GPU: A matriz de processadores

Paulo Henrique Junqueira Amorim
Instituto de Computação
Universidade Estadual de Campinas
Campinas, SP, Brasil
paulojamorim@gmail.com

Francisco Romulo da Silva Araújo
Instituto de Computação
Universidade Estadual de Campinas
Campinas, SP, Brasil
franciscoromulo@gmail.com

Resumo
A unidade de processamento gráfico ou *graphics processing unit* (GPU), tornou-se uma peça importante em sistemas computacionais nos últimos anos. Inicialmente apenas com a proposta de controlar a exibição de imagens na tela. Hoje, além de placas com um alto poder de processamento gráfico, as GPU's são também processadores que podem ser utilizados para programação paralela em conjunto com altas larguras de banda. A facilidade de programação e o poder de processamento têm aumentado a cada ano. Com isso pesquisadores de diversas áreas e até mesmo o mercado financeiro tem utilizado a GPU para solucionar e modelar problemas que demandariam um tempo maior se processados pela CPU. O presente artigo apresenta a GPU sob um enfoque arquitetural e descreve os aspectos básicos para uma aplicação ser executada.

Keywords
GPU, Streaming Processor, Mutithreads

1. INTRODUÇÃO
Entre o final dos anos 80 e o inicio dos anos 90, os gráficos em um computador eram executados pelo controlador VGA, um simples controlador de vídeo e memória conectado a DRAM [12]. Em 1997, os controladores VGA começaram a adicionar recursos para aceleração 3D por hardware, como a rasterização de triângulos [9] e o sombreamento (*shading*) [11]. A geração seguinte (1999-2000), já suportava a transformação de vértices, iluminação e mapeamento de textura. Em 1999 a NVIDIA lançou a Geforce 256 e para diferenciá-la das outras placas, que só tinham a capacidade de rasterização, usou o termo *graphics processing unit* para comercializá-la como sendo "a primeira GPU do mundo".
O objetivo inicial das primeiras GPU's era otimizar o tempo de geração de gráficos 2D e 3D, imagens e vídeos, que seriam utilizados pelos sistemas operacionais (*window-based*) para sua exibição no monitor, removendo essa carga da CPU. Nas últimas duas décadas, as unidades de processamento gráfico estão cada vez mais presentes, seja em computadores pessoais, *laptop's*, vídeo games, dispositivos móveis ou até mesmo em automóveis.
Ao longo dos anos, a programação da GPU tornou-se mais fácil com o advento das interfaces de programação, que serão apresentadas nos próximos capítulos. Além disso, por ser um processador com alto poder de paralelismo, os fabricantes de GPU's também tem mantido o foco em outras áreas aonde o processamento paralelo pode ser utilizado, como simulações por análise de elementos finitos e processamento de sinais. Com isso surgiu um novo termo, GP-GPU, que são GPU's de propósito geral. Existem GPU's que são produzidas sem a saída de vídeo na placa, normalmente essas placas contêm vários chip's de GPU's destinados ao mercado de processamento de alto desempenho, como é o caso de algumas GPU's da arquitetura NVIDIA Tesla [3]. No outro extremo da computação, as GPU's marcam presença nos *smartphones* e *tablets*, como o chip da NVIDIA Tegra [5] voltado para dispositivos móveis.

2. PIPELINE TRADICIONAL (GRÁFICOS)
Quando a GPU é utilizada em aplicações tradicionais para seu propósito específico, como jogos e softwares especializados para computação gráfica, tipicamente recebe como entrada um conjunto de primitivas geométricas, basicamente triângulos em um cenário 3D, com o sistema de coordenadas cartesianas em conjunto com seus respectivos mapas de cores. Através de vários passos, essas primitivas são sombreadas e mapeadas na tela, onde serão construídas para a criação da cena final em 2D.
As primitivas são formadas por vértices individuais, cada vértice normalmente é transformado no espaço da cena e em seguida sombreado. Os vértices ou um grupo de vértices podem ser calculados independentemente, construindo um triângulo em seguida, peça fundamental na composição dos objetos 3D.
Um dos últimos passos antes de exibir a imagem na tela é a rasterização, que consiste no processo de calcular quais pixels o triângulo estará contido e a sua respectiva interpolação, pois o vértice do triângulo pode está no meio do pixel. Cada triângulo é chamado de fragmento, esses fragmentos compõem a cena final e essa construção tipicamente é realizada em paralelo.
Além dos itens citados anteriormente, existem implementações em algumas GPU's para simulação de física, como por exemplo, quando uma pedra é jogada em um vidro. O vidro é quebrado e é necessário determinar quais triângulos serão separados do objeto para dar o efeito de rompimento.

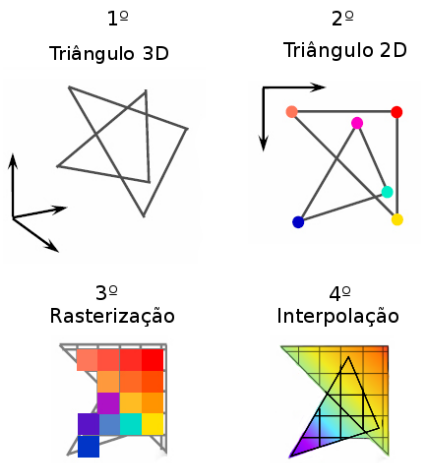

Figure 1: Fases para transformar triângulo em pixel. Adaptado de [8]

3. ARQUITETURA DA GPU

Nessa seção será apresentada uma descrição arquitetural das GPU's, sua matriz de processadores, funcionamento e o sistema de memória.

3.1 Uma visão macro

Na arquitetura de um computador de 1990 (circa 1990), a "GPU"da época, simplesmente controlava a memória *framebuffer*, uma memória que armazenava a imagem a ser exibida na tela, e a operação de exibição da imagem no monitor. O controlador era conectado por um barramento PCI até a *north bridge*.

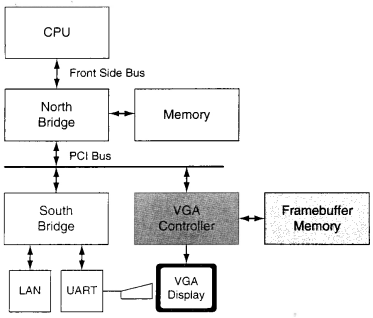

Figure 2: Arquitetura com controlador VGA. Extraída de [12]

As figuras 3 e 4 exibem duas configurações comumente utilizadas atualmente. Na arquitetura Intel (fig.3), a GPU é ligada a *north bridge* via barramento PCI-Express 16x, proporcionando uma taxa de transferência de 16 GB/s. Já na arquitetura AMD (fig.4), a GPU é conectada ao chipset, também via PCI-Express com a mesma largura de banda. O chipset conecta-se a *north bridge* via *HyperTransport* [6]. Em alguns sistemas mais econômicos são utilizadas arquiteturas UMA (*Unied Memory Architecture*) que utilizam somente a memória principal do sistema, omitindo a memória da GPU. Em sistemas GPU de alto desempenho, utilizam-se duas ou mais placas gráficas interligadas, como é o caso da tecnologia SLI (*scalable link InterConnect*) da NVIDIA [4] e o CrossFire da ATI [2].

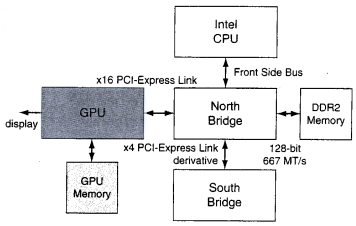

Figure 3: Arquitetura Intel com GPU. Extraída de [12]

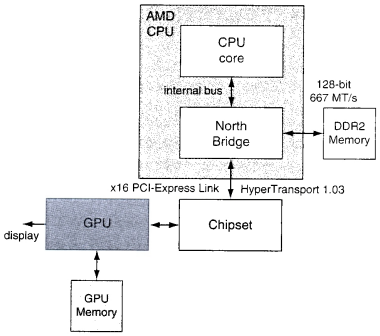

Figure 4: Arquitetura AMD com GPU. Extraída de [12]

3.2 A matriz de processadores

Uma GPU tradicional contém vários núcleos processadores (*cores*). Como estudo de caso, adotaremos a arquitetura Tesla da NVIDIA, mais precisamente o modelo geforce GTX-280. Com 1400 milhões de transistores, a GTX-280, possui 933 GFLOPS de poder de processamento e uma taxa de transferência de memória de 113.3 GB/sec. Para termos idéia do poder computacional, podemos compará-la ao processador Intel Core i7 que possui 731 milhões de transistores, 102 GFLOPS de poder de processamento e uma taxa de transferência de memória de 25.6 GB/sec.
A GTX-280 possui 240 *streaming processors* (SP), que são responsáveis pelo processamento dos *vertex shader* e *pixel shader*, e 10 *texture processor cluster* (TPC), cada um com 3 *Single Program Muliple Data* (SPDM) ou *streaming multiprocessors* (SM). Em cada SPDM ou SM contém 8 unidades de ponto flutuante, inteiros e registradores, sendo conhecidos como *streaming processors* (SP) ou *shader cores*. Além das 8 SP's, o SM ainda é formado por 2 *special function units* (SFU). De acordo com [7], essas unidades são responsáveis por cálculos como logaritmo, transformações de perspectiva, cálculos de iluminação, distância entre vetores, interpolações, entre outros. Em cada SP tem um pequeno registrador com 16KB, já nos SM's existe 1 bloco de memória compartilhada com 160KB e uma cache de primeiro nível, chamado de *texture L1*.

3.3 Multiprocessadores multithreads

Cada *streaming multiprocessor* pode executar até 768 *threads* [10]. Segundo [12], as novas GPU's são altamente

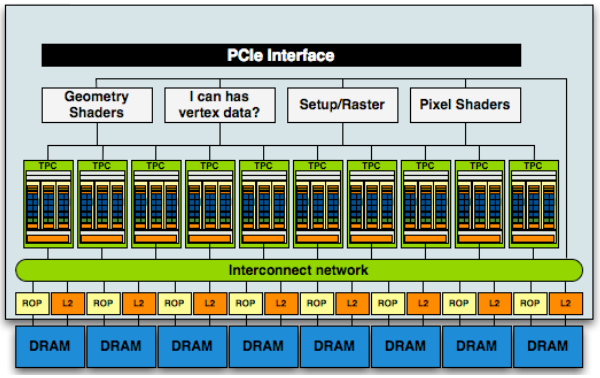

Figure 5: Arquitetura NVIDIA Tesla - GTX-280. Extraída de [1].

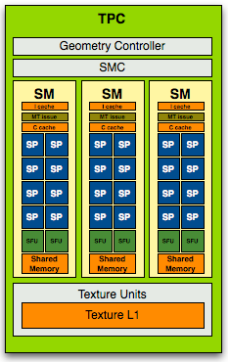

Figure 6: Texture Processor Cluster - GTX-280. Extraída de [1].

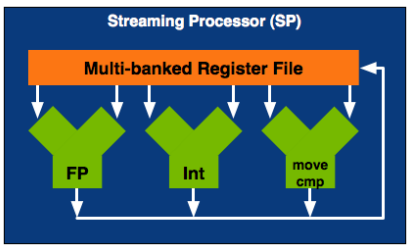

Figure 7: Streaming processor - GTX-280. Extraída de [1].

multithreading para alcançar algumas metas como, compensação da latência de *load* e *fetch* de memória, virtualização de processadores físicos em *threads* ou em blocos de *threads* para prover transparência aos usuários.
Um *fetch* requer uma grande parte da banda de acesso da DRAM, consequentemente causando grande latência. A arquitetura *multithreading* ajuda a minimizar esse efeito, pois enquanto uma *thread* aguarda o resultado do *fetch*, a outra *thread* pode ir realizando o processamento normalmente, onde cada SP poderá executar até 4 *threads*. Para gerenciar todo esse conjunto de *threads* e de softwares diferentes, é adotada uma arquitetura chamada de *single-instruction multiple-thread* (SIMT), responsável por criar, agendar, gerenciar e executar *threads* concorrentes. Cada grupo de *threads* que são executadas em paralelo são chamadas de *wrap*, nome originário de *weaving*, a primeira tecnologia de *threads* paralelas. Cada *wrap* tem 32 *threads* para 1 SM que contém 8 SP's.
No agendamento das *threads*, é selecionado o *wrap* que executará determinada instrução. Cada instrução executa 4 conjuntos de 8 *threads* em 4 ciclos de processador. O agendamento é realizado utilizando o algoritmo de *round-robin*, ou seja, atribui a mesma fração de tempo para todas as *threads* que serão executadas, que por sua vez serão mapeadas em uma fila circular.

3.4 Conjunto de instruções

A arquitetura Tesla da NVIDIA, define instruções que normalmente não são encontradas em CPU's. O conjunto de instruções é chamado de *parallel thread execution* (PTX) e compreende operações típicas como, ponto flutuante, inteiro, lógica, acesso a memória, controle de fluxo, e também funções especiais como cálculo de seno e raiz quadrada, além de operações de textura. Operandos fonte são valores escalares de 32 ou 64 bits em registradores, um valor imediato ou uma constante. Instruções de memória e textura transferem escalares ou vetores de 2-4 componentes, até o total de 128 bits. As funções especiais são limitadas a 32 bits (*floating-point*).

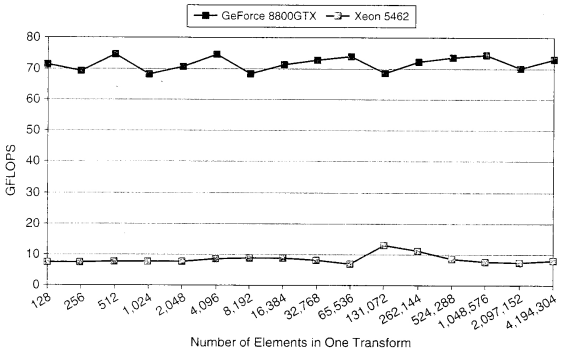

Figure 8: Comparação de implementação da Transformada de Fourier em GPU e CPU. Extraído de [12]

A figura 8 exibe a comparação feita em [12] da execução da transformada de Fourier rápida (*Fast Fourier Transforms*) entre uma GPU GeForce 8800 GTX 1.35GHz e uma CPU quad-Core Intel Xeon E5462 2.8GHz. Podemos observar que a transformada de Fourier quando implementada na GPU, teve um desempenho muito superior em relação a CPU. Isso acontece principalmente devido às instruções especiais da GPU, que possui instruções de seno e co-seno, pois a transformada é resumidamente, um conjunto de soma de senos e co-senos.

3.5 Sistema de Memória

Fora a GPU em si, o sistema de memória é o fator mais importante para determinar o desempenho de um sistema gráfico. O sistema de memória deve suportar altas taxas de transferências, a GTX-280, por exemplo, possui uma taxa de 1107 MHz, sendo possível processar em média 50 pixels por ciclo de clock. Em uma GPU existem vários níveis de memória, memória principal (DRAM), memória compartilhada, memória local e memória de constantes. De acordo

com [12], o sistema de memória de uma GPU deve ser capaz de satisfazer as seguintes características: o sistema deve prover uma grande largura de banda, consequentemente exigindo uma grande quantidade de pinos entre a GPU e seus dispositivos de memória; suporte a utilização de técnicas de compressão de dados, mesmo com perdas (quando avaliado pelo programador); utilizar vários níveis de cache para reduzir a quantidade de tráfico *off-chip* necessário e assegurar que ciclos gastos movendo dados são usados sempre que possível.
Para atender a demanda da largura de banda, a DRAM é arranjada em vários blocos, na GTX-280 são 8 blocos. Cada bloco possui um controlador de memória (*memory management unit - MMU*) independente.
As memórias compartilhadas ficam localizadas dentro dos *streaming multiprocessors*, elas são visíveis somente para as *threads* que pertencem aos seus respectivos *streaming processors*. Um dos benefícios desse tipo de organização é que não existe competição pela largura de banda com a memória principal.
As memórias locais são visíveis somente pelas threads que as estão utilizando, enquanto as memórias de constantes são visíveis por todas as *threads*. As memórias de constantes guardam valores escalares, matrizes de rotação, espelhamento, entre outros.

$$[x'y'z'1] * M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure 9: Matriz de rotação. Exemplo de matriz que é armazenada na memória de constantes

As modernas GPU's também são capazes de traduzir endereços virtuais para físicos. Todas as unidades de processamento geram endereços virtuais de 40 bits. Para instruções de *load* e *store* são utilizados 32 bits, sendo estendido para 40 bits, adicionando um offset. Esse trabalho é realizado pela MMU.
Em conjunto com o sistema de memória, o *raster operation processor* (ROP) captura os dados dos *streaming multiprocessors* para exibi-los, mas antes é realizada a compressão dos dados (compressão sem perda de cor e profundidade) e a interpolação. Essa operação é realizada em uma memória própria e após finalizada, os dados são enviados via *interconnection network* para serem exibidos na tela.

4. PROGRAMAÇÃO

Como mostrado nas seções anteriores, em aspectos de hardware a GPU é um poderoso processador paralelo, no entanto é necessário desenvolver aplicativos voltados (paralelizáveis) para essa arquitetura, a fim de tirar o máximo proveito. Os 2 maiores fabricantes de GPU's disponibilizam bibliotecas de programação para propósito geral, o *Compute Unied Device Architecture* (CUDA), mantido pela NVIDIA e a *Open Computing Language* (openCL), mantida pelo consórcio tecnológico Khronos. É possível executar o código de ambas as bibliotecas em uma CPU, entretanto o desempenho em uma GPU será superior, visto que sua arquitetura explora muito melhor o paralelismo de dados e de *threads*.

Podemos observar um código tradicional em linguagem C para CPU

```
void add_matrix
(float* a, float* b, float* c, int N)
{
  int index;
  for(int i = 0; i < N; i++)
    for(int j = 0; j < N; j++)
    {
      index = i + j*N;
      c[index] = a[index] + b[index];
    }
}

int main()
{
   add_matrix( a, b, c, N);
}
```

A mesma lógica anterior mas em CUDA

```
__global__ add_matrix
(float* a, float* b, float* c, int N)
{
 int i = blockIdx.x * blockDim.x + threadIdx.x;
 int j = blockIdx.y * blockDim.y + threadIdx.y;
 int index = i + j*N;
 if (i < N && j < N)
   c[index] = a[index] + b[index];
}

int main()
{
 dim3 dimBlock(blocksize, blocksize);
 dim3 dimGrid(N/dimBlock.x, N/dimBlock.y);
 add_matrix<<<dimGrid, dimBlock>>>(a,b,c,N);
}
```

5. INTEL LARRABEE

Em meados de 2008 a Intel anunciou um sistema de um único chip que contém CPU e GPU, mais precisamente uma *accelerated processing unit* (APU), com o codinome de Larrabee [13]. Esse protótipo tem como objetivo tornar a computação paralela (normalmente realizada pela GPU e *Clusters de CPU*) mais programável em aplicações de propósito geral, em particular. Apesar das GPU's estarem cada vez mais satisfazendo essa necessidade, ainda possuem algumas limitações, como um melhor controle dos processos por parte do programador. Tipicamente em uma GPU os processos são agendados pela lógica de rasterização dos pixels.
A arquitetura do Larrabee é composta por múltiplas CPU's baseada no x86 e cada *core* contém uma unidade de processamento vetorial de 512 bits, capaz de processar até 16 operações de ponto flutuante por vez (*single precision*). Possui um *ring bus* de 1024 bits para comunicação entre *cores* e memória, 32 KB de cache para texturas e uma cache L2 que também é utilizada no processo de renderização. Em um cálculo de desempenho máximo (possível), considerando 32 *cores* 2GHz, teoricamente alcançaria 2TFLOPS, mas em uma demonstração no fim de 2009, Larrabee foi capaz de alcançar 1006 GFLOPS.
Ao contrário de uma GPU, o Larrabee não possui implementações específicas no hardware para rasterização, renderização e interpolação, tais funcionalidades são implementadas pelo seu compilador, de acordo com [13].

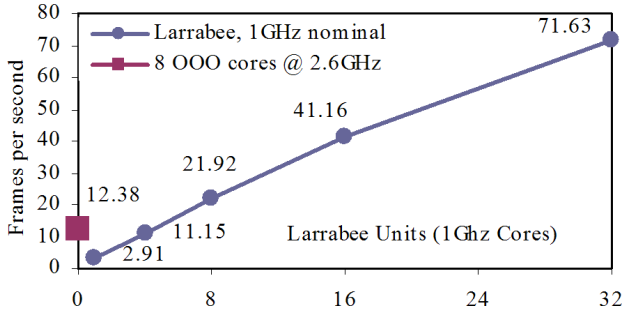

Figure 10: Comparativo entre processador Intel Xeon 2.6GHz com 8 *cores* e Intel Larrabbe de 1 GHz com diversos *cores*. Figura extraída de [13]

6. CONCLUSÃO

Nesse trabalho foi apresentada a arquitetura básica de uma GPU tradicional, alguns aspectos de sua programação e características do processador Intel Larrabbe.

As GPU's apareceram como uma solução interessante para aplicações que possam ser paralelizáveis, principalmente por terem um custo relativamente baixo em relação à quantidade de CPU's necessárias para gerar o mesmo poder computacional. Outro fator importante é a preocupação dos fabricantes em manter bibliotecas de fácil utilização para explorar sua capacidade de processamento, além da compatibilidade com a CPU.

A exemplo do que tem ocorrido nas últimas décadas com outros itens de hardware que foram incorporados a CPU, certamente o destino das GPU's dada a evolução da microeletrônica, seja compartilhar o mesmo chip com a CPU, o que já é possível observar no Intel Larrabbe que veio para reforçar ainda mais essa teoria.

7. REFERÊNCIAS

[1] AnandTech - NVIDIA's 1.4 billion transistor GPU: GT200 arrives as the GeForce GTX 280 & 260. http://www.anandtech.com/show/2549/2, 6 2011.

[2] ATI CrossFireX. http://game.amd.com/us-en/crossfirex_about.aspx, 6 2011.

[3] High performance computing (hpc) - supercomputing with nvidia tesla. http://www.nvidia.com/object/tesla_computing_solutions.html, 6 2011.

[4] NVIDIA SLI technology for NVIDIA Quadro graphics boards. http://www.nvidia.com.br/object/quadro_sli_br.html, 6 2011.

[5] Processador móvel nvidia tegra. http://www.nvidia.com.br/object/tegra-2-br.html, 6 2011.

[6] A. Castonguay and Y. Savaria. Architecture of a hypertransport tunnel. In *Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on*, page 4 pp., 0-0 2006.

[7] D. De Caro, N. Petra, and A. Strollo. High-performance special function unit for programmable 3-d graphics processors. *Circuits and Systems I: Regular Papers, IEEE Transactions on*, 56(9):1968 –1978, sept. 2009.

[8] Minh Tri Do Dinh. Gpus - graphics processing units. *Vertiefungsseminar Architektur von Prozessoren, SS 2008*, pages 3 – 4, 2008.

[9] H. Fuchs, S.M. Pizer, Li Ching Tsai, S.H. Bloomberg, and E.R. Heinz. Adding a true 3-d display to a raster graphics system. *Computer Graphics and Applications, IEEE*, 2(7):73 –78, sept. 1982.

[10] P. Maciol and K. Banas. Testing tesla architecture for scientific computing: The performance of matrix-vector product. In *Computer Science and Information Technology, 2008. IMCSIT 2008. International Multiconference on*, pages 285 –291, oct. 2008.

[11] Marc Olano, Kurt Akeley, John C. Hart, Wolfgang Heidrich, Michael McCool, Jason L. Mitchell, and Randi Rost. Real-time shading. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, New York, NY, USA, 2004. ACM.

[12] D.A. Patterson and J.L. Hennessy. *Computer organization and design: the hardware/software interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Elsevier Morgan Kaufmann, 2008.

[13] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM Trans. Graph.*, 27:18:1–18:15, August 2008.