Multi-GPU and the Wavelet Transform

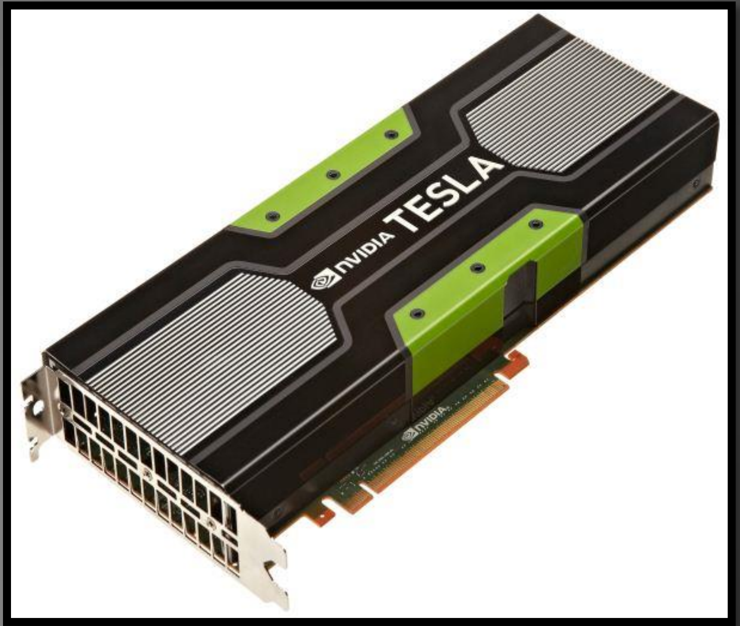
Andre Kessler [6.338/18.337]

(SpaceX Combustion Group)

DECEMBER 9, 2013

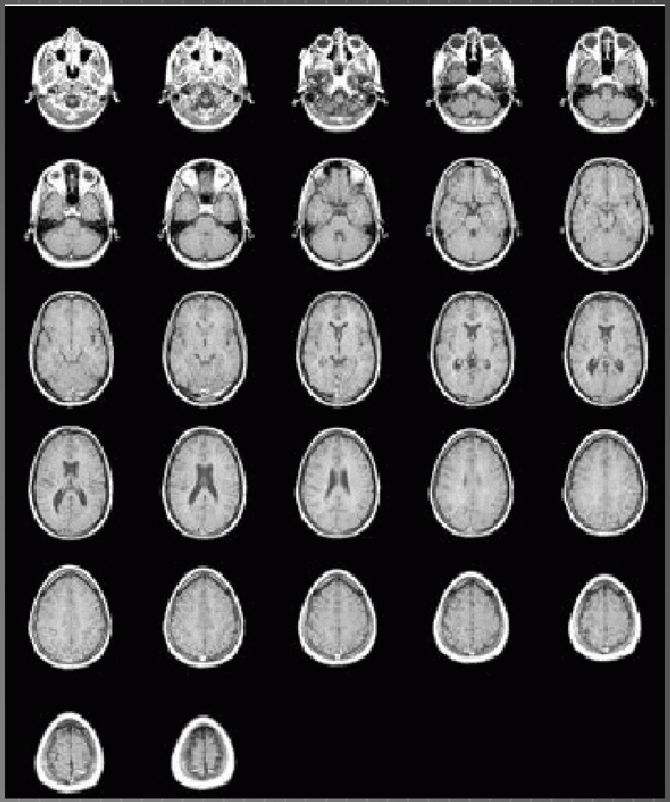
THE GRAPHICS PROCESSING UNIT

- Good for big computation
  - NVIDIA's Tesla K20 has…
  - **1.17** Tflops double / **3.52** Tflops single
- Not so great for big data
  - NVIDIA's Tesla K20 has...
  - Just 5 GB
- Improving, but not quickly enough
  - Next-gen K40 has 12 GB



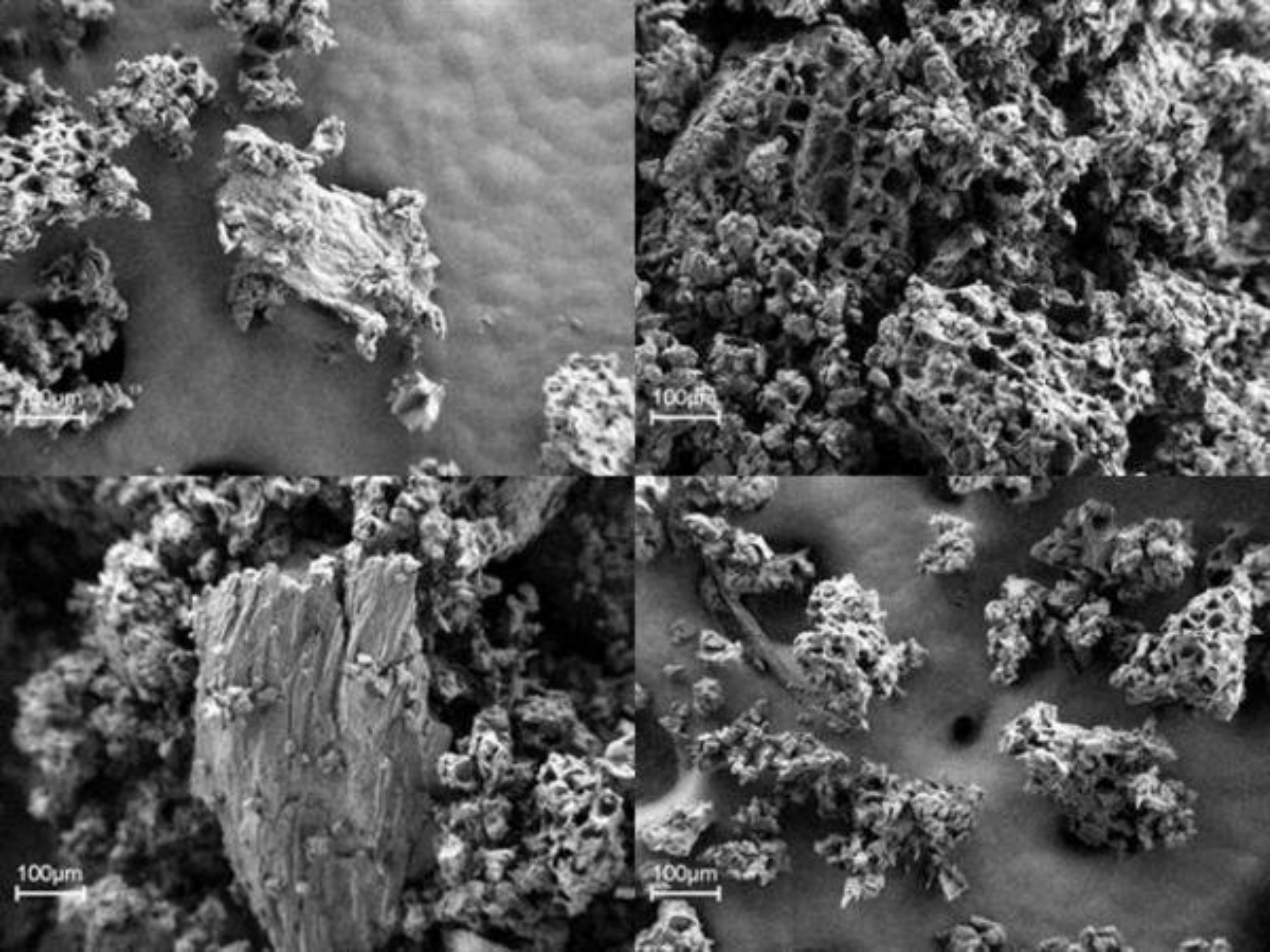
THE PROBLEM OF 3-D DATA

- Very high fidelity 3-d data takes up a lot of space.
- Simple grayscale voxel field with a single float per point:
  - Up to **N < 1,700**
- If one double per point,
  - Up to **N < 850**
- If RGBA data, halve again:
  - Up to **N < 425**



http://www.mathworks.com/products/demos/image/3d_mri/mri_hori.gif

Following slides: http://www.home-barista.com/reviews/titan-grinder-project-scanning-electron-microscope-sem-analysis-of-ground-coffee-t4205.html


100μm
100μm
100μm
100μm

WAVELET TRANSFORM

- First simple example:
- $(a, b) \to (\mu = (a+b)/2\,,\ \delta = b - a)$
- (Following example from *Ripples in Mathematics*)

56	40	8	24	48	48	40	16
48	16	48	28	**-16**	**16**	**0**	**-24**
32	38	**-32**	**-20**	**-16**	**16**	**0**	**-24**
35	**6**	**-32**	**20**	**-16**	**16**	**0**	**-24**

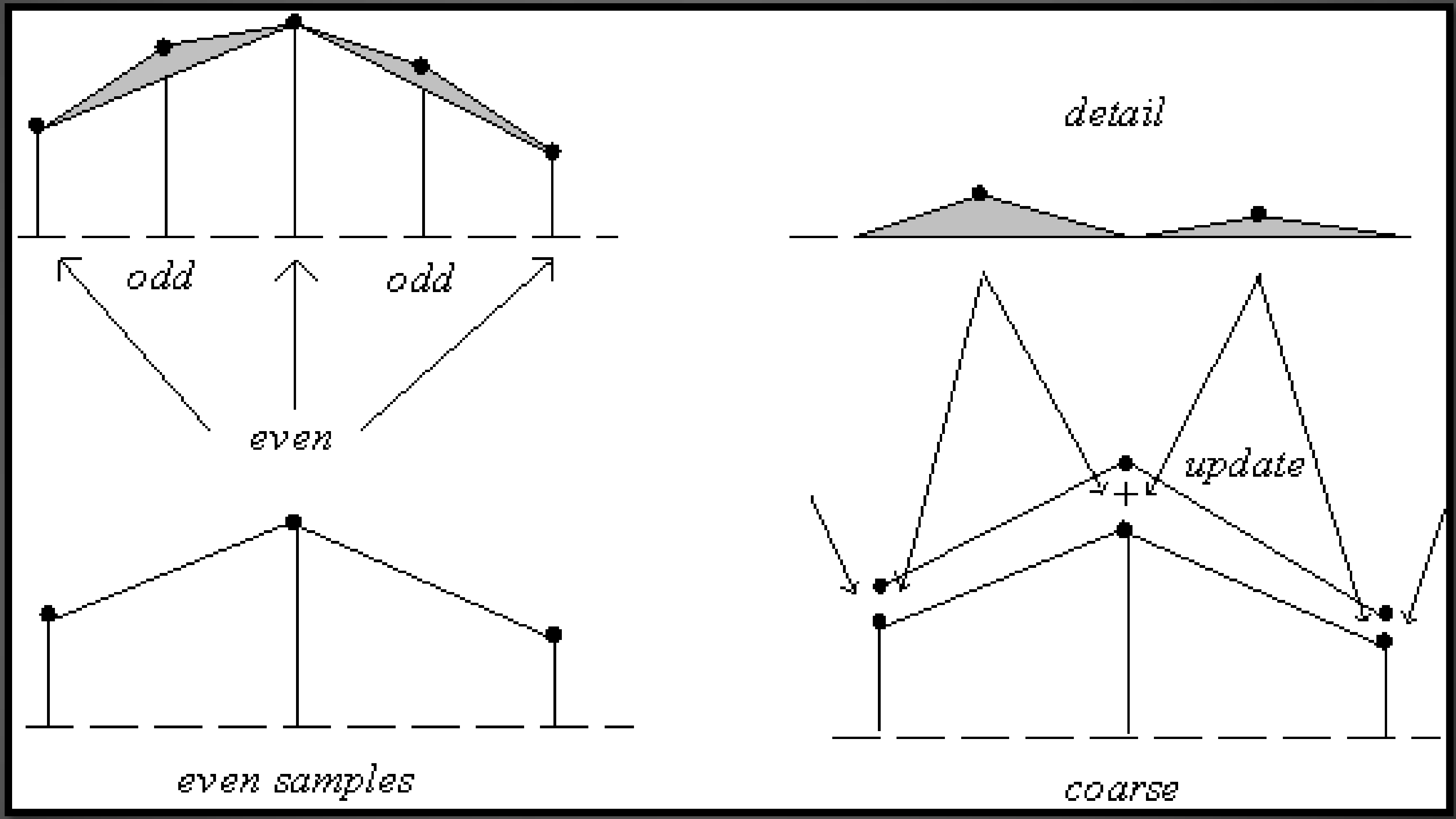
WAVELET TRANSFORM

56	40	8	24	48	48	40	16
48	16	48	28	**-16**	**16**	**0**	**-24**
32	38	**-32**	**-20**	**-16**	**16**	**0**	**-24**
35	**6**	**-32**	**20**	**-16**	**16**	**0**	**-24**

- The idea is that we can turn our data into a set of
  - ***Coarse data –*** in this case, we've got one (35 on the left)
  - ***Detail coefficients*** – in this case, the 7 entries to the right
- Notice the detail coefficients are smaller than the original data. Now we'll compress w/ a high-pass filter.

WAVELET TRANSFORM

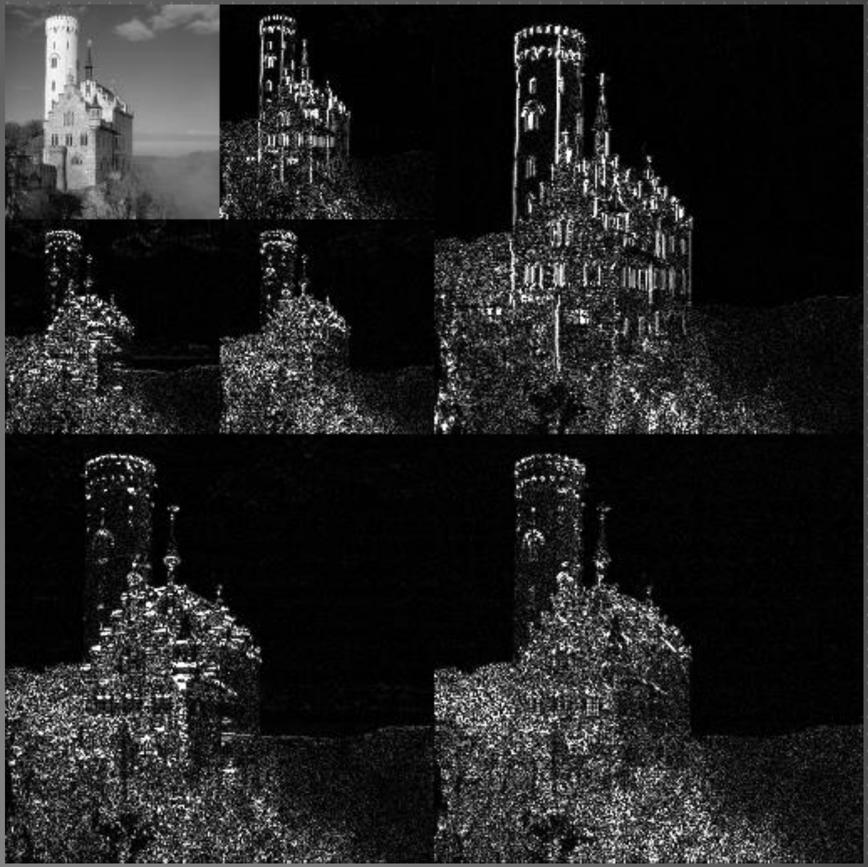
- Again, an example:



http://www.ima.umn.edu/industrial/97_98/sweldens/fourth.html

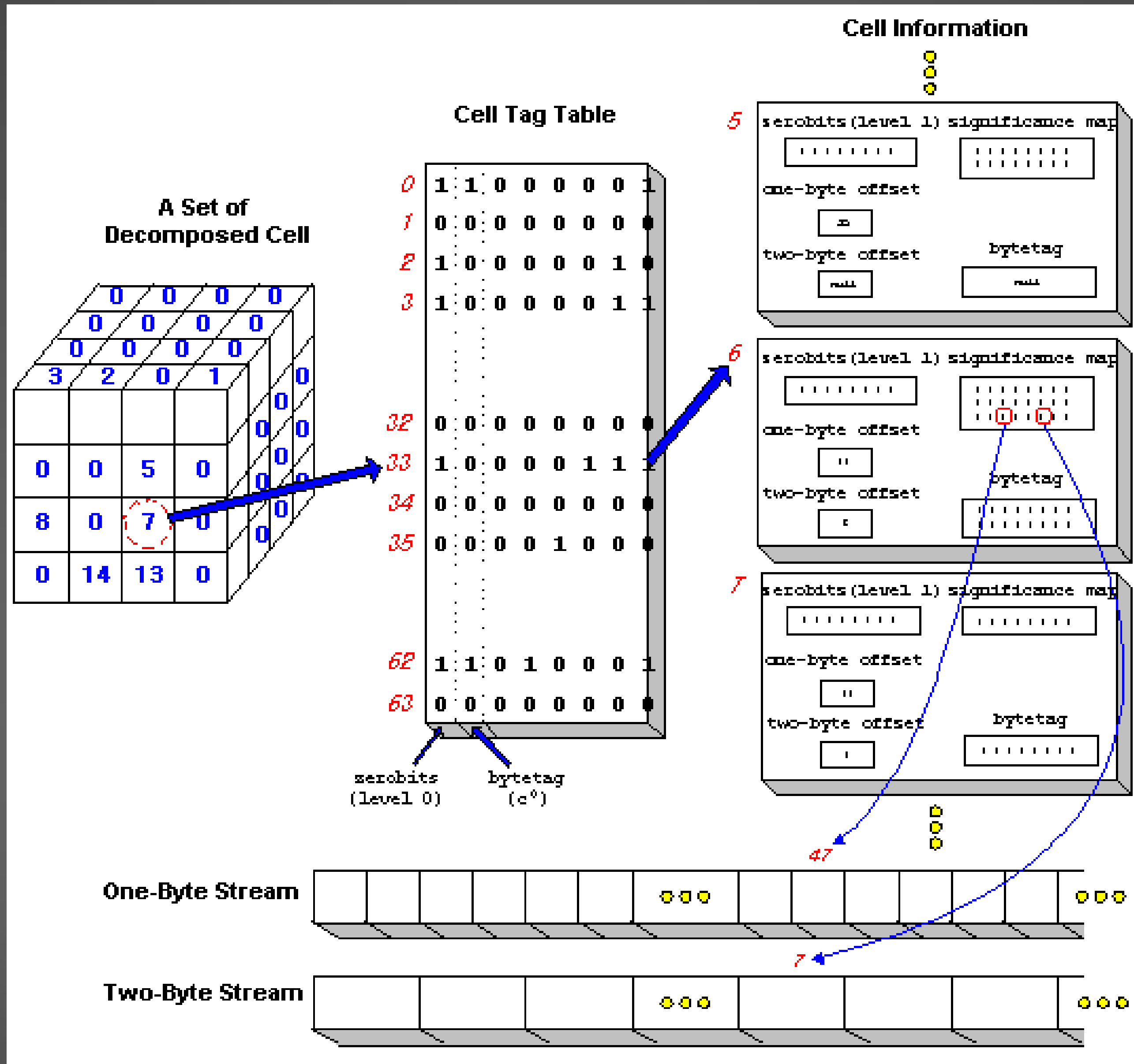
WAVELET TRANSFORM – JPEG2000

- Compression with wavelets was the choice for the ill-fated JPEG2000 standard
- ".jp2"
- There is also a JP3D standard for 3D data



http://upload.wikimedia.org/wikipedia/commons/e/e0/Jpeg2000_2-level_wavelet_transform-lichtenstein.png

ZEROTREE/ZEROBIT ENCODING



WAVELETS + GPUS

- Why is this combination particularly attractive?
- Computation is cheap
  - *Compress/decompress* is very cheap; host to device memory reads are terribly slow
- So you can compress your data, selectively decode a part and do your computation, then recompress

GPU MEMORY TRANSFERS

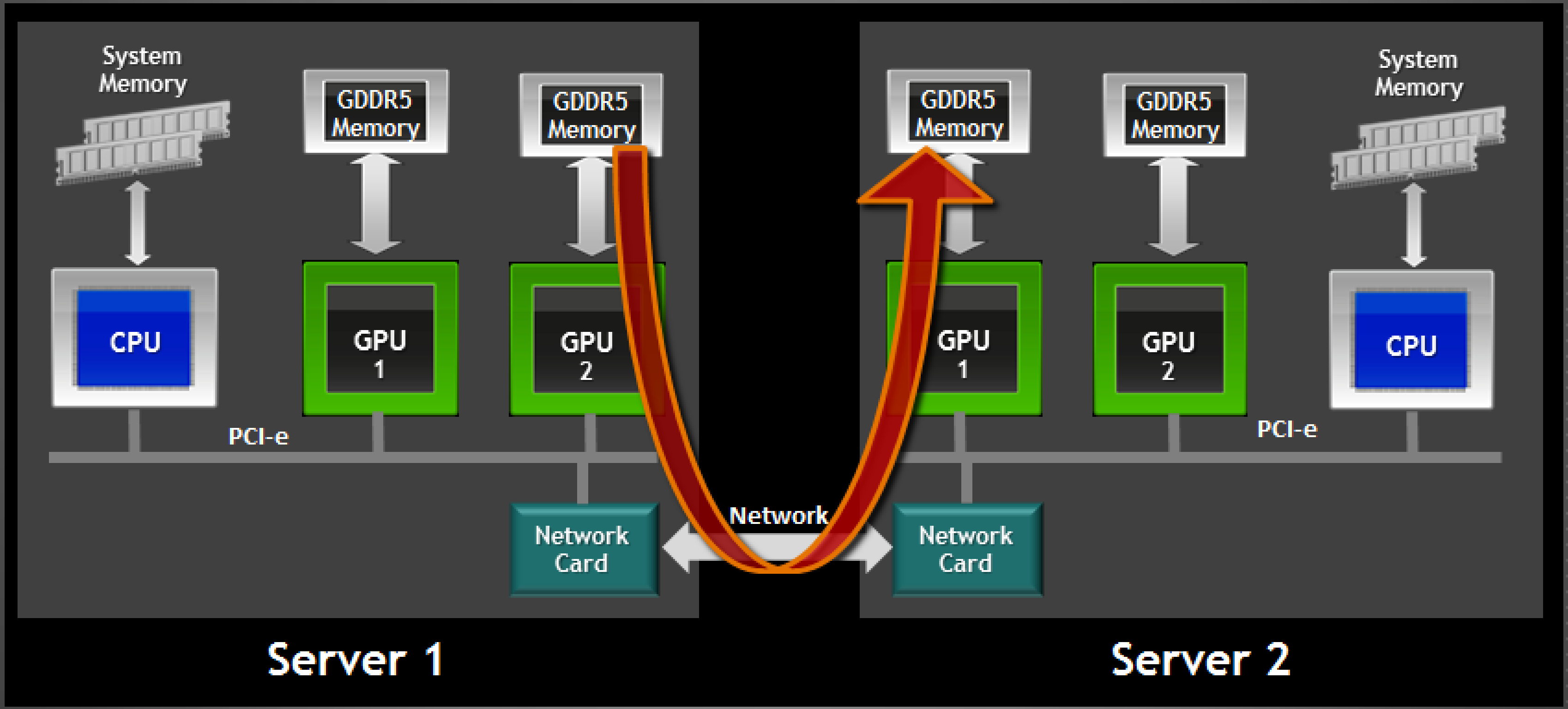
- From host memory: 250 GB/s
- Coalesced reads are absolutely necessary
- Fetching cache lines at a time (float4)

SPACEX

- Part of a cluster for propulsion analysis
- 4x computer nodes
  - Hooked up with Infiniband
  - 4x Tesla K20 each
    - (1.17 Tflops single / 3.52 Tflops double / 5 GB)
  - GPUDirect (Mellanox)
- **Overall, 18 Tflops double / 56 Tflops single**
- **Only 96 GB total GPU RAM**

MULTI-GPU PROGRAMMING

- Peer-to-peer addressing
- Unified virtual addressing
- GPUDirect (https://developer.nvidia.com/gpudirect)



CONTROL FLOW OF PROJECT

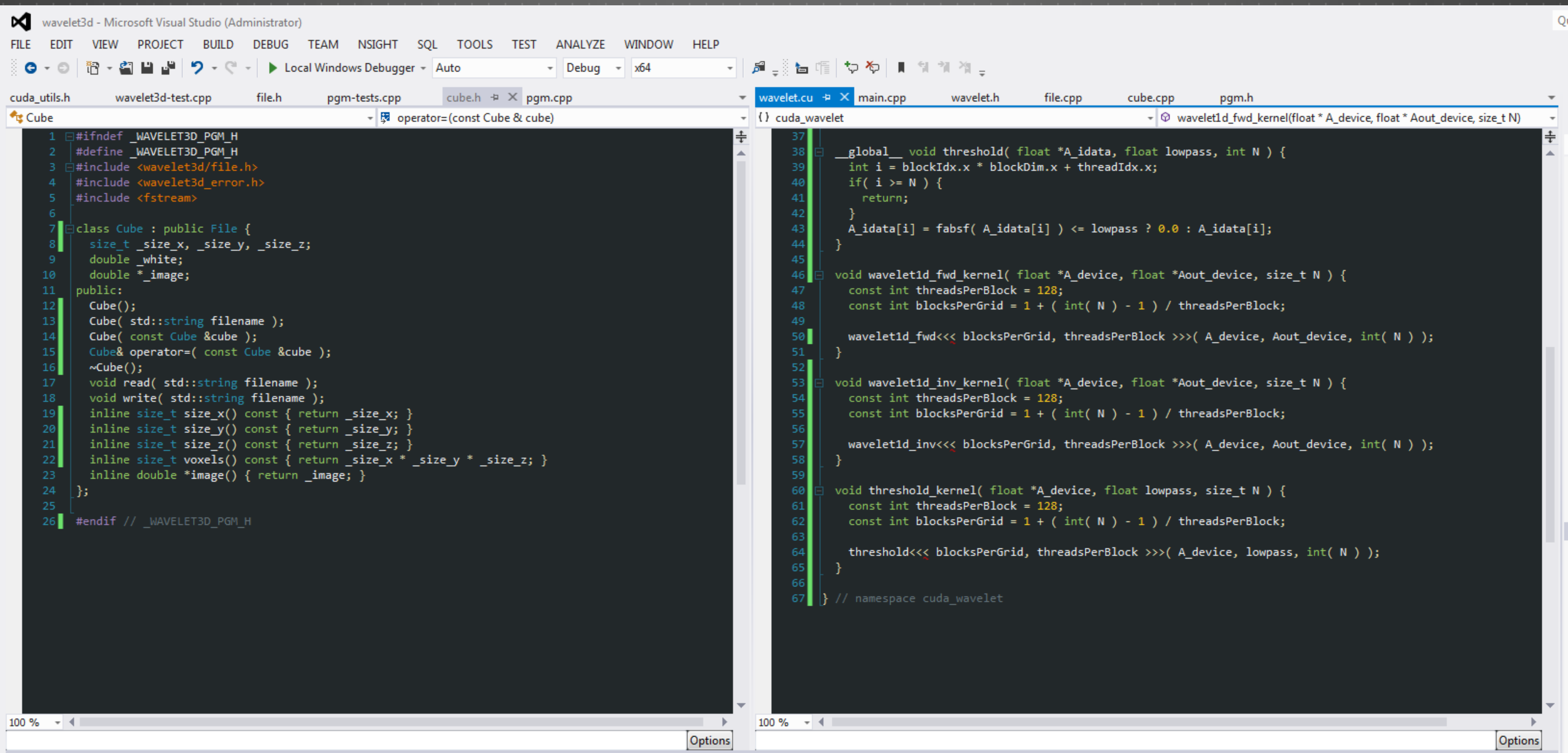
1. Binary data file read (3d "pgm")
2. Stream to GPU, saturating global device memory
3. Compress the data in the GPU
4. 90% or more of the RAM is now free – stream in more, and compress.

PERFORMANCE TRICKS

- 3D data, better than 2d data, can be fetched in two cache lines:
  - One voxel cube and its 7 "minor" neighbors fits in two cache lines, and therefore is very efficient to fetch.

THE CODE

- General development was done in Visual Studio due to excellent CUDA debugging tools (“Nsight”), but actual performance testing done on cluster running Ubuntu



NSIGHT PERFORMANCE ANALYSIS

wavelet3d - wavelet3d_d131209_001_Capture_000.nvreport

wavelet3d_d131209_...pture_000.nvreport

CUDA Launches | Hierarchy | Flat

Filter

	Function Name	Grid Dimensions	Block Dimensions	Start Time (µs)	Duration (µs)	Occupancy	Registers per Thread	Static Shared Memory per Block (bytes)	Dynamic Shared Memory per Block (bytes)	Cache Configuration Executed	Local Memory per Thread (bytes)	Device Name
1	wavelet1d_fwd	{1, 1, 1}	{128, 1, 1}	654,130.061	100.288	75.00 %	34	0	0	PREFER_SHARED	0	Quadro K1000M
2	wavelet1d_fwd	{1, 1, 1}	{128, 1, 1}	942,326.829	93.536	75.00 %	34	0	0	PREFER_SHARED	0	Quadro K1000M
3	wavelet1d_fwd	{1, 1, 1}	{128, 1, 1}	1,197,253.485	91.040	75.00 %	34	0	0	PREFER_SHARED	0	Quadro K1000M
4	threshold	{1, 1, 1}	{128, 1, 1}	1,321,520.621	4.128	100.00 %	10	0	0	PREFER_SHARED	0	Quadro K1000M
5	wavelet1d_inv	{1, 1, 1}	{128, 1, 1}	1,479,782.029	8.832	75.00 %	34	0	0	PREFER_SHARED	0	Quadro K1000M
6	wavelet1d_inv	{1, 1, 1}	{128, 1, 1}	1,635,792.109	8.832	75.00 %	34	0	0	PREFER_SHARED	0	Quadro K1000M
7	wavelet1d_inv	{1, 1, 1}	{128, 1, 1}	1,789,757.517	8.800	75.00 %	34	0	0	PREFER_SHARED	0	Quadro K1000M

All Kernel-Level Experiments

Select this experiment group to collect kernel-level experiments. Please note that this template adds significant overhead to the target application. When this group is selected, the following experiments will be run.

Experiment	Description
Achieved FLOPS	Calculates the achieved single/double floating point operations per second.
Achieved IOPS	Calculates the achieved integer operations per second.
Achieved Occupancy	Calculates the occupancy achieved at runtime of the kernel.
Branch Statistics	Collects efficiency metrics for the kernel's usage of flow control.
Instruction Statistics	Collects instructions per clock cycle (IPC), instructions per warp (IPW) and SM activity.
Issue Efficiency	Collects efficiency metrics for issuing the kernel's instructions.
Memory Statistics - Global	Provides information about the global memory requests, transactions, and bandwidth.
Memory Statistics - Local	Provides information about the local memory requests, transactions, and bandwidth.
Memory Statistics - Atomics	Provides information about atomic operations and the resulting memory transactions.
Memory Statistics - Shared	Provides information about the shared memory requests, transactions, and bandwidth.
Memory Statistics - Texture	Provides information about about texture memory usage, such as texture fetch rates and texture bandwidth.
Memory Statistics - Caches	Provides information about the efficiency of the L1/L2 caches.
Memory Statistics - Buffers	Provides information about memory accesses to device memory as well as system memory.
Pipe Utilization	Collects utilization metrics for the functional pipes of each SM.

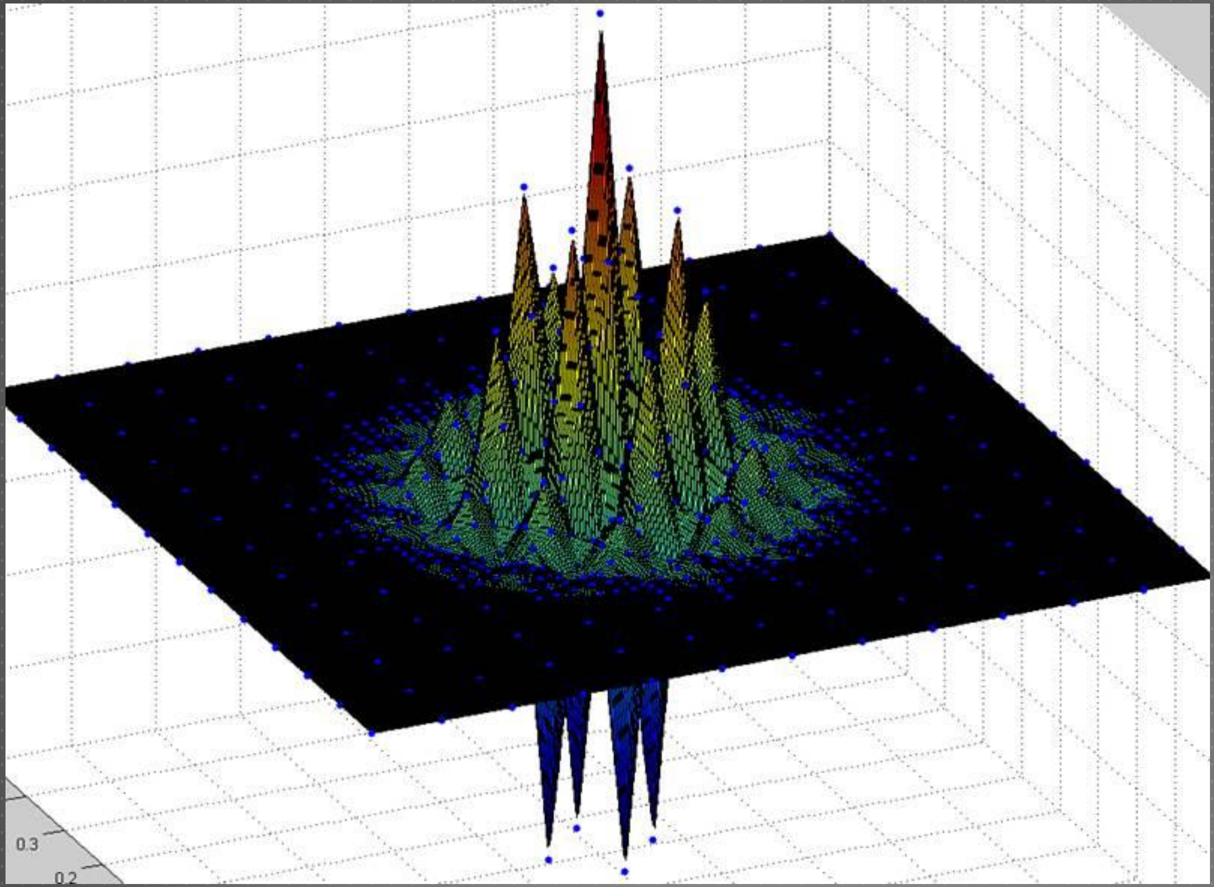
OVERLAPPING MEMCPYS

- Stagger for best time usage!

```
for( int i = 0; i < numGPUs; ++i ) {
    CUDART_CHECK( cudaSetDevice( i ) );
    CUDART_CHECK( cudaMalloc( <<<>>> );
    CUDART_CHECK( cudaMemcpyAsync( <<<>>>, cudaMemcpyHostToDevice ) );
}
for ( int i = 0; i < numGPUs; ++i ) {
    CUDART_CHECK( cudaSetDevice( i ) );
    wavelet3d_fwd_kernel( <<<>>> );
    CUDART_CHECK( cudaMemcpyAsync( <<<>>>, cudaMemcpyDeviceToHost ) );
}
```

2D COMPRESSION RESULT

- Compression ratio: 185.97 (max err: 0.003)



2D → 3D

- Compression ratio will only improve, drastically.
- *Particularly effective* for data which represents "lower dimensionality" in a higher-dimensional space.

FUTURE (SOON) WORK

- Utilization of all compute nodes
- Actual compliant implementation of the real JP3D standard – easier to import data
- More types of wavelets – Bezier patches, Daubechies for more vanishing moments
  - Much more accurate than Haar/similar and needed for JP3D standard.

QUESTIONS?