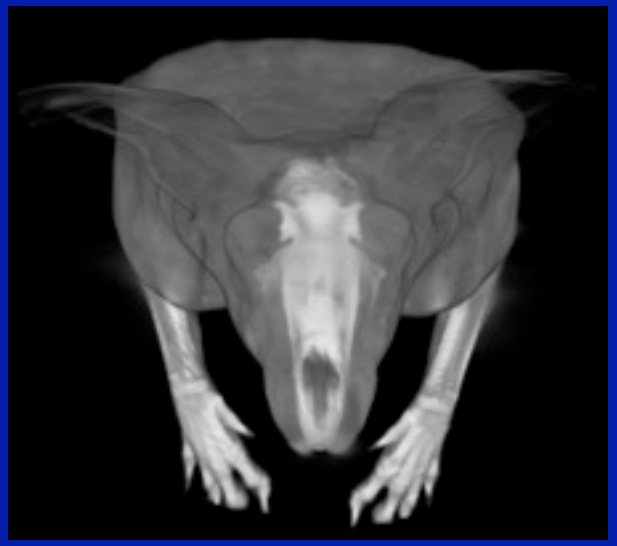
Medical Image Reconstruction using Graphics Hardware (CUDA)



RSL Meeting

November 11th, 2009

Benjamin Keck[1,2], Hannes G. Hofmann[1], Holger Scherl[2], Markus Kowarschik[2] and Joachim Hornegger[1]

[1] Pattern Recognition Lab (Computer Science 5)
Friedrich-Alexander-University Erlangen-Nuremberg, Germany

[2] Siemens Healthcare, CV,
Medical Electronics & Imaging Solutions, Erlangen, Germany

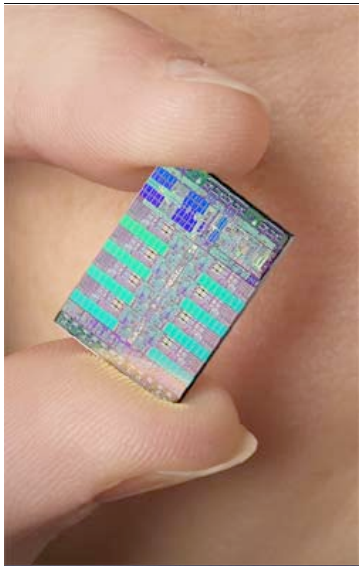
Outline

- **Hardware architectures (CPUs, CELL, GPUs)**
- **GPGPU: general-purpose computation on Graphics Hardware**
- **Compute Unified Device Architecture: Introduction**
- **S(i/a)mple program**
- **Performance results:**
  - 2-D multi-scalar motion filter
  - Filtered back-projection (FDK)
  - Generalized autocalibrating partially parallel acquisitions (GRAPPA)
  - Iterative reconstruction
- **Outlook: NVIDIA‘s *Fermi* Architecture**

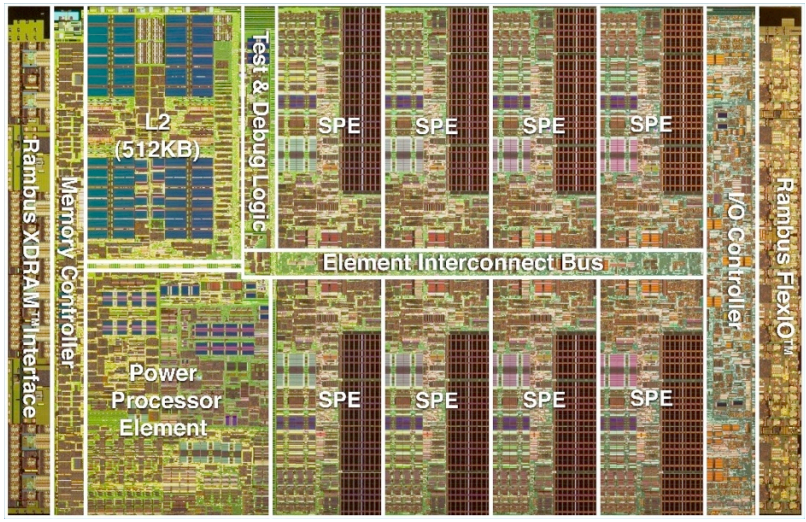
Latest CPU technology: Intel Core i7

- Today
- 4 cores per processor @ 3.33 GHz
- 64 KB L1 cache per core
- 256 KB L2 cache per core
- 8 MB L3 cache shared by all 4 cores
- Memory bandwidth of up to 40 GB/s
- ~102 GFlops peak performance

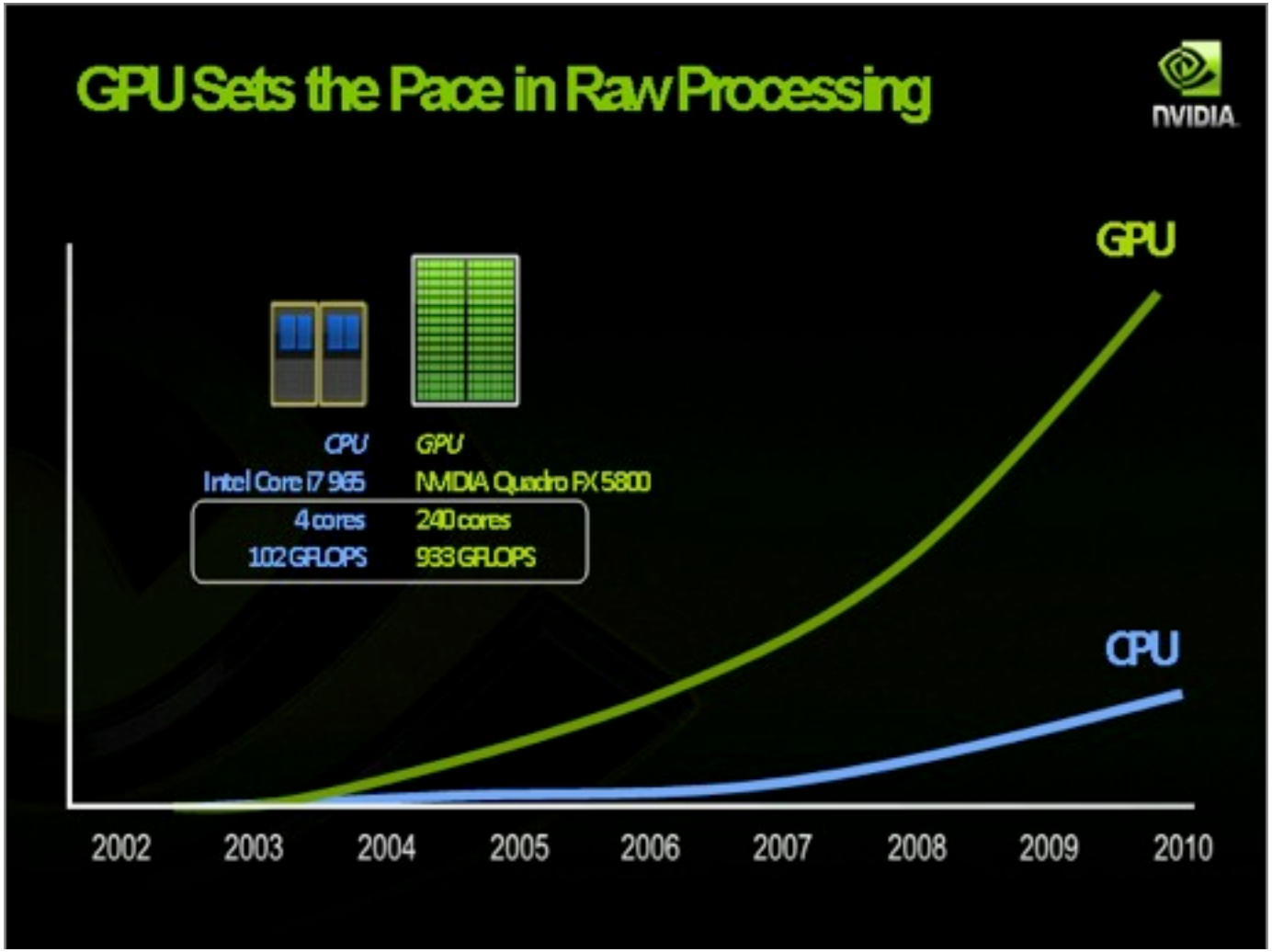
CELL Broadband Engine Architecture (CBEA)



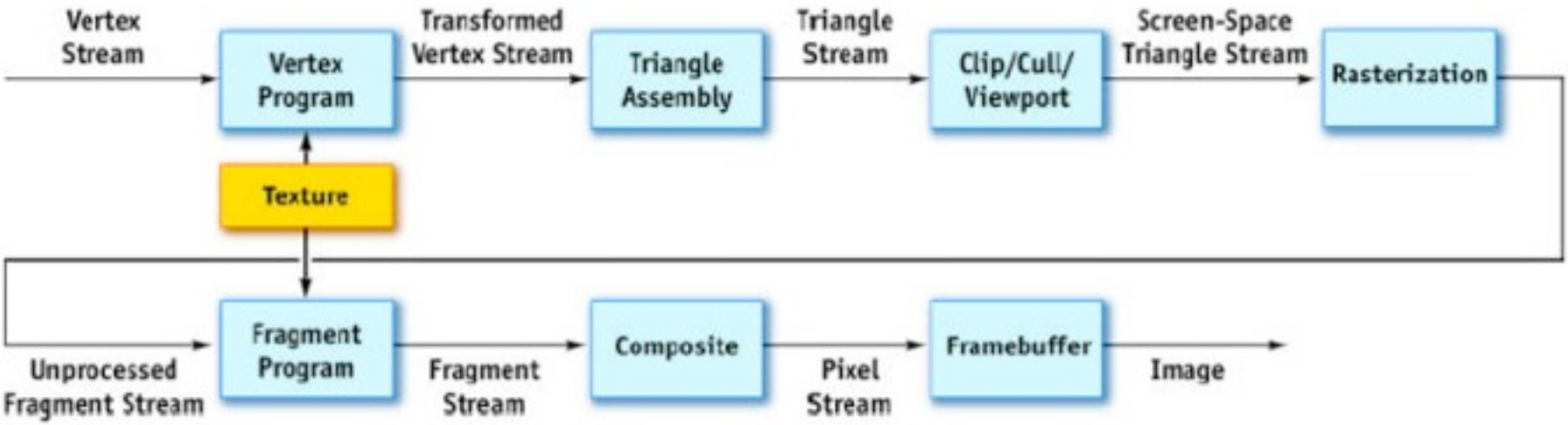
- Early 2006
- *Supercomputer on a chip*
- Multi-core microprocessor
  (1 CPU + 8 RISC vector units)
- *10x performance for many applications*
- Running at 3.2 GHz
- Peak performace ~205 GFlops (SPEs)
- Each SPE: 4 madd per cycle
- Dual XDR memory controller
  (25.6 GB/s)



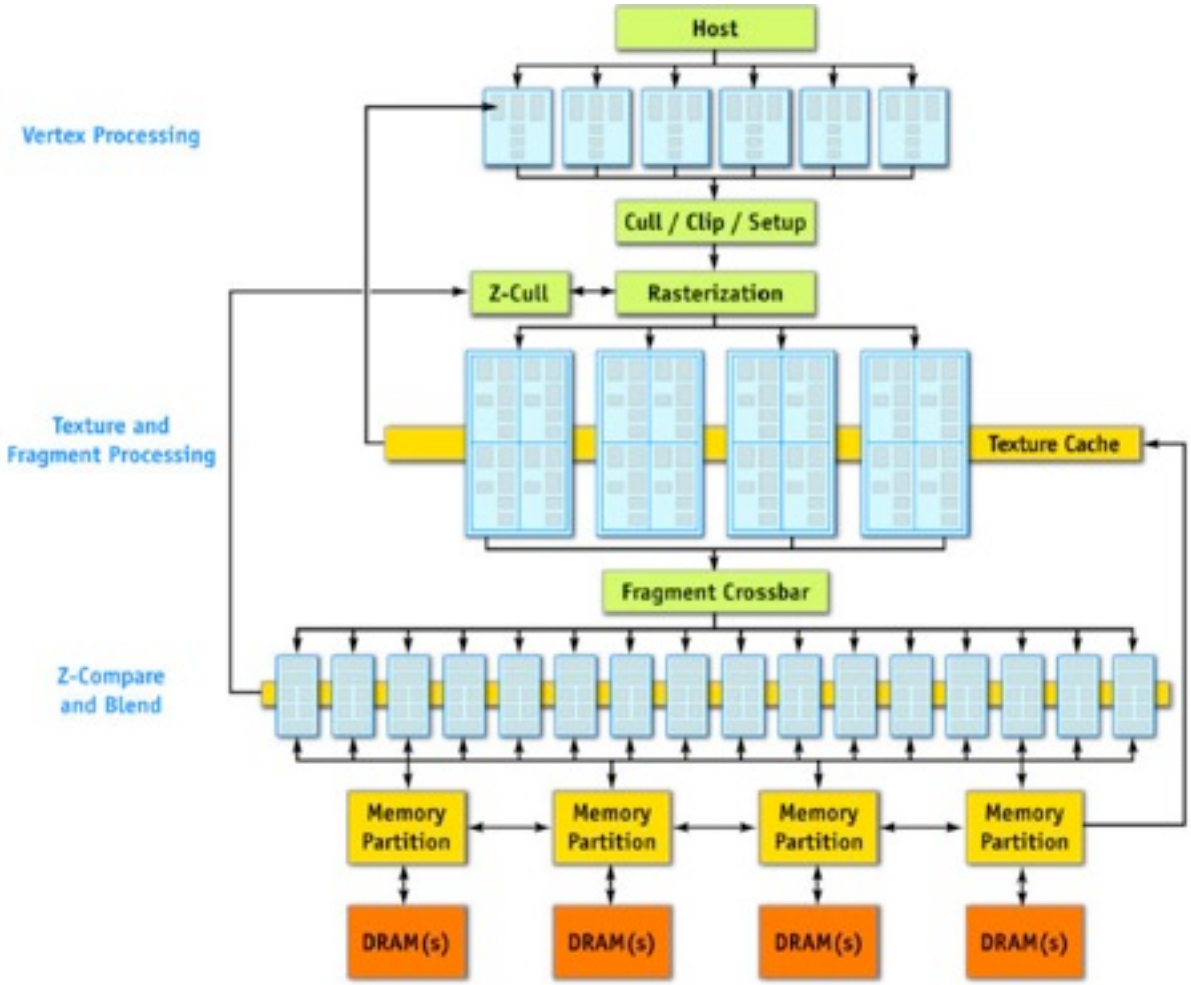
What‘s about the graphics processing unit?



The *former* GPU pipeline

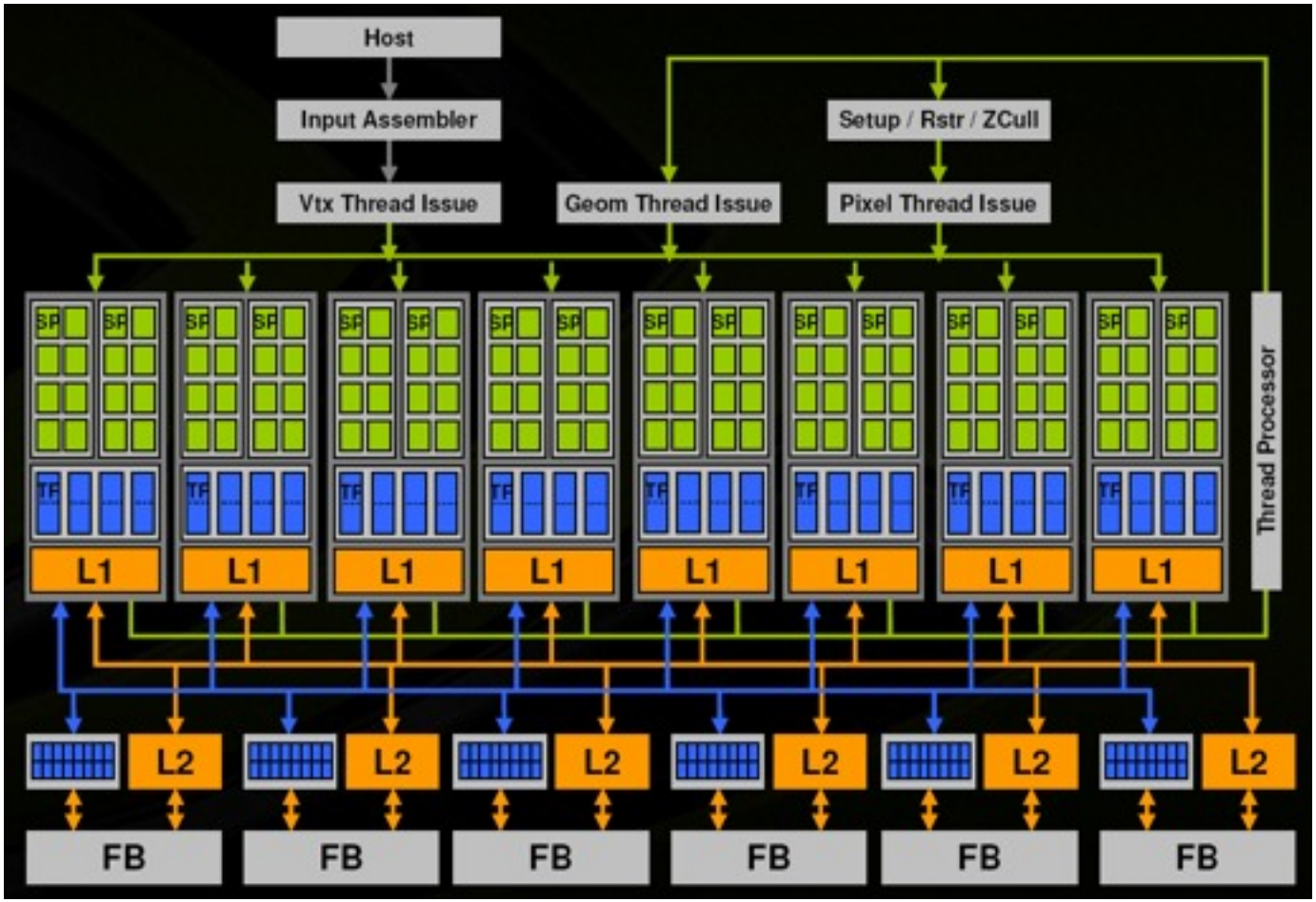


GPU architecture (NVIDIA GeForce 6800)

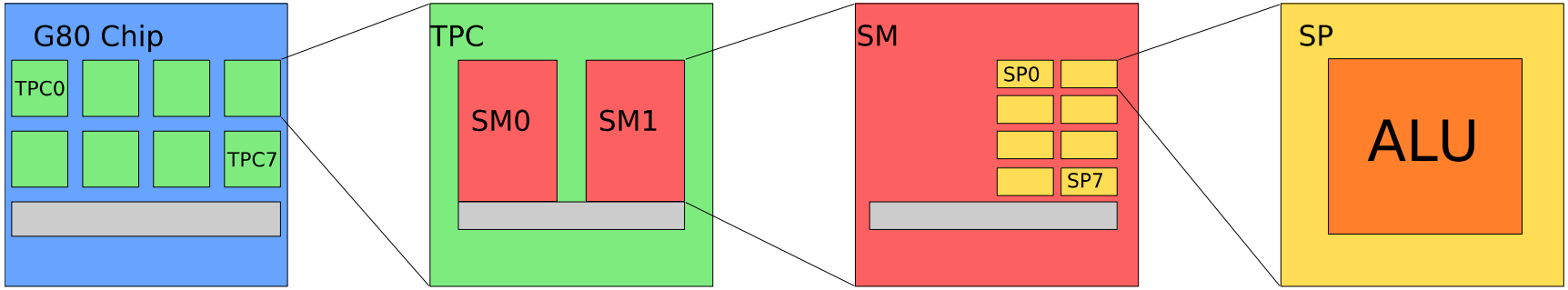


Unified GPU architecture (NVIDIA GeForce 8800 GTX)

- **November 2006**



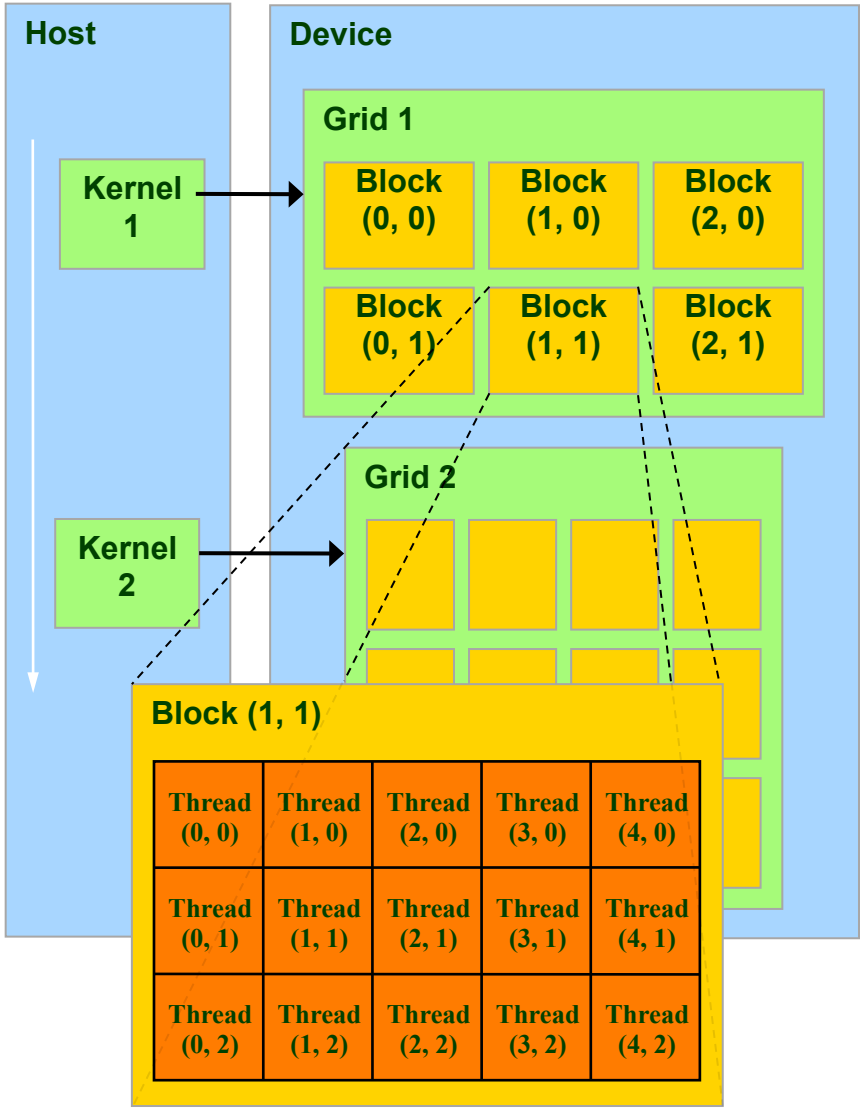
NVIDIA‘s G80 architecture



- **G80 based GeForce 8800 brought several key innovations to GPU Computing**
- **Compute unified device architecture (CUDA):**
  - **ALU: streamprocessor, up to 4 operations pipelined**
  - **SM: streaming multiprocessor, up to 8 identical threads in parallel & up to 768 threads on the scheduler´s ready list**
  - **TPC: thread processor cluster, 2 multiprocessors**
  - **G80-chip: 8 TPCs = 16 multiprocessors = 128 ALUs, up to 12288 threads in flight**

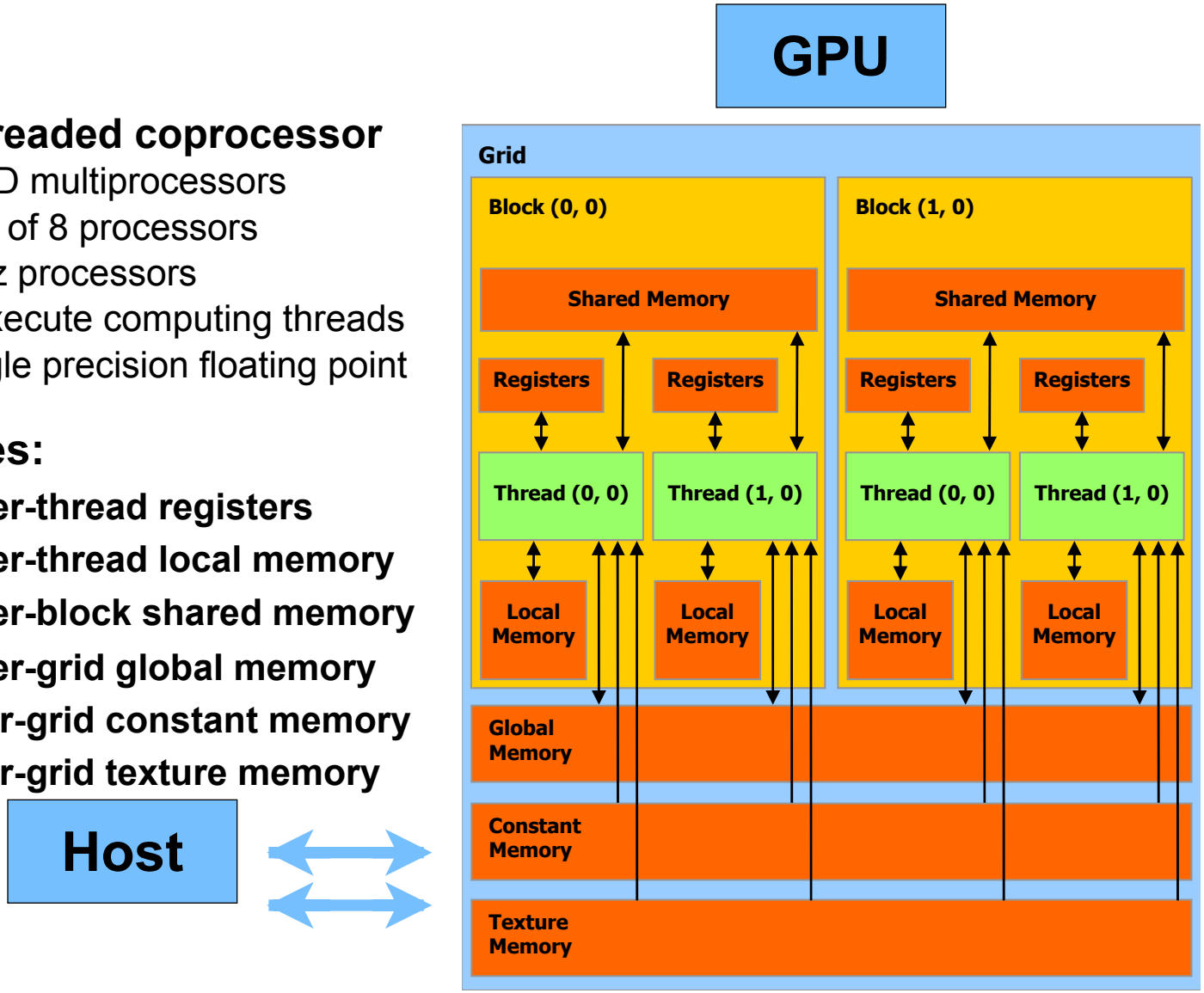
CUDA programming model - a highly multi-threaded coprocessor

- **Thread:** concurrent code and associated state executed on the CUDA device (in parallel with other threads)
  - unit of parallelism in CUDA
- **Warp:** a group of threads executed physically in parallel (SIMD)
- **Block:** a group of threads that are executed together and can share memory on a single multiprocessor
- **Grid:** a group of thread blocks that execute a single CUDA program logically in parallel



CUDA programming model - memory management

- **Highly multi-threaded coprocessor**
  - Set of 16 SIMD multiprocessors
  - Each consists of 8 processors
  - 128, 1.35 GHz processors
  - Processors execute computing threads
  - IEEE 754 single precision floating point
- **Memory Spaces:**
  - **Read/write per-thread registers**
  - **Read/write per-thread local memory**
  - **Read/write per-block shared memory**
  - **Read/write per-grid global memory**
  - **Read only per-grid constant memory**
  - **Read only per-grid texture memory**



S(i/a)mple code

```
// copy and increment a image
for (y=0; y<sizeY; y++){
    for (x=0; x<sizeX; x++){
        dst[y*sizeX+x] = src[y*sizeX+x] + 1.0f;
}}
```

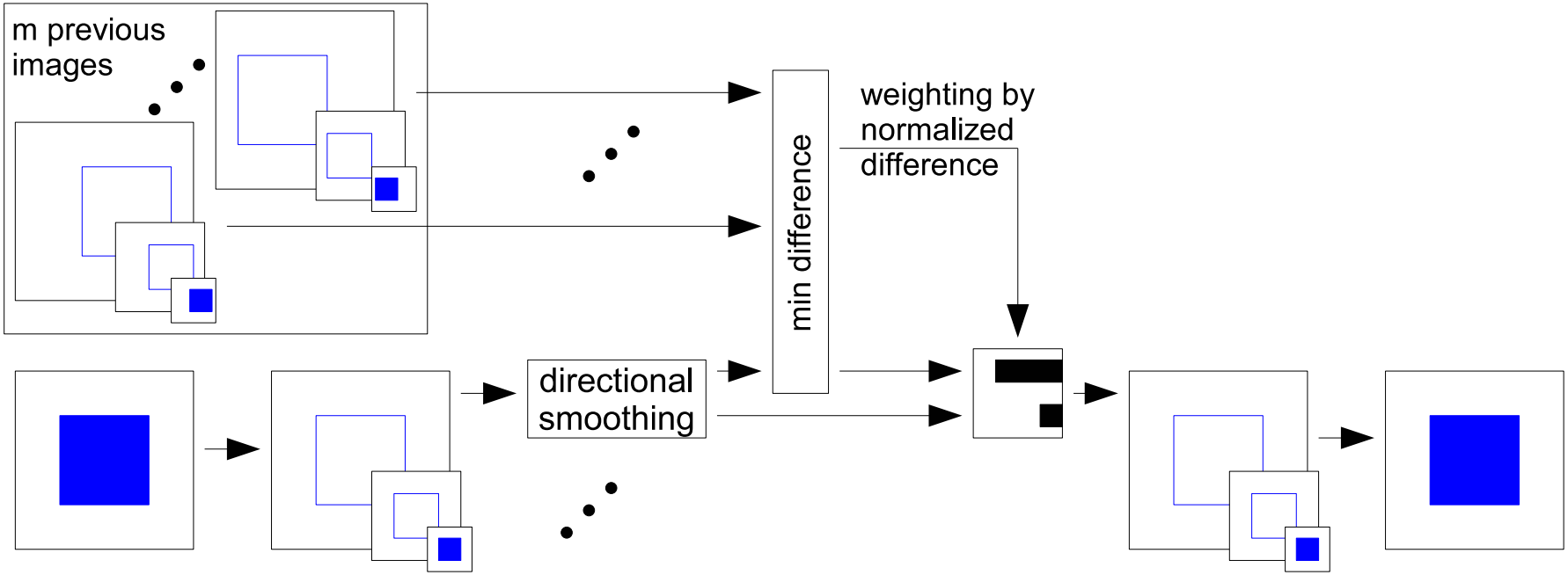
```
}

...
// 32 * 32 = 1024; 4 * 256 = 1024
dim3 threadblocksize (32, 4, 1);
dim3 gridsize(32, 256, 1);

// the kernel-call starts (32 * 4) * (32 * 256) threads
kernel<<<gridsize, threadblocksize>>>(src, dst, 1024, 1024);
...
```

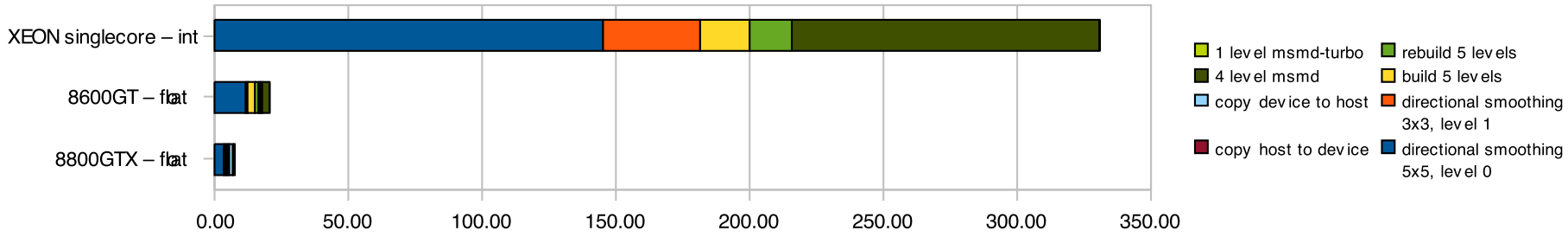
Multi-scale motion filter

- **Combination of different filters:**
  - Multi-resolution
  - Time dependent
  - Directional smoothing

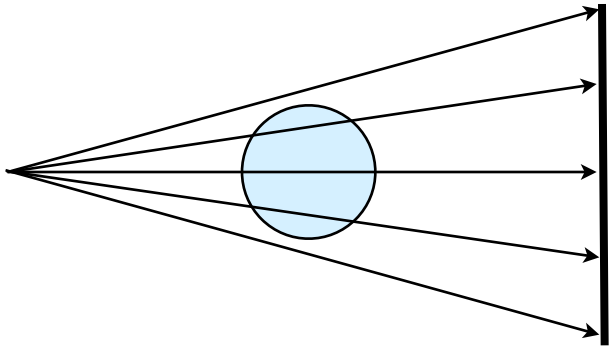


Comparison: CPU - GPU

size of pictures 960x960		System 1 AMD X2-4200+ 2,2 Ghz; 1GB int – singlecore	**8600 GT** 32 ALUs int	float	System 2 Intel Xeon 5150 2,66 Ghz; 2GB int – singlecore	**8800 GTX** 128 ALUs int	float
reduce	ms/pic	1.60	0.37	0.36	1.11	0.13	0.13
	frames/sec	624.38	2698.10	2745.62	904.00	7650.19	7859.02
	factor		**4.32**	**4.40**		**8.46**	**8.69**
expand	ms/pic	17.00	1.22	1.13	12.17	0.31	0.29
	frames/sec	58.82	820.82	883.28	82.20	3231.20	3477.08
	factor		**13.95**	**15.02**		**39.31**	**42.30**
build Laplace	ms/pic	20.39	1.71	1.70	14.10	0.47	0.44
	frames/sec	49.04	585.16	588.54	70.90	2144.83	2291.35
	factor		**11.93**	**12.00**		**30.25**	**32.32**
rebuild Laplace	ms/pic	18.59	1.33	1.25	13.05	0.33	0.32
	frames/sec	53.78	751.72	798.57	76.60	3041.91	3086.60
	factor		**13.98**	**14.85**		**39.71**	**40.30**
directional smoothing 5x5	ms/pic	211.56		11.34	145.34		3.62
	frames/sec	4.73		88.18	6.88		275.98
	factor			**18.65**			**40.11**



Filtered back-projection principle



filter each projection line-by-line

for all projections:
    for all voxels:
        do a back-projection

FDK – filtering FFT performance

- **Filtering**
  - 414 projections each 1024 x 1024 pixels
  - FFT size: 2048

Filtering	Time [s]	pps
NVIDIA GeForce 8800 GTX		
cuFFT	**3.0**	**138.00**
Cell processor 3.2 GHz		
optimized FFT	**0.82**	**503.03**

FDK – back-projection performance

- **Back-projection volume: 512^3 float voxels**

Back-projection	time [s]	pps
NVIDIA GeForce 8800 GTX (nearest neighbor or bilinear interpolation)		
projection-based (brute force)	7.81	53.01
projection-based (incremental)	7.06	58.64
Cell processor 3.2 GHz		
nearest neighbor	11.85	34.94
bilinear interpolation	20.99	19.73

additional time needed for data transfers (CUDA):

projection data transfer	1.07
volume data transfer	0.89

DEMO

FDK – speed difference GTX / QuadroFX

	GeForce 8800 GTX	QuadroFX 5600
Core Clock	575 MHz	600 MHz
Shader Clock	1350 MHz	1400 MHz
Memory Clock Speed	1800 MHz	1600 MHz
Memory Bandwidth	86,4 GB/s	76,8 GB/s

$$7{,}19\text{s} * \frac{1800\text{MHz}}{1600\text{MHz}} = 7{,}98\text{s}$$

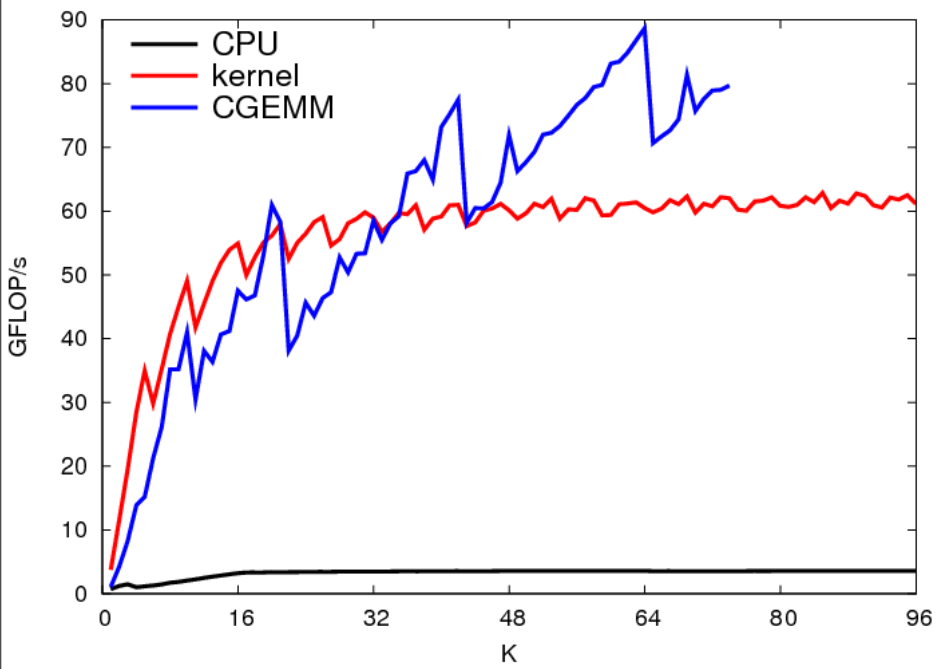
GTX: 7,19s

QuadroFX: 8,14s

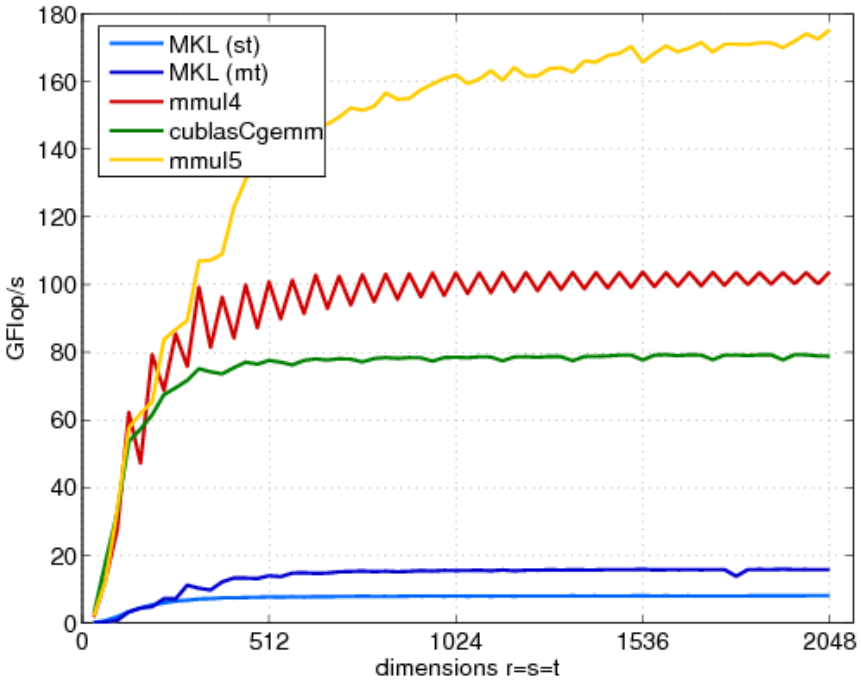
memory bandwidth limited ?!

Ultra: 5,99s

GRAPPA performance prospect



k-space interpolation

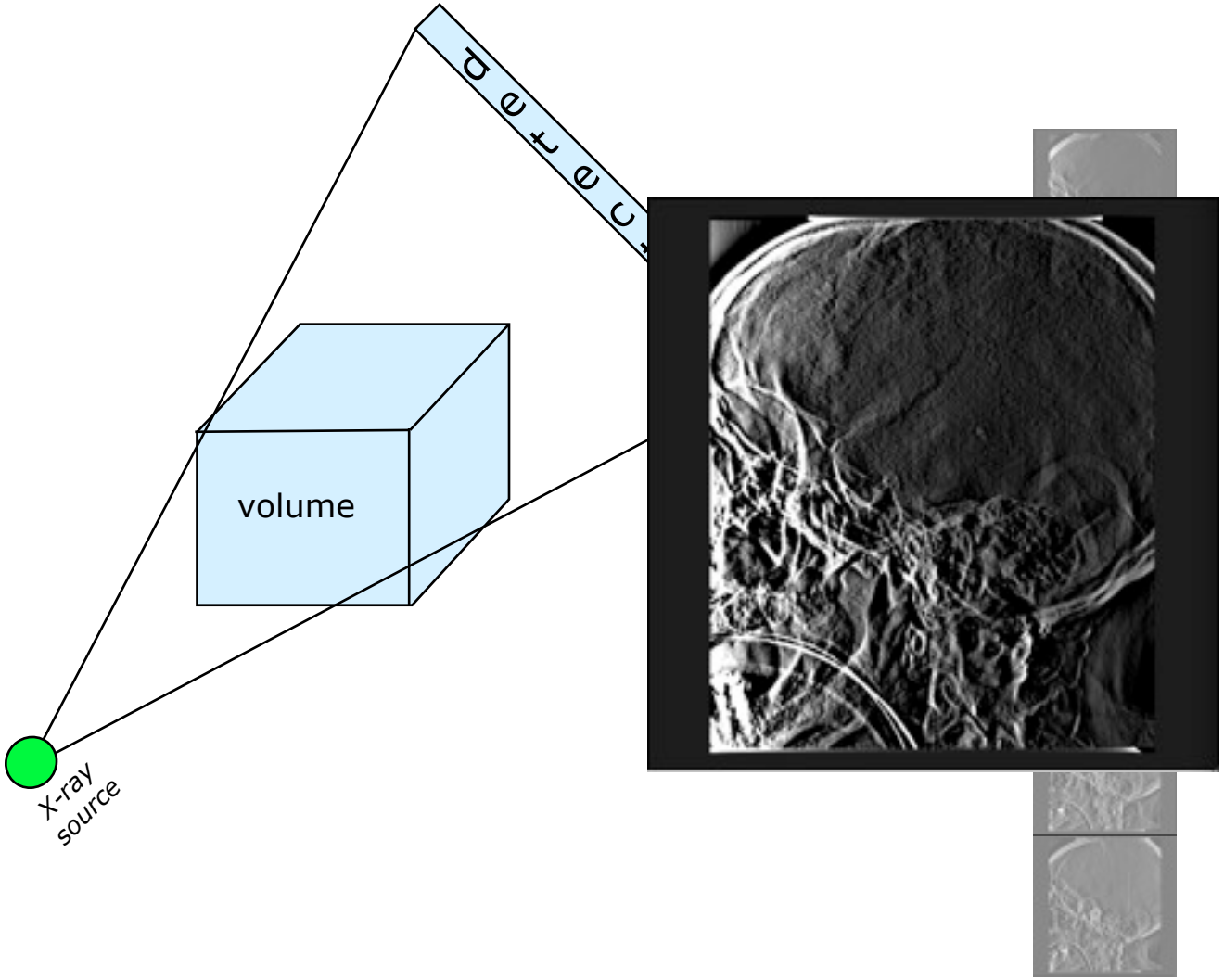


grappa recon.

by Robert Grimm & Matthias Schneider

Simultaneous algebraic reconstruction technique (SART)

- Acquisition

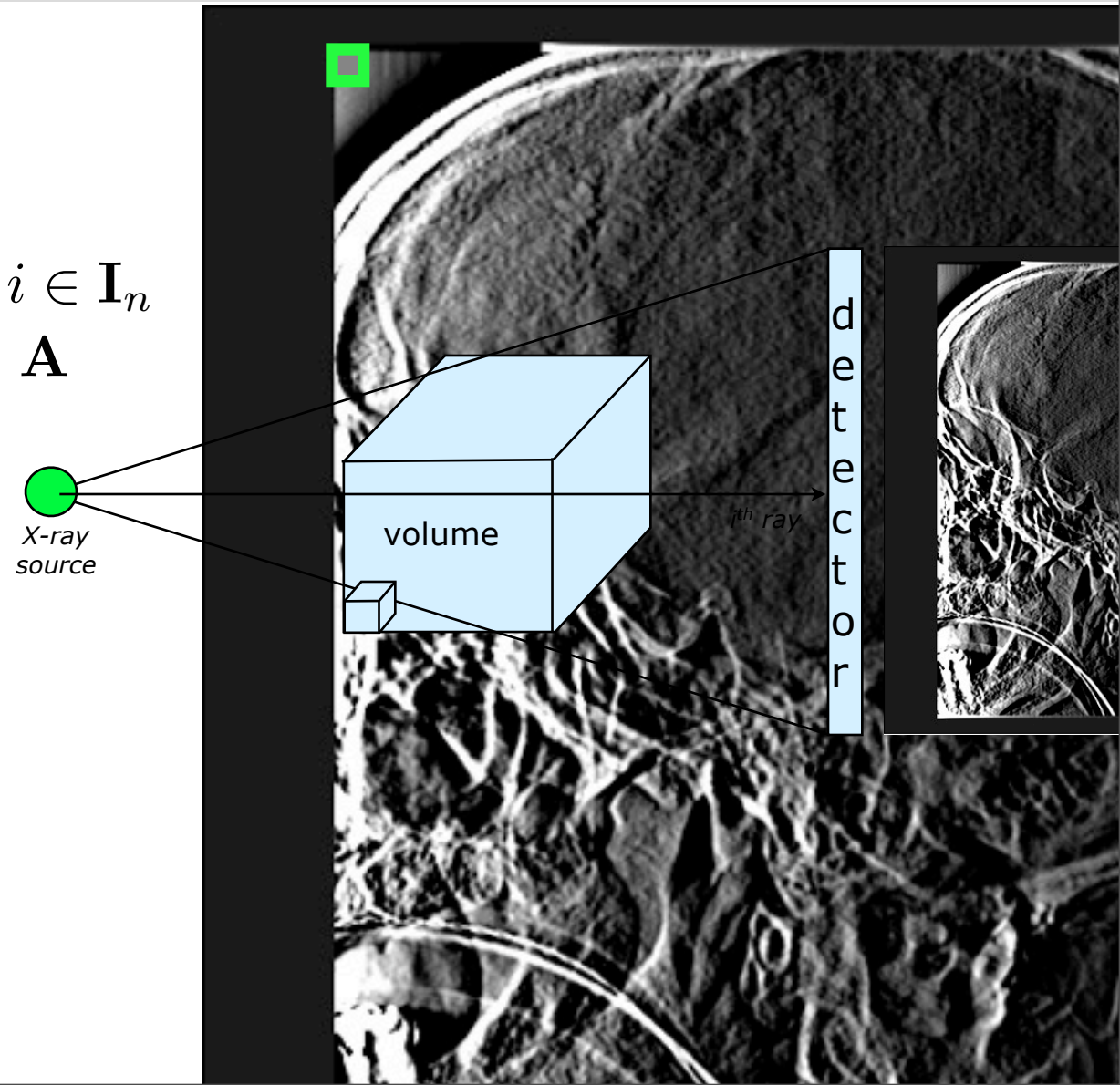


Benjamin Keck

SART: theory and implementation

- Definitions:

- voxel x_j, $j \in \mathbf{J}$
- projections $n \in \mathbf{N}$
- projection value y_i, $i \in \mathbf{I}_n$
- system matrix $a_{ij} \in \mathbf{A}$
- iterations $k \in \mathbf{K}$



SART: formula

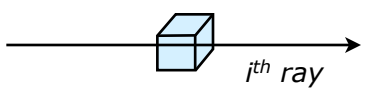
new value = old value + correction term

SART: correction term

$$\lambda \frac{\sum_{i=1}^{\mathbf{I}_n} a_{ij} \left(\frac{y_i - \sum_{j=1}^{\mathbf{J}} a_{ij} x_j{}^{k,n-1}}{\sum_{j=1}^{\mathbf{J}} a_{ij}} \right)}{\sum_{i=1}^{\mathbf{I}_n} a_{ij}}$$

sum over all intersections of a specific voxel with all rays of a specific projection

- Voxel based back-projection: only one ray hits each voxel



SART: correction term

$$\lambda \left(\frac{y_i - \sum_{j=1}^{\mathbf{J}} a_{ij} x_j^{k,n-1}}{\sum_{j=1}^{\mathbf{J}} a_{ij}} \right)$$

total intersection length for a specific ray

- Assumption:
  constant total intersection length for each ray

$$\hat{\lambda} = \lambda \cdot \mathbf{const.}$$

SART: implementation

$$x_j^{k,n} = x_j^{k,n-1} + \hat{\lambda}$$

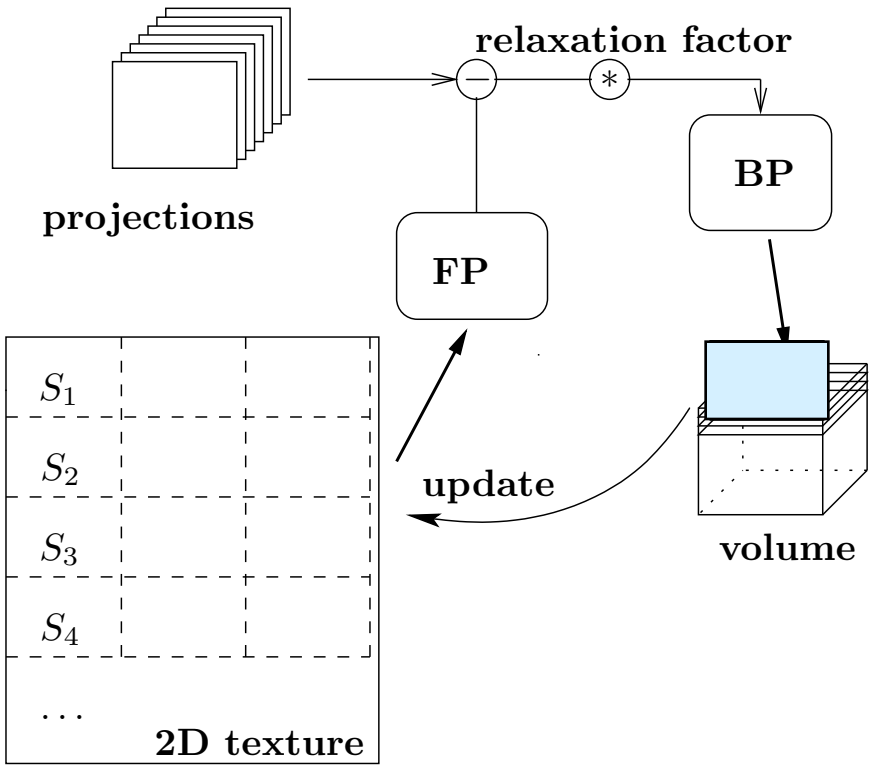
corrective projection

SART using CUDA ≥ 1.1

- **Back-projection (BP): voxel-driven approach**

- **Forward-projection (FP):**
  - Based on ray casting (eligible on GPUs)
  - Unmatched pair forward-projector and back-projector

- **Texture update procedure:**
  - Slice-wise copy
  - Slow (~1s for a 512^3 vol.)



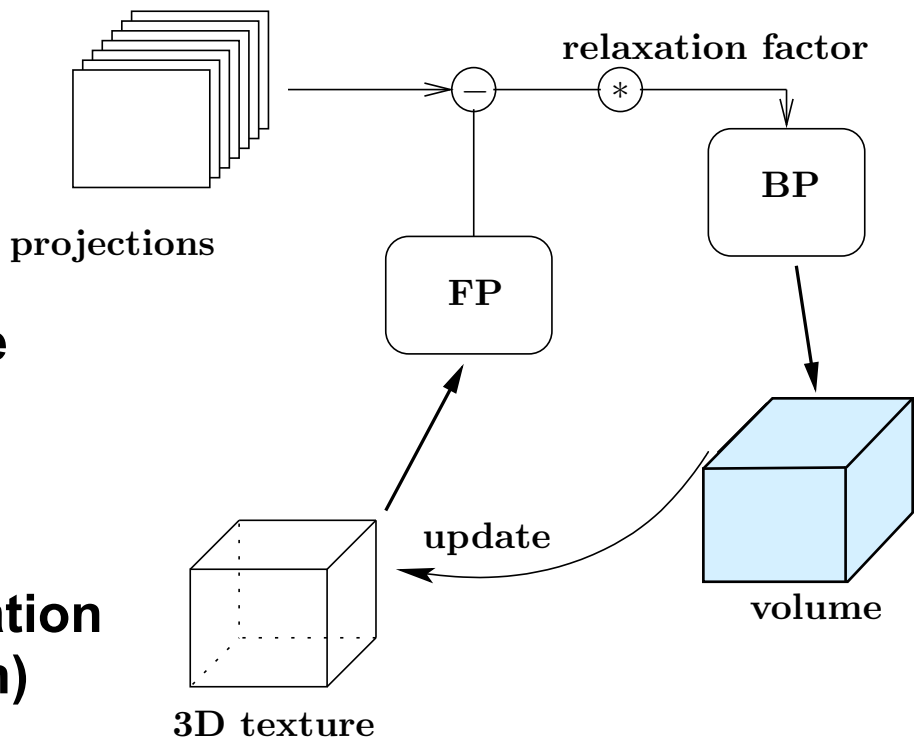
SART using CUDA ≥ 2.0

- **Back-projection see FDK**

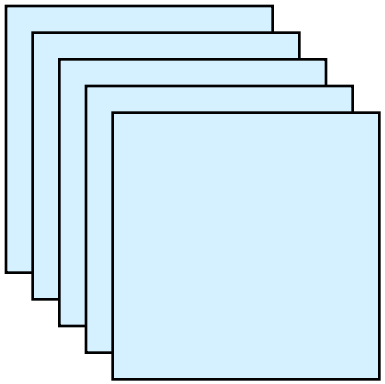
- **Difference in forward-projection**
  - CUDA 2.0 supports 3D textures
  - Enabled hardware support for trilinear interpolation

- **Easier texture update procedure**
  - Single instruction copy
  - Update approx. 10 times faster

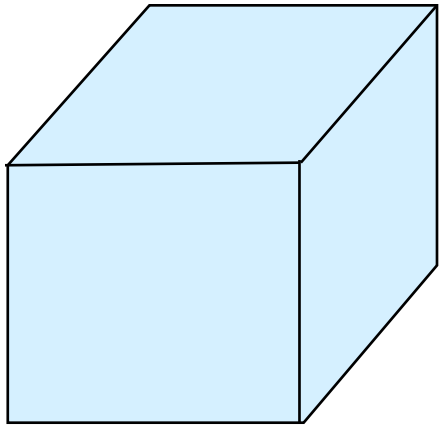
- **Drawback 3-D texture size limitation (2048 elements in each direction)**



Experimental Setup



Projections:
228 projections
à 256x128 pixel



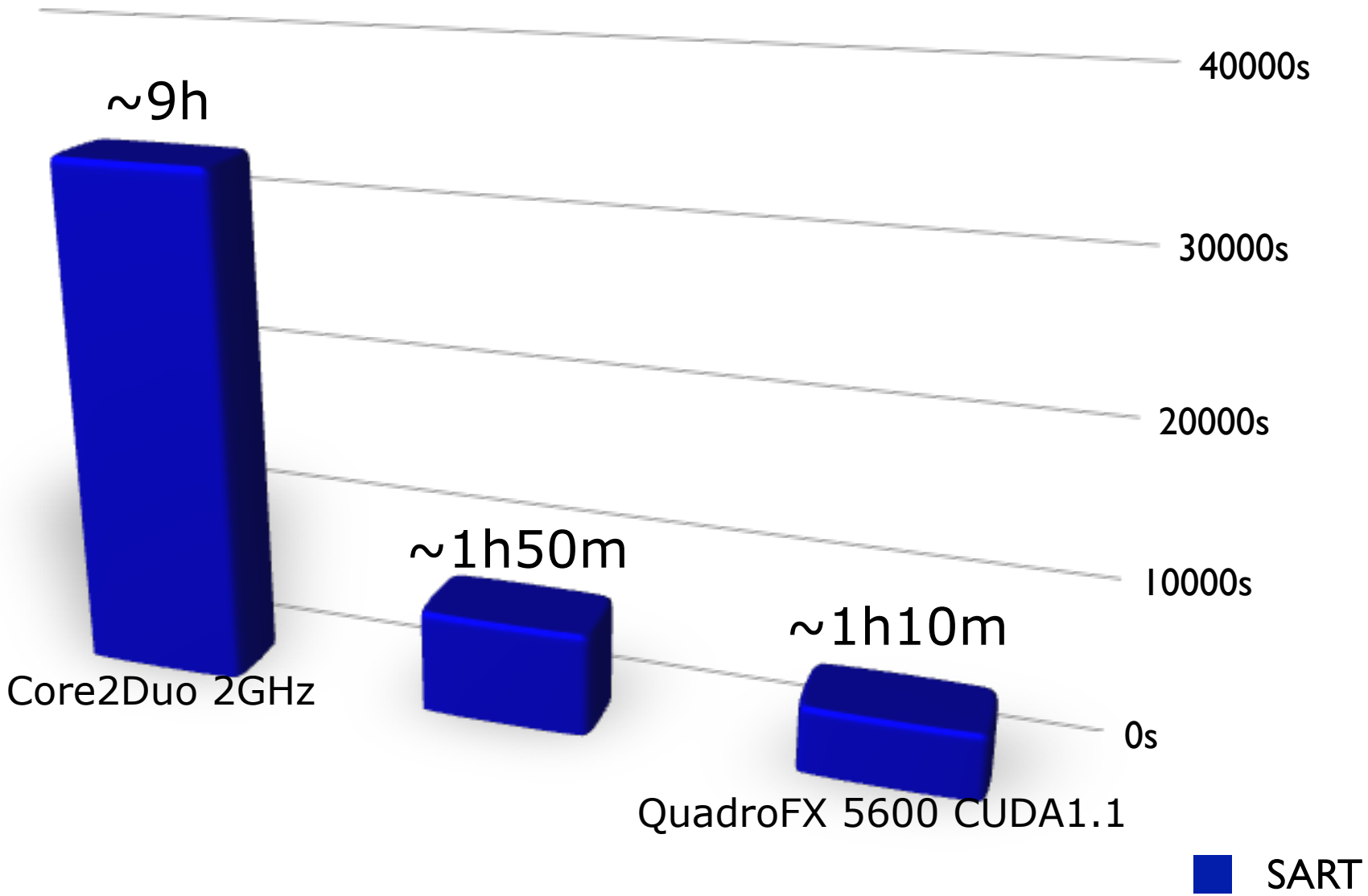
Volume:
512x512x350

- Performing 20 iterations
- Step size used in ray cast algorithm: 0.3 of uniform voxel size

Compared systems:

Off-the-shelf PC: Intel Core2Duo @ 2 GHz	Workstation: Two Intel QuadCore @ 2.33 GHz	GPU: NVIDIA QuadroFX 5600

SART: performance comparison I

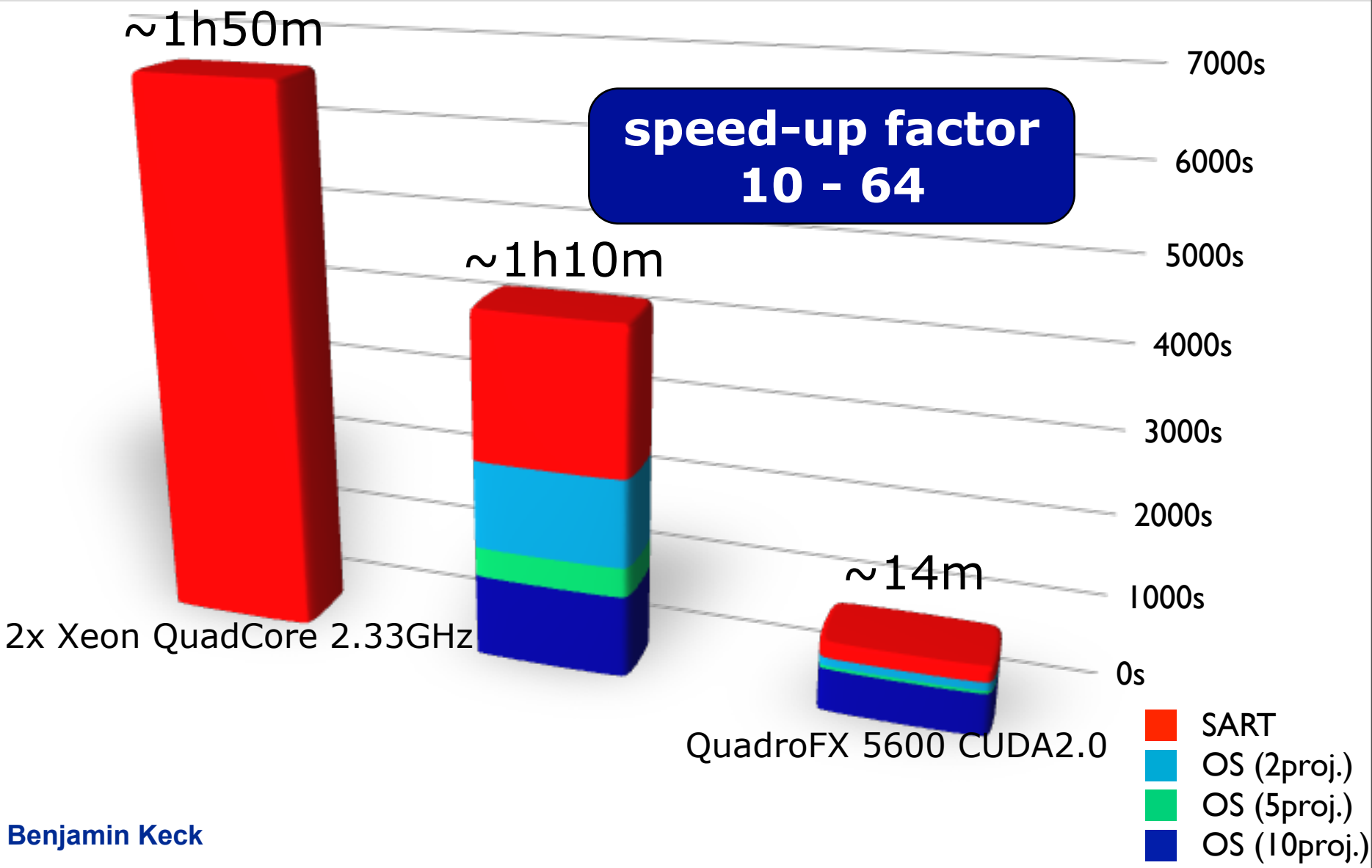


SART: performance

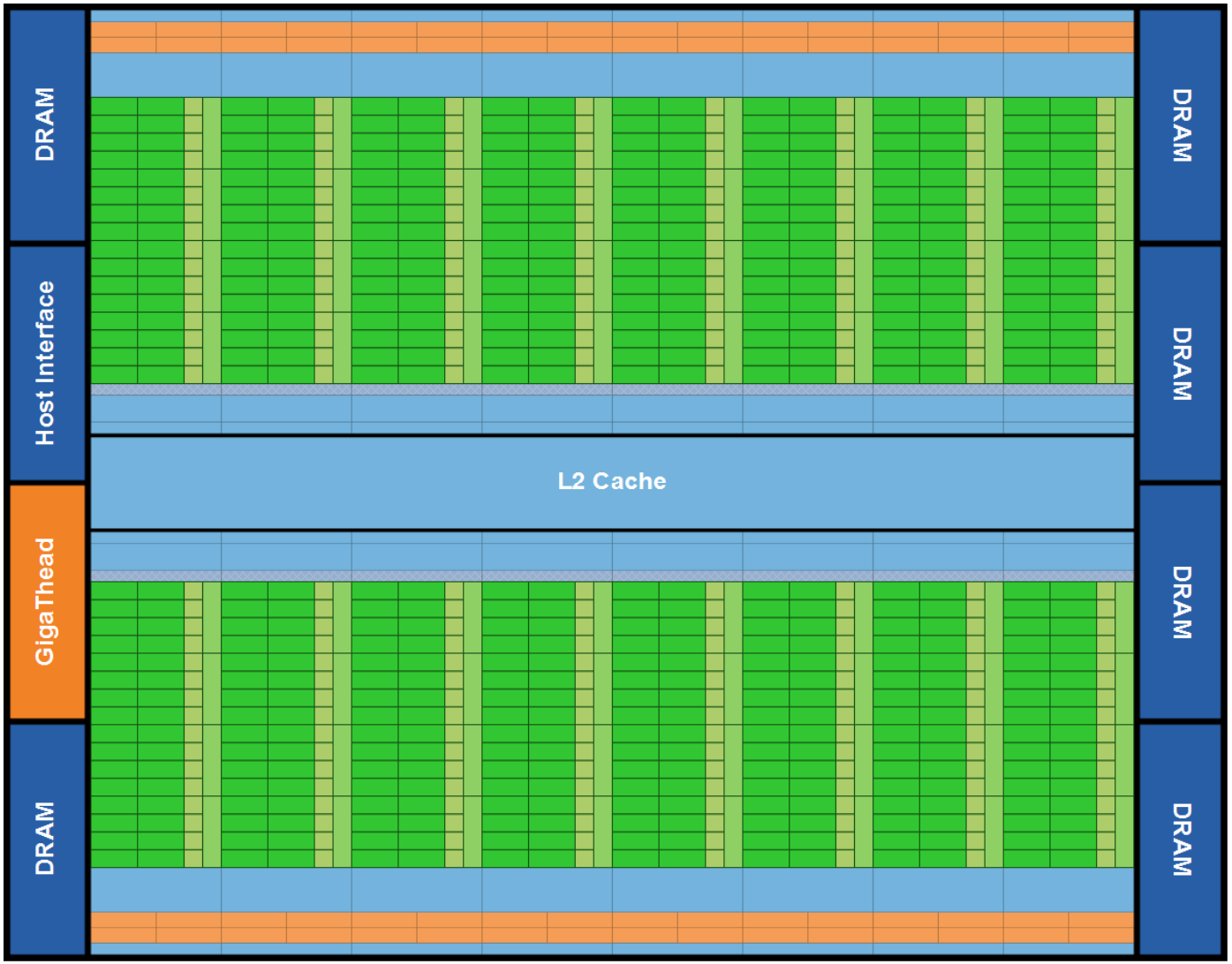
	$512 \times 512 \times 350$ voxels			
Hardware/ Method	Intel Core2Duo 2 GHz Time [s]	2×Intel Xeon QuadCore 2.33 GHz Time [s]	QuadroFX 5600 CUDA 1.1 Time [s]	QuadroFX 5600 CUDA 2.0 Time [s]
SART	**32968**	6630	4234	844
OS(2proj.)	”	”	2435	661
OS(5proj.)	”	”	1359	551
OS(7proj.)	”	”	1156	530
OS(10proj.)	”	”	998	**514**

- OS optimization reduces GPU specific runtime up to 76% (CUDA 1.1), 39% (CUDA 2.0)
- CUDA 2.0 implementation (SART) outperforms CUDA 1.1 (OS 10proj.)
- Speedup factor GPU vs. CPU: 64x - 12x (PC resp. Workstation)

SART: performance comparison II

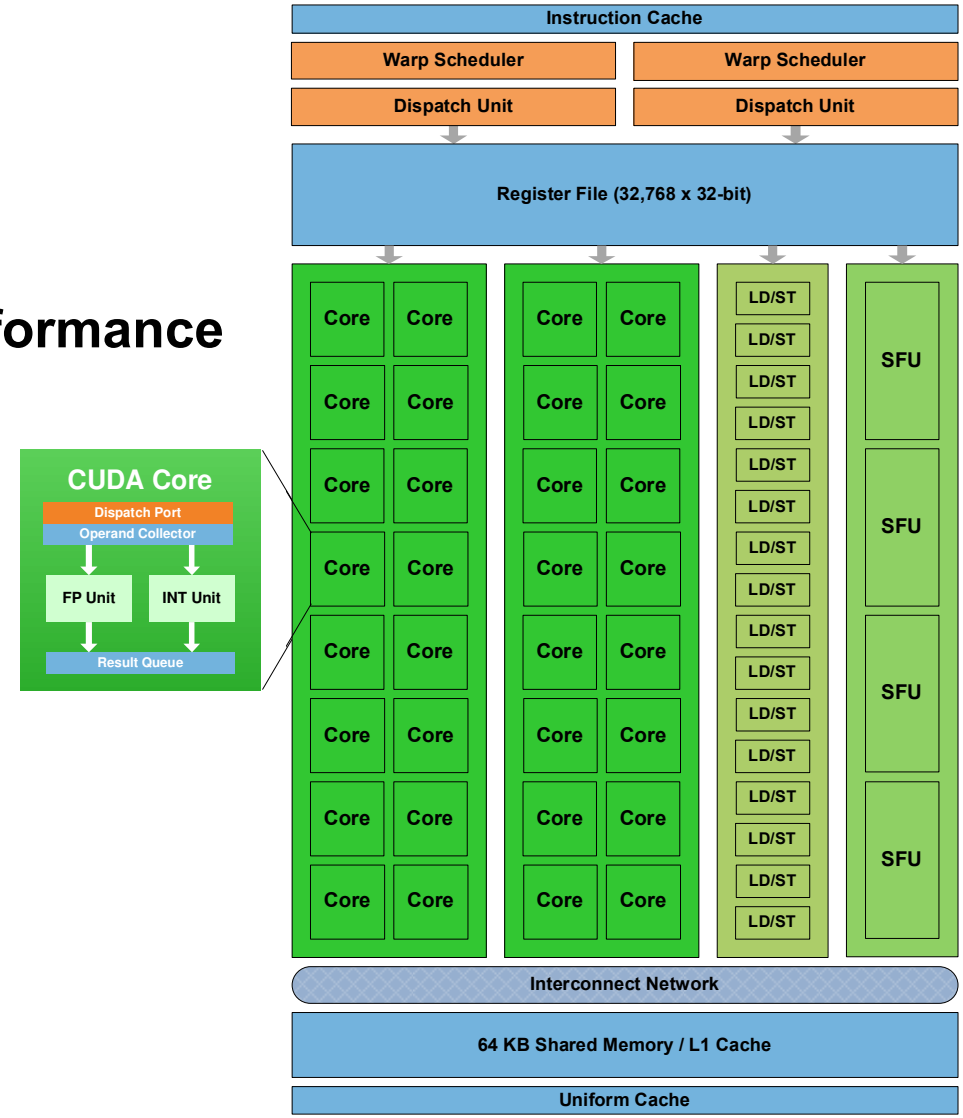


NVIDIAs “Fermi“ architecture



NVIDIAs “Fermi“ architecture

- Much more CPU like GPU
- Outstanding 512 cores
- Improved double precision performance
- ECC support
- True cache hierarchy
- More shared memory
- Faster context switching
- Faster atomic operations

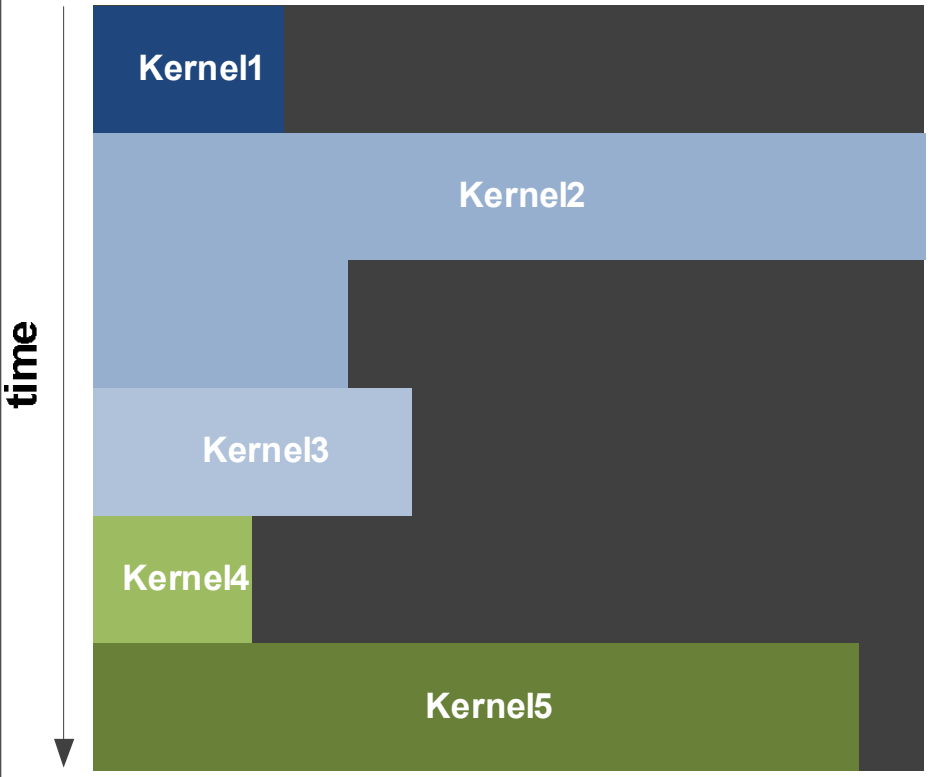


Fermi Streaming Multiprocessor (SM)

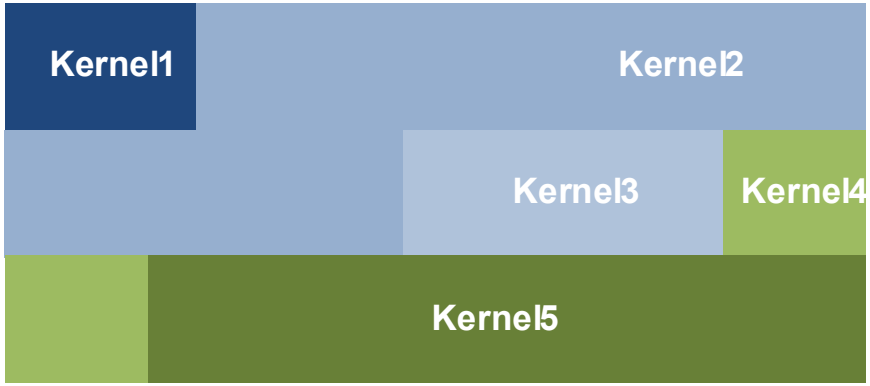
NVIDIAs “Fermi“ architecture

GPU	G80	GT200	Fermi
Transistors	681 million	1.4 billion	3.0 billion
CUDA Cores	128	240	512
Double Precision Floating Point Capability	None	30 FMA ops / clock	256 FMA ops /clock
Single Precision Floating Point Capability	128 MAD ops/clock	240 MAD ops / clock	512 FMA ops /clock
Special Function Units (SFUs) / SM	2	2	4
Warp schedulers (per SM)	1	1	2
Shared Memory (per SM)	16 KB	16 KB	Configurable 48 KB or 16 KB
L1 Cache (per SM)	None	None	Configurable 16 KB or 48 KB
L2 Cache	None	None	768 KB
ECC Memory Support	No	No	Yes
Concurrent Kernels	No	No	Up to 16
Load/Store Address Width	32-bit	32-bit	64-bit

NVIDIAs “Fermi“ architecture



Serial Kernel Execution



Concurrent Kernel Execution

NVIDIAs “Fermi“ architecture

Nexus CUDA Samples.90 (Debugging) - Microsoft Visual Studio (Administrator)

File Edit View Project Build Debug Nexus Tools Test Window Help

Debug | Win32 | d_vbo_buffer | CUDA (0,0,0), (0,0,0)

Process: [4560] GPU - matrixMul.e | Thread: [2772376] <No Name> | Stack Frame: Module: 60956632 - [0] _Z9n

Solution Explorer - matrixMul

Solution 'Nexus CUDA Samples.90' (3 projects)
- matrixMul
  - inc
  - src
    - matrixMul.cu

NVIDIA Nexus - CUDA Focus Picker

		Dimensions
Block:	0, 0, 0	8, 5, 1
Thread:	0, 0, 0	16, 16, 1

Examples
#129 for block index 129
10 for coordinates 10, 0
10, 5 for coordinates 10, 5

OK Cancel

Solution Explorer | Class View

matrixMul_kernel.cu

```
// Loop over all the sub-matrices of A and B
// required to compute the block sub-matrix
for (int a = aBegin, b = bBegin;
        a <= aEnd;
        a += aStep, b += bStep) {

    // Declaration of the shared memory array As used to
    // store the sub-matrix of A
    __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

    // Declaration of the shared memory array Bs used to
    // store the sub-matrix of B
    __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

    // Load the matrices from device memory
    // to shared memory; each thread loads
    // one element of each matrix
    AS(ty, tx) = A[a + wA * ty + tx];
    BS(ty, tx) = B[b + wB * ty + tx];

    // Synchronize to make sure the matrices are loaded
    __syncthreads();
```

Nexus CUDA Device Summary

Name	Details
Devices	
Device 0	
Device 1	
Context 2772376	Device 0
Module 60956632	c:/ProgramData/NVIDIA Nexus 1.0/Samples/CUDA/Debug
Grid	_Z9matrixMulPfS_S_ii<<<(8,5),(16,16,1), 0>>>
Block 0	Warp Mask: 0x000000FF
Warp 0	Active Mask: 0xFFFFFFFF, PC: 0x000703E8, matrixMul_k
Warp 1	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 2	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 3	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 4	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 5	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 6	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,
Warp 7	Active Mask: 0xFFFFFFFF, PC: 0x000703E8,

Locals

Name	Value	Type
blockDim	{x = 16, y = 16, z = 1}	const dim3
gridDim	{x = 8, y = 5, z = 1}	const dim3
As	0x00000024 {{0.20108646, 0.23432112, 0.2616657, 0.18860439, ...}, {0.88125247	float[16][16] __shared__
[0]	0x00000024 {0.20108646, 0.23432112, 0.2616657, 0.18860439, ...}	float[16] __shared__
[1]	0x00000064 {0.88125247, 0.21982482, 0.15710929, 0.15753655, ...}	float[16] __shared__
[2]	0x000000a4 {0.55427718, 0.1802118, 0.76696068, 0.56581318, ...}	float[16] __shared__
[3]	0x000000e4 {0.60716575, 0.673513, 0.26108584, 0.37244788, ...}	float[16] __shared__
Bs	0x00000424 {{0.80645162, 0.41080967, 0.12955107, 0.26792198, ...}, {0.1797546	float[16][16] __shared__
a	0	int
b	0	int
bx	0	int
by	0	int
tx	0	int

Autos | Locals | Threads | Modules | Watch 1

Memory 1

Address: 0x00000024 | Columns: Auto

```
0x00000024  9c e9 4d 3e e0 f1 6f 3e 0c f9 85 3e 82 21 41 3e 6c e7 35 3f 7f   œéM>àño>.ù.>.!A>lç5?.
0x00000039  63 3f 3f 97 4d 4b 3f 91 59 48 3f 0a e1 04 3e 1f 69 0f 3f fc 11   c??—MK?'YH?.á.>.i.?ü.
0x0000004E  fe 3d 1c d5 0d 3f 5a 3d ad 3e e4 27 72 3f 8a f9 44 3e ca c7 64   þ=.Ó.?Z=->ä'r?ŠùD>ÊÇd
0x00000063  3f c3 99 61 3f c2 19 61 3e 42 e1 20 3e 43 51 21 3e a4 11 52 3e   ?Ã™a?Â.a>Bá >CQ!>¤.R>
0x00000078  eb 43 75 3f 8a ed c4 3e ba dd dc 3e 54 2f 2a 3f 23 b5 91 3e 49   ëCu?ŠíÄ>ºÝÜ>T/*?#µ'>I
0x0000008D  75 24 3f c5 b1 62 3f 9a 3b 4d 3f a9 bf 54 3f 88 33 44 3f 47 65   u$?Å±b?š;M?©¿T?^3D?Ge
0x000000A2  a3 3e 1c e5 0d 3f 71 89 38 3e 89 57 44 3f 22 d9 10 3f 13 71 09   £>.å.?q.8>.WD?"Ù.?.q.
0x000000B7  3e ba c1 dc 3d dc e3 6d 3f 9c c1 cd 3e 76 c1 3a 3e 70 c9 37 3f   >ºÁÜ=Üãm?œÁÍ>vÁ:>pÉ7?
0x000000CC  21 4d 10 3f d6 37 6b 3f db 7d 6d 3f d9 85 6c 3f 42 3d 21 3f 47   !M.?Ö7k?Û}m?Ù.l?B=!?G
0x000000E1  7d 23 3f 37 6f 1b 3f 59 6b 2c 3f 0b ad 85 3e 7d b1 be 3e b7 69   }#?7o.?Yk,?.-.>}±.>·i
0x000000F6  5b 3e 10 35 88 3e 21 6f 10 3f 04 2d 82 3e 89 51 c4 3d 5b 99 ad   [>.5^>!o.?.-.>.QÄ=[™-
0x0000010B  3e 2e 15 97 3e cd ab 66 3f 23 91 11 3f c6 f1 62 3f 20 e9 0f 3e   >..—>Í«f?#'.?Æñb? é.>
0x00000120  7f 49 3f 3f dc 11 ee 3d 84 c9 41 3e da 01 6d 3c d8 dd 6b 3f ee   .I??Ü.î=.ÉA>Ú.m<ØÝk?î
0x00000135  01 f7 3b ee fb 76 3f 2f 81 17 3d 6e f3 36 3f 48 1d a4 3e ef b1   .÷;îûv?/..=nó6?H.¤>ï±
```

Memory 1 | Call Stack | Breakpoints | Output | Pending Checkins

Ready | Ln 110 | Col 1 | Ch 1 | INS

3:57 PM 8/25/2009

NVIDIA Nexus integrated development environment

Matlab with CUDA

- **MATLAB plug-in for CUDA:**
  - acceleration of standard MATLAB 2D FFTs (single-precision)
  - CUDA/MEX example plug-in and build environment
  - available at http://developer.nvidia.com/object/matlab_cuda.html