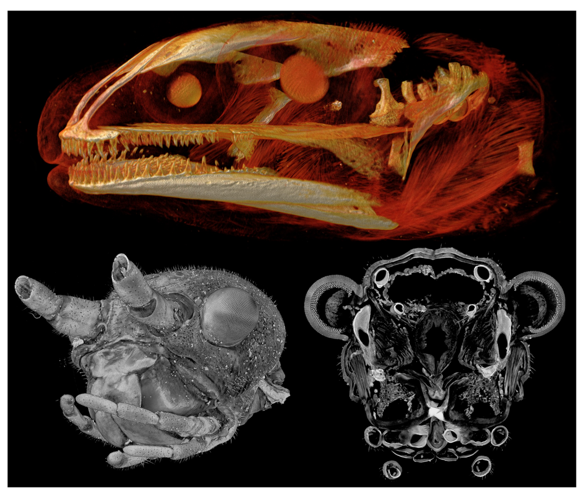
A GPU-based Architecture for Real-Time Data Assessment at Synchrotron Experiments

S. Chilingaryan[1], M. Caselle[1], T. Farago[1], A. Kopmann[1], A. Mirone[2], T. dos Santos Rolo[1], U. Stevanovic[1], M. Vogelgesang[1]

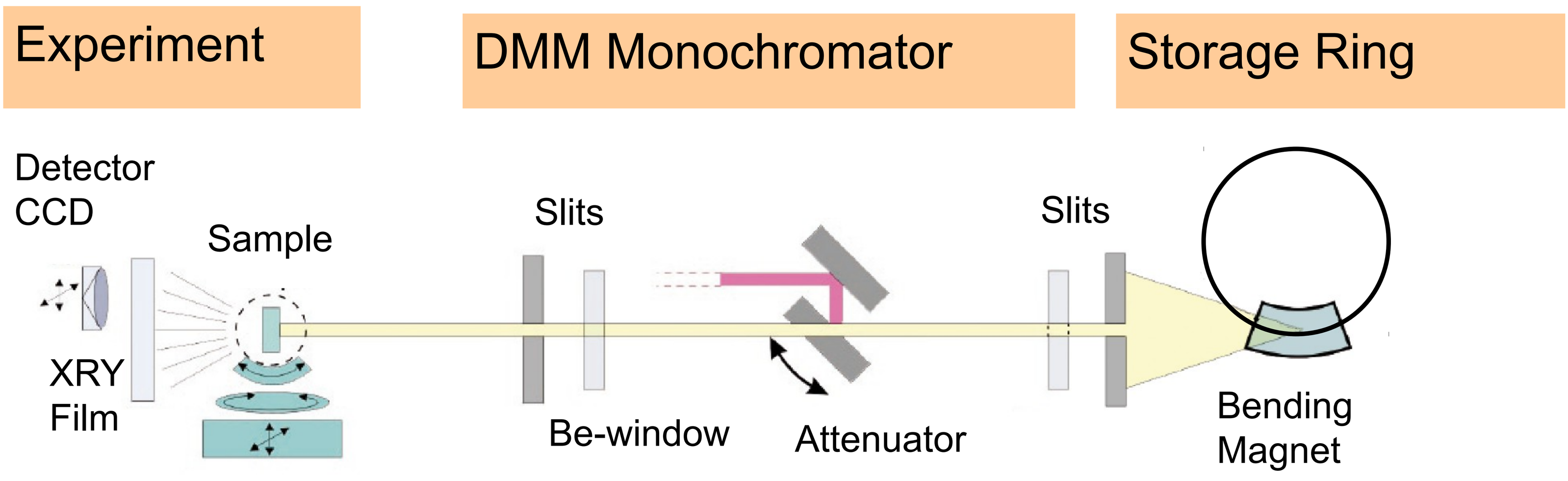
High Speed X-Ray Imaging

X-ray tomography has been proven to be a valuable tool for understanding internal, otherwise invisible, mechanisms in biology, materials research and other fields. Detectors employed at modern synchrotrons are able to deliver images with high resolution and at high frame rates generating up to several gigabytes per second. The ability to process this information in real-time and present to the users without long processing delays is extremely important for synchrotron operation. It will increase experiment throughput and enable image-based control of dynamical processes under study. We have developed a GPU-based platform for high speed tomography optimized for continuous operation with streamed data. Our system consists of dedicated hardware platform, a camera abstraction layer, a pipelined parallel programming framework, and a high-speed implementation of tomographic reconstruction. Using only a single GPU server we are able to handle the full throughput of CameraLink interface with 850 MB/s.

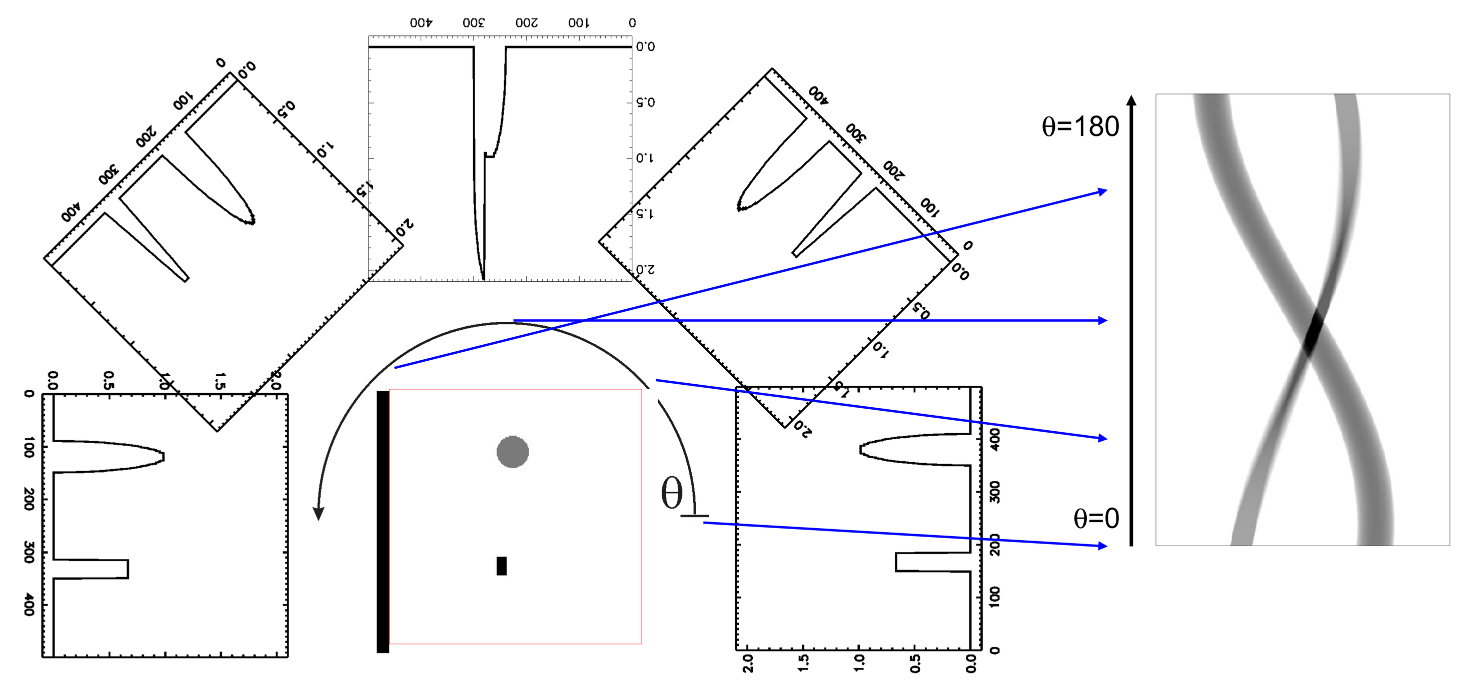


Heads of a newt larva showing bone formation and muscle insertions (top) and a stick insect (bottom), acquisition time 2s.

Tomography at Synchrotron Light Sources



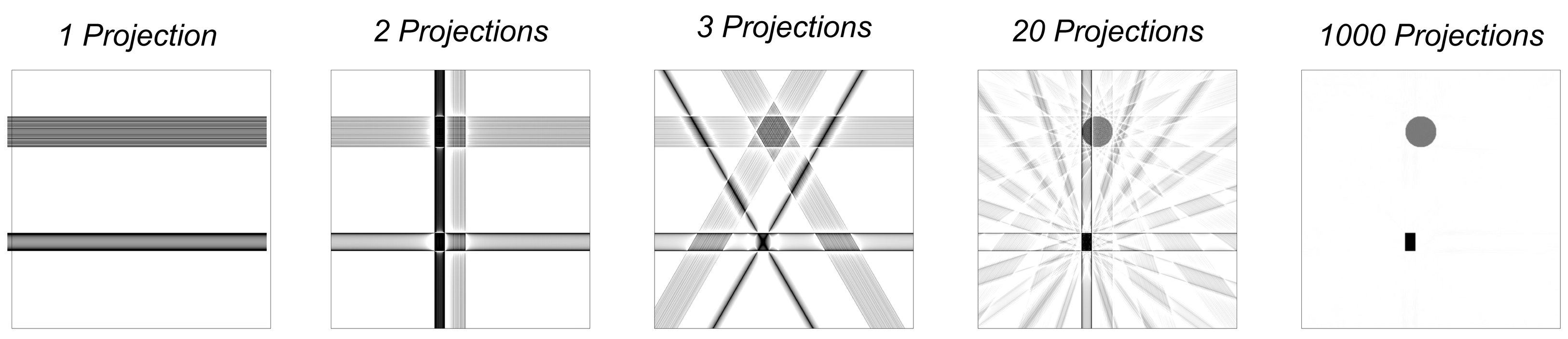
The sample, evenly rotating in the front of a pixel detector, is penetrated by X-rays produced in the synchrotron



The pixel detector registers series of parallel 2D projections of the sample density at different angles.

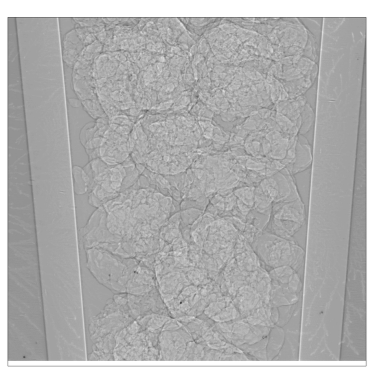
3D Image Reconstruction

According to the Back Projection algorithm, the pixel at position *(x,y,z)* is computed by $\sum_{p=1}^{P} I_p(x\cdot\cos(pa)-y\cdot\sin(pa),z)$, where *P* is the number of projections *α* is the angle between projections, and I_p is the image of *p*-th projection.



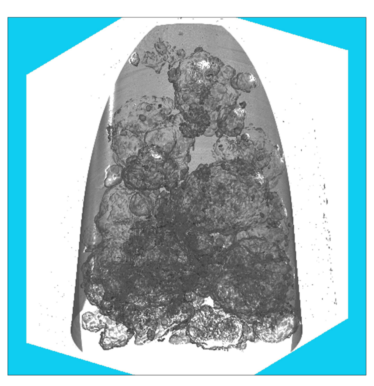
Filtered back-projection is used to reconstruct 3D images from a manifold of 2D projections. The projection values are smeared back over 2D cross sections and integrated over all projection angles. To reduce blurring effect the projections are filtered in the Fourier space before being back projected.

Typical Setup

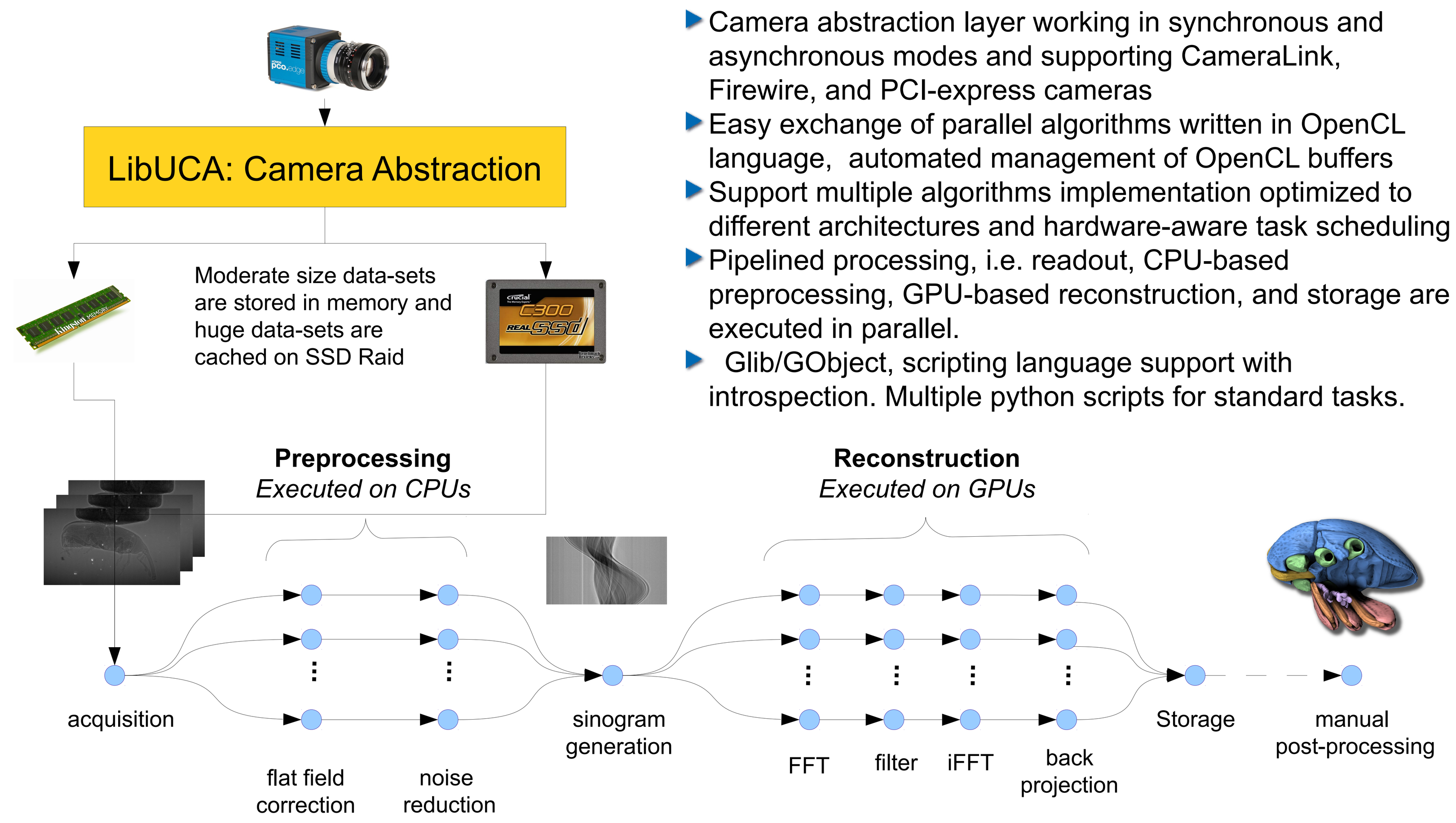


Sample: Plastic holder with porose polyethylene grains
Source data: 24GB (2000 projections, 3 Mpix, 32 bits)
3D Image: 11GB (3 Gpix, 32 bits)
Complexity: 53 Tflop back-projection + 0.6 Tflop filtering

Goal: Reconstruct 3D image in 1 minute

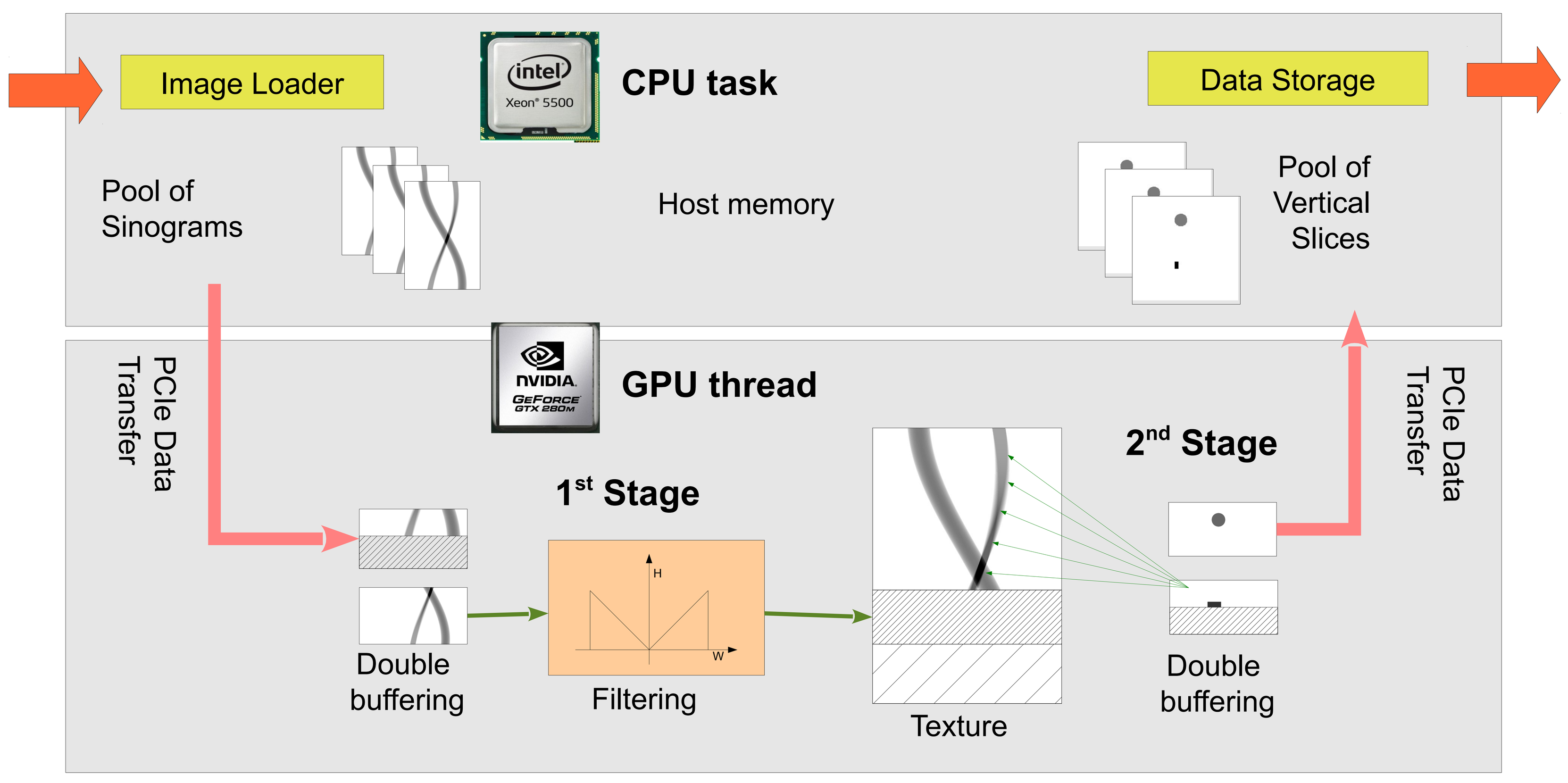


Open Source Parallel Processing Framework



- Camera abstraction layer working in synchronous and asynchronous modes and supporting CameraLink, Firewire, and PCI-express cameras
- Easy exchange of parallel algorithms written in OpenCL language, automated management of OpenCL buffers
- Support multiple algorithms implementation optimized to different architectures and hardware-aware task scheduling
- Pipelined processing, i.e. readout, CPU-based preprocessing, GPU-based reconstruction, and storage are executed in parallel.
- Glib/GObject, scripting language support with introspection. Multiple python scripts for standard tasks.

Filtered Back Projection



- Four stage pipeline is used. Data is prefetched from SSDs, preprocessed on CPUs, reconstructed using all GPUs, and stored to the real-time storage.All 4 stages are executed in parallel.
- Pinned (unswappable) memory buffers are used to speed-up data transfers. The slice is split in blocks and the transfer of next block is interleaved with reconstruction of the current one.
- Batched mode of Fourier library is used for better performance. Two real convolutions are computed using a single complex FFT transform
- An-architecture specific variations of back projection algorithm is to better utilize GPU resources

Tunning Back Projection for Hardware Architectures

GT200
Base version
Uses texture engine

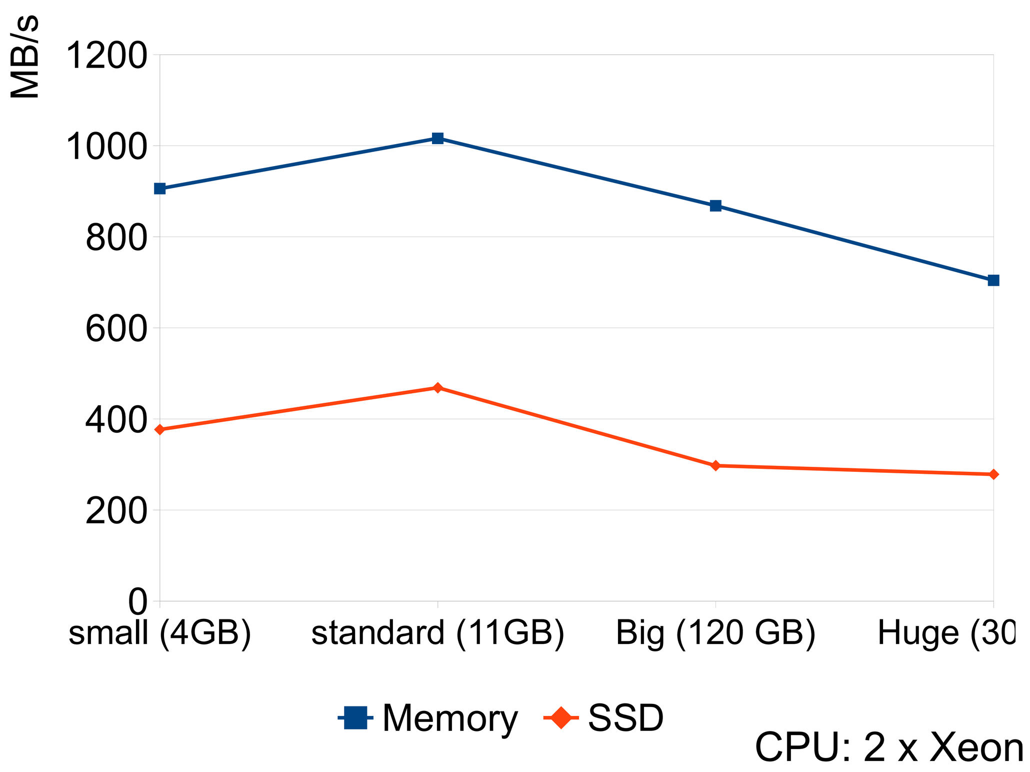
Fermi +100%
High computation power, but low speed of texture unit
Reduce load on texture engine: use shared memory to cache the fetched data and, then, perform linear interpolation using computation units.

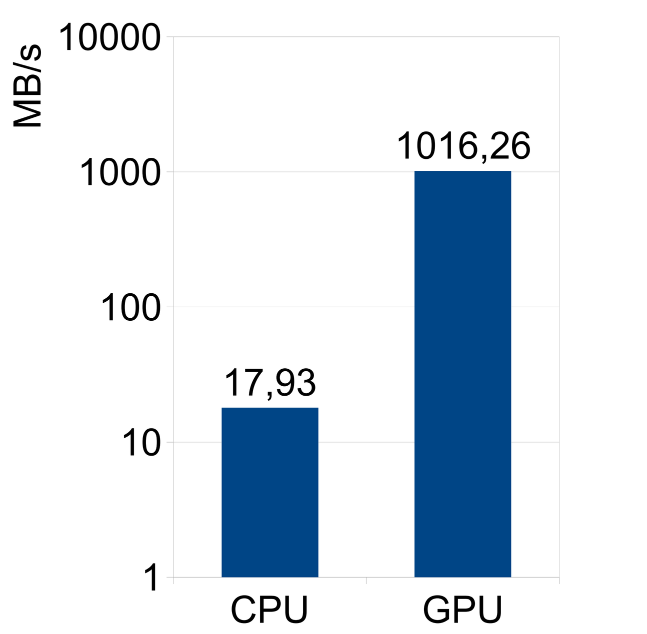
Kepler +75%
Low bandwidth of integer instructions, but high register count
Uses texture engine, but processes 16 projections at once and 16 points per thread to enhance cache hit rate

VLIW +530%
Executes 5 independent operations per thread
Computes 16 points per thread in order to provide sufficient flow of independent instructions to VLIW engine

GCN +95%
High performance of texture engine and computation nodes
Balance usage of texture engine and computation nodes to get highest performance

Performance





CPU: 2 x Xeon X5650 (total 12 cores at 2.66 Ghz)
GPUs: 2 x GTX 580 + 4 x GTX580 External

Contact: http://ufo.kit.edu
Suren Chilingaryan <Suren.Chilingaryan@kit.edu>
Andreas Kopmann <Andreas.Kopmann@kit.edu>

[1] *Karlsruhe Institute of Technology, Karlsruhe, Germany*
[2] *European Synchrotron Radiation Facility, Grenoble, France*