A evolução da GPGPU: arquitetura e programação

Conrado S. Miranda

Laboratório de Identificação e Controle de Sistemas Dinâmicos
Departamento de Projeto Mecânico
Faculdade de Engenharia Mecânica
Universidade Estadual de Campinas

14 de junho de 2010

1. Introdução

2. Programação

3. Arquitetura

4. Performance

Histórico das GPUs

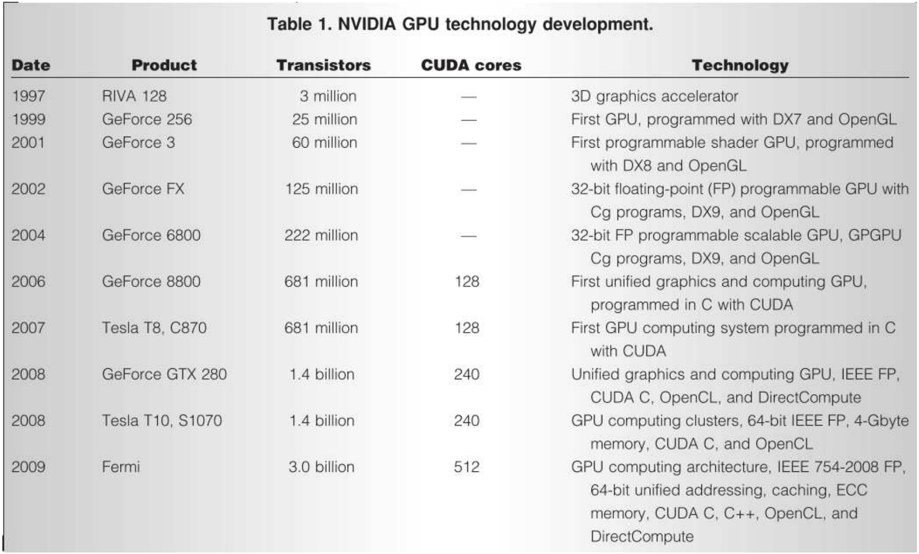
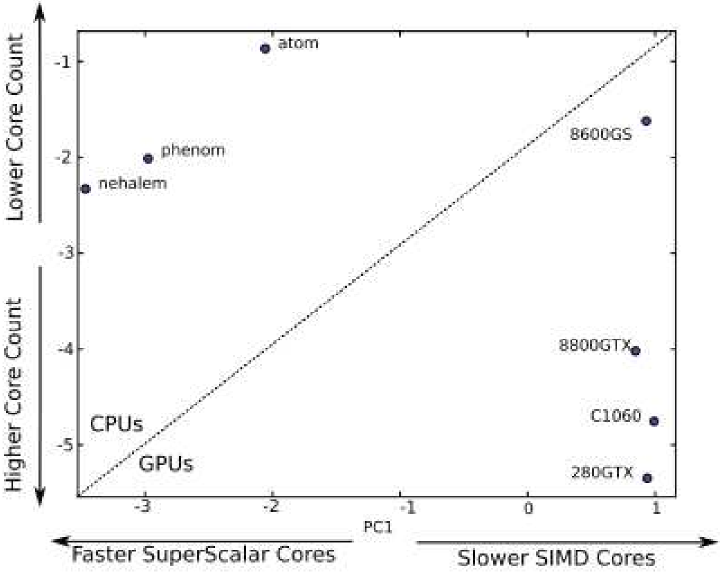


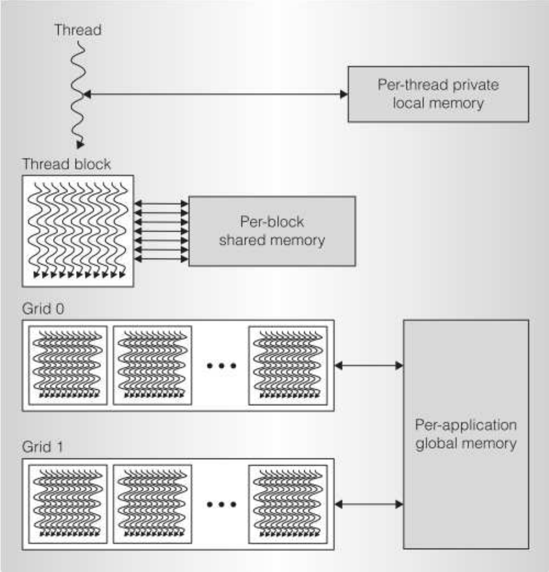
Table 1. NVIDIA GPU technology development.

Date	Product	Transistors	CUDA cores	Technology
1997	RIVA 128	3 million	—	3D graphics accelerator
1999	GeForce 256	25 million	—	First GPU, programmed with DX7 and OpenGL
2001	GeForce 3	60 million	—	First programmable shader GPU, programmed with DX8 and OpenGL
2002	GeForce FX	125 million	—	32-bit floating-point (FP) programmable GPU with Cg programs, DX9, and OpenGL
2004	GeForce 6800	222 million	—	32-bit FP programmable scalable GPU, GPGPU Cg programs, DX9, and OpenGL
2006	GeForce 8800	681 million	128	First unified graphics and computing GPU, programmed in C with CUDA
2007	Tesla T8, C870	681 million	128	First GPU computing system programmed in C with CUDA
2008	GeForce GTX 280	1.4 billion	240	Unified graphics and computing GPU, IEEE FP, CUDA C, OpenCL, and DirectCompute
2008	Tesla T10, S1070	1.4 billion	240	GPU computing clusters, 64-bit IEEE FP, 4-Gbyte memory, CUDA C, and OpenCL
2009	Fermi	3.0 billion	512	GPU computing architecture, IEEE 754-2008 FP, 64-bit unified addressing, caching, ECC memory, CUDA C, C++, OpenCL, and DirectCompute

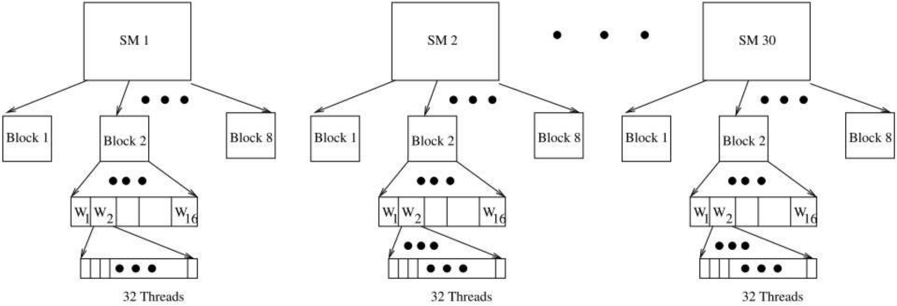
Comparação entre CPUs e GPUs



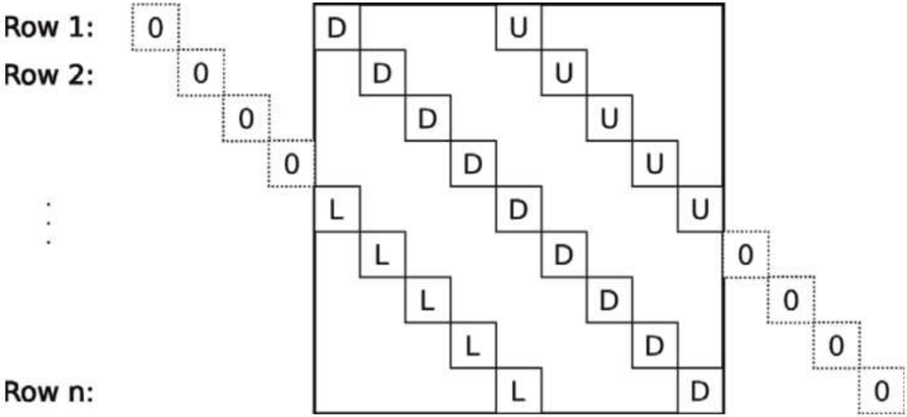
Memórias disponíveis para uma thread


Thread
Per-thread private local memory
Thread block
Per-block shared memory
Grid 0
Grid 1
Per-application global memory

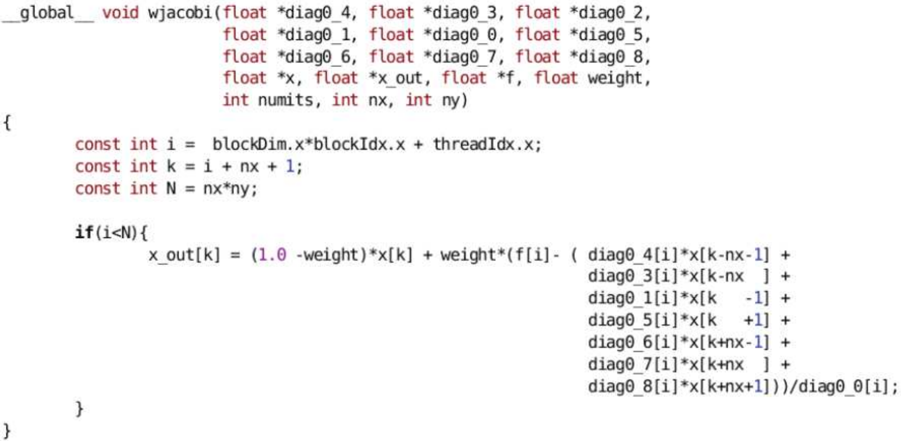
Hierarquia de threads



Organização da matriz de Jacobi



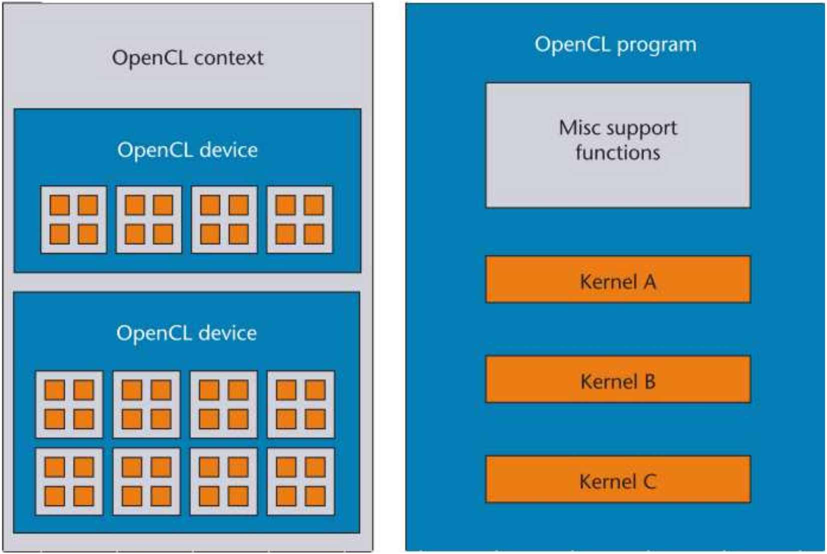
Implementação em CUDA da iteração de Jacobi



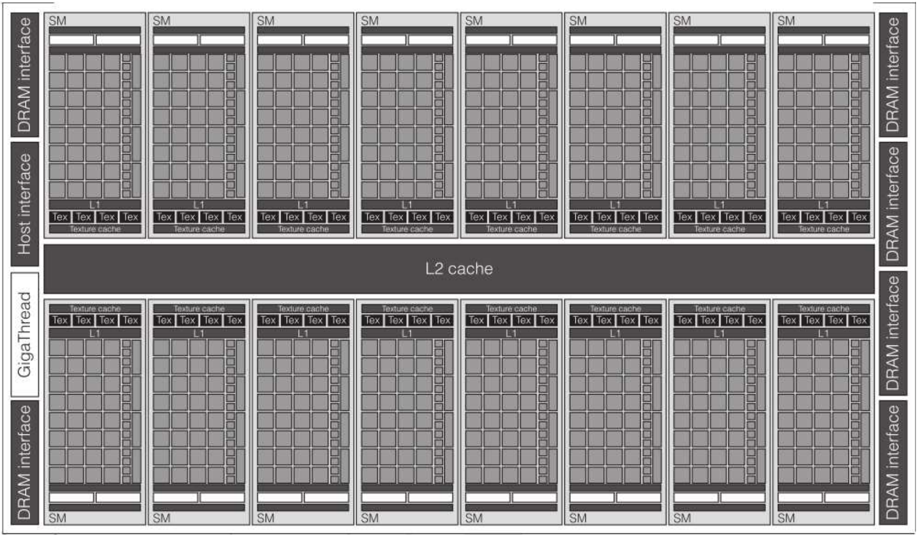
```
__global__ void wjacobi(float *diag0_4, float *diag0_3, float *diag0_2,
                        float *diag0_1, float *diag0_0, float *diag0_5,
                        float *diag0_6, float *diag0_7, float *diag0_8,
                        float *x, float *x_out, float *f, float weight,
                        int numits, int nx, int ny)
{
        const int i =  blockDim.x*blockIdx.x + threadIdx.x;
        const int k = i + nx + 1;
        const int N = nx*ny;

        if(i<N){
                x_out[k] = (1.0 -weight)*x[k] + weight*(f[i]- ( diag0_4[i]*x[k-nx-1] +
                                                                diag0_3[i]*x[k-nx  ] +
                                                                diag0_1[i]*x[k   -1] +
                                                                diag0_5[i]*x[k   +1] +
                                                                diag0_6[i]*x[k+nx-1] +
                                                                diag0_7[i]*x[k+nx  ] +
                                                                diag0_8[i]*x[k+nx+1]))/diag0_0[i];
        }
}
```

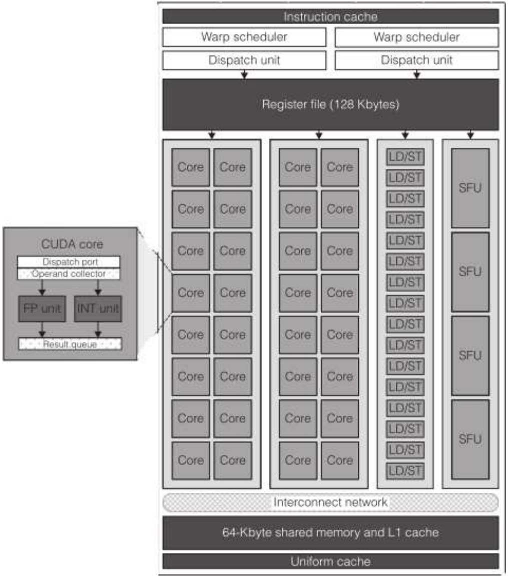
Modelo de programa em OpenCL



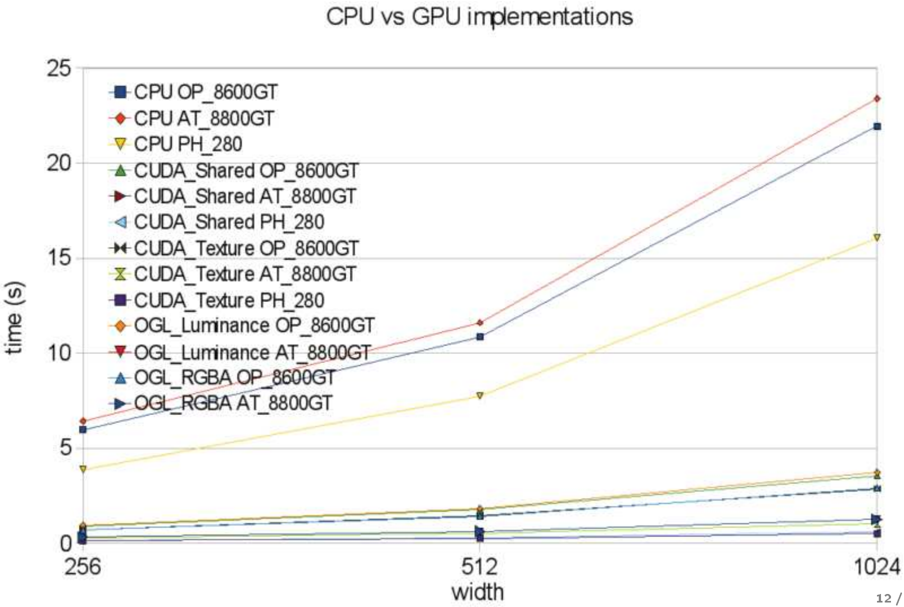
Arquitetura da GPU Fermi



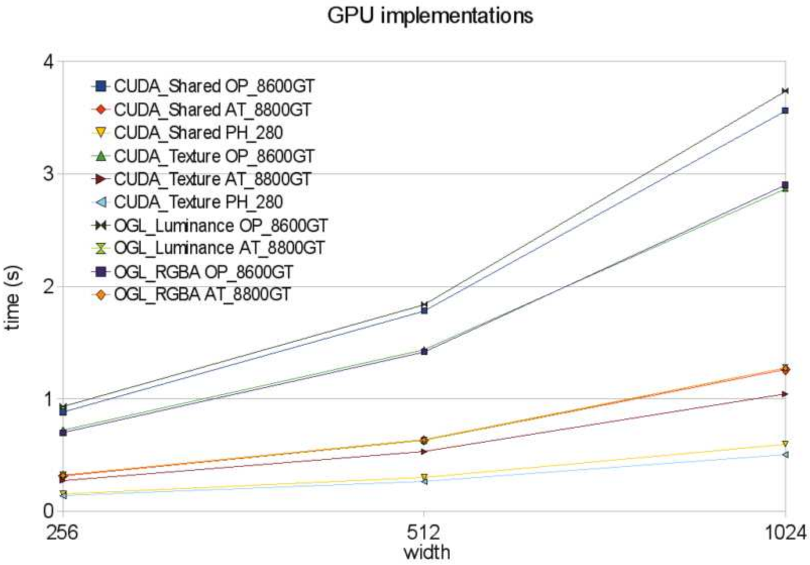
Arquitetura de um multiprocessador da GPU Fermi



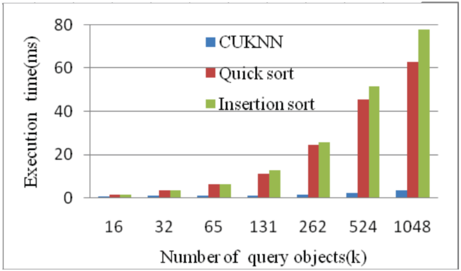
Performance da iteração de Jacobi

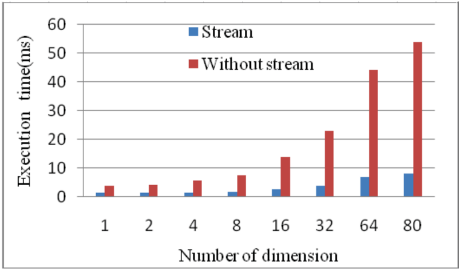


Performance da iteração de Jacobi



Performance do CUKNN





Performance da memória

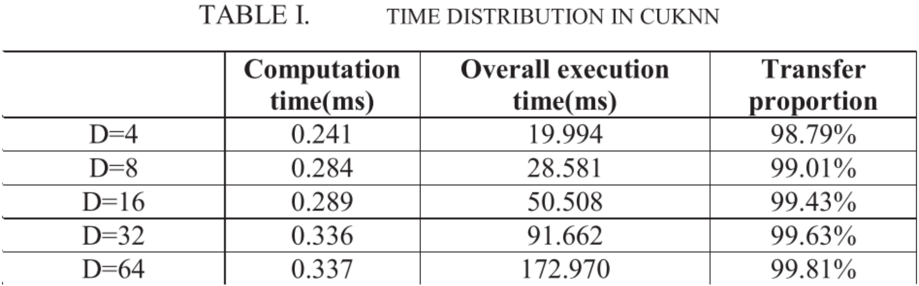


TABLE I. TIME DISTRIBUTION IN CUKNN

	Computation time(ms)	Overall execution time(ms)	Transfer proportion
D=4	0.241	19.994	98.79%
D=8	0.284	28.581	99.01%
D=16	0.289	50.508	99.43%
D=32	0.336	91.662	99.63%
D=64	0.337	172.970	99.81%