O potencial das Graphics Procesing Units

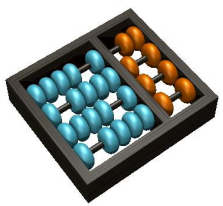
MO401 - Arquitetura de Computadores

Alunos:
Juan Salamanca Guillén - ra134063
Rafael Auler - ra045840
Junior Fabian Arteaga - ra123542
Professor: Prof. Paulo Centoducatte

Universidade Estadual de Campinas
Instituto de Computação

18 de junho de 2012

UNICAMP



Sumário

Sumário

Introdução

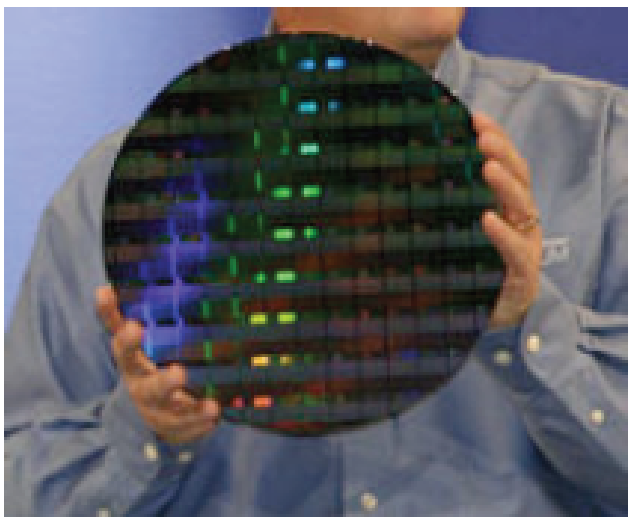
- As GPUs evoluíram nesta última década: Processadores programáveis de alto desempenho.
- Novos termos: GPGPU e GPU Computing.
- Aplicações aproveitadas pela GPU têm:
  - Expressivos requisitos computacionais.
  - Altamente paralelizáveis.
  - Priorizam vazão à latência.

Por que GPUs?

- Cada vez é mais difícil extrair ILP do fluxo de instruções.
- A parte de controle domina os microprocessadores.
  - É complexo, difícil de construir e verificar.
  - Toma muito tempo.
  - Não faz cálculos .
    - Intel Core Duo: 48 GFLOPS
    - NVIDIA G80: 330 GFLOPS
- Industria está indo de instruções por segundo a instruções por watt.

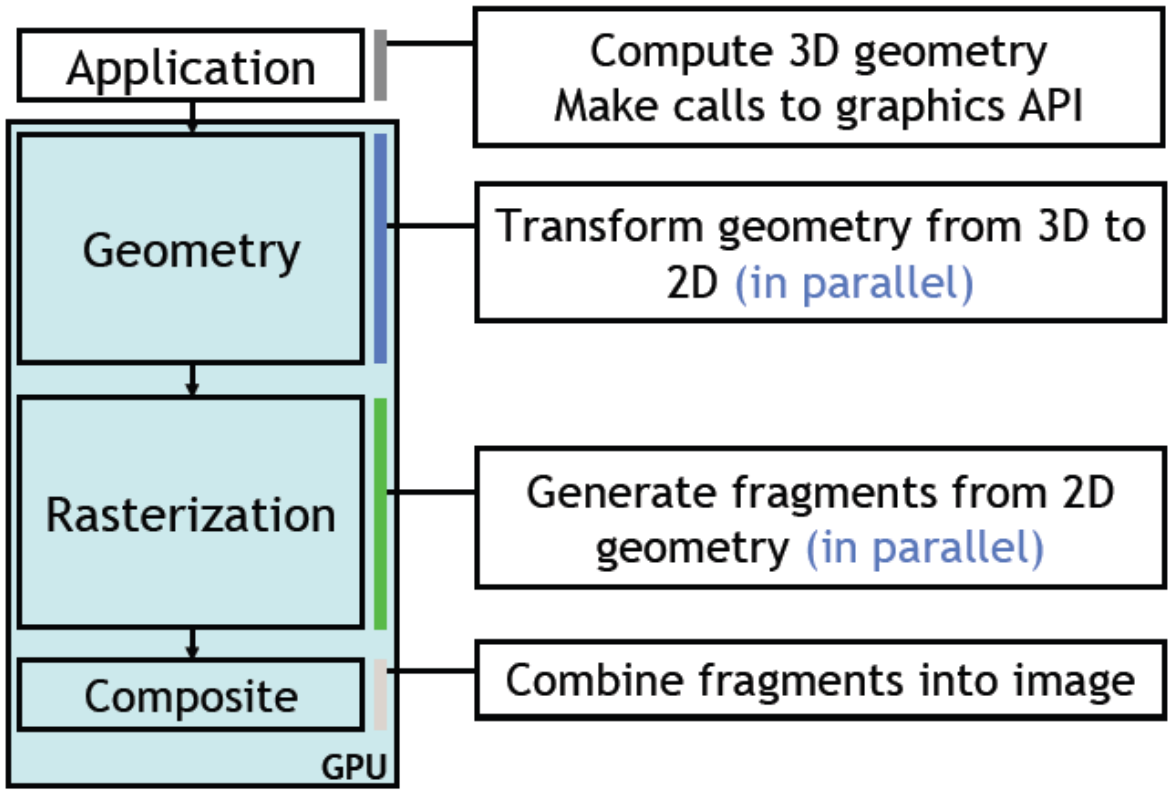
Indo para o paralelismo

- Agora estamos num tempo de inovação de arquiteturas.
  - GPU nos permitem explorar centenas de processadores agora, mas 10 anos antes não.
- A maioria de fabricantes de CPU suportam multicore.
- Interesse por programação de propósito geral usando GPUs.
- As universidades devem ensinar a pensar em paralelo?



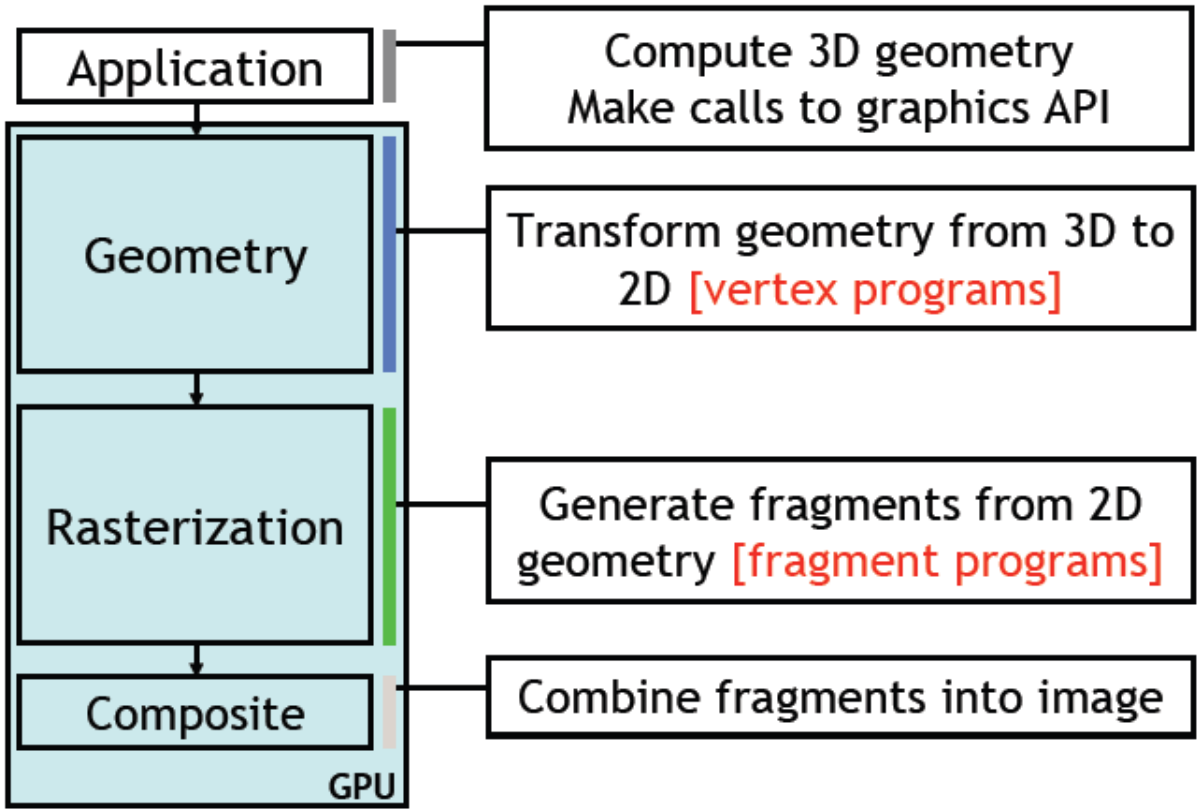
Sumário

O pipeline de renderização



- Pipeline de funções fixas com partes configuráveis.

O programável pipeline



- Dispositivo programável com funcões fixas de suporte.

O pipeline de gráficos mapeado

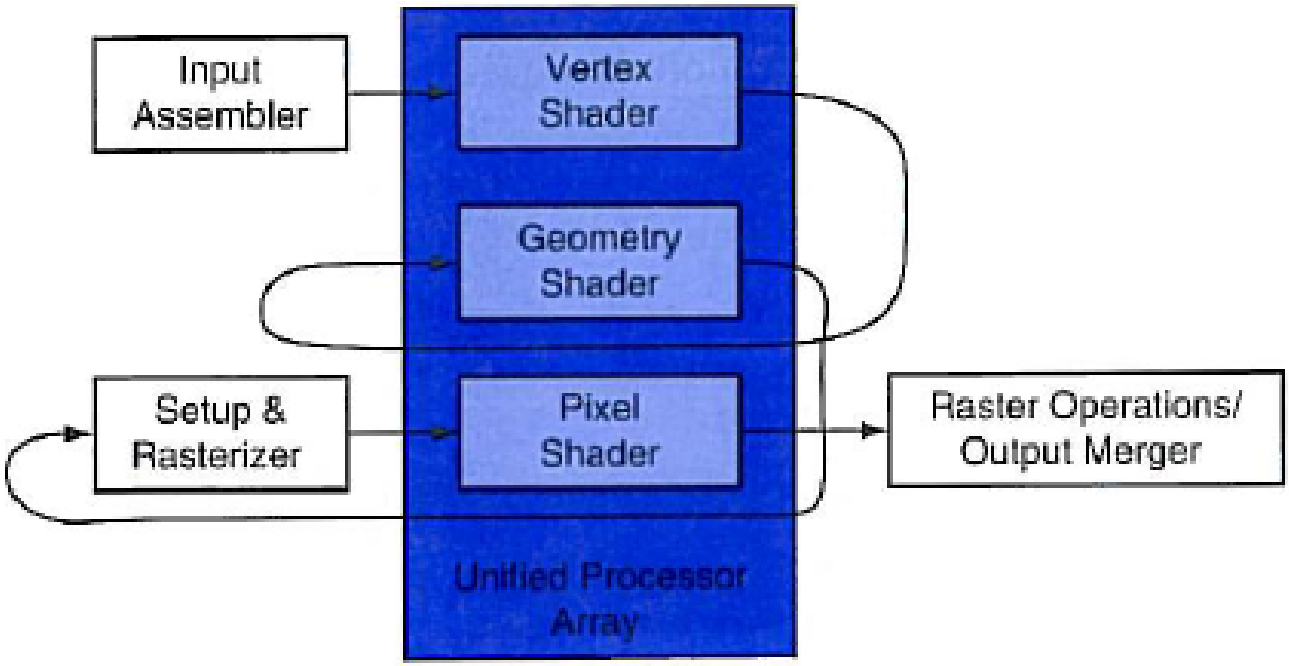


Figura : O pipeline de gráficos com os estágios programáveis mapeados à matriz de processadores. Hennessy e Patterson (2008).

Arquitetura

- Um longo pepelline com muitos estágios, cada um acelerado por hardware específico.
- Está baseada numa matriz de muitos processadores programáveis.
- Organização tipica: uma matriz de streaming processors cores (SP) distribuídos em alguns multithreaded streaming multiprocessors (SM).
- São SIMD (uma instrução para varios dados), aind mais são SIMT (uma instrução é executada por múltiplas threads).
- Conjunto de instruções especializado.
- Sistema de memoria com varios níveis de memoria:
  - Memoria principal fora do chip (DRAM).
  - Memoria compartilhada.
  - Memoria local.
  - Memoria de constantes.

Arquitetura

- Construído sobre unidades programáveis.
- Unified shader.

Figura : Arquitetura da GPU NVIDIA GetForce 8800 GTX. Owens et al. (2008).

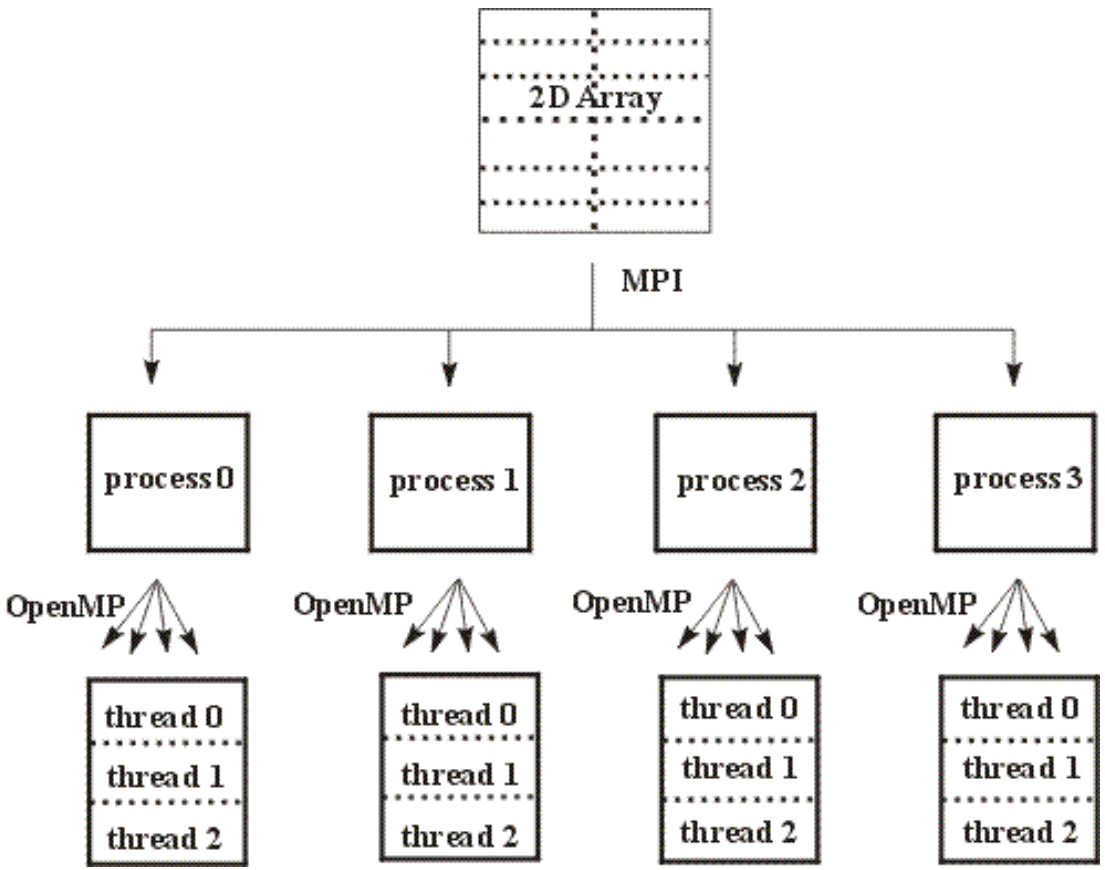
Arquitetura

- GPU é uma arquitetura massivamente paralela.
  - Muitos problemas são mapeados bem ao estilo que trabalham as GPUs.
  - GPUs têm grande quantidade de capacidade para fazer aritmética.
  - O incremento da programabilidade no pipeline.
- Novas características são mapeadas bem a GPGPU.
  - Unified shaders.
  - Acesso direito para usar suas unidades de processamiento em novos APIs.
- Desafíos
  - Cómo usar melhor o hardware da GPU?
  - Novas técnicas, modelos de programação, linguagems, ...

Sumário

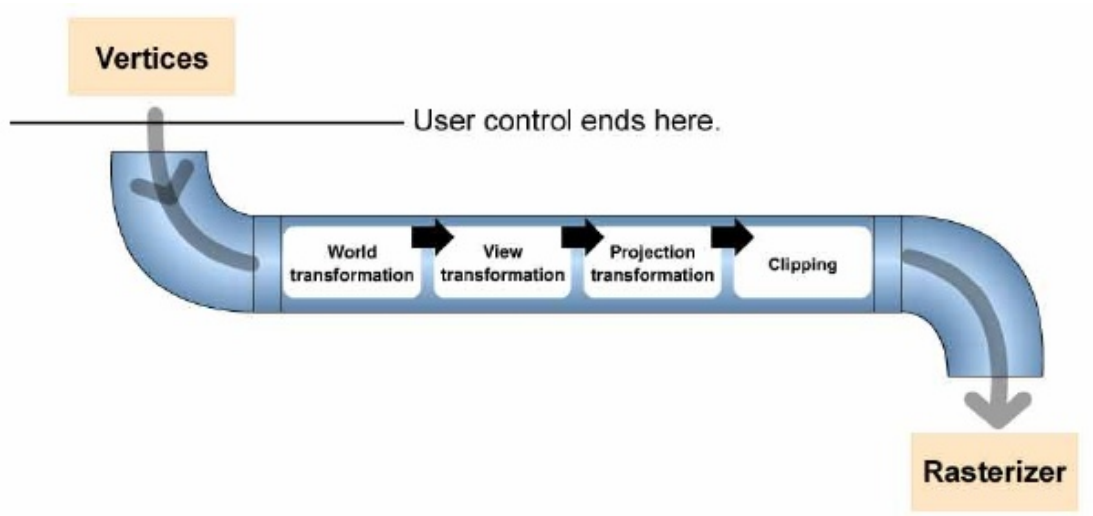
Programação de GPUs

- Busca por abstração mais fácil
- Programação paralela: desafios
- MPI e OpenMP



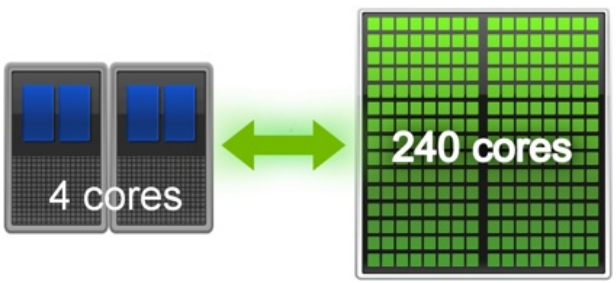
Programação de GPUs

- Mercado de jogos eletrônicos levaram GPUs a virar *commodity*
- Programação paralela em GPUs: aspectos iniciais



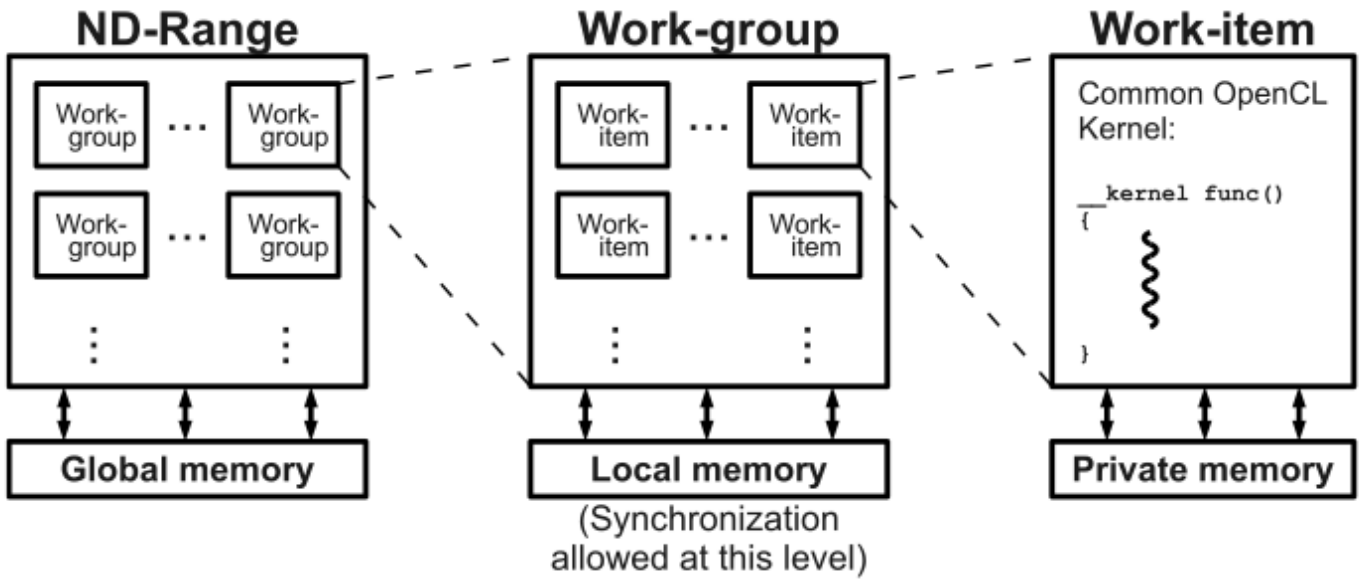
Programação de GPUs

- Fabricantes se dão conta de que GPUs podem ser usadas para computação de propósito geral.
- OpenCL (Open Computing Language) e CUDA (Compute Unified Device Architecture)



OpenCL

- Surgiu com a Apple
- Consórcio Khronos Group (NVidia, ATI, Intel, Apple etc.)



OpenCL

```
faca_algo_na_cpu();
kernels (num_bloco, num_thread);
faca_outra_coisa_na_cpu();
sincronize();
```

- Kernels lembram vagamente diretivas paralelas OpenMP
- Não há sincronização entre blocos: independentes

OpenCL - Kernel

```
__kernel void
vectorAdd(__global const float * a,
__global const float * b,
__global float * c)
{
// Indice do vetor
int nIndex = get_global_id(0);
c[nIndex] = a[nIndex] + b[nIndex];
}
```

OpenCL - Host

```
int main(void){
// Informa ao runtime que queremos despachar
// a tarefa para uma GPU
cl_context context = clCreateContextFromType(0,
CL_DEVICE_TYPE_ GPU, ...
// get the list of GPU devices associated
// with context
clGetContextInfo(...
// create a command-queue
cl_cmd_queue cmd_queue = clCreateCommandQueue(...
cl_mem memobjs[1];
// Aloca buffer de entrada
memobjs[0] = clCreateBuffer(...
// Cria o programa e o kernel
cl_program program = clCreateProgramWithSource(...
cl_int err = clBuildProgram(...
cl_kernel kernel = clCreateKernel(...
// Configura argumentos do kernel e o executa
err = clSetKernelArg(...
err = clEnqueueNDRangeKernel(....
// Lê resultado
err = clEnqueueReadBuffer(
...
}
```

Sumário

NVIDIA

- NVIDIA é uma empresa multinacional que fabrica peças de computador - *(GeForce)*. AMD - *Radeon*.
- As primeiras GPUs foram projetadas como aceleradores gráficos, suportando somente pipelines de função fixa específicos.
- A programação para as GPUs da primeira geração não era nada fácil.
- Brook: o primeiro modelo de programação na linguagem C com construções de paralelismo de dados.

NVIDIA - *CUDA*

- Um hardware extremamente rápido tinha que ser combinado a ferramentas intuitivas de software e hardware.
- NVIDIA apresentou a CUDA *(Compute Unified Device Architecture)* em 2006, a primeira solução do mundo para computação de propósito geral em GPUs.
  - O CUDA não é uma nova linguagem de programação, mas uma biblioteca C.
  - CUDA é vista como um co-processador genérico, e não apenas gráfico.

NVIDIA - *CUDA*

- A grande novidade é a maneira eficiente que o CUDA possibilita o desenvolvimento de aplicações que exploram ao máximo o paralelismo de dados.
- Um mesmo trecho de código é executado em paralelo para pequenos blocos de dados, com a existência de várias pequenas caches e níveis de hierarquia de memoria que escondem a latência de acesso.

Figura : Diferença arquiteturais entre CPU e GPU. NVIDIA (2008)

NVIDIA - *CUDA*

- Desde a sua introdução em 2006, CUDA têm sido amplamente utilizada através de milhares de aplicações e trabalhos de pesquisa.
- Atualmente, mais de 300 milhões de GPUs instaladas em notebooks, workstations, clusters e supercomputadores utilizam CUDA.
- Aplicações em diferentes áreas: astronomia, biologia, química, física, manufatura, finanças etc.

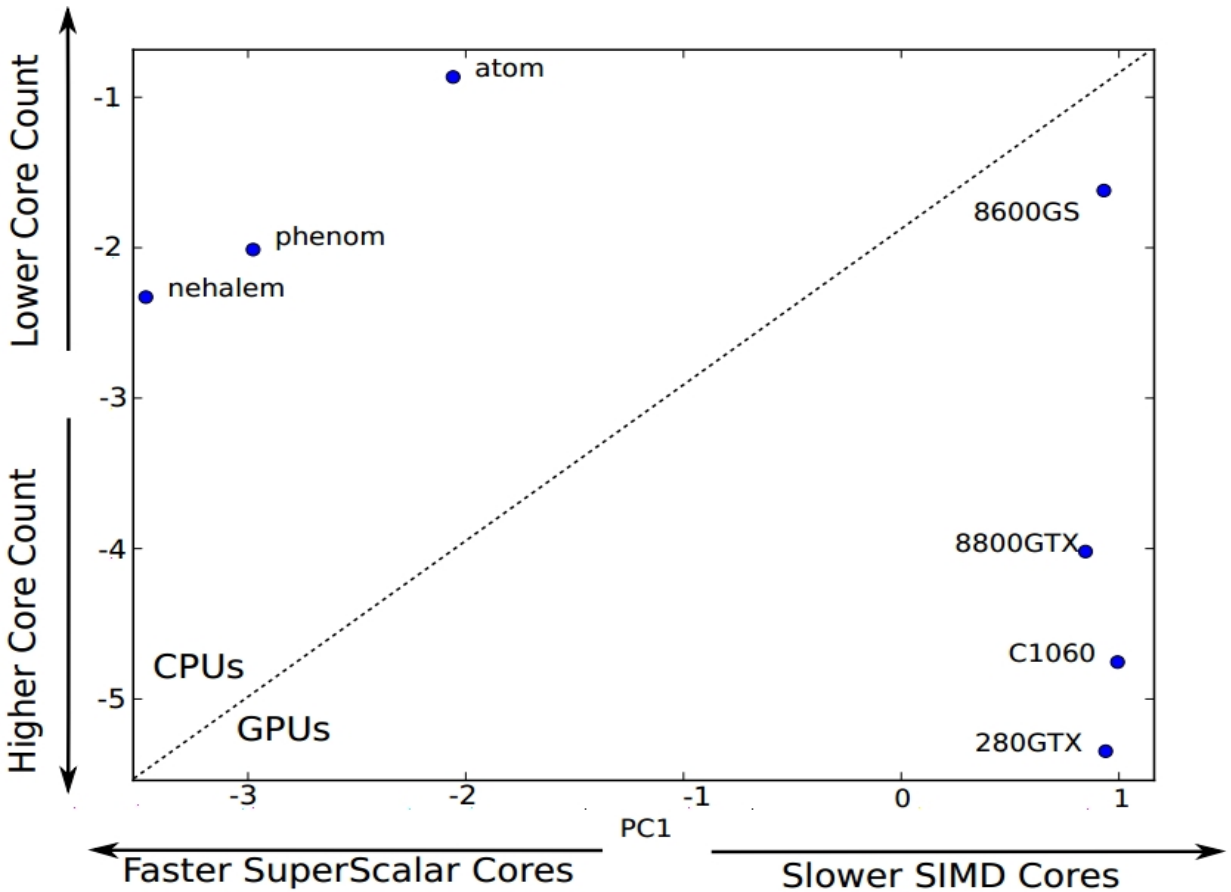
NVIDIA - *CUDA*

- Alguns exemplos:
  - Identificação de placas ocultas em artérias.
  - Análise do fluxo de tráfego aéreo.
  - Visualização de moléculas.

Sumário

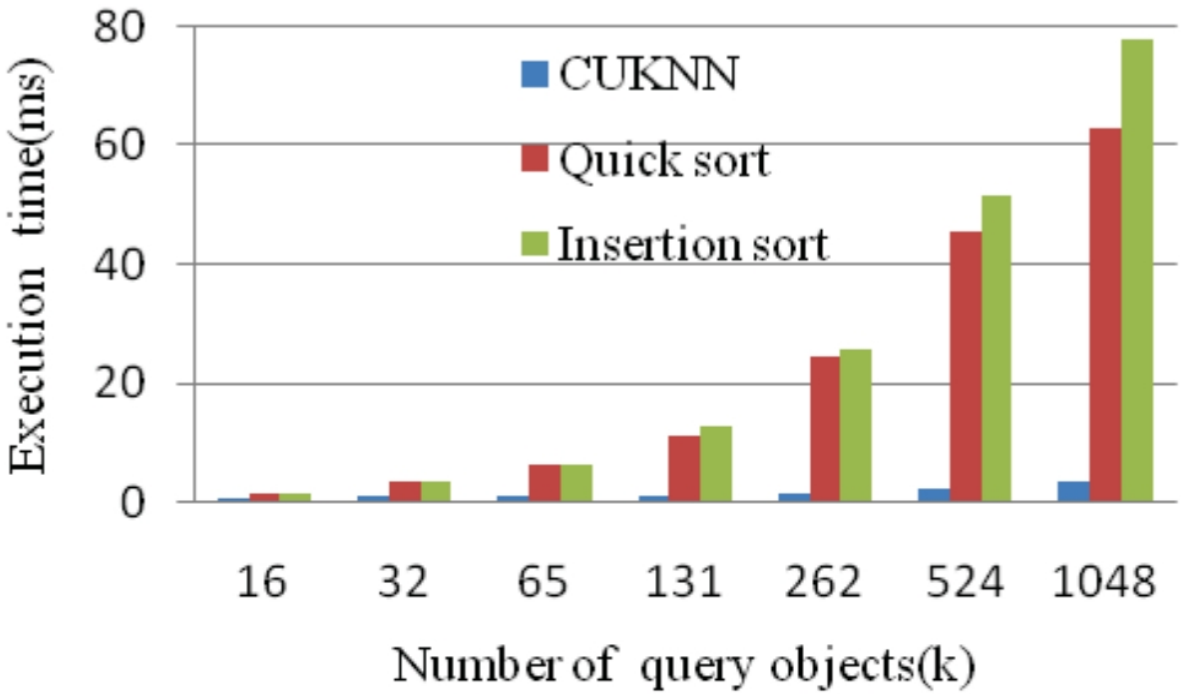
Comparação entre CPUs e GPUs

- Uma técnica de modelagem da performance entre uma CPU e uma GPU é apresentada no trabalho de Kerr et al. (2010).



Comparação entre CPUs e GPUs

- Outra aplicação para medir as diferenças entre CPUs e GPU é a implementação do algoritmo que calcula os K vizinhos mais próimos (KNN).
- No trabalho de Lian et al. (2009), os autores propõem o algoritmo de CUKNN.



Comparação entre CPUs e GPUs

- Em Minh (2008) o autor apresenta a seguinte tabela para medir as diferenças entre GPUs e CPUs.
- *std:sort vs. Bitonic Merge Sort* (uma versão do merge sort implementada em paralelo).

std:sort Pentium 4 3.0 GHz			**Bitonic Merge Sort:** Float Data NVIDIA GeForce 6800 Ultra		
N	**Sorts/Sec**	**Keys/Sec**	**N**	**Sorts/Sec**	**Keys/Sec**
256^2	82.5	5.4 M	256^2	90.07	6.1 M
512^2	20.6	5.4 M	512^2	18.3	4.8 M
1024^2	4.7	5.0 M	1024^2	3.6	3.8 M

Comparação entre CPUs e GPUs

- No trabalho de Adams et al. (2007) os autores utilizaram GPU para resolver as equações de Maxwell em campos eletromagnéticos, mediante a execução de simulações de Diferenças finitas no domínio do tempo *(Finite Difference Time Domain - FDTD)*.
- Os autores verificaram que uma GPU *GeForce 8800 GTX* é 429 vezes mais rápida que uma CPU *Opteron 270*. Enquanto que, uma Core 2 Duo T7600 (2.33 Ghz.) foi unicamente 2 vezes mais rápida que a CPU.

Sumário

Conclusões

- A programação em GPU não é realizada seguindo o modelo tradicional de programação de CPU, mas para obter maior eficiência devemos trabalhar com o modelo conhecido como SIMD ou SIMT.
- Para conseguir altas velocidades com a GPU é necessário:
  - Formatar os vetores em arranjos bidimensionais.
  - Executar processamento de grandes quantidades de dados.
  - Executar grande número de operações simples por itens de dados.

Conclusões

- A abordagem da NVIDIA com CUDA é de uma linguagem de alto nível e fácil de utilizar para desenvolver aplicações reais.
- Uma das maiores vantagens de utilizar GPU é que o custo é muito menor em comparação com um computador equivalente para realizar a mesmas tarefas.
- As GPUs tem um poder de processamento maior e um gasto energético menor.

Sumário

Referências

1. J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. *GPU Computing*. Proceedings of the IEEE, 96(5):879;899, Maio 2008.
2. NVIDIA. *CUDA: Programming Guide 2.0 - NVIDIA*, 2008.
3. A. Kerr, G. Diamos, and S. Yalamanchili. Modeling GPU-CPU workloads and systems. In Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, pages 31-42, New York, USA, 2010. ACM.
4. S. Liang, Y. Liu, C. Wang, and L. Jian. A CUDA-based parallel implementation of K-nearest neighbor algorithm. In Cyber-Enabled Distributed Computing and Knowledge Discovery, 2009. pages 291-296, oct. 2009.
5. Minh Tri. GPUs - Graphics Processing Units. Institute of Computer Science - University of Innsbruk, 2008.
6. S. Adams, J. Payne, and R. Boppana. Finite Difference Time Domain (FDTD) Simulations using Graphics Processors. In Proceedings of the 2007 DoD High Performance Computing Modernization Program Users Group Conference, pages 334-338, Washington, USA, 2007. IEEE Computer Society.
7. D. Patterson and J. Hennessy. Computer organization and design: the hardware/software interface. The Morgan Kaufmann Series in Computer Architecture and Design, 2008.

Obrigado!