Análise e Testes de Algoritmos Utilizando Sistemas de Arquiteturas Híbridas CPU/GPU

Danilo, Silva Maciel
Univem - Marília, Brasil
danilo.maciel@univem.edu.br
Mauricio Duarte
maur.duarte@gmail.com
Univem / FATEC

***Abstract* — Há pouco tempo atrás, a principal utilização de placas de vídeos (GPU) era para processamento de jogos eletrônicos e processamento de vídeos, tornando a GPU uma das partes mais ociosas de um computador, pelos menos para a maioria dos usuários. Mas nos últimos anos tem se observado um aumento muito grande no campo de pesquisa e utilização de GPUs fora deste contexto, assim como a CPU, a GPU também evoluiu, mas de uma forma bem mais rápida. Com a evolução destes dois componentes vem a ideia de coloca-los para trabalharem juntos, este trabalho busca tentar demonstrar através de algoritmos que exijam um processamento pesado da CPU, a utilização do conjunto CPU e GPU, comprovando através de testes o eventual ganho ou não de performance na execução destes algoritmos e também sua eficiência**

Keywords—CPU, GPU; algoritmos; performance; eficiência.

I. INTRODUÇÃO

A computação de alto desempenho exige sempre um hardware potente e acabam, de certa forma, embasando pesquisas na área de desenvolvimento de hardware que a cada geração ser torna mais complexo e poderoso, gerando uma evolução constante de componentes que em sua nova versão, conseguem superar a geração passada com muita facilidade, evolução esta que na grande maioria das vezes é alavancada por componentes como a CPU (Central Processing Unit ou Unidade Central de Processamento) e a GPU (Graphics Processing Unit ou Unidade de Processamento Gráfico) que aumentaram consideravelmente suas capacidades computacionais nos últimos anos. Facilmente encontramos Processadores (CPU) de até 8 núcleos físicos acessíveis a usuários comuns, mas por outro lado acabou ficando estagnada em relação a velocidade de seu clock, e os fabricantes acabaram contornando isso, introduzindo mais núcleos dentro de um único processador.

A GPU por sua vez, que antes era utilizada somente como um simples processador gráfico, evoluiu para um coprocessador paralelo, que é capaz de executar milhares de operações simultaneamente resultando em uma capacidade computacional enorme, que na grande maioria das vezes supera o poder de processamento de muitas CPUs que se encontram atualmente no mercado. Com o avanço que as tecnologias GPUs sofreram, vários benefícios ficaram disponíveis, entre eles pode-se citar as unidades programáveis, as GPUS possuem milhares de unidades programáveis (Stream Processors ou Cuda Cores) que conseguem realizar cálculos simultaneamente, como por exemplo, a GPU da ATI R9 290x que possui 2816 Stream Processors ou a Nvidia gtx980 que possui 2048 Cuda Cores. A vantagem da GPU também está no fato do surgimento das APIS (Application Programming Interface) que possuem suporte ao processamento em paralelo,

possibilitando aproveitar a capacidade computacional das GPUs como um todo. Algumas dessas APIS que se destacam são a OMP (Open Multi Processing) e MPI (Message-Passing Interface), são utilizadas para trabalhar com paralelismo de tarefas utilizando a CPU, CUDA (Compute Unified Device Architecture) e OpenCL (Open Computing Language) para trabalhar com paralelismo utilizando a GPU.

Houve também crescimento no desenvolvimento de algoritmos que trabalham de forma híbrida, utilizando CPU e GPU tirando proveito do paralelismo entre estas arquiteturas, apesar de suas arquiteturas serem distintas. Com este tipo de algoritmo é possível aproveitar o que cada arquitetura oferece, pois, a programação híbrida torna possível que conjuntos de instruções possam ser executados na arquitetura que melhor se adapte, desta forma se ganha poder de processamento a um custo acessível.

O objetivo do presente trabalho será demonstrar as vantagens e desvantagens da utilização de algoritmos em arquiteturas híbridas, para isso serão gerados testes e documentação dos resultados obtidos. Sendo assim, será realizado um estudo sobre a exploração da GPU como um coprocessador no auxílio a CPU, se concentrando no desenvolvimento de algoritmos híbridos e convencionais, tornando possível a realização de testes em uma arquitetura híbrida (CPU e GPU) e arquitetura convencional (somente CPU), explorando cada arquitetura utilizando OpenCL.

II. REVISÃO BIBLIOGRÁFICA

A. *Arquitetura de Computadores*

Os computadores são compostos por diversos componentes, que trabalham em conjunto para entregar um resultado que satisfaça uma condição inicial. Estes componentes são organizados em grupos distintos como CPU, Memória Principal, controladores de entrada e saída de dados e barramentos. Dentre estes grupos um em especial chama a atenção, a CPU. Segundo Tanembaun [1], CPU é o 'cérebro' do computador. Sua função é executar programas armazenados na memória principal, buscando suas instruções, examinando-as e então executando-as uma após a outra. A CPU é composta por partes menores, como demonstra a figura 1-A.

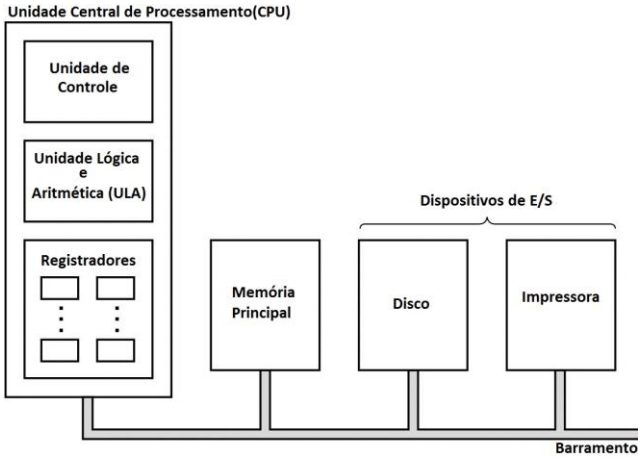

Figura 1-A - Organização de Computador Simples.
Fonte Tanenbaum[1]

Algumas das partes de uma CPU e suas responsabilidades:

- **Unidade de Controle**: busca as instruções na memória e as decodificam.[1]
- **ALU:** trabalha com os dados presentes na instrução, realizando operações lógicas e aritméticas quando necessário.[1]
- **Registradores**: que trabalham com informação temporária, com acesso rápido na leitura e escrita.[1]

Segundo Tanembaun[1], a CPU também contém um pequena memória de alta velocidade usada para armazenar resultados temporários e para um certo controle da informação. Essa memória é composta por uma quantidade de registradores, cada um dele com um certo tamanho e função. Normalmente, todos os registradores têm o mesmo tamanho. Cada registrador pode conter um número, até algum máximo determinado pelo tamanho do registrador. Registradores podem ser lidos e escritos em alta velocidade porque são interno a CPU[1].

A arquitetura de CPU mais popular atualmente, é baseada na arquitetura de Von Neumann, demonstrada na figura 1-B. Este tipo de arquitetura possui registradores que ficam conectados a ULA, fornecendo informações para que a ULA consiga realizar as devidas operações, e um registrador na saída da ULA, que é chamado de Registrador Acumulador. Esta disposição de componentes é chamada Caminho de Dados[1], e segundo Tanembaum é essencial em todos os computadores.

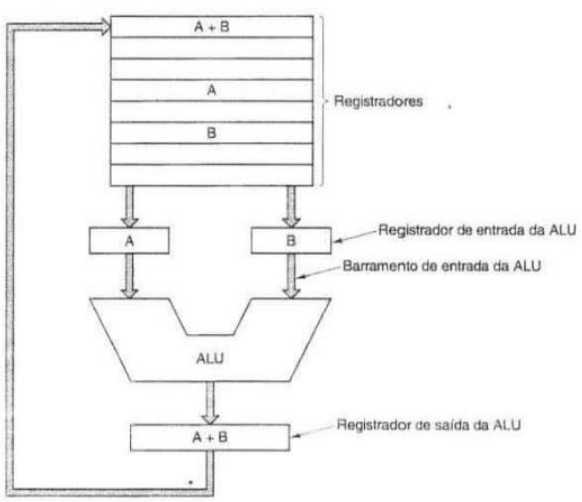

Figura 1-B - Arquitetura de Von Neumann(caminho de dados). Fonte Tanenbaum[1].

Segundo Tanembaum, para a execução de uma instrução, a CPU necessita executar os seguintes passos:

1. Trazer a próxima instrução da memória principal até o registrador(contador de programa).
2. Alterar o registrador contador de programa para indicar a próxima instrução.
3. Determinar o tipo de instrução.
4. Se a instrução usar uma palavra da memória, determinar onde está palavra está.
5. Trazer a palavra para dentro de um registrador da CPU, se necessário.
6. Voltar à etapa 1 para iniciar a execução da instrução seguinte.

Estes passos são chamados de buscar-decodificar-executar e é fundamental para a operação de todos os computadores[1].

B. Arquitetura Multicore

Segundo D.G. Victor [2], a tecnologia Multicore é uma resposta aos problemas causados pela miniaturização dos componentes no núcleo de um processador. Sinteticamente, a arquitetura multicore consiste em agregar dentro de um único chip de processador dois ou mais núcleos de execução, possibilitando - com o software apropriado - a efetiva execução de múltiplas threads[2]. Figura1-C.

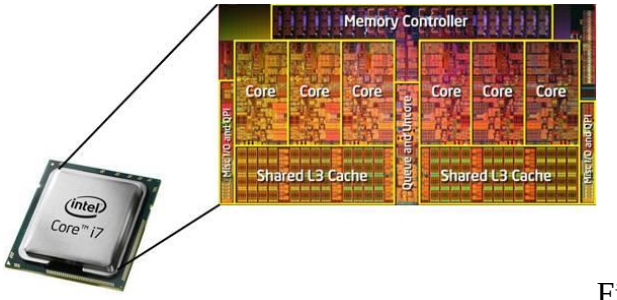

Figura 1-C: CPU com mais de um núcleo na mesma pastilha. Fonte Kim[3].

C. Unidade Gráfica de ProcessamentoGPU

Nos primórdios da computação, os computadores não utilizavam interface gráfica, os textos e tudo o que era exibido na tela era o processador quem processava. Mas isso foi mudando com o tempo, principalmente depois do lançamento do Xerox 810 Star, que possuía interface gráfica e tornou as coisas mais complexas para somente o processador trabalhar. A solução para desafogar o processador neste quesito, foi a criação de outra central de processamento que ficasse a cargo somente do processamento gráfico. Um dos primeiros projetos foi da IBM e se chamava PGA(Professional Graphics Controller) que entregava uma resolução de 640x320pixels com uma paleta de 256 cores, espetacular para a época[4]. Mas mesmo assim não foi o suficiente para fazer as GPUs serem essenciais, mas as coisas começaram a mudar quando as empresas começaram a fabricar placas de vídeos 3d, e com isso uma dos primeiros motivos surgiram para reforçar a ideia que uma GPU seria essencial, os jogos eletrônicos. O termo GPU só foi empregado de forma definitiva quando a empresa Nvidia e Amd em 1999, lançaram suas placas de vídeos no mercado tornando essas placas acessíveis a maioria dos usuários[4].

As GPUs evoluíram rapidamente em seu poder computacional, e segundo Vianna e Gomes[5], recentemente estão sendo incorporadas áreas de programação onde o desenvolvedor pode substituir as funções fixas por suas próprias rotinas, dando assim maior liberdade ao programador para criar efeitos mais interessantes, usufruindo do poder de processamento paralelo da GPU e de suas otimizações em nível de hardware. Essas áreas programáveis são chamadas de shaders, e que estão sendo exploradas para processamento genérico e não só para imagens, como por exemplo o que a api CUDA faz, que atualmente é uma das que mais se destacam na área[5]. Um dos pontos altos para as GPUs é a forma como ela realiza um processamento, em uma CPU os núcleos são otimizados para o processamento em série sequencial, enquanto que na GPU consiste em milhares de núcleos menores, mais eficientes projetados para lidar com múltiplas tarefas simultaneamente[6]. E segundo a Nvidia[6], GPUs têm milhares de núcleos para processar cargas de trabalho paralelas de forma eficiente. Uma comparação com a quantidade de cores é demonstrado na figura 1-D.

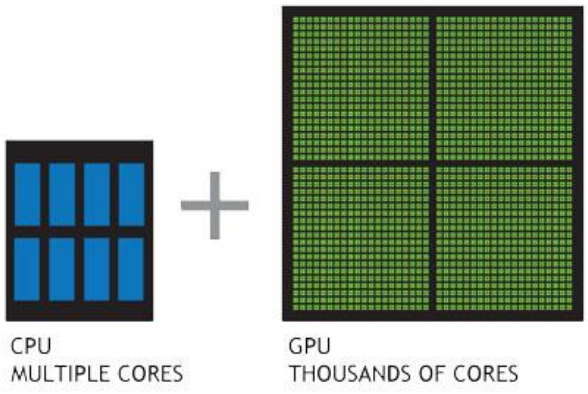

Figura 1-D - Comparação da quantidade de Cores em cada uma das arquiteturas. Fonte: Nvidia[6].

D. Processamento Híbrido CPU/GPU

Com o surgimento de diversas APIS(Application Programming Interface) com suporte ao paralelismo entre arquiteturas, está sendo possível utilizar a real capacidade das arquiteturas da CPU e GPU[7] . O principal objetivo da programação paralela híbrida é tirar proveito das melhores características de cada arquitetura visando alcançar um melhor desempenho[7], distribuindo o fluxo de processamento a arquitetura que melhor se adapte aquele tipo de processamento.

Um dos usos deste recurso seria o demonstrado pela empresa Nvida, onde pode-se transferir parte do processamento intensivo do aplicativo para a GPU, enquanto o resto do código continua sendo executado pela CPU[6] ou vice-versa. Figura 1-E.

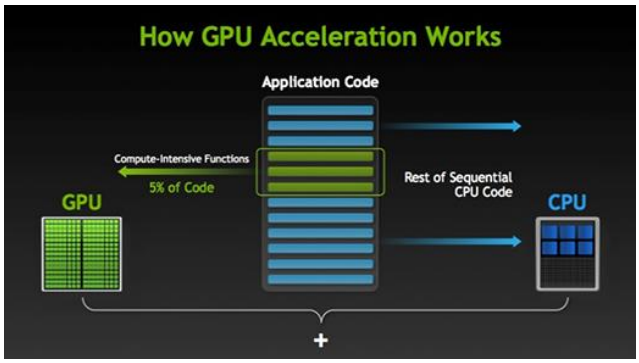

Figura 1-E. Parte do Código sendo processado pela GPU e o restante pela CPU. Fonte: Nvidia[6].

No final obtemos o código processado por ambas as arquiteturas, cada um na sua especialidade. Este processo é imperceptível ao usuário final[6].

Dentre essas APIS, se destacam algumas que visam explorar este cenário. São elas OpenMP(Open Multi Processing), MPI(Message Passing Interface) paralelização de CPU, CUDA(Compute Unified Device Archiecture) e OpenCL(Open Computing Language) para paralelização em GPU[6].

E. Algoritmos Distribuídos/Paralelos

Como comentado anteriormente, a combinação da arquitetura da CPU com a arquitetura GPU depende de uma peça fundamental para que realmente funcione. Os algoritmos necessariamente tem que serem escritos de forma que venham aproveitar essa arquitetura híbrida de forma correta.

Os Sistemas Operacionais atuais já trabalham com a divisão de um algoritmo comum para funcionar em mais de um processador e são chamados de sistemas multiprocessados[9], mas esse algoritmo necessita ter descrito em seu código que em algum ponto determinando a criação de uma ou mais Threads, cada programa em execução no sistema operacional é considerado um ou mais processos[9] e este processo pode ser dividido em partes menores chamadas Threads.

Um algoritmo Paralelo funciona neste contexto, este algoritmo tem em seu código, trechos que descrevem onde deverá ocorrer a paralelização de recursos da máquina.

III. METODOLOGIA

Para a realização dos testes, que é base desta pesquisa, os algoritmos serão escritos no padrão OpenCL 2.0, que será utilizado ao invés de outros padrões por suportar o paralelismo de CPU e GPU e também por se tratar de um padrão criado com a proposta de poder ser executado em qualquer plataforma que implemente OpenCL, não ficando restrito a uma solução criado por uma empresa específica e um hardware em específico. Segundo a Amd[8], as aplicações aceleradas com o OpenCL podem acessar o poder de processamento combinada dos núcleos da GPU e CPU de um computador sob uma única plataforma unificada. E o OpenCL tem a capacidade de realizar tarefas de cálculos intensivo em paralelo, utilizando virtualmente qualquer processador multicore[7].

O OpenCL é uma API independente de plataforma que permite aproveitar as arquiteturas de computação paralelas disponíveis como CPUs multicore e GPUs, esta API tem foco na portabilidade e foi criada pelo Khronos Group em 2008, ela define uma especificação aberta em que os fornecedores de hardware podem implementar, ela por ser independente de plataforma se torna mais complexa que outras APIs para o desenvolvimento[7], mas possui um amplo suporte de fabricantes como Nvidia, Amd e Intel por exemplo, tendo elas suas próprias implementações de OpenCL e seus próprios kits de desenvolvimento(SDK). A especificação OpenCL é feita em 3 partes: linguagem, camada de plataforma e runtime[7].

- **Especificação da linguagem**: esta especificação descreve a sintaxe e a API para escrita do código em OpenCL[7].
- **Camada de Plataforma**: fornece ao desenvolvedor acesso à rotinas que buscam o número e os tipos de dispositivos no sistema[7].
- **Runtime**: possibilita ao desenvolvedor enfileirar comandos para a execução nos dispositivos e fica responsável por gerenciar os recursos disponíveis[7].

Para a realização dos testes, será usado um computador com processador Intel CoreI5 3550 de 4 núcleos físicos, 8 Gb de memória DDR3 1066Mhz, SSD de 480Gb Sandisk Ultra II velocidade de escrita de até 550Mbs e de até 500Mbs na leitura. Também será

usado uma placa de vídeos Saphire ATI Radeon R9 290X com 2816 Stream Processors, 4Gb de Ram GDDR5, 512Bit.

IV. PROJETO

Este projeto de pesquisa visa analisar, testar e documentar o desempenho de algoritmos escritos no padrão OpenCL e executados em CPU e CPU + GPU, criação de gráfico dos resultados. Alguns trabalhos que foram encontrados com a proposta semelhante deste projeto.

A. *Programação Paralela Híbrida em CPU e GPU: Uma Alternativa na Busca por Desempenho[7]*

Este trabalho focou em análise e testes de desempenho utilizando a junção das arquiteturas de CPU e GPU utilizando APIs para a realização de paralelização de CPU e para a paralelização de GPU. Foi utilizada para a paralelização de CPU as APIs OpenMP e MPI, já para a paralelização de GPU foi utilizada as APIs CUDA e OpenCL. Os autores realizaram os testes com combinações e são eles :

- Execução de algoritmo híbrido utilizando OpenMP e CUDA, denominado OMP/CUDA.
- Execução de algoritmo híbrido utilizando OpenMP e OpenCL, denominado OMP/OCL.
- Execução de algoritmo híbrido utilizando MPI e CUDA, denominado MPI/CUDA.
- Execução de algoritmo híbrido utilizando MPI e OpenCL, denominado MPI/OCL

Neste trabalho indicado, ele apresentou diversos testes com diferentes tamanhos de cargas e com vetores de tamanhos diferentes e quantidade de repetições. Tudo foi documentado

B. *Ambientes de Programação Paralela Híbrida[9]*

Este trabalho explora a contextualização sobre a programação paralela em arquitetura híbrida. Em que o autor apresenta uma visão geral sobre as arquiteturas paralelas híbridas atuais e apresenta classes de ferramentas de programação que visam facilitar o desenvolvimento e a executar códigos nestas arquiteturas.

Este trabalho apresenta vários tipos de arquiteturas , algumas mais antigas e outras mais novas como por exemplo a arquitetura de processadores Amd Fusion, que contém na mesma pastilha os processadores da arquitetura de CPU x86 e de arquitetura GPU. Algumas das ferramentas apresentadas neste trabalho são:

- **CUDA:** plataforma paralela de propósito geral em GPUs fabricada pela NVIDIA[9].
- **OpenCL:** padrão de programação paralela em ambientes heterogêneos que possibilita o desenvolvimento de aplicações que executem sobre um conjunto de fabricantes, como CPUs e GPUs[9].

- **Thrust:** biblioteca de algoritmos paralelos para programação CUDA.
- **HMPP:** ambiente de programação que oferece suporte à aceleradores por meio de diretivas de compilação. Neste ambiente o código fonte não contém referência ao hardware em que está sendo executado, gerando código portável graças as diretivas.

C. *Análise de Paralelismo Em Arquitetura Multicore Com Uso de Unidade de Processamento Gráfico[10]*

Este trabalho busca demonstrar que a GPU não precisa ser necessariamente somente utilizada para geração de gráficos, com auxílio da API CUDA é possível que a GPU seja utilizada em aplicações de propósitos gerais. Gerando testes e análises o autor buscou demonstrar que a dupla de arquitetura CPU e GPU, podem gerar aumento no desempenho de aplicações que visam tirar proveito dessas arquiteturas simultâneas, explorando o paralelismo destas duas arquiteturas.

O autor utilizou algoritmos de ordenação de vetores para a geração de testes e expôs os resultados através de gráficos e tabelas.

D. *A proposta deste Projeto*

Este trabalho tem o objetivo de gerar os testes baseados em algoritmos que exigem esforço pesado de utilização dos processadores da CPU e da GPU. Mas acima de tudo gerar testes que consiga tirar proveito da arquitetura híbrida, visto que atualmente praticamente todos os computadores possuem mesmo que um modelo básico de GPU em seus computadores. Utilizando o padrão OpenCL, que foi a ferramenta escolhida para a geração de código para os testes, pois o mesmo pode ser executado em diversas plataformas com um mínimo de modificação de código, gerando assim código com portabilidade. A figura 1-F demonstra a ideia básica por trás dessa arquitetura híbrida, a CPU e a GPU cada uma com suas quantidades de cores individuais.

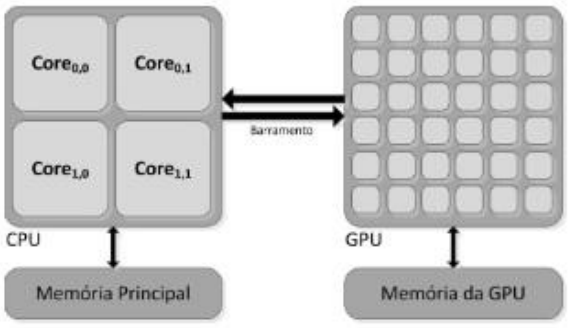

Figura 1-F. Arquitetura Híbrida CPU e GPU. Fonte: Pinto[9].

Apesar de alguns modelos de processadores(CPU) conter vários núcleos físicos(MULTICORE), a GPU por sua vez, tem a vantagem de atualmente conter milhares de Núcleos físicos(Stream Processors) que trabalham em paralelo e podem ser programados, o que torna interessante o uso de cada arquitetura específica na operação em que ela for mais otimizada, fazendo uso do paralelismo entre arquiteturas além do paralelismo de núcleos dentro de cada PU(Unidade de Processamento).

Através da utilização do padrão OpenCL, em que é possível explorar o paralelismo de CPU e GPU independente de fabricante, será desenvolvido algoritmos para testes que

visam explorar este paralelismo e também o paralelismo em uma arquitetura multicore (CPU), os testes realizados serão documentados e serão usados como base para a geração de gráficos que venham demonstrar através de forma gráfica o ganho ou não de desempenho na utilização desta arquitetura. Comprovando que tipo de cenário seria ideal a utilização do mesmo e seus benefícios. Da mesma forma apontar o cenário em que a utilização do mesmo venha a não ter benefícios e até mesmo perca de desempenho, chegando a não ser aconselhável o seu uso. A figura 1-G demonstra um trecho de código com processamento paralelo utilizando OpenCL[11].

```
1: __kernel void
2: vecmul(__global double* vectorA, __global double* vectorB,
3:         __global double* result_vector )
4: {
5:     int i = get_global_id(0);
6:     double a = vectorA[i];
7:     double b = vectorB[i];
8:     result_vector[i] = a * b;
9: }
```

Figura 1-G. Trecho de código em OpenCL. Fonte [11].

V. RESULTADOS

Cada teste foi executado 100 vezes, com os valores coletados foi possível realizar o cálculo da média aritmética e também foi calculado o desvio padrão amostral sobre o tempo. Essa abordagem foi utilizada porque existe sempre a utilização dos dispositivos pelo Sistema Operacional, não sendo possível executar o teste sem que algum tipo de processamento esteja ocorrendo no dispositivo fora do contexto do teste, influenciando o resultado do teste realizado. A CPU é a que mais sofre alteração como na maioria dos testes foi possível verificar com a utilização do desvio padrão, ficando a GPU de certa forma com os resultados com menos variações. Desta forma não foi considerada a utilização dos dispositivos pela Sistema Operacional, já que não é possível dedicar todo o processamento do dispositivo para o teste.

No final de cada teste foi gerado um arquivo de texto com os valores de todos os testes realizados com o adicional do desvio padrão que se encontra em anexo junto ao final do trabalho.

Os testes foram executados com o mesmo tamanho de vetor entre as arquiteturas, sendo na CPU/GPU dividida igualmente entre as duas unidades de processamento.

A operação realizada no kernel é uma subtração de um valor contido em uma posição de um vetor pelo valor contido na posição de outro vetor diferente. Os vetores são preenchidos de forma aleatória pelo próprio algoritmo.

```
__kernel void Subtracao(__global const int* a, __global const int* b,__global int* c)
{
    int id = get_global_id(0);

    c[id] = a[id] - b[id];
};
```

Figura 2 . Kernel utilizado para os testes sem dependência de dados. Fonte: Elaborada pelo autor.

A. Teste utilizando um vetor de 1.000 posições

O primeiro teste foi utilizado um vetor de 1.000 posições e foi obtido o resultado contido na tabela 1.

Dispositivo	Vetor 1.000 Posições	Variação
CPU	0,011	0,003
GPU	0,044	0,004
CPU / GPU	0,041	0,002

Tabela 1 - Resultado Vetor de 1.000 posições.
Fonte: Elaborada pelo autor.

No primeiro teste realizado, a CPU conseguiu superar a GPU e a CPU/GPU. A CPU conseguiu ser aproximadamente 300% mais rápida que a GPU e aproximadamente 272% mais rápida que a CPU/GPU. Já a diferença entre a GPU e a CPU/GPU foi de aproximadamente 7%. A Figura 1-I ilustra através de gráficos, os resultados obtidos.

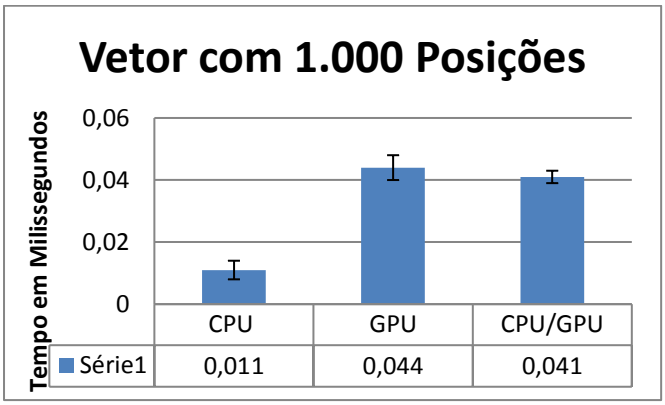

Figura 1-I. Gráfico Vetor de 1.000 Posições.
Fonte: Elaborada pelo autor.

B. Teste utilizando um vetor de 10.000 posições

O segundo teste foi utilizado um vetor de 10.000 posições e foi obtido o resultado contido na tabela 2.

Dispositivo	Tamanho do Vetor 10.000	Variação
CPU	0,037	0.006
GPU	0.046	0.005
CPU / GPU	0.043	0.007

Tabela 2 - Resultado Vetor de 10.000 posições.
Fonte: Elaborada pelo autor.

O teste realizado com um vetor de 10.000 posições, a CPU continua sendo mais rápida que a GPU e a CPU/GPU. A diferença entre eles foi amenizada, mais ainda existe. Neste cenário a CPU conseguiu ser aproximadamente 24% mais rápida que a GPU e aproximadamente 16% mais rápida que a CPU/GPU. A CPU/GPU conseguiu superar a GPU em aproximadamente 7% novamente. A Figura 1-J ilustra através de gráficos, os resultados obtidos.

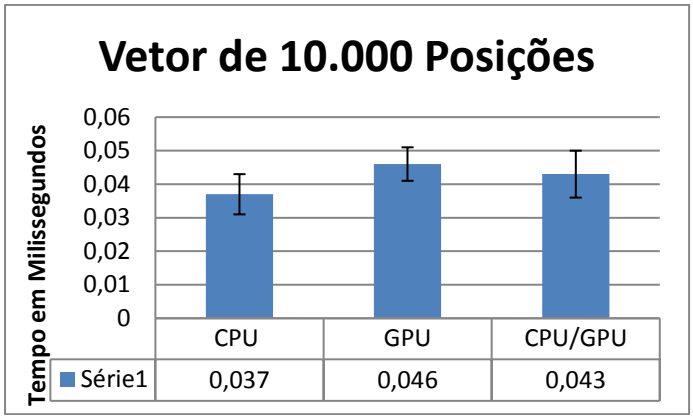

Figura 1-J. Gráfico Vetor de 10.000 Posições.
Fonte: Elaborada pelo autor.

C. *Teste utilizando um vetor de 20.000 posições*

Neste foi utilizado um vetor de 20.000 posições e foi obtido o resultado contido na tabela 3.

Dispositivo	Tamanho do Vetor 20.000	Variação
CPU	0,067	0,002
GPU	0,045	0,001
CPU / GPU	0,047	0,005

Tabela 3 - Resultado Vetor de 20.000 posições
Fonte: Elaborada pelo autor.

Neste teste, houve uma mudança nos resultados. A GPU começou a demonstrar que consegue trabalhar melhor com carga maiores de trabalho. A GPU superou a CPU e a CPU/GPU, chegando a ser aproximadamente 49% mais rápida que a CPU e aproximadamente 4% mais rápida que a CPU/GPU. Já a CPU/GPU superou a CPU, sendo aproximadamente 42% mais rápida que ela. Neste teste foi possível observar que a CPU/GPU está se aproximando da GPU sozinha. A Figura 1-K ilustra através de gráficos, os resultados obtidos.

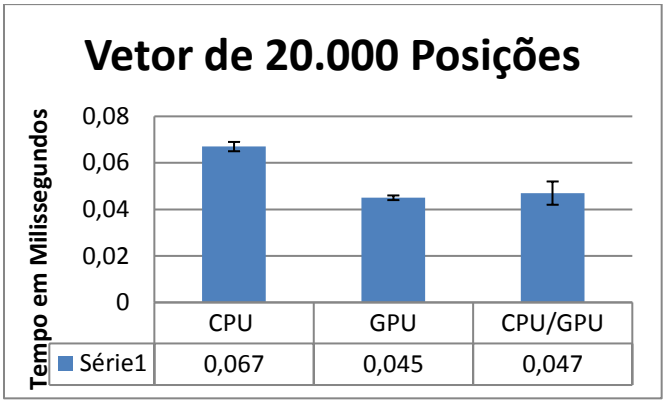

Figura 3. Gráfico Vetor de 20.000 Posições.
Fonte: Elaborada pelo autor.

D. Teste utilizando um vetor de 22.000 posições

Como no teste de 20.000 a CPU/GPU se aproximou bastante da GPU, no quarto teste foi utilizado um vetor de 22.000 posições e foi obtido o resultado contido na tabela 4

Dispositivo	Tamanho do Vetor 22.000	Variação
CPU	0,065	0,006
GPU	0,045	0,010
CPU / GPU	0,044	0,005

Tabela 4 - Resultado Vetor de 22.000 posições.
Fonte: Elaborada pelo autor.

.

Neste novo teste, agora com 22.000 posições, foi encontrado o ponto aproximado em que CPU/GPU superou a CPU e a GPU. A combinação CPU/GPU foi aproximadamente 35% mais rápida que CPU sozinha e aproximadamente 2% mais rápida que a GPU. A Figura 1-L ilustra através de gráficos, os resultados obtidos.

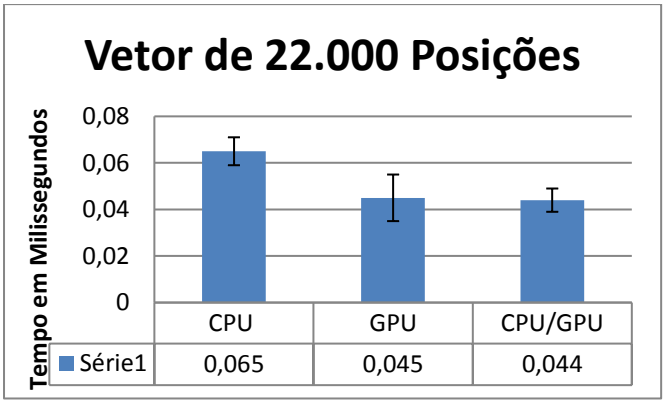

Figura 1-L. Gráfico Vetor de 22.000 Posições.
Fonte: Elaborada pelo autor.

E. Teste utilizando um vetor de 24.000 posições

Como no teste de 22.000 a CPU/GPU conseguiu um tempo menor que as outras arquiteturas, o teste agora foi feito com 24.000 posições e foi obtido o resultado contido na tabela 5.

Dispositivo	Tamanho do Vetor 24.000	Variação
CPU	0,074	0,009
GPU	0,045	0,006
CPU / GPU	0,047	0,006

Tabela 5 - Resultado Vetor de 24.000 posições.
Fonte: Elaborada pelo autor.

Neste teste, a o tempo que CPU/GPU utilizou para processar o vetor, começou aumentar novamente. O que demonstra que o teste anterior se mostrou mais propício para o cenário CPU/GPU. A CPU foi a que mais demorou para processar o vetor. A Figura 1-M ilustra através de gráficos, os resultados obtidos.

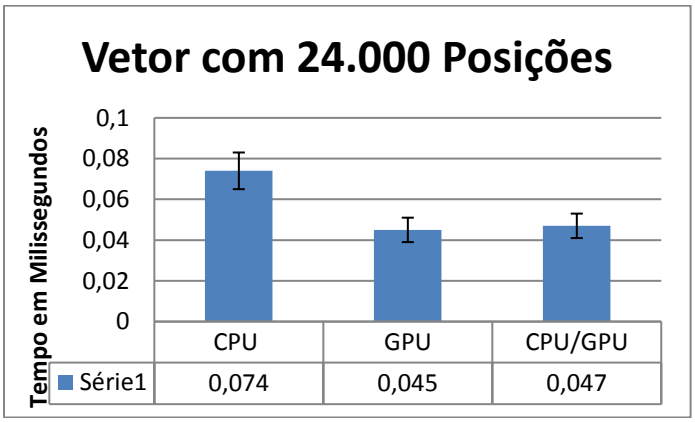

Figura 1-M. Gráfico Vetor de 24.000 Posições.
Fonte: Elaborada pelo autor.

F. Teste utilizando um vetor de 100.000 posições

Para o quinto teste foi utilizado um vetor de 100.000 posições e foi obtido o resultado contido na tabela 6.

Dispositivo	Tamanho do Vetor 100.000	Variação
CPU	0,201	0,016
GPU	0,047	0,001
CPU / GPU	0,172	0,009

Tabela 6 - Resultado Vetor de 100.000 posições.
Fonte: Elaborada pelo autor.

Neste teste a GPU voltou a ser mais rápida que as outras arquiteturas. Ela chegou a ser aproximadamente 327% mais rápida que a CPU e aproximadamente 266% mais rápida que CPU/GPU. A CPU/GPU conseguiu ser aproximadamente 17% mais rápida que a CPU.

Novamente em um cenário que possui uma carga maior, a GPU conseguiu realizar o processamento sem que houvesse uma alteração muito grande no tempo que ela levou para processar. A Figura 1-N ilustra através de gráficos, os resultados obtidos.

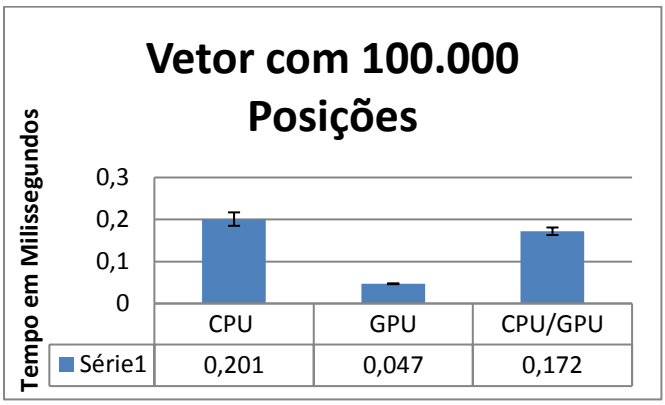

Figura 1-N. Gráfico Vetor de 100.000 Posições.
Fonte: Elaborada pelo autor.

G. Teste utilizando um vetor de 1.000.000 posições

Para o sexto teste, o vetor teve um aumento proposital para que fosse possível a análise em cima das três arquitetura, o vetor utilizado possui 1.000.000 posições e foi obtido o resultado contido na tabela 7.

Dispositivo	Tamanho do Vetor 1.000.000	Variação
CPU	2,822	0,350
GPU	0,087	0,006
CPU / GPU	0,953	0,038

Tabela 7 - Resultado Vetor de 1.000.000 posições.
Fonte: Elaborada pelo autor.

Foi possível observar um aumento considerável no tempo da CPU para processar o vetor, a GPU por sua vez, não teve um aumento na mesma proporção e superou a CPU e a CPU/GPU. A GPU conseguiu ser aproximadamente 3.100% mais rápida que CPU e 995% mais rápida que CPU/GPU. A CPU/GPU superou em aproximadamente 196% a CPU. A Figura 1-O ilustra através de gráficos, os resultados obtidos.

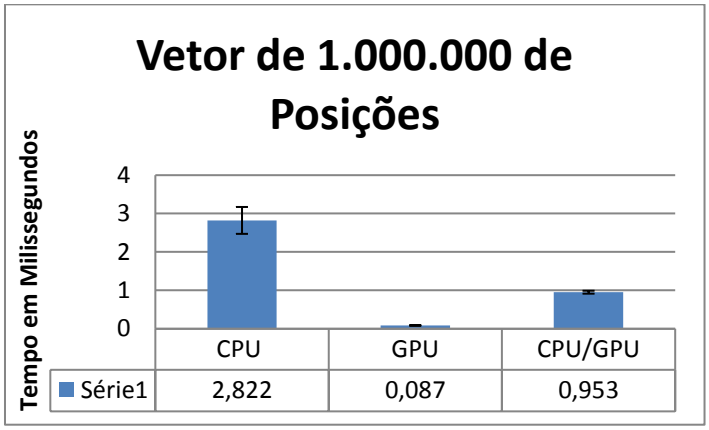

Figura 4. Gráfico Vetor de 1.000.000 Posições.
Fonte: Elaborada pelo autor.

H. Teste utilizando um vetor de 10.000.000 posições

O último teste foi utilizado um vetor de 10.000.000 posições e foi obtido o resultado contido na tabela 8.

Dispositivo	Tamanho do Vetor 10.000.000	Variação
CPU	29,114	3,968
GPU	0,512	0,073
CPU / GPU	9,029	0,687

Tabela 8 - Resultado Vetor de 10.000.000 posições.
Fonte: Elaborada pelo autor.

No último teste a diferença entre a GPU e as outras duas arquiteturas somente aumentou. A GPU foi aproximadamente 5.587% mais rápida que a CPU e 1.663% mais rápida que a CPU/GPU. A CPU/GPU conseguiu ser aproximadamente 222% mais rápida que a CPU. A Figura 1-P ilustra através de gráficos, os resultados obtidos.

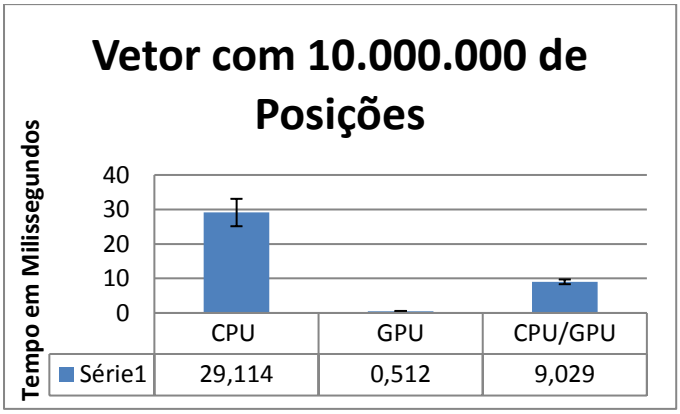

Figura 1-P. Gráfico Vetor de 10.000.000 Posições.
Fonte: Elaborada pelo autor.

Com o resultado dos testes, fica-se entendido que vetores pequenos não são interessantes de serem processados pela GPU, já com vetores de tamanho grande fica num cenário mais interessante a utilização da GPU.

No caso da CPU, ela demonstrou nos testes que consegue trabalhar muito bem com vetores pequenos mas que ao aumentar o tamanho do vetor, como por exemplo, acima de vinte mil posições, ela demonstra que precisa de mais tempo para processar o vetor que as outras arquiteturas, ficando definido que a para a CPU é interessante o processamento de vetores pequenos.

No caso da união das arquiteturas, foi concluído que ela consegue processar os vetores propostos de forma a ficar entre a CPU e a GPU, mas que em um determinado cenário ela consegue se destacar mais que as outras arquiteturas. Por isso, foi realizado mais alguns testes com vetores de tamanho próximo a vinte mil, pois neste tamanho de vetor as arquiteturas processaram o vetor com um tempo muito próximo uma da outra. Com isso, chegou-se a um valor aproximado de vetor em que a CPU/GPU conseguiu processar o vetor em um tempo menor que as arquiteturas da CPU e GPU.

REFERENCIAS BIBLIOGRÁFICAS

[1] T. S. Andrew, “Organização Estruturada de Computadores” , 5º Edição.

[2] D.G. Victor, Avaliação de Técnicas de Engenharia de Software para Modelagem de Programas Parelelos em Ambientes Multicore, 2008

[3] J.Kim. Disponível em :
< httpshpc.postech.ac.kr~jangwooresearchresearch.html>. Acessado em 29 de Maio de 2015.

[4] D. KerBer. Disponível em: <http://adrenaline.uol.com.br/2012/09/07/19438/entenda-o-que--uma-placa-de-v-deo--gpu->. Acessado em 30 de Maio de 2015.

[5] F. V. Vianna, T.E. Gomes, ShaderLabs: Desenvolvimento ágil de uma IDE para OpenGL Shaders.

[6] O que é Computação com GPU?. Disponível em <http://www.nvidia.com.br/object/what-is-gpu-computing-br.html>. Acesso em 30 de Maio de 2015.

[7] A.L. Stefanello, C. C. Machado, D. Rosa, M.Sulzbach, R.W. Moerschbacher, T.R.Sarturi, Programação Parelela Híbrida em CPU e GPU: Uma Alternativa na Busca por Desenpenho,RS - Brasil, 2013.

[8] A linguagem aberta para GPU. Disponível em:< http://www.amd.com/pt-br/solutions/professional/hpc/opencl#recursos>. Acessado em: 31 de Maio de 2015.

[9] V.G. Pinto, Ambientes de Programação Paralela Híbrida, Porto Alegre, 2011.

[10] J.H.M. Cimino,Análise de Paralelismo Em Arquitetura Multicore Com Uso de Unidade de Processamento Gráfico, Marília, 2011.

[11] A.L.R. Tupinambá, DistributedCL: middleware de processamento distribuído em GPU com interface da API OpenCL, Rio de Janeiro, 2013.