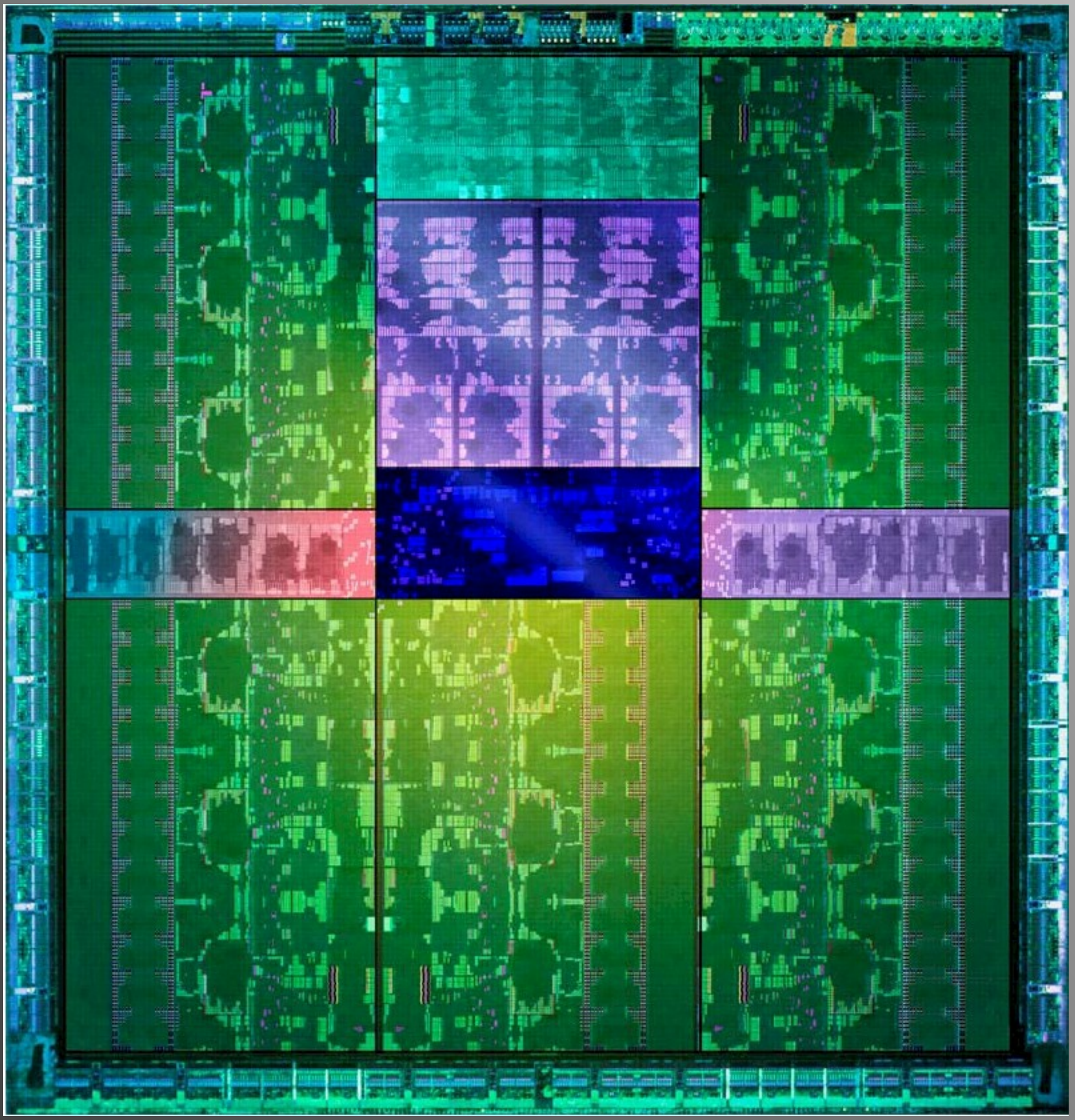
How many GPUs do we really need ?



Arnaud Sevin, Damien Gratadour & Julien Brule
LESIA, Observatoire de Paris

Outline

- Possible GPU-based architectures
- GPU environment
- Measuring latency and jitter
- Problem scale
- Pure compute performance
- Jitter
- Future works

GPU-based architectures

- GPUs : accelerators / coprocessors = device in a host (CPU)
- Connection via PCIe lanes (x16 Gen3 :128 Gb/s)
- Several setups for full bandwidth
  - Desktop : single socket (40 PCIe lanes) + 1/2 GPUs
  - Workstation : dual-socket + 2/4 GPUs
  - Cluster : proprietary dual-socket nodes (2-3 GPUs) + infiniband interconnect (40Gb/s)
- Other setups throughput oriented
  - PCIe switches behind IOH : up to 8 GPUs for a single IOH (shared bandwidth)
- Through IOH : 40Gb/s in & 32Gb/s out (measured on Fermi with Gen2)
- New Intel X79 chipset (no IOH) up to 48Gb/s in & out (measured on Kepler with Gen3)

GPU-based architectures

- NVIDIA, leader in GPGPU : CUDA architecture & toolkit
  - Tesla Fermi : 16 Streaming Multiprocessor x 32 cores = 512 cuda cores
  - Tesla Kepler K10 : 8 SM x 192 cores = 1536 cuda cores x 2 GPU chips (+ on-board PCIe switch)
  - Up to 6GB on board memory
- Stream processing
  - Several kernel launch instances concurrently
  - 2 copy engines : bi-directional. Asynchronous copy + compute overlap (hide host-to-device / device-to-host memcopy latency + removes device memory limitation)
  - 1 compute engine queue + concurrent kernels : increase performance, maximize GPU utilization for small kernels (Fermi up to 16 concurrent kernels)
  - Get as close as possible to theoretical peak throughput : ~1 TFLOPS

GPU environment

- Choice of OS/kernel
  - SL 6.3 64bits (2.6.32) / nvidia driver 304.54 (cuda5 drivers)
  - Based on SL 6.3 (3.2.23-rt37) / nvidia driver 304.54 RT patched with Red Hat Enterprise MRG tools
- Multi-GPU : peer-to-peer
  - Workstation : CUDA provides peer-to-peer communications between GPUs on the same motherboard.
  - Single IOH / single node : GPUdirect
  - Limited impact on dual-IOH (30% in bandwidth)
- Multi(>2)-GPUs : MPI / openMP + CUDA
  - MPI support for Unified Virtual Adressing (peer-to-peer)
  - GPUdirect over infiniband (shared pinned memory from the host)

Measure latency & jitter

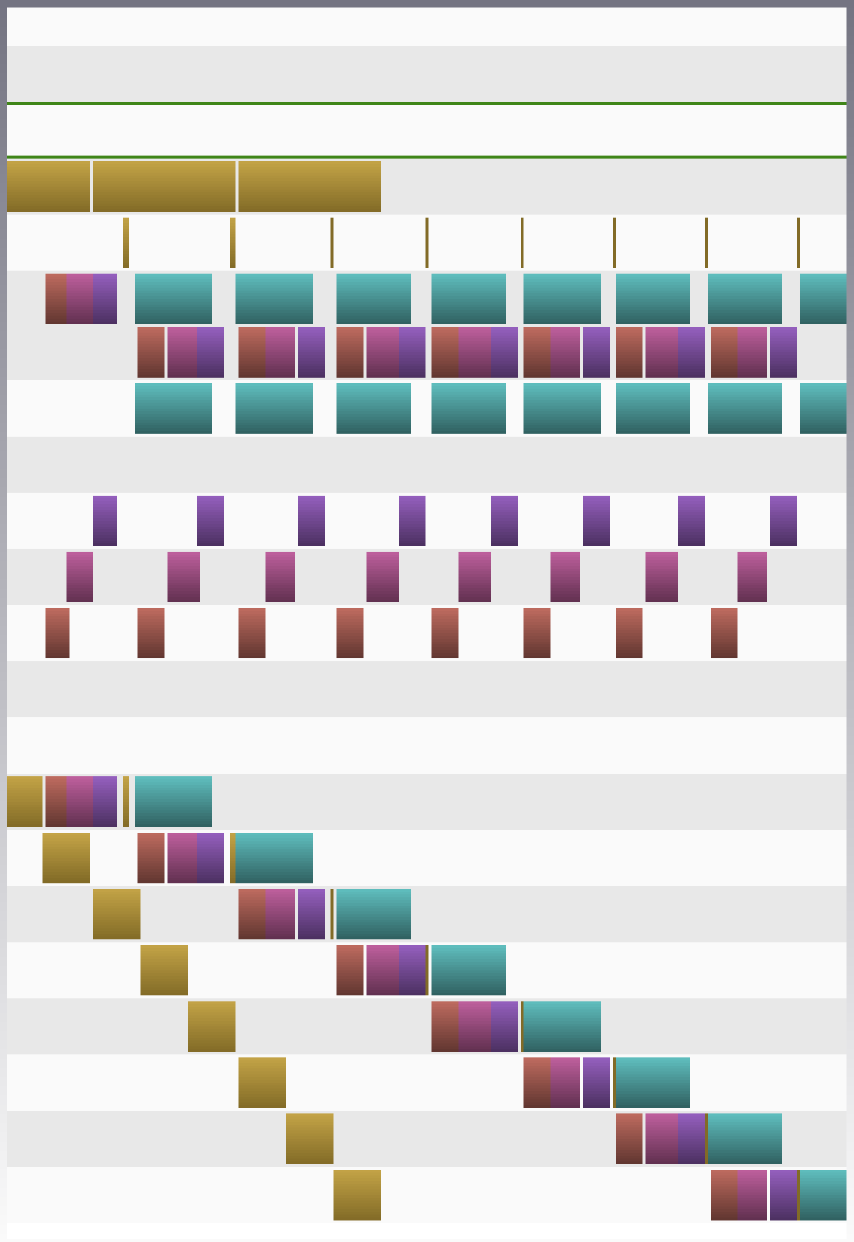
- NVIDIA profiling tools
  - NVVP (graphical) / nvprof (command line) : profiling at the GPU (device) level
  - Better understand CPU-GPU interaction, identify bottlenecks or concurrency opportunities
  - Monitor multi-processor occupancy, optimize code
- TAU / PAPI / CUPTI
  - TAU : Generic tools for heterogeneous systems : profiling at the host level (kernel level profiling available)
  - PAPI CUDA : provides access to hardware counters on NVIDIA GPUs. Based on NVIDIA CUPTI the Cuda Performance Tools Interface. Provide device level access
  - Intrinsically intrusive but limited impact

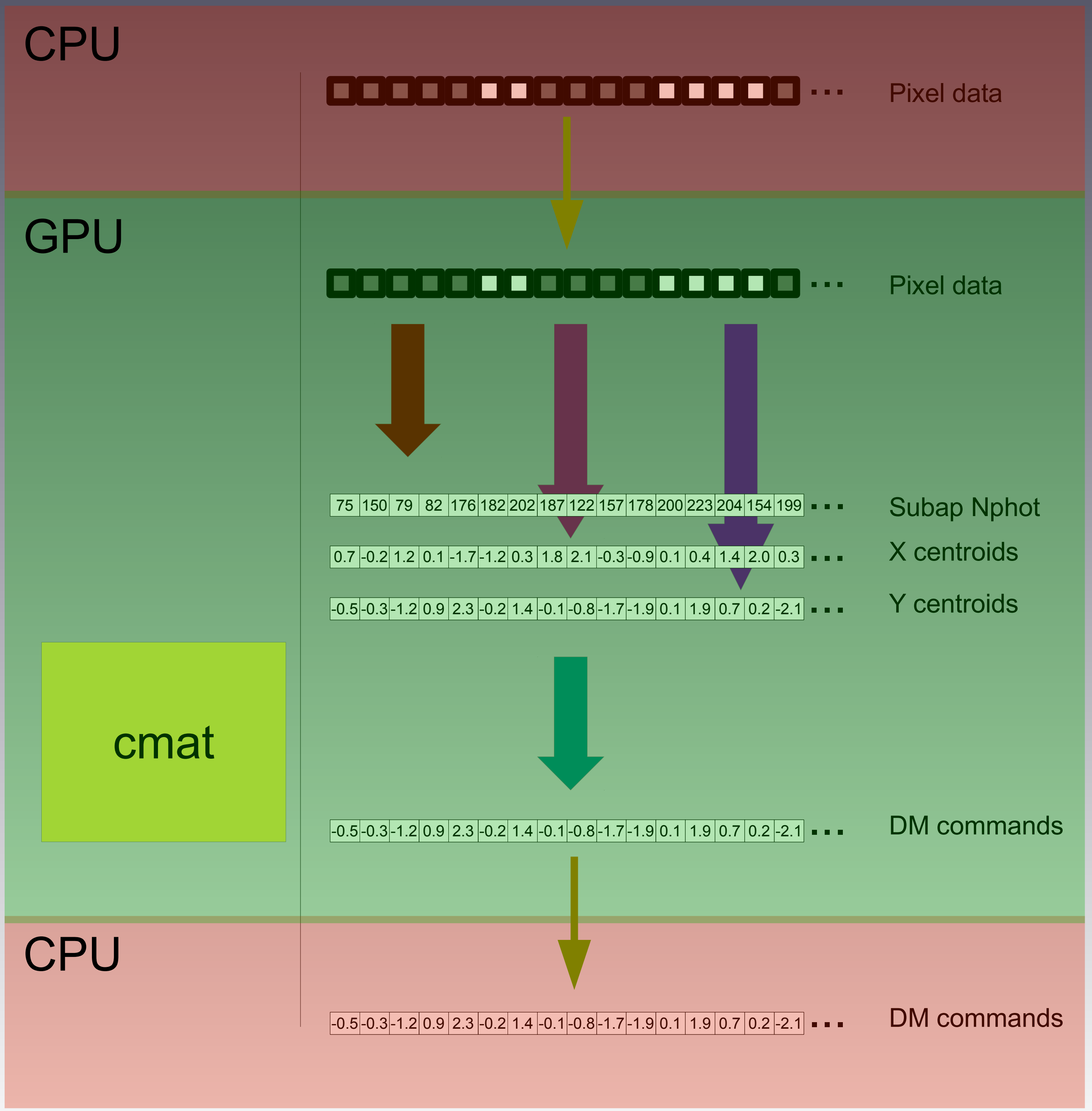
Problem scale

- E-ELT : 40m telescope
- SCAO system : 80x80 subaps with 6x6 pixels
  - ~ 0.25 Mpix per transfer (1Mb) => ~ 1Gb/s (1kHz)
  - 10k slopes ~0.2Mb and 5k DM commands
  - 10k x 5k command matrix : ~1.5Gb
  - MVM : 163MFLOP => 163 GFLOPS (1kHz)
- Multiple WFS / multiple DMs / LGS
  - MAORY : up to 6 LGS WFS with 80x80 subaps, 12x12 for LGS and 3 Dms
  - Bandwidth requirement : ~15Gb/s (1kHz)
  - Command matrix up to ~20Gb
  - Throughput requirement (1kHz) for MVM : 1.5TFLOPs

Benchmarking

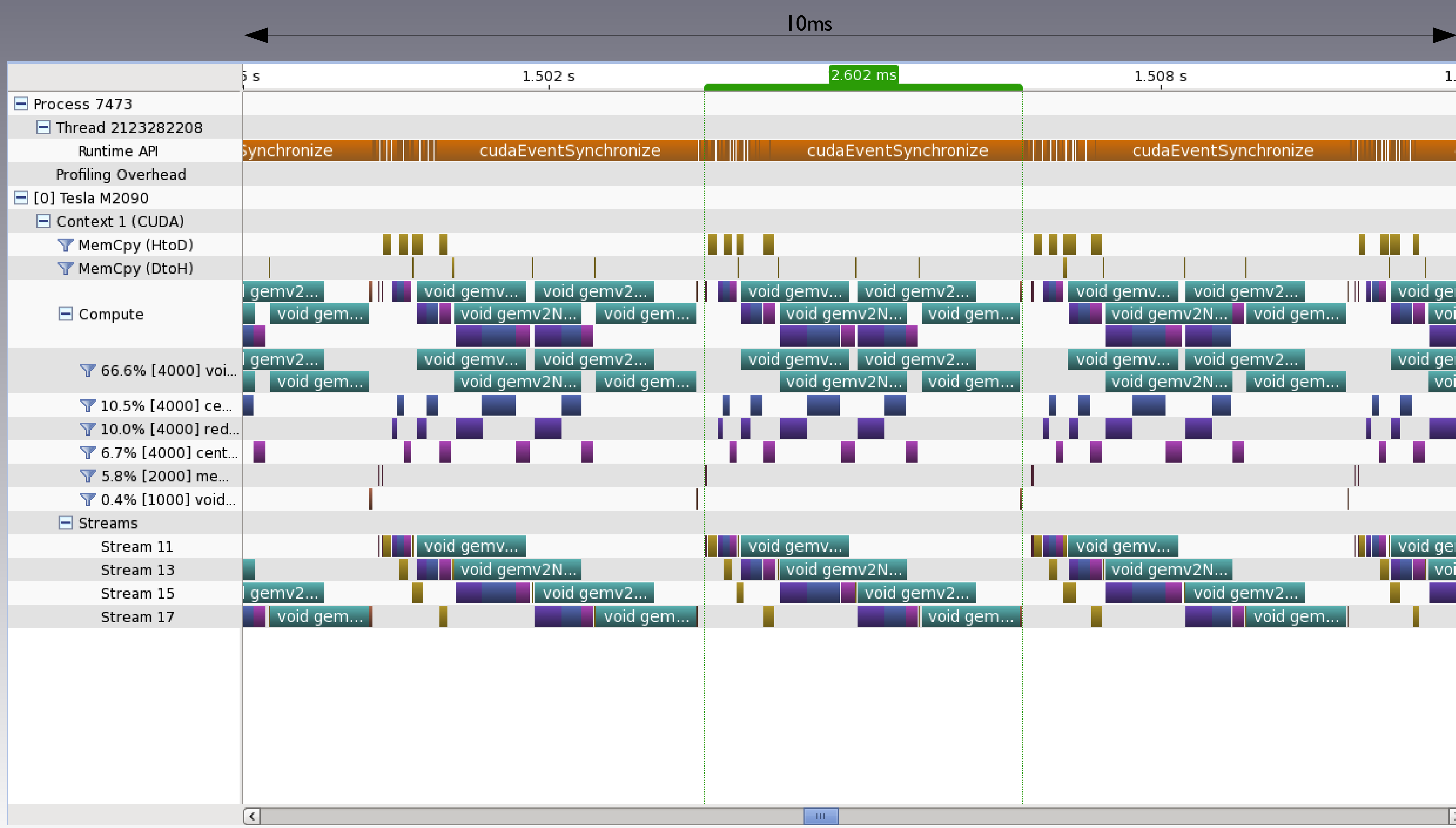
- NVIDIA Visual Profiler: profiling overhead
- Simulating the core RTC process





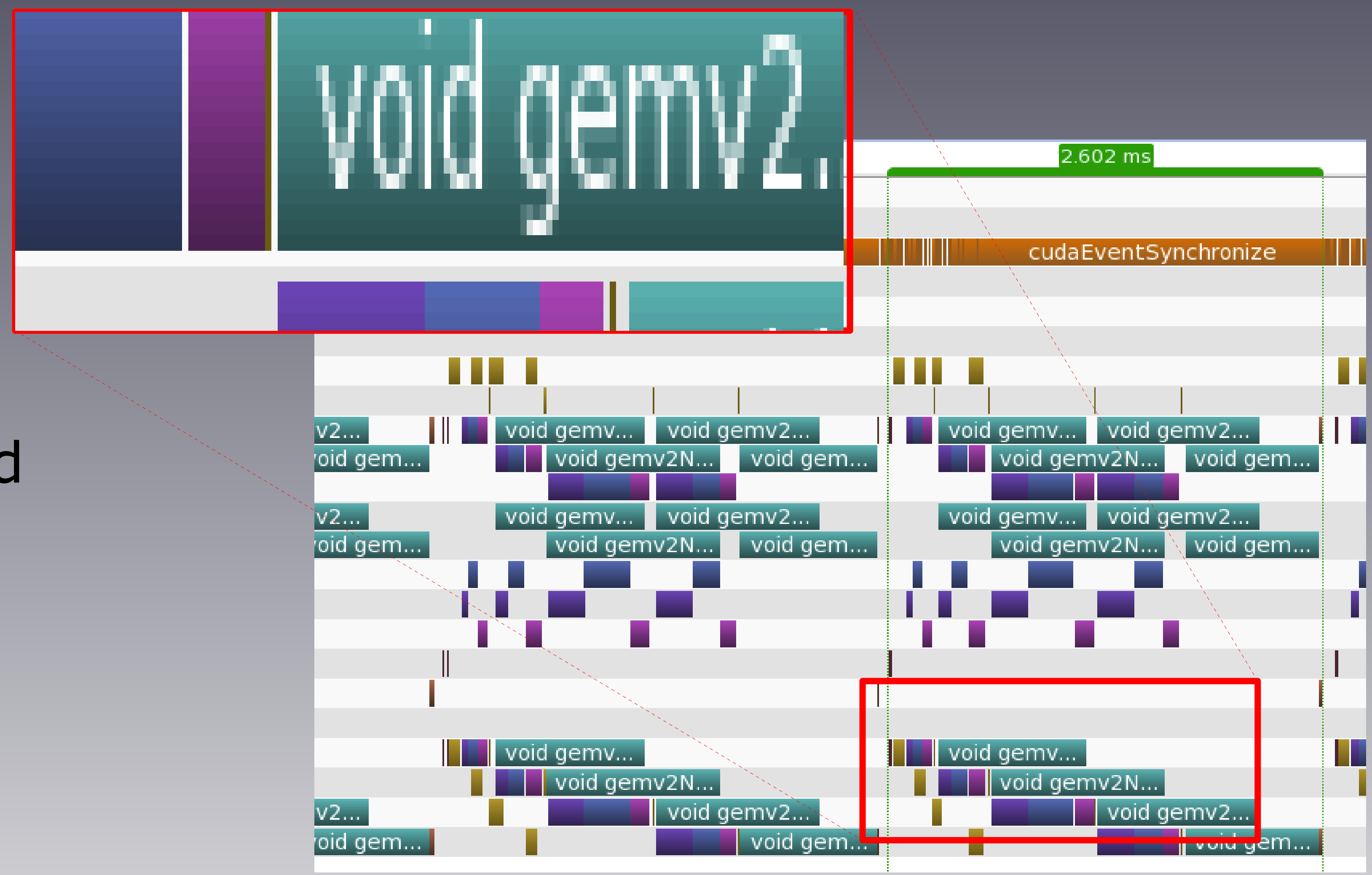
Pure compute performance

- 80x80 (square pupil, no obstruction => upper limit), 6x6 pixels
  - 1 GPU 4 streams



Pure compute performance

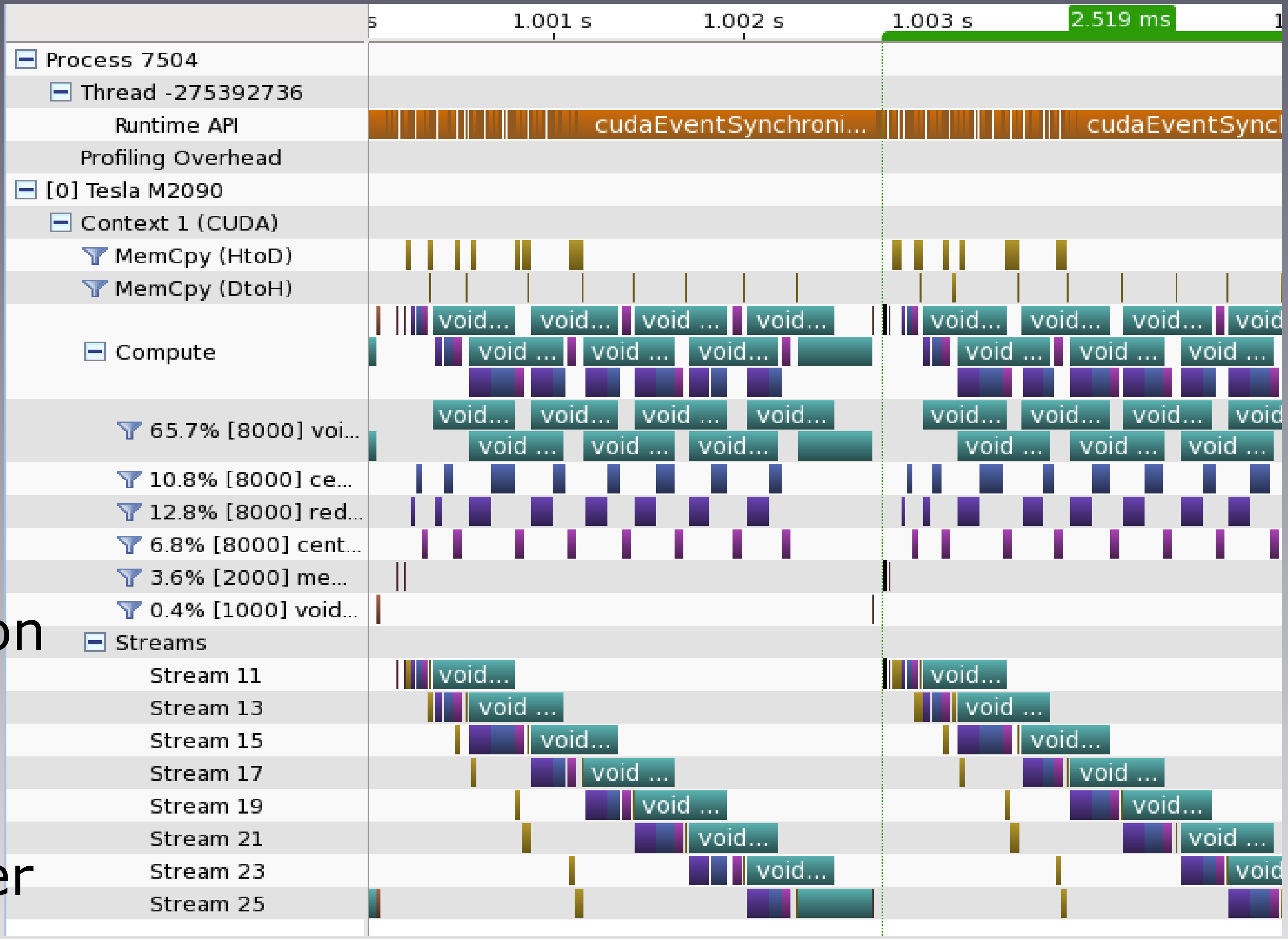
- 80x80 (square pupil, no obstruction => upper limit), 6x6 pixels
  - 1 GPU 4 streams
  - Good level of concurrency (memcopy latency hidding + increased performance)
  - Exec. Time dominated by MVM



  - Centroiding has high multi-processor occupancy
  - Throughput limited (pixel data copy takes about 500μs)

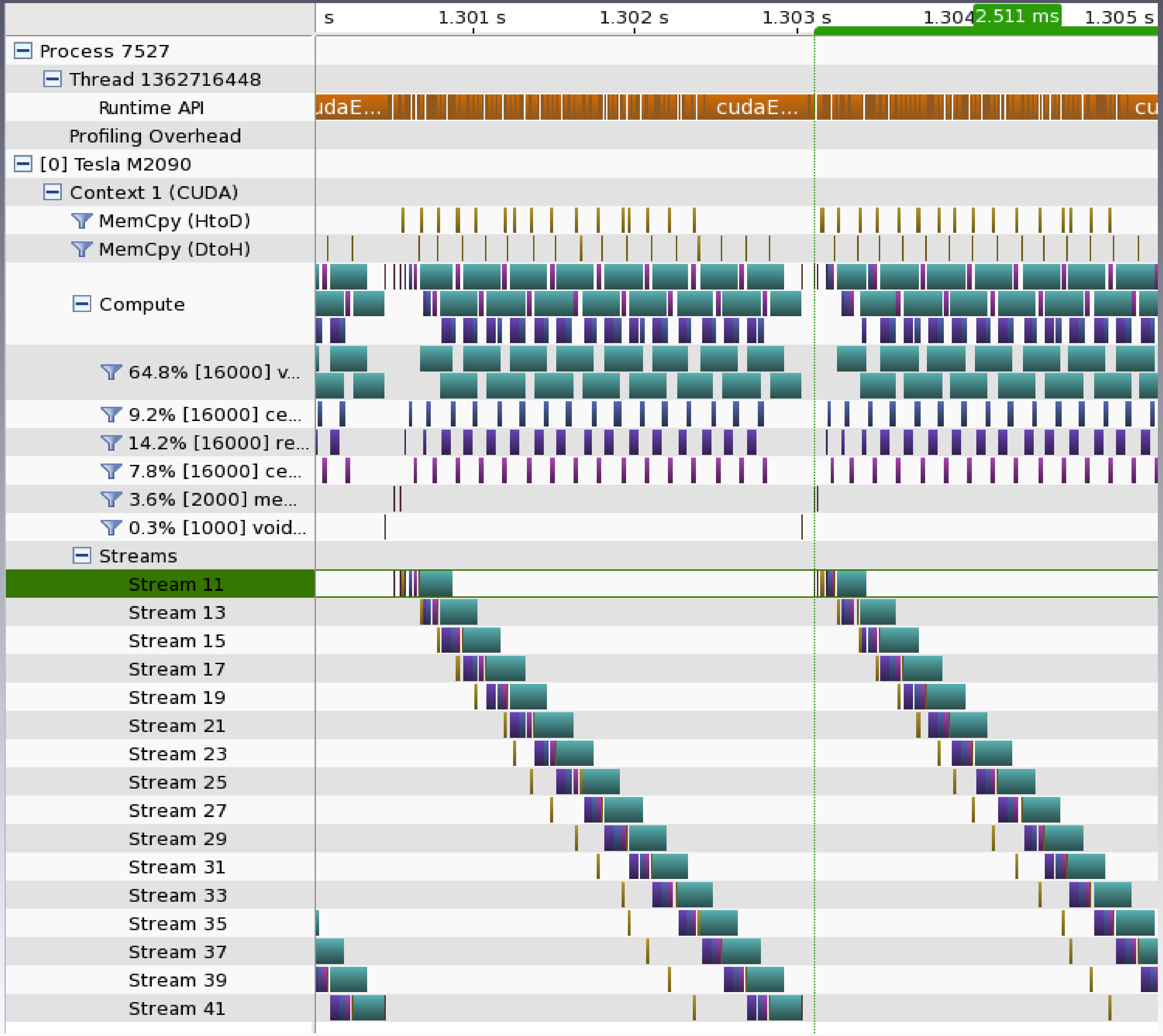
Pure compute performance

- 80x80 (square pupil, no obstruction => upper limit), 6x6 pixels
  - 1 GPU 8 streams
  - Slight performance gain (<2.602ms)
  - Lower kernel concurrency level (load too small)
  - Explicit synchronization may be required
  - Pixel copy takes longer (smaller chunks, sub-optimal) >1ms



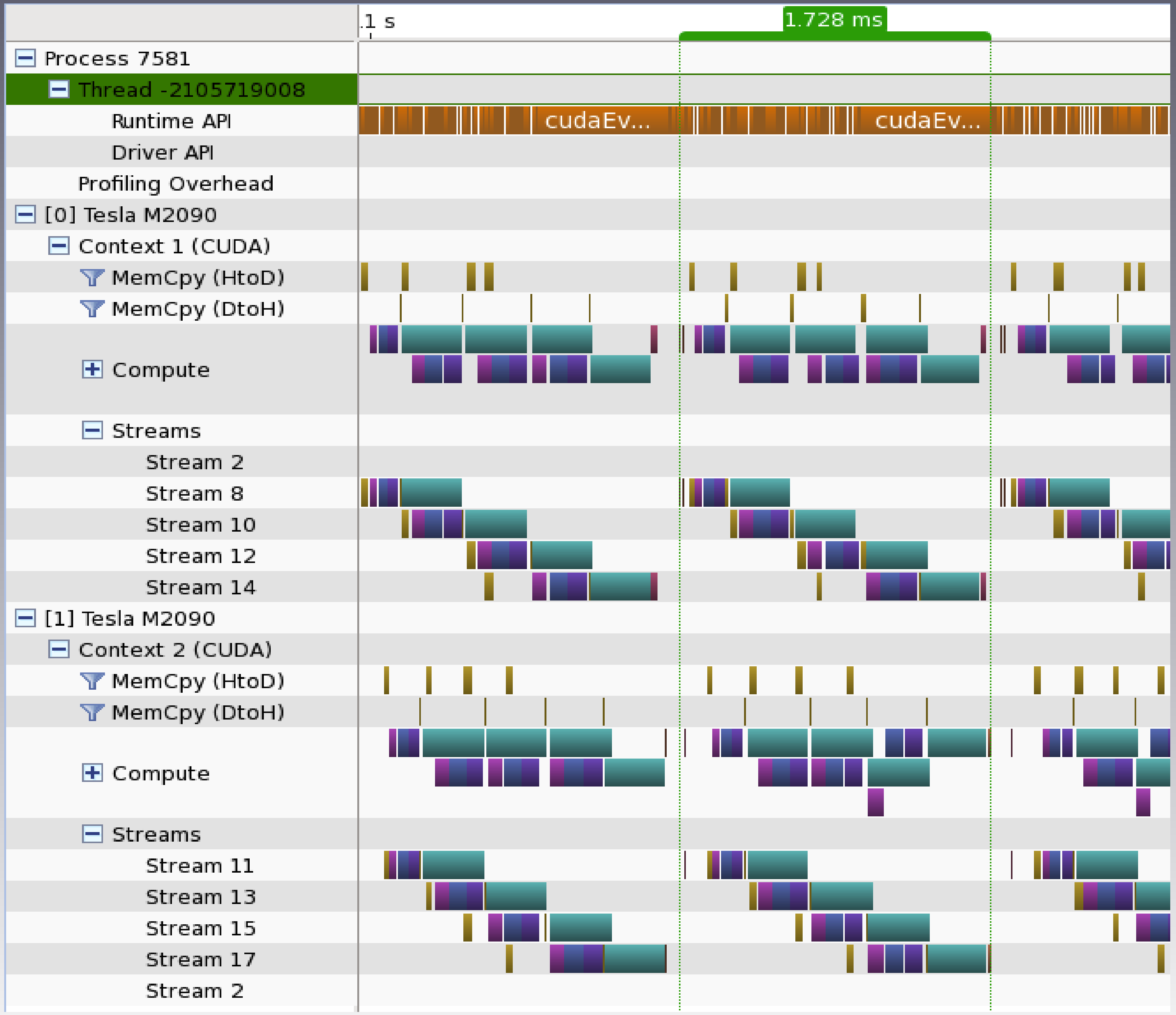
Pure compute performance

- 80x80 (square pupil, no obstruction => upper limit), 6x6 pixels
  - 1 GPU 16 streams
  - Same performance
  - Pixel copy is dispatched over the whole process



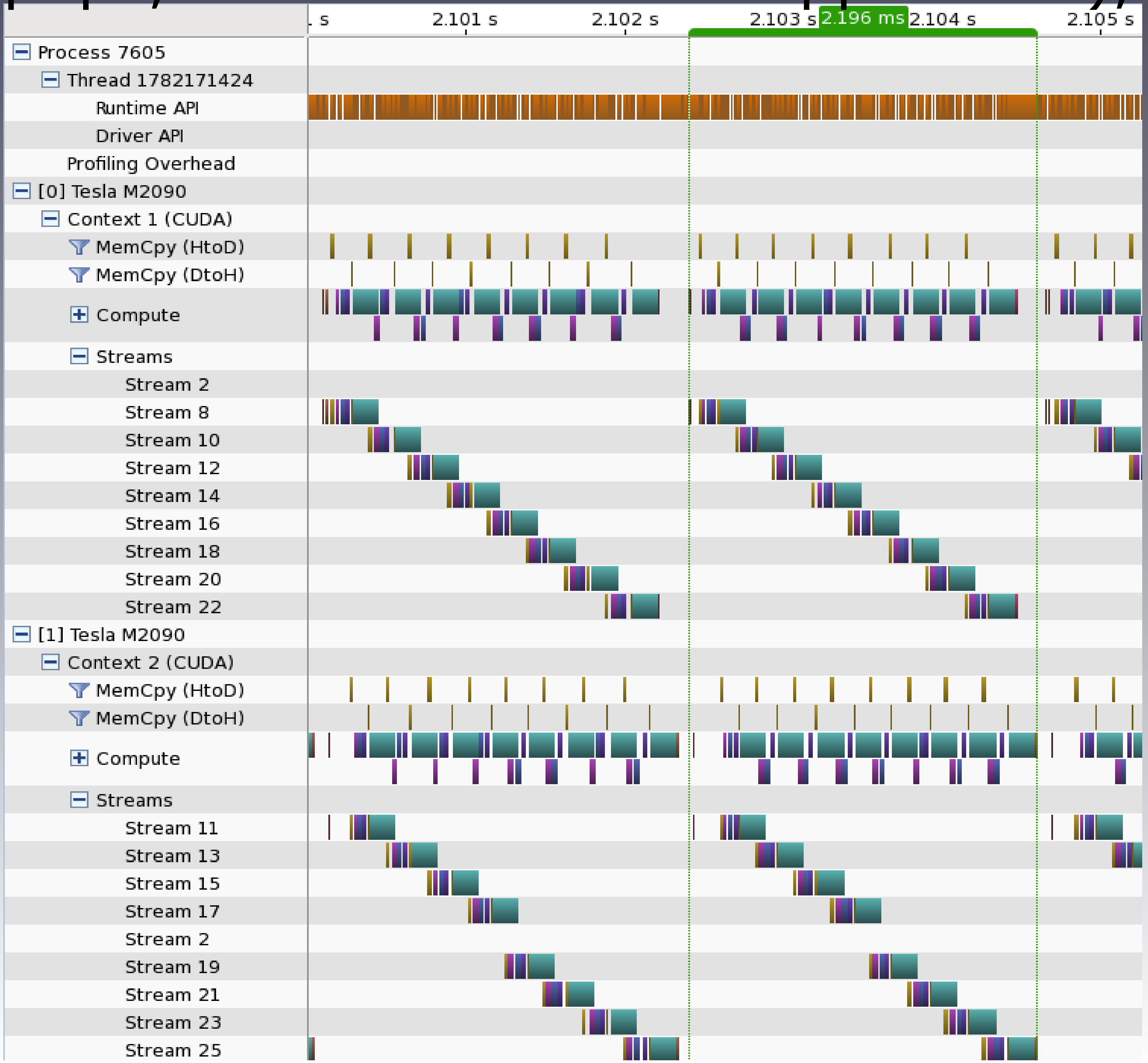
Pure compute performance

- 80x80 (square pupil, no obstruction => upper limit), 6x6 pixels
  - 2 GPUs 4 streams
  - Increased throughput
  - Pixel copy takes ~1ms
  - Best trade-off
  - >500 Hz with profiling overheads



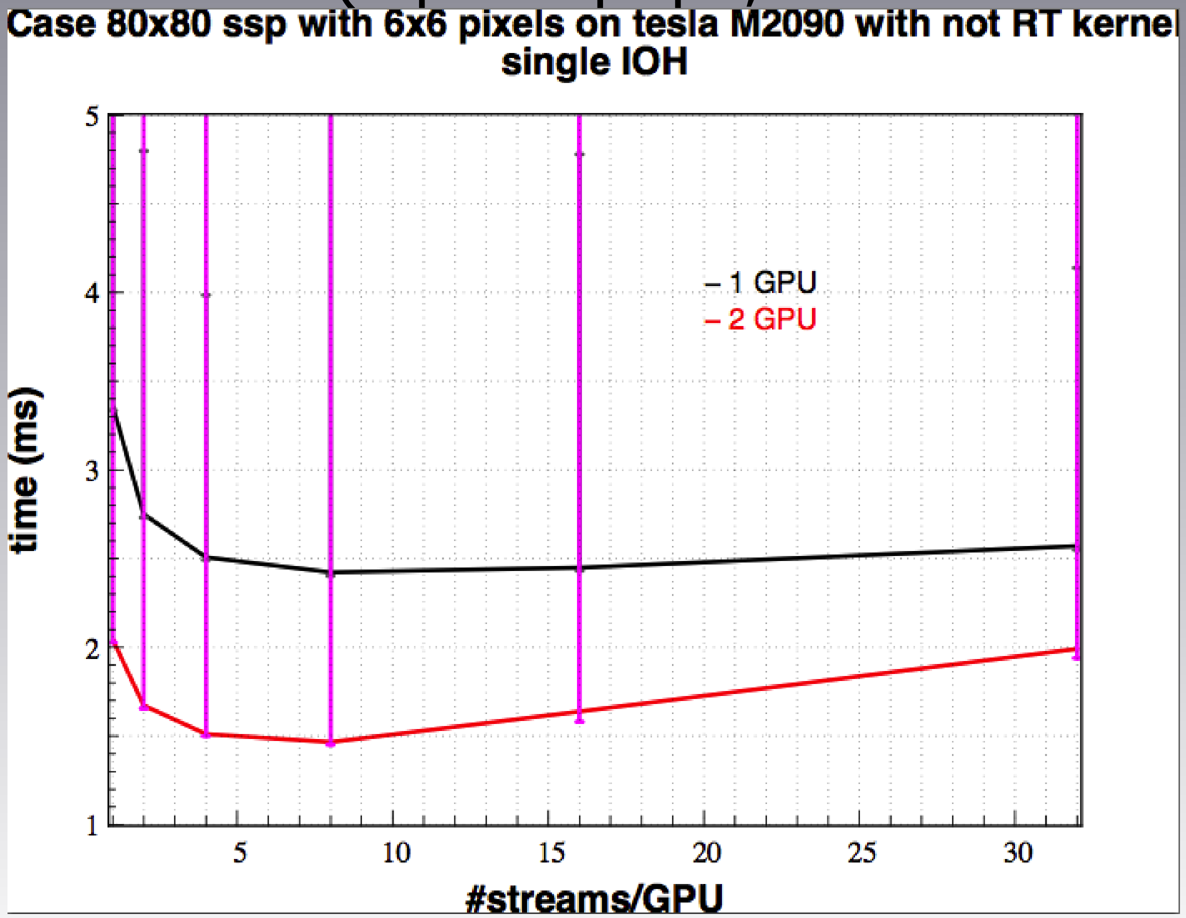
Pure compute performance

- 80x80 (square pupil, no obstruction => upper limit), 6x6 pixels
  - 2 GPUs 8 streams
  - Lower performance
  - bad concurrency
  - Chunks too small
  - Sub-optimal (low occupancy)

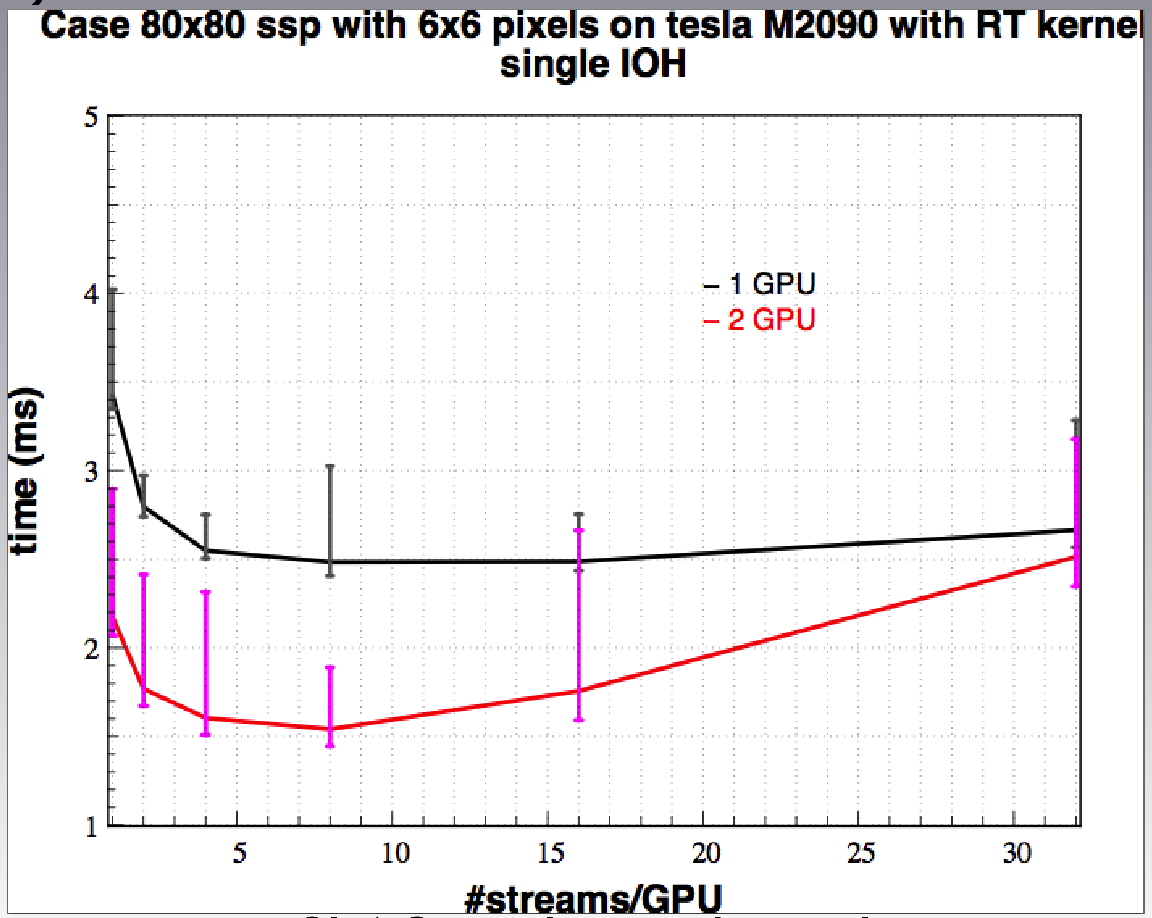


Jitter

- Need for RT kernels
  - SL6.3 without / with RT-kernel, patched NVIDIA drivers, CPU shielding and affinity on HP SL390s motherboards (no RT scheduling)
  - Dramatic reduction of jitter, slight throughput gain
  - Throughput / jitter ensure real-time operations at > 500Hz for SCAO with 2 GPUs (square pupil, no obstruction)



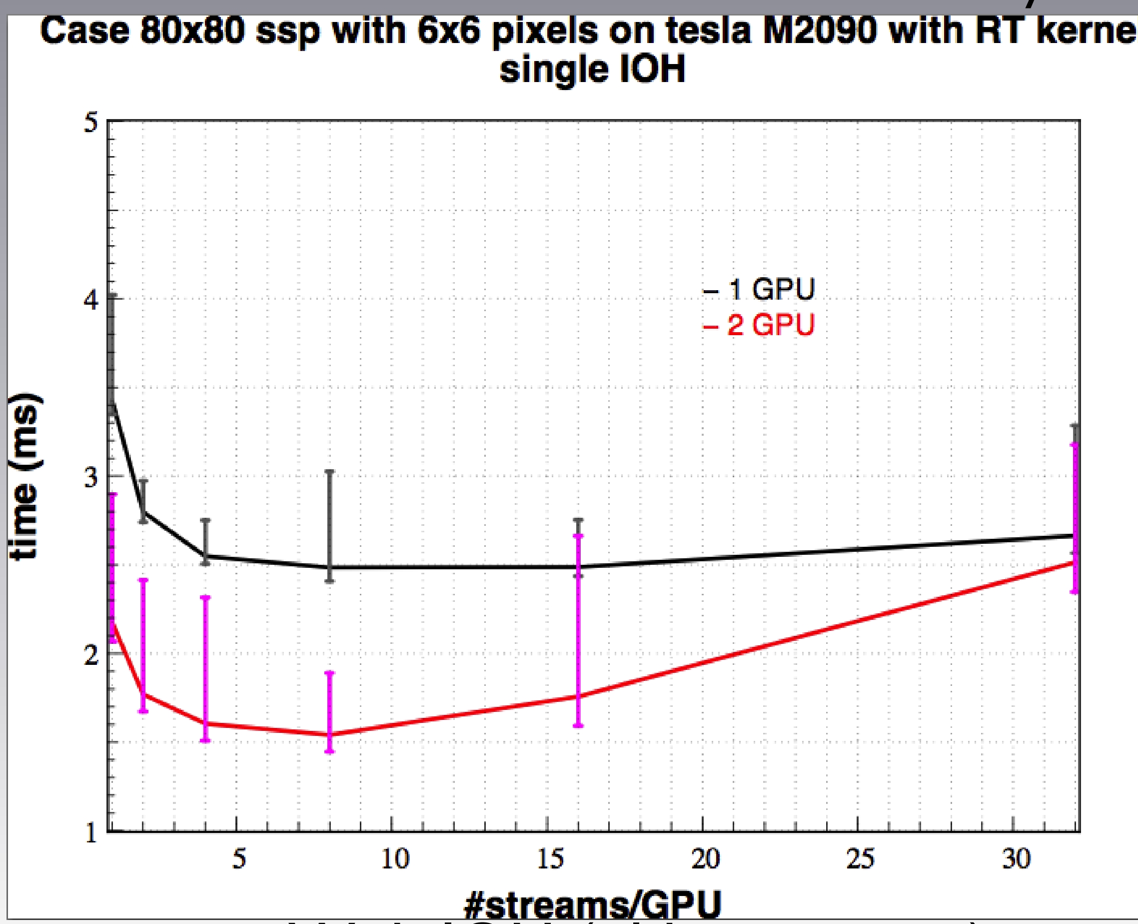
SL6.3, Non real-time kernel



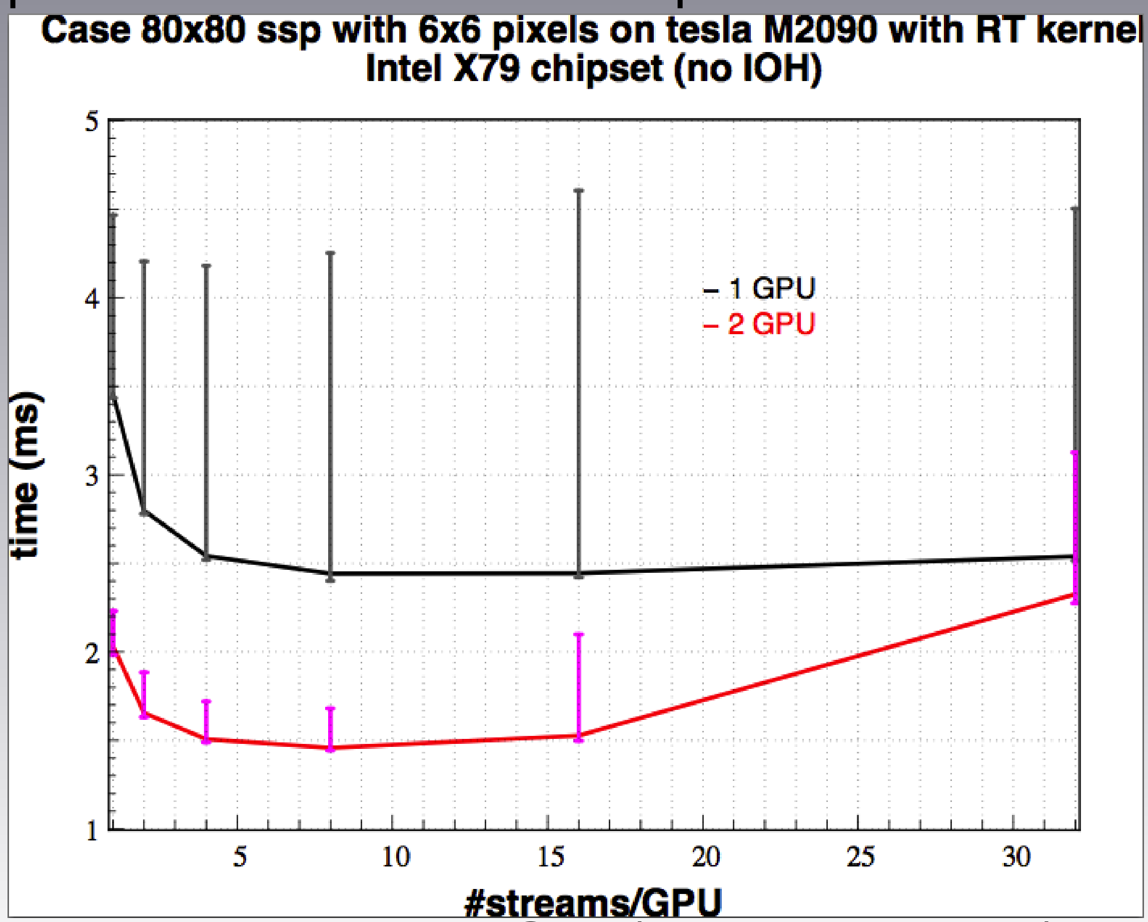
SL6.3, real-time kernel

Jitter

- Impact of new Intel X79 chipset (PCIe lanes on the socket, no IOH)
  - Not bandwidth limited so limited gain in performance
  - Large impact on the single GPU case (why ?)
  - Lower jitter on dual GPU case
  - Are CUDA 5.0 & NVIDIA driver fully compatible with this new chipset ?



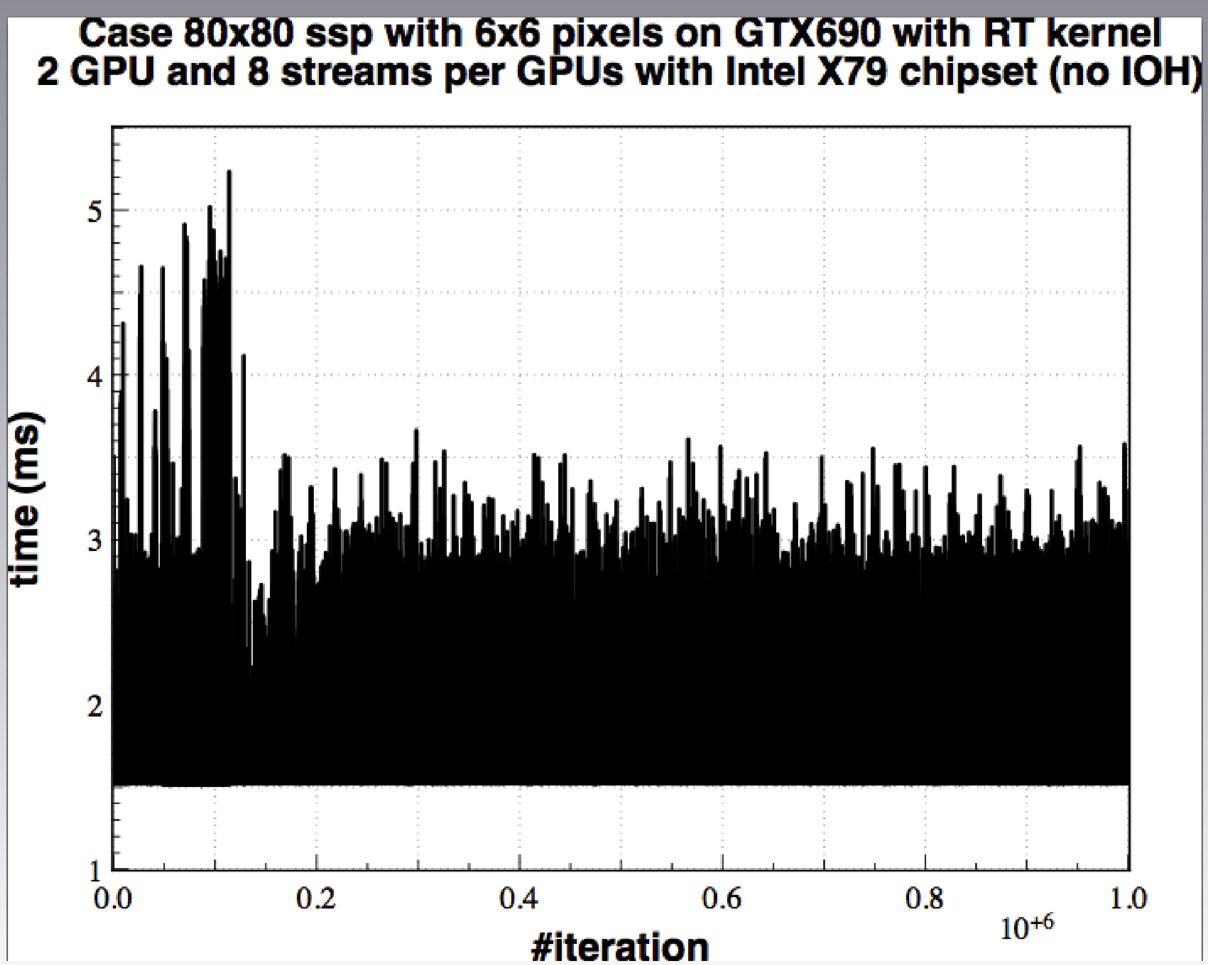
With IOH (old generation)

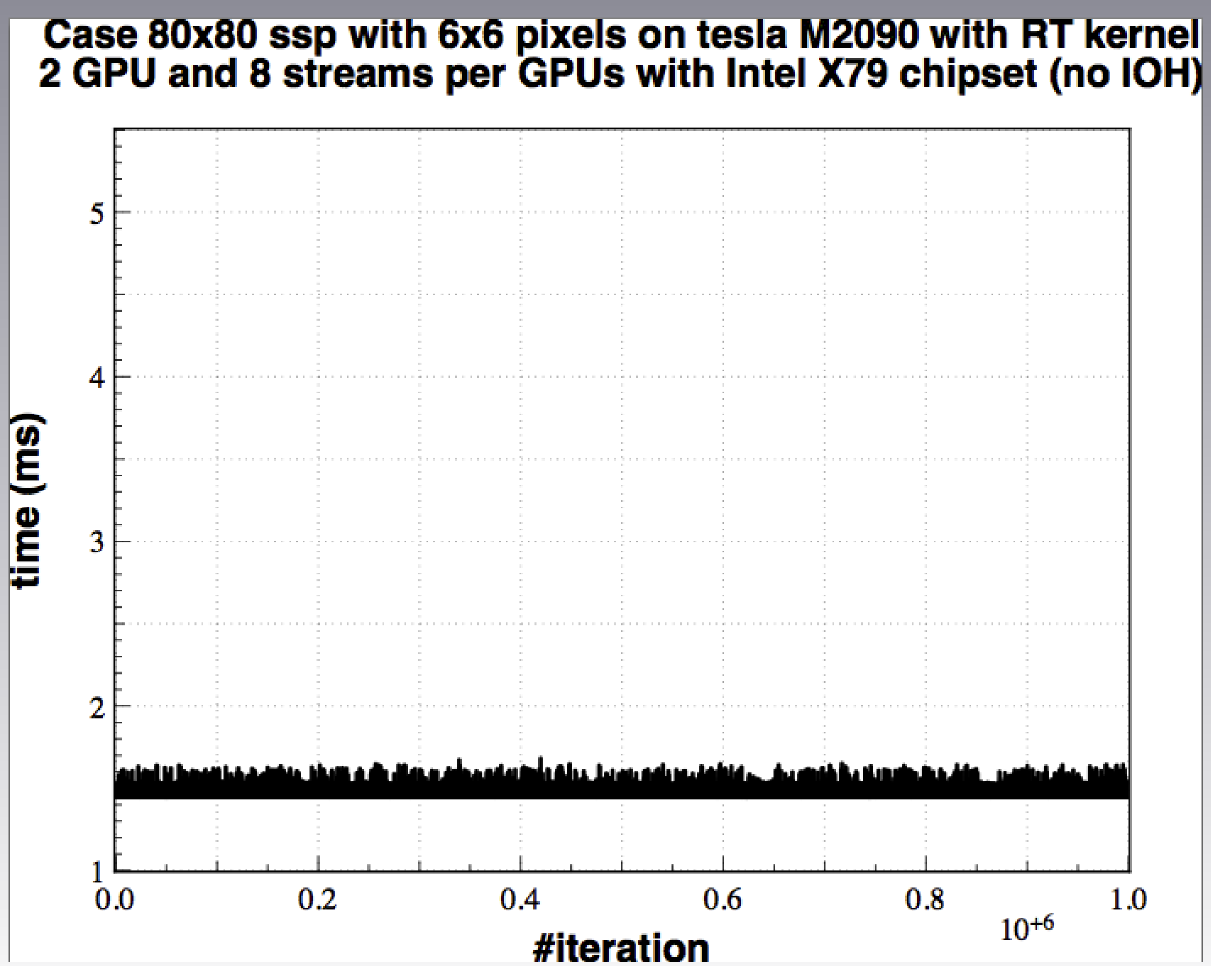


Without IOH (new generation)

Jitter

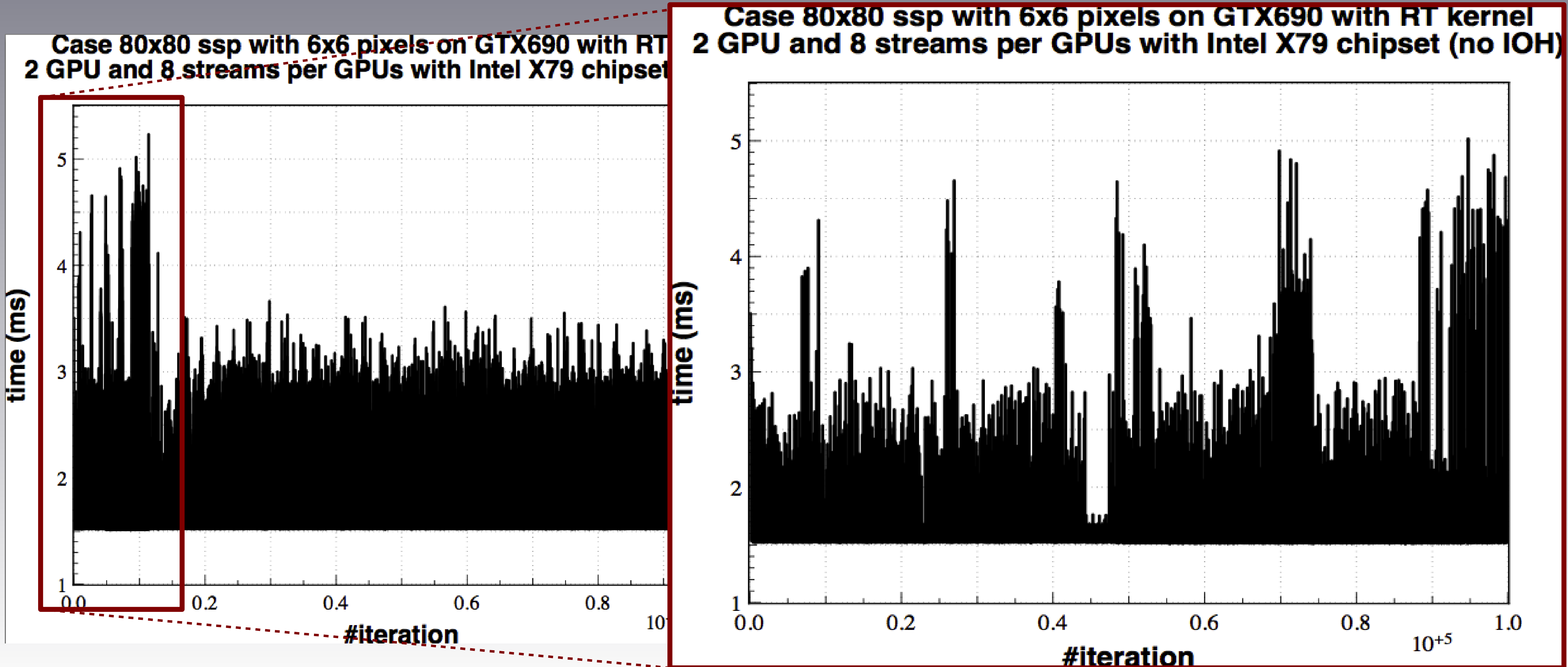
- Using Geforce cards (gamer oriented), to reduce cost
  - 1Msample : about 1/2h operations at 600Hz
  - SL6.3 with RT-kernel & options
  - 2 GPUs (Geforce GTX 690, PCIe switch on board)





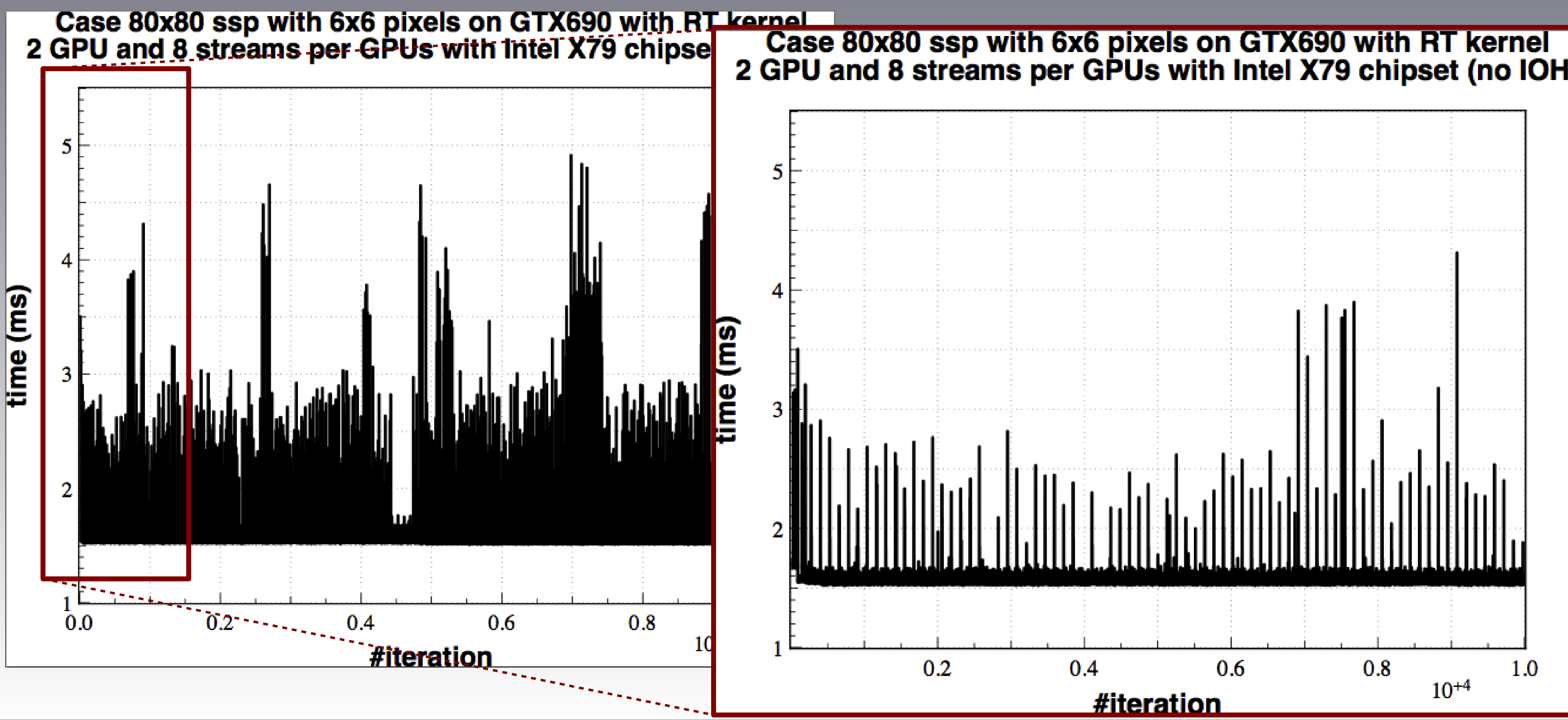
Jitter

- Using Geforce cards (gamer oriented), to reduce cost
  - 1Msample : about 1/2h operations at 600Hz
  - SL6.3 with RT-kernel & options
  - 2 GPUs (Geforce GTX 690, PCIe switch on board)



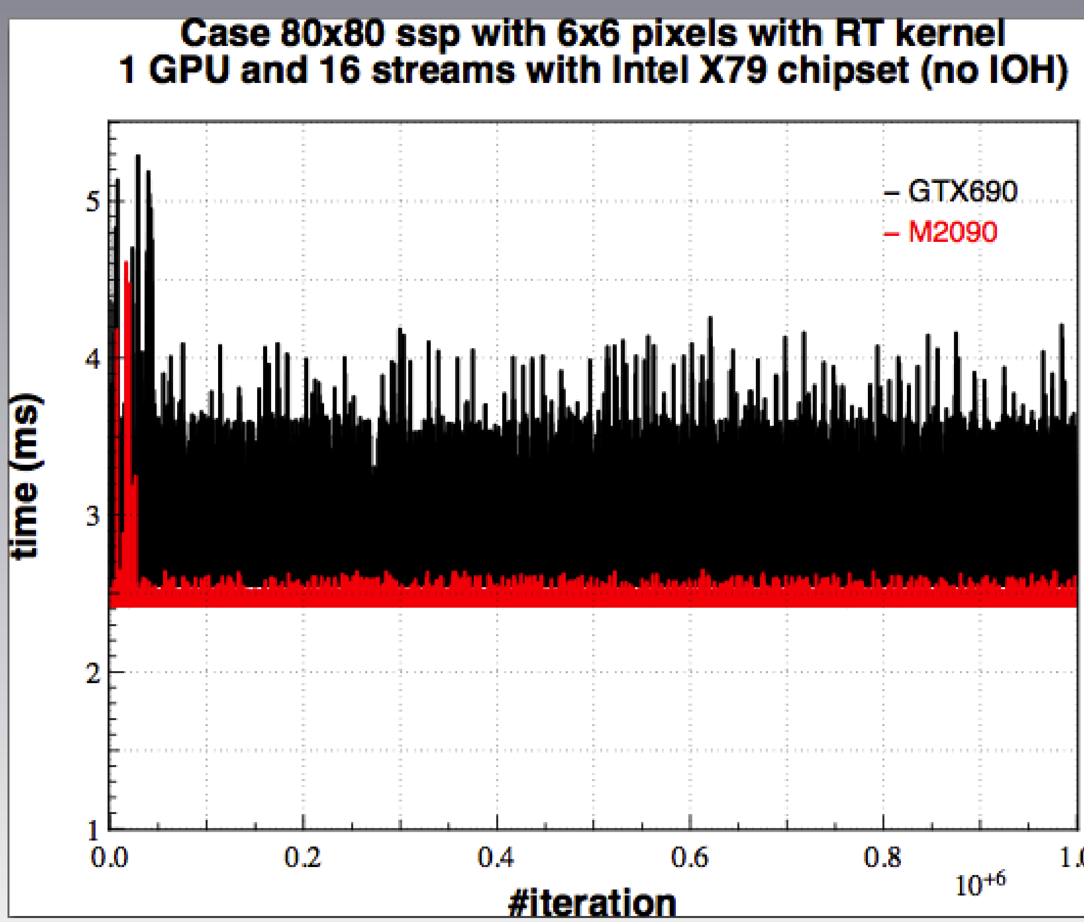
Jitter

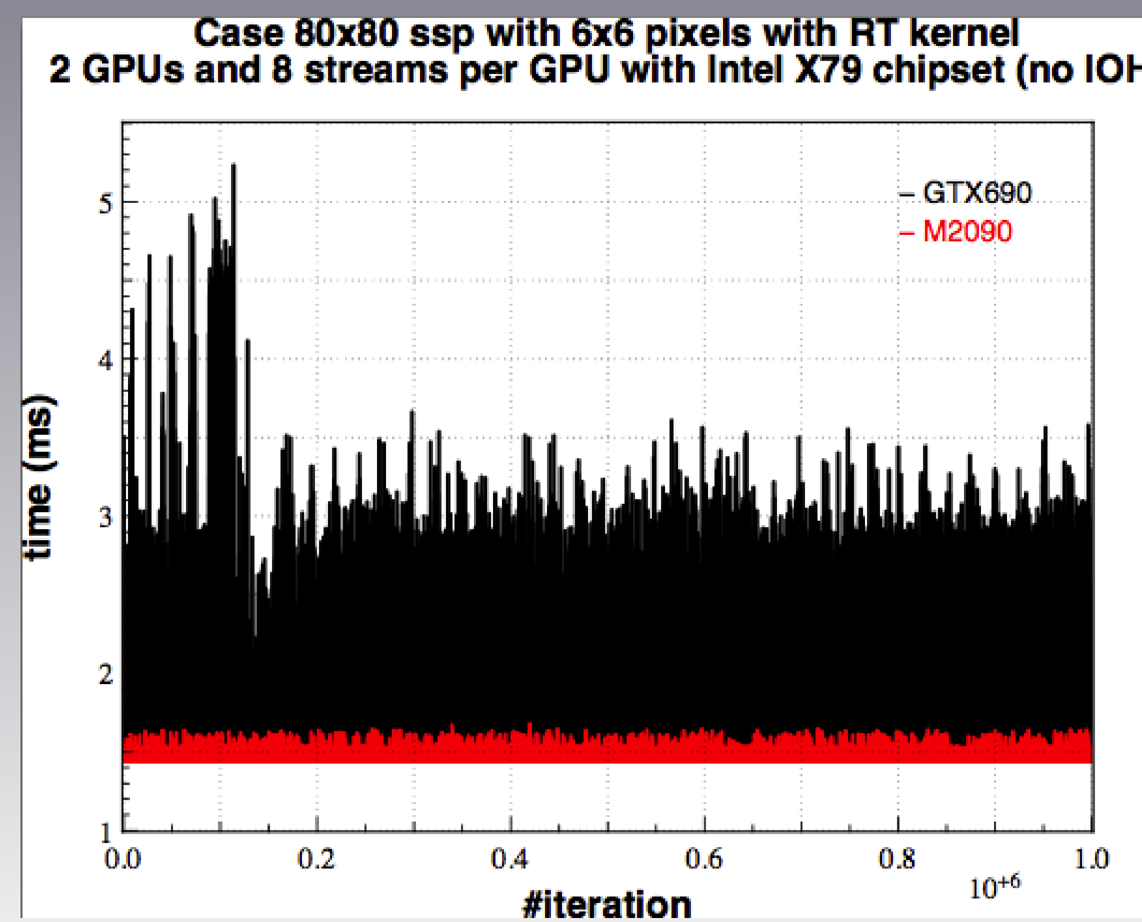
- Using Geforce cards (gamer oriented), to reduce cost
  - Large jitter at all scale (several ms)
  - Pure throughput is ok but jitter incompatible with RT operations
  - Geforce ok for simulations but not compatible with RT



Jitter

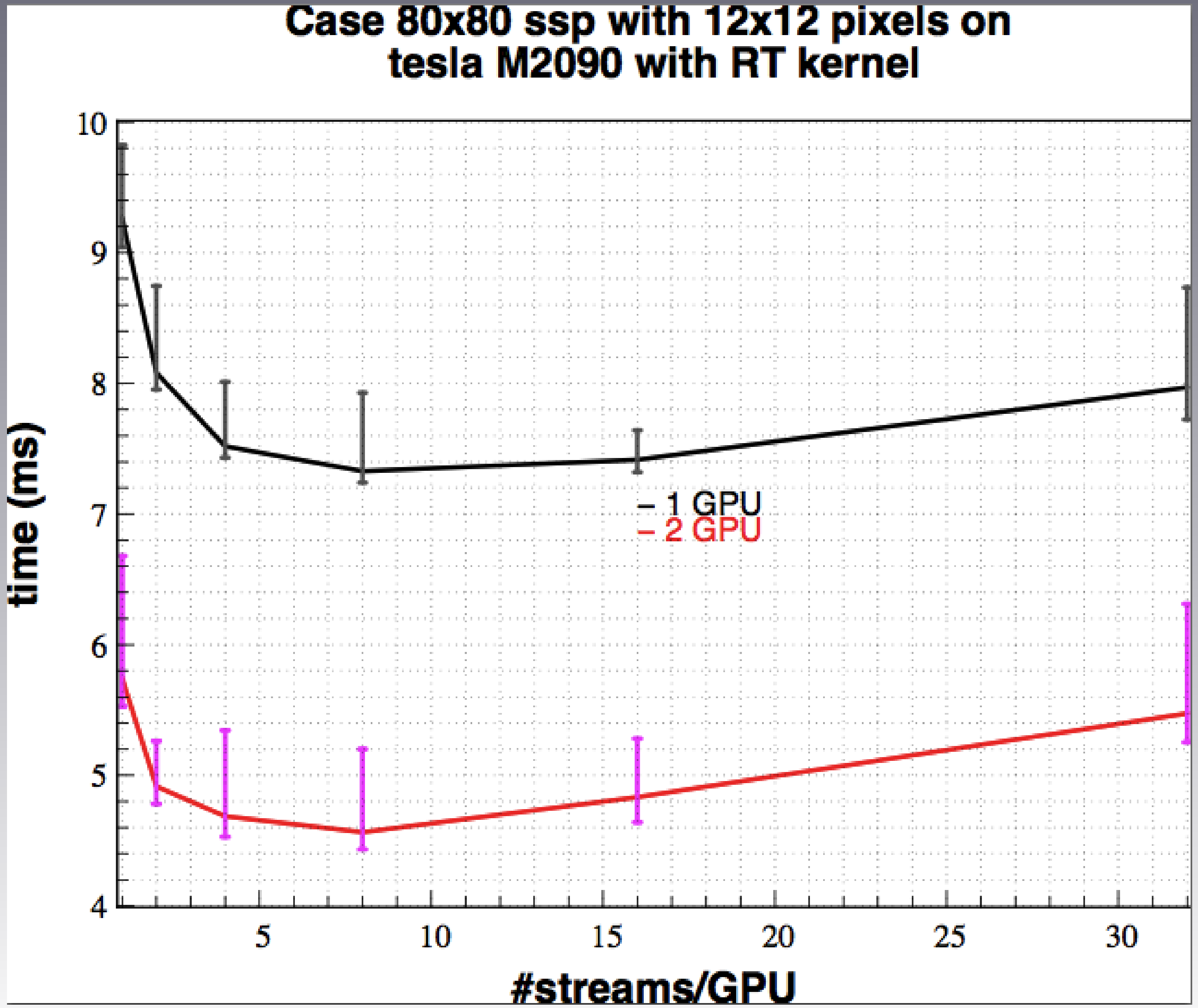
- Using Geforce cards (gamer oriented), to reduce cost
  - Using only one GPU gives lower throughput
  - 2 GPUs : increased jitter on GeForce compensate gain in throughput (not compatible with RT, no point to use 2 GPUs)
  - Part of jitter (single GPU) may be due to hardware (X79 chipset)





The MAORY case

- MCAO with 6 LGS WFS 80x80 (baseline = 12x12 pixels) and 3 DMs
- 6 dual-GPU nodes: 1/WFS (cmd vector integrated out of the nodes)
- Close to 200Hz



Conclusions

- Throughput is there
  - RT operations for SCAO 80x80 @>500Hz (2 GPUs)
  - CUDA framework allows optimizations (N streams) and memcopy latency hiding
- Means to reduce jitter
  - Use of RT kernel with nvidia RT patch
  - Use of professional Tesla boards (M2090, K10, K20, K20X, …)
  - Long-run performance (including jitter) compatible with RT operations on Tesla boards
- GeForce have similar throughput, but...
  - Huge jitter (>2ms) → performance not stable
  - OK for simulations but incompatible with RT

Future works

- Use host level (kernel level) for profiling:
  - Absolute performance and jitter measurements
- Kernel module for data exchange
  - The first step is to feed the GPU with a custom kernel module with GPU direct compatibility
  - Next the GPU can feed a second kernel module (also GPU direct compatible)
  - Accurate precision of the real jitter
- Acquisition interface through serial protocol
- Studies on MPI/RT