GPU TECHNOLOGY CONFERENCE

3D BACKPROJECTION

MEETING THE CHALLENGE FOR PERFORMANCE IN MEDICAL IMAGING

LARS NYLAND
JULIEN DEMOUTH
SKY WU
FEIWEN ZHU
NVIDIA

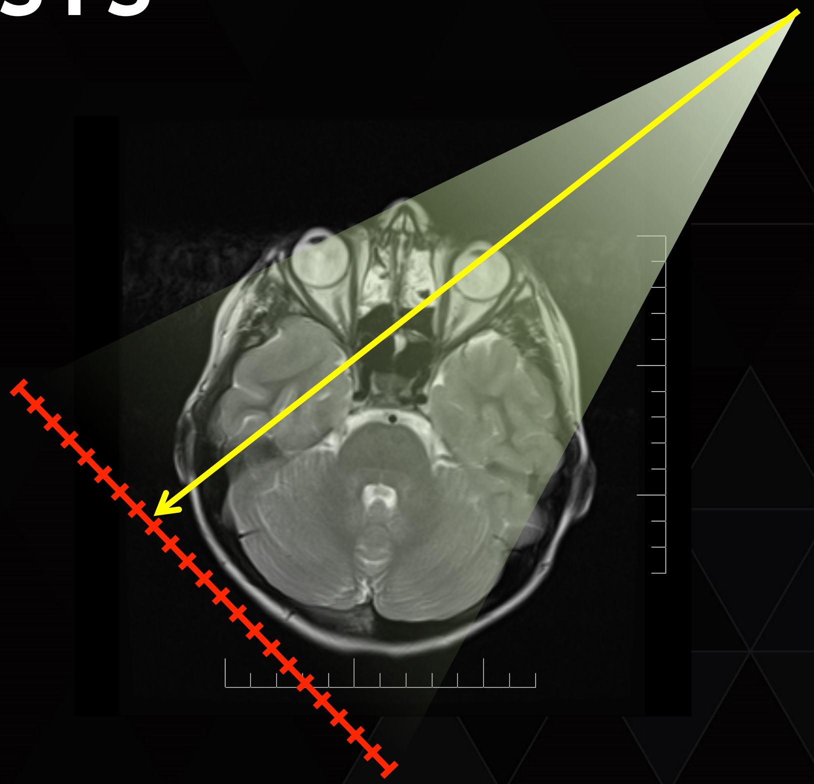
SUPPORTING APPLICATIONS

- Julien Demouth -- DevTech
  - Understanding customer applications and how they map to NVIDIA hardware
- Lars Nyland, Sky Wu, and Feiwen Zhu -- Compute Architecture Group
  - Study how applications perform on GPUs
  - Propose capabilities and performance enhancements
  - Work with DevTech to improve customers' codes
- Example: Medical Imaging
  - DevTech and Compute Arch spend significant time understanding performance of several codes
- Occasionally enlist help of compiler team

GPU TECHNOLOGY CONFERENCE

MEDICAL IMAGING INTERESTS

- Goals
  - To see inside the body without cutting it open
- Methods
  - Computed Tomography (CT)
  - Magnetic Resonance Imaging (MRI)
  - Ultrasound
- 2d, 3d
  - Lines over time (2d reconstruction)
  - Images over time (3d reconstruction)

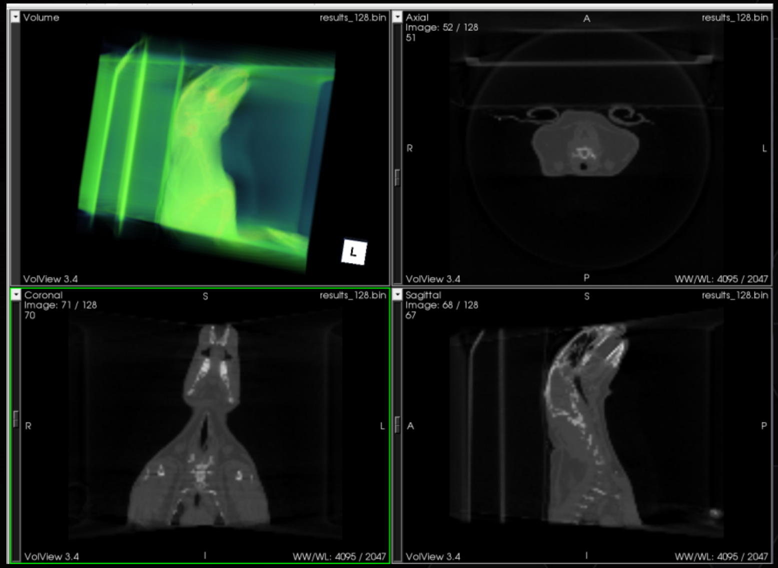


CT RECONSTRUCTION

- Sequence of 2d images rotating around patient

Projected images

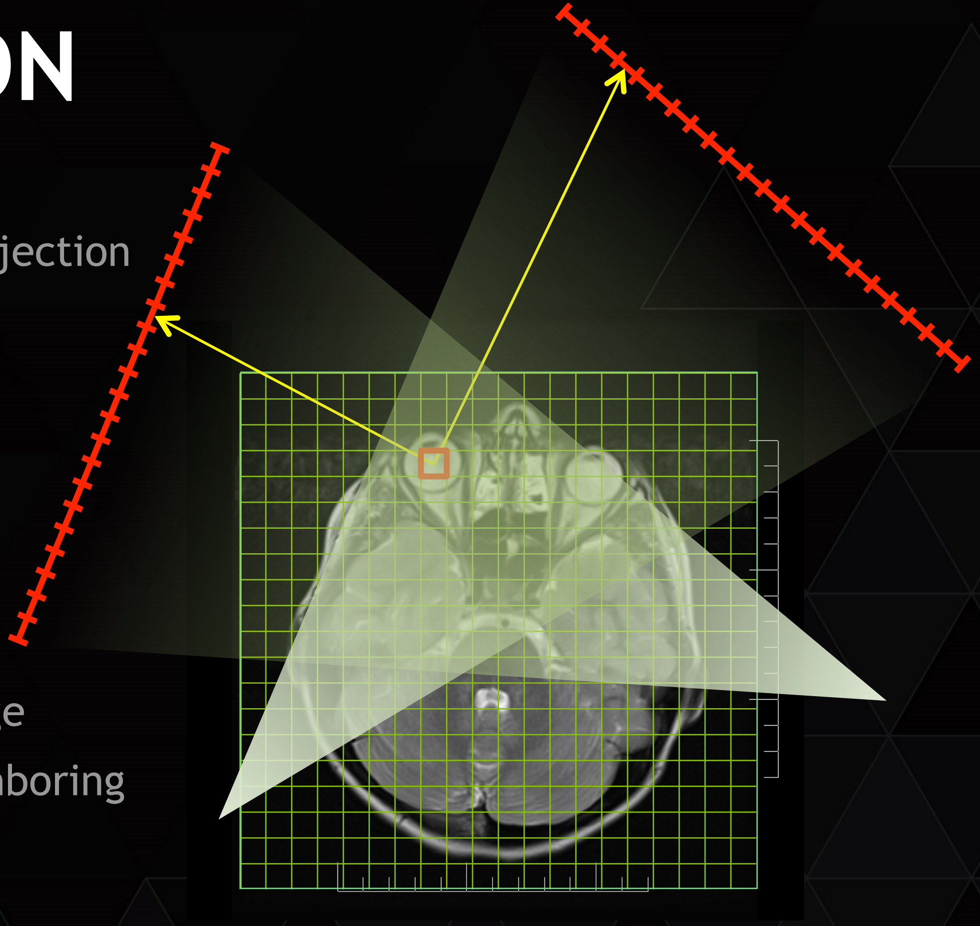
Filtered Backprojection



Reconstructed 3D volume

FILTERED BACKPROJECTION

- Input
  - A list of projected images and the corresponding projection matrices
- Output
  - A reconstructed 3D cube
- Algorithm
  - For each projected image:
    - For each voxel in the 3D cube:
      - Project the center of the voxel on the image
      - Compute the bilinear interpolation of neighboring pixels
      - Add the value to the voxel (with a weight)



RABBITCT BENCHMARK

- Filtered backprojection benchmark: http://www5.cs.fau.de/research/projects/rabbitct/
  - C. Rohkohl *et al. RabbitCT---an open platform for benchmarking 3D cone-beam reconstruction algorithms. Med. Phys. 36*, 3940 (2009)

- 496 projected images of size 1248x960

Ranking

Problem size: 256 | **512** | 1024

Rank	Algorithm Description	q_{rmse}* [HU]	Error Hist.†	PSNR‡ [dB]	Time+ [s]	Performance*
1	**Elmer** Submitter: Sam Hawker Institution: Nikon Metrology RabbitCT dataset version: 2 Date of submission: 2014-04-26 Elmer is a multi-GPU implementation using CUDA 5.5	0.16		88.29	0.4	100.00%
2	**Thumper** Submitter: Timo Zinßer Institution: Siemens AG RabbitCT dataset version: 2 Date of submission: 2013-06-11 Thumper is a CUDA-based back-projection implementation.	0.17		87.39	0.6	66.67%

GPU TECHNOLOGY CONFERENCE

OPTIMIZED FILTERED BACKPROJECTION

- We implemented
  - T. Zinsser and B. Keck. *Systematic Performance Optimization of Cone-Beam Back-Projection on the Kepler Architecture*. *Proceedings of the 12th Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2013*

- Key ideas
  - Each kernel launch works on a batch of images
  - Each thread works on several voxels (each threads works on 4 x 2 voxels)

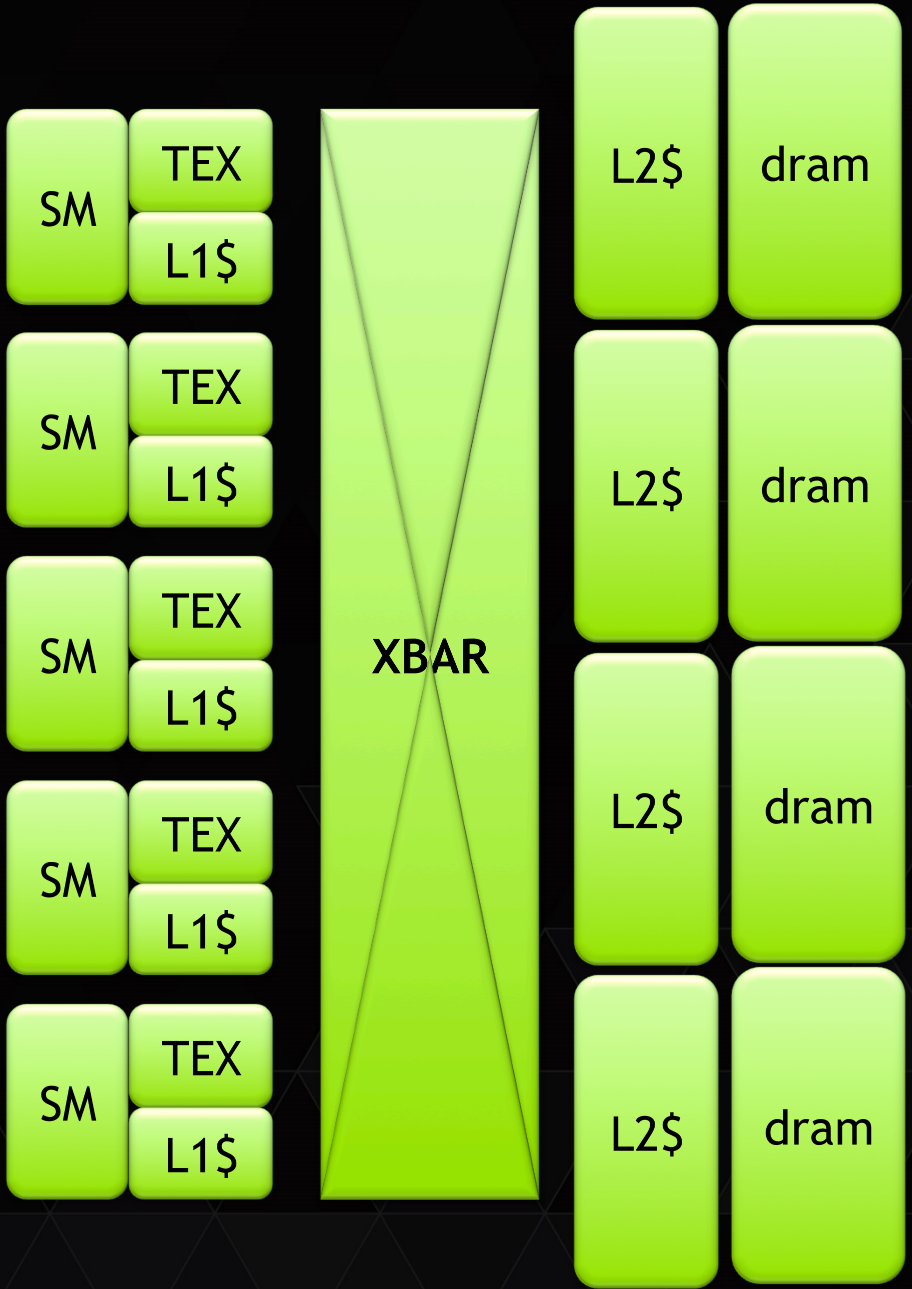
- Not implemented
  - Smart strategy to partially hide the first H2D and last D2H copies

ARCHITECTURE DIGRESSION

- Kepler
  - 1st architecture to optimize for power
  - 4 texture units/sm
- Maxwell
  - Improve performance/watt
  - 2 texture units/sm
  - Can support more SMs in same power
- Expectations
  - Maxwell's clocks are faster
  - Maxwell GPUs have more SMs
  - Nearly as many texture units between Tesla K40 and Quadro M6000

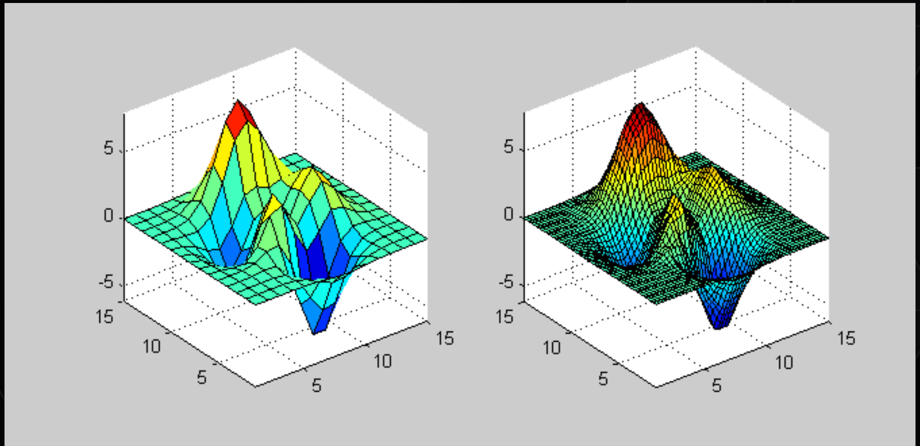
GPU ORGANIZATION

- SM does the bulk of the computation
  - Math, loop control
  - LD/ST memory ops
- Other units can perform computations
  - Texture has interpolation hardware
  - L2$ has atomic memory units
- You can use them to lighten the load on SM
  - Specialized computations
  - Close to memory values



DEDICATED INTERPOLATION HARDWARE

- GPUs have interpolation hardware in Texture Units
  - Developed for graphics to interpolate texture images
  - Available in **Cuda** for 1, 2 and 3d interpolation
- Similar to MATLAB's *interp1*, *interp2*, and *interp3* functions
  - Nearest, linear, and cubic interpolations available
- Interpolation Operation
  - Index array with floats, not integers
  - Example: A[1.3][9.74] -> tex2d(A, 1.3, 9.74)
  - Returns 1-4 values at interpolated location
  - Source data is bytes, half-words, or floats
  - Safe out-of-bounds handling

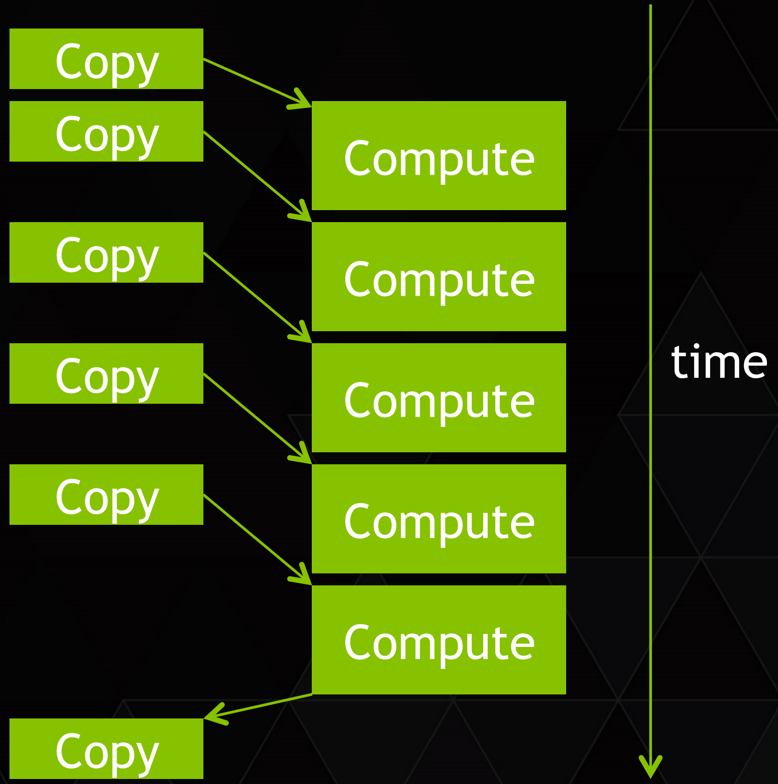


ATOMIC MEMORY OPERATIONS

- Update values in memory with no interference
  - Where "update" means "do some math using memory value as input and output"
- Ex. atomicAdd(&A[i], t);
  - Perform A[i] += t; with no interference
  - If no return value, operation is "fire and forget" (SM does not wait)
  - Otherwise, returns old value of A[i], like a load operation
  - Available types for atomicAdd():
    - Int32, fp32

BACKPROJECTION CPU-GPU STRATEGY

- Overlap copy and compute
- CPU copies 1st batch of images to GPU
- Launches 1st kernel
- Without waiting, copy next batch of images
- Launch next kernel
- Repeat until done
- Copy final results back to CPU



BACKPROJECTION IMPLEMENTATION

- Each thread works on 8 voxels (4 in Y dimension, 2 in Z dimension)

```
int x = (blockIdx.x*blockDim.x + threadIdx.x);
int y = (blockIdx.y*blockDim.y + threadIdx.y)*4;
int z = (blockIdx.z)*2;
```

- Iterate over the images

```
float p0 = 0.f, p1 = 0.f, p2 = 0.f, p3 = 0.f, ...;
for( int i = 0 ; i < NUM_IMAGES_PER_BATCH ; ++i )
{
  // Project and update the 8 voxels
}
atomicAdd(&dst[(z+0)*CUBE_SIDE_2 + y*CUBE_SIZE + x], p0);
atomicAdd(&dst[(z+1)*CUBE_SIDE_2 + y*CUBE_SIZE + x], p1);
atomicAdd(&dst[(z+2)*CUBE_SIDE_2 + y*CUBE_SIZE + x], p2);
atomicAdd(&dst[(z+3)*CUBE_SIDE_2 + y*CUBE_SIZE + x], p3);
...
```

VOXEL PROJECTION IMPLEMENTATION

- Homogeneous coordinates for the 8 voxels (only 1st and 2nd shown)

```
float u0 = d_proj[12*i+0]*x + d_proj[12*i+3]*y + d_proj[12*i+6]*z + d_proj[12*i+ 9];
float v0 = d_proj[12*i+1]*x + d_proj[12*i+4]*y + d_proj[12*i+7]*z + d_proj[12*i+10];
float w0 = d_proj[12*i+2]*x + d_proj[12*i+5]*y + d_proj[12*i+8]*z + d_proj[12*i+11];

float u1 = u0 + 1.0f*d_proj[ID][12*i+6]; // 2nd voxel, increase in z.
float v1 = v0 + 1.0f*d_proj[ID][12*i+7];
float w1 = w0 + 1.0f*d_proj[ID][12*i+8];
```

- Project and update the voxels in registers (only 1st voxel shown)

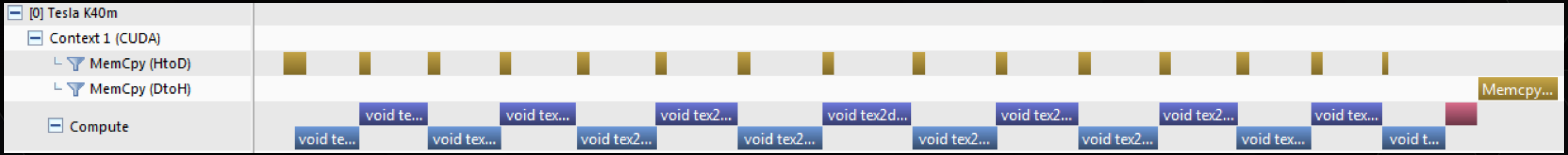
```
float inv_w0 = 1.f / w0;

float texu0 = u0*inv_w0 + 0.5f;
float texv0 = v0*inv_w0 + 0.5f;

p0 += tex2DLayered<float>(src, texu0, texv0, i) * (inv_w0*inv_w0);
```

APPLICATION PROFILE

- NVIDIA Tesla K40m (CUDA 7.0), cube 512^3, data in fp32



- Limited by kernel performance
  - Reconstructed cube, stored on the device
  - Projected images and matrices, transferred in batches (32 per kernel)
- H2D copies at ~12GB/s over PCIE 3.0
  - Hidden behind kernel computation

GPU TECHNOLOGY CONFERENCE

APPLICATION PERFORMANCE

- Performance in GUPS (higher is better), input data in fp16 / fp32

Cube	K40m (Kepler)	M6000 (Maxwell)	Limiter
128	10 / 5	10 / 5	H2D/D2H copies
256	63 / 30	73 / 33	Kernels
512	**102 / 82**	**151 / 112**	**Kernels**
1024	108 / 106	155 / 154	Kernels

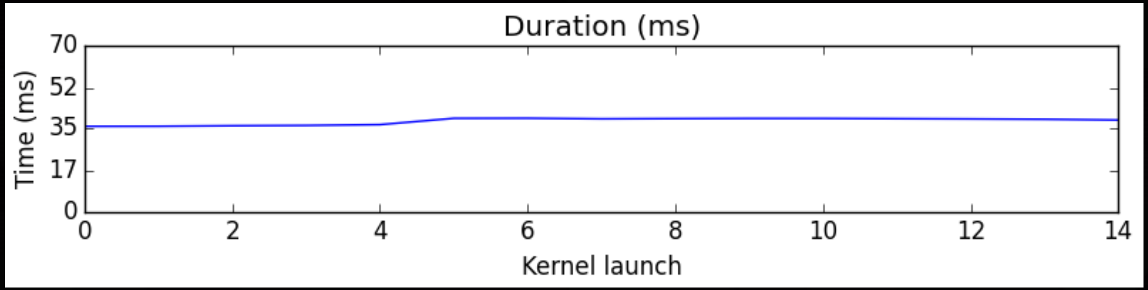
- For 512^3, Intel Xeon Phi 5110P reaches 9 GUPS (fp32) [1] (~12x slower than M6000)

[1] J. Hofmann *et al. Comparing the performance of different x86 SIMD instruction sets for a medical imaging application on modern multi- and manycore chips*. In *Proceedings of the 2014 Workshop on Programming models for SIMD/Vector processing* (WPMVP '14). 2014

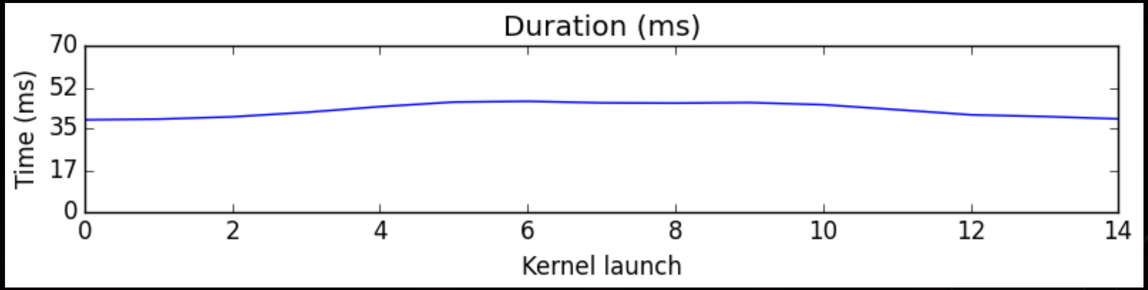
PERFORMANCE ON K40

- Kernel performance varies during reconstruction

fp16



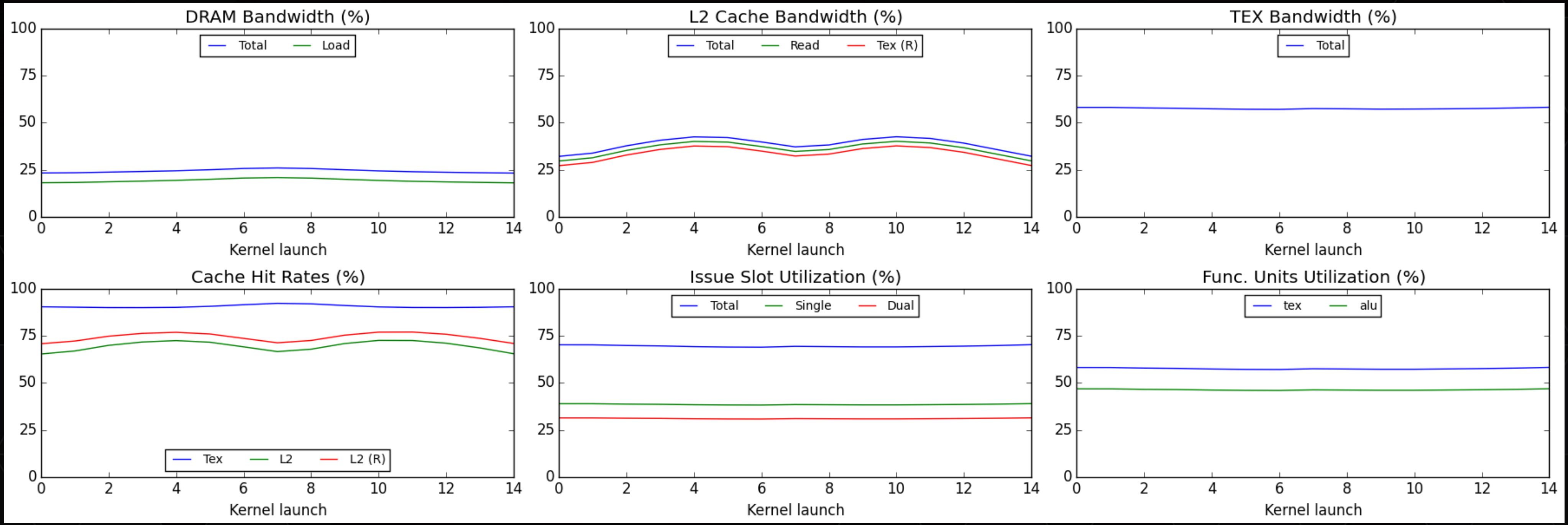
fp32



- Execution time depends on projection matrix

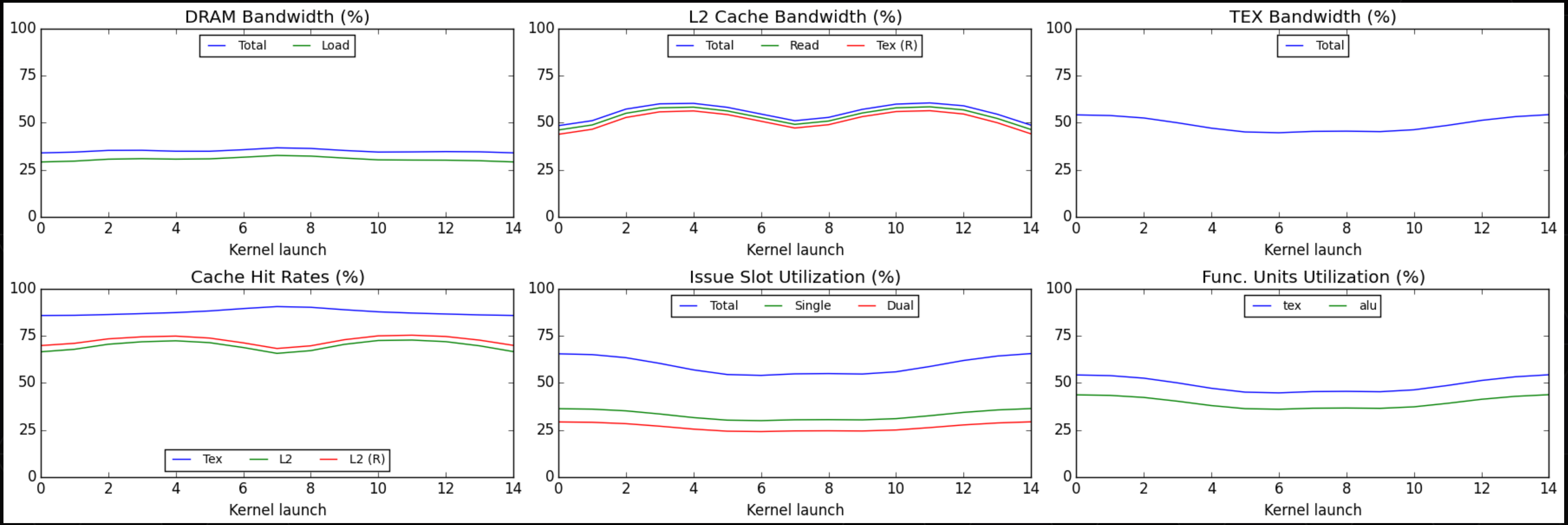
KERNEL METRICS (FP16, K40)

- No clear performance limiter. Latency bound.



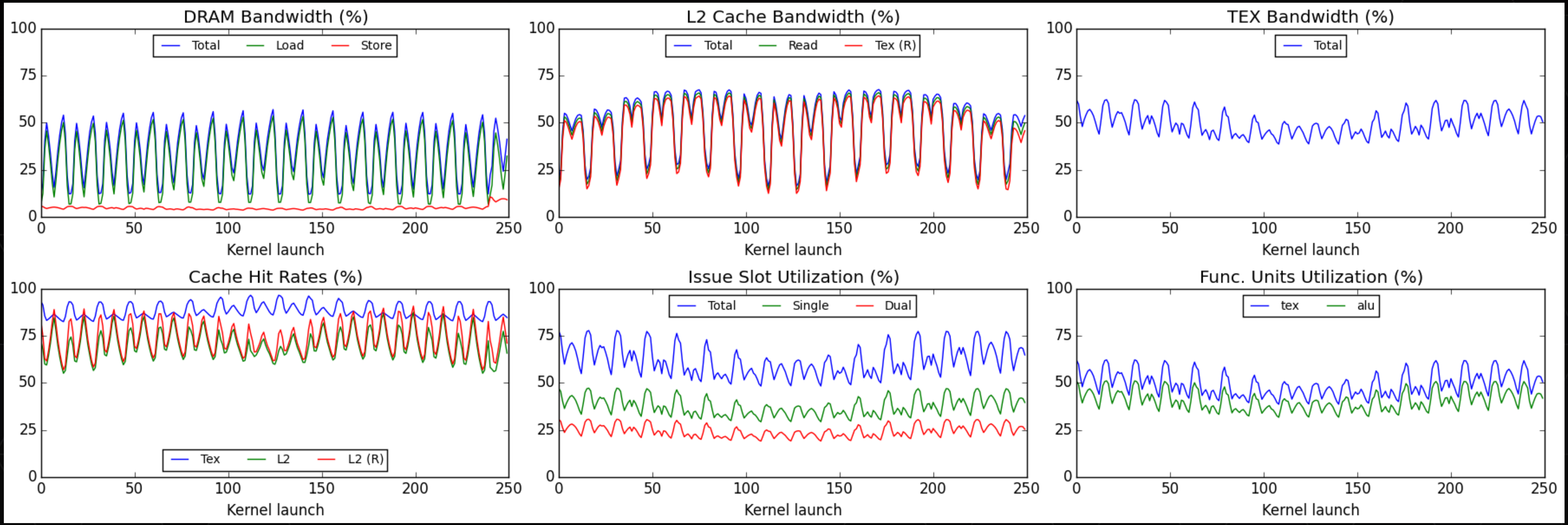
KERNEL METRICS (FP32, K40)

- Still no clear performance limiter. Latency bound.



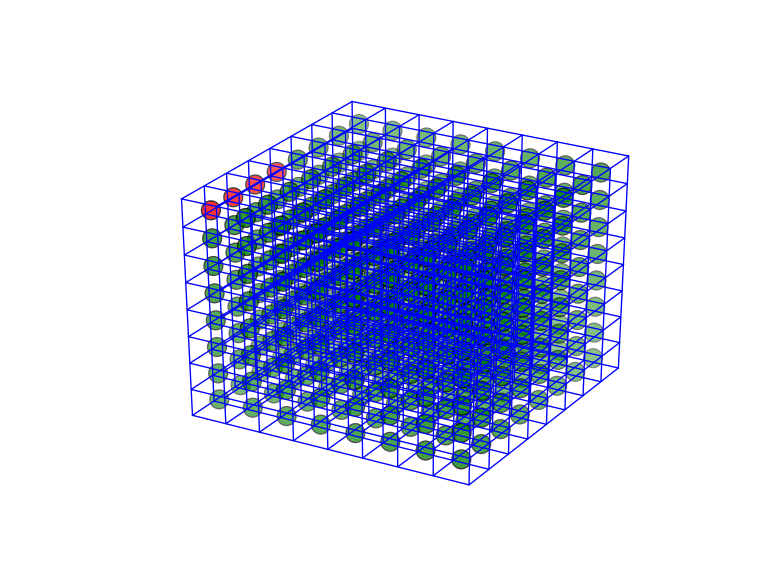
DETAILED PERFORMANCE

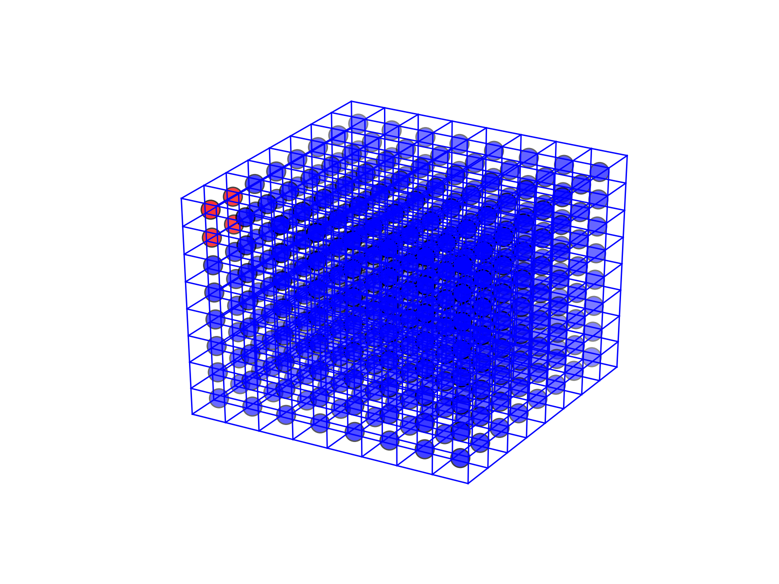
- Split the kernel. Each kernel computes 16 slices in the Z dimension



AUTO-TUNING

- Ideas
  - Process different numbers of voxels per thread (2, 4, 8, 16)
  - Distribute the voxels per thread differently: In Y dimension, in Z dimension, in both





AUTO-TUNING

- Improved performance

Cube	K40m (with)	K40m (without)
128	10 / 5	10 / 5
256	86 / 46	63 / 30
512	**117 / 96**	**102 / 82**

- Vary block sizes and vary # of voxels computed per each thread (4 or 8)

PERFORMANCE TECHNIQUE SUMMARY

- Batch data and work
  - Send some data, do some work, repeat and overlap
  - Nearly all data copies hidden behind work
  - Provides scaling to any size problem
- Multiple output voxels per thread
  - Reduces redundant work
  - More efficient use of registers
- Memory accesses exhibit spatial and temporal locality
  - By mapping of voxels to threads and voxels to thread groups
- Use arithmetic hardware outside the SM
  - Texture performs interpolation
  - L2 performs per-voxel sums with atomics

CONCLUSIONS

- Medical Imaging Applications run well on GPUs
- Customer collaboration with NVIDIA
  - Better performance today
  - Better support in the future

NVIDIA REGISTERED DEVELOPER PROGRAMS

- Everything you need to develop with NVIDIA products
- Membership is your first step in establishing a working relationship with NVIDIA Engineering
  - Exclusive access to pre-releases
  - Submit bugs and features requests
  - Stay informed about latest releases and training opportunities
  - Access to exclusive downloads
  - Exclusive activities and special offers
  - Interact with other developers in the NVIDIA Developer Forums

REGISTER FOR FREE AT: developer.nvidia.com

BACKUP: ACHIEVED OCCUPANCY

39 active warps / clock for both fp16 and fp32