Uma introdução para computação paralela de modelos massivos

Adriano Brito Pereira 1021752
apereira @ inf.puc-rio.br

Departamento de Informática
Novembro / 2010

- Resultados obtivos com Manta Framework x Cuda

- Cenário em 2006 (Manta) x Cenário em 2009.

- Conclui-se que o paper sobre o Boeing 777 foi reforçado para divulgação do Manta framework.

• Diferentes objetivos produzem diferentes designs

• GPU assume que a carga de trabalho é altamente paralela
• CPU deve ser boa em tudo, paralela ou não

• CPU: minimização da latência experimentada por 1 thread
• GPU: maximização da vazão de todas as threads

• Idéias arquiteturais: SIMT (Single Instruction Multiple Thread) – threads executam em grupos de 32 (warps). Hardware multithreading. Threads com todos os recursos necessários para executar.

✓ Motivação

• Quantidade de dados produzidos pela engenharia de design excede as capacidades de renderização

• Modelos geométricos de dados podem produzir dezenas de gibabytes.

• Como o uso de paralelismo e uma arquitetura escalável pode ajudar a realizar tarefas no mundo real.

✓ Objetivo

• Aprender como desenvolver programas que processem modelos massivos através de paralelismo

• Adquirer alta performance

• Garantir os aspectos funcionais e manuteníveis do modelo

• Garantir escalabilidade para os modelos futuros

✓ Conhecimento técnico envolvido

- Princípios e padrões da programação paralela
- Arquitetura física de processadores e suas condições
- API, frameworks, ferramentas e técnicas

✓ Tecnologia mínima utilizada

- PC multiprocessado com um GPU habilitado com CUDA

✓ Relembrar é viver

• “O número de transistores em um circuito integrado duplica a cada 2 anos.” - Lei de Moore

• Paralelismo a nível de instrução (especulação, execução fora de ordem,...)

• Paralelismo a nível de dado (vetores unitários, SIMD, GPU,...)

• Paralelismo a nível de thread (multithreading, multicore, intel core2, AMD Phenom, Nvidia Fermi,...)

✓ A nova Lei de Moore

- Computadores não ficam mais rápidos, apenas mais espertos
- Você deve repensar seus algoritmos para serem paralelos
- Paralelismo por dado é a solução mais escalável

• Diferentes objetivos produzem diferentes designs

• GPU assume que a carga de trabalho é altamente paralela
• CPU deve ser boa em tudo, paralela ou não

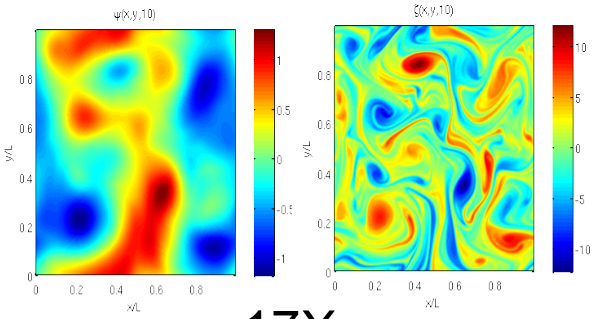
• CPU: minimização da latência experimentada por 1 thread
• GPU: maximização da vazão de todas as threads

• Idéias arquiteturais: SIMT (Single Instruction Multiple Thread) – threads executam em grupos de 32 (warps). Hardware multithreading. Threads com todos os recursos necessários para executar.

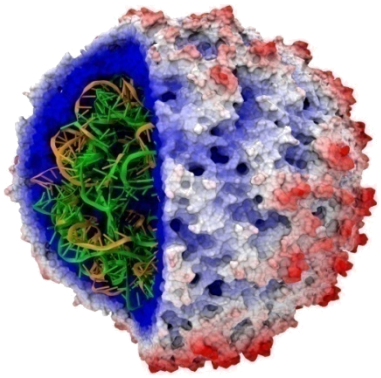
- CUDA
- Modelo de programação paralela escalável
- Familiar ao desenvolvimento em C/C++
- Computação em série-paralela heterogênea

- CUDA (Motivações)

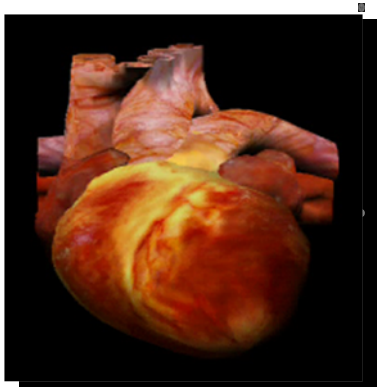
45X



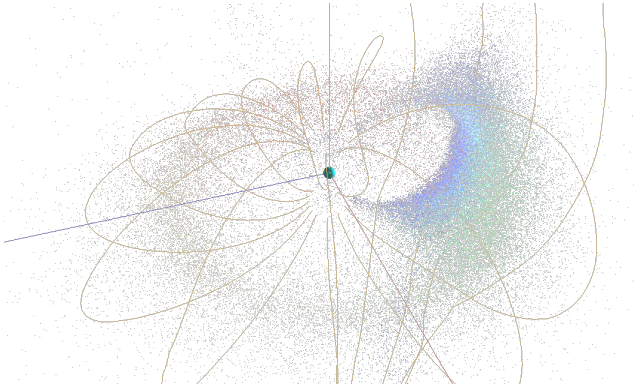
17X



110-240X



13–457x



100X

- CUDA - Vantagens
- Permite os programadores focarem em algoritmos paralelos
- Não é uma linguagem mecânica para paralelismo
- Bom para mapeamento com hardware
- Altamente escalável em número de threads

- CUDA - Abstrações
- Hierarquia de threads concorrentes
- Sincronização primitivas
- Modelo de memória compartilhada para threads que cooperam entre si.

- CUDA – Modelo de paralelismo
- Virtualização da memória física.
- Schedulamento em hardware não preemptiva.
- Blocos independentes

- CUDA – Exemplo de implementação

```
__global__ void my_kernel() { }
__device__ float my_device_func() {

__constant__ float my_constant_array[32];
__shared__   float my_shared_array[32];

dim3 grid_dim(100, 50);  // 5000 thread blocks
dim3 block_dim(4, 8, 8); // 256 threads per block
my_kernel <<< grid_dim, block_dim >>> (...); // Launch kernel

dim3 gridDim;   // Grid dimension
dim3 blockDim;  // Block dimension
dim3 blockIdx;  // Block index
dim3 threadIdx; // Thread index
void __syncthreads(); // Thread synchronization
```

- CUDA – Exemplo de implementação

Example: `vector_addition`

```
__global__ void vector_add(float* A, float* B, float* C)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    C[i] = A[i] + B[i];
}


int main()
{

    vector_add<<< N/256, 256>>>(d_A, d_B, d_C);
}
```

Projetos em andamento

Application	Description	Source	Kernel	% time
H.264	SPEC ‘06 version, change in guess vector	34,811	194	35%
LBM	SPEC ‘06 version, change to single precision and print fewer reports	1,481	285	>99%
RC5-72	Distributed.net RC5-72 challenge client code	1,979	218	>99%
FEM	Finite element modeling, simulation of 3D graded materials	1,874	146	99%
RPES	Rye Polynomial Equation Solver, quantum chem, 2-electron repulsion	1,104	281	99%
PNS	Petri Net simulation of a distributed system	322	160	>99%
SAXPY	Single-precision implementation of saxpy, used in Linpack’s Gaussian elim. routine	952	31	>99%
TPACF	Two Point Angular Correlation Function	536	98	96%
FDTD	Finite-Difference Time Domain analysis of 2D electromagnetic wave propagation	1,365	93	16%
MRI-Q	Computing a matrix Q, a scanner’s configuration in MRI reconstruction	490	33	>99%

Conclusões finais

• Resultados de paralelismo geralmente obtém melhores resultados.

• GPUs são massivamente paralelas a muitos cores de processadores.

• Paralelismo e escalabilidade são cruciais para o sucesso de renderização de modelos massivos.

✓ Bibliografia e fontes consultadas

Abe Stephens, 1 Solomon Boulos, 2 James Bigler, 1 Ingo Wald, 1 Steven Parker 1, **An Application of Scalable Massive Model Interaction using Shared-Memory Systems**, Eurographics Symposium on Parallel Graphics and Visualization (2006)

Parallel-Computing-Seminar-Report - http://www.scribd.com/doc/14564472/Parallel-Computing-Seminar-Report

CUDA – Parallel Computer Architecture
http://www.speedup.ch/workshops/w38_2009/pdf/speedup_workshop.pdf

Yongpeng Zhang, Frank Mueller, Xiaohui Cui, Thomas Potok
North Carolina State University, **A Programming Model for Massive Data Parallelism with Data Dependencies, 2007 - http://moss.csc.ncsu.edu/~mueller/ftp/pub/mueller/papers/pmea09.pdf**