YADIN, A. **Computer Systems Architecture.**Taylor& Francis Group, LLC, 2016.

YIN, R. K. **Estudo de caso: Planejamento e Métodos.**Bookman, 2001.