JÜLICH
FORSCHUNGSZENTRUM

JÜLICH
APPLICATION LAB
NVIDIA

First Experiences with Maxwell GPUs

October 8, 2014 | Andrew V. Adinetz

Agenda

- New in Maxwell
- Shared-memory atomics
  - histograms, filtering
- PANDA Triplet Finder

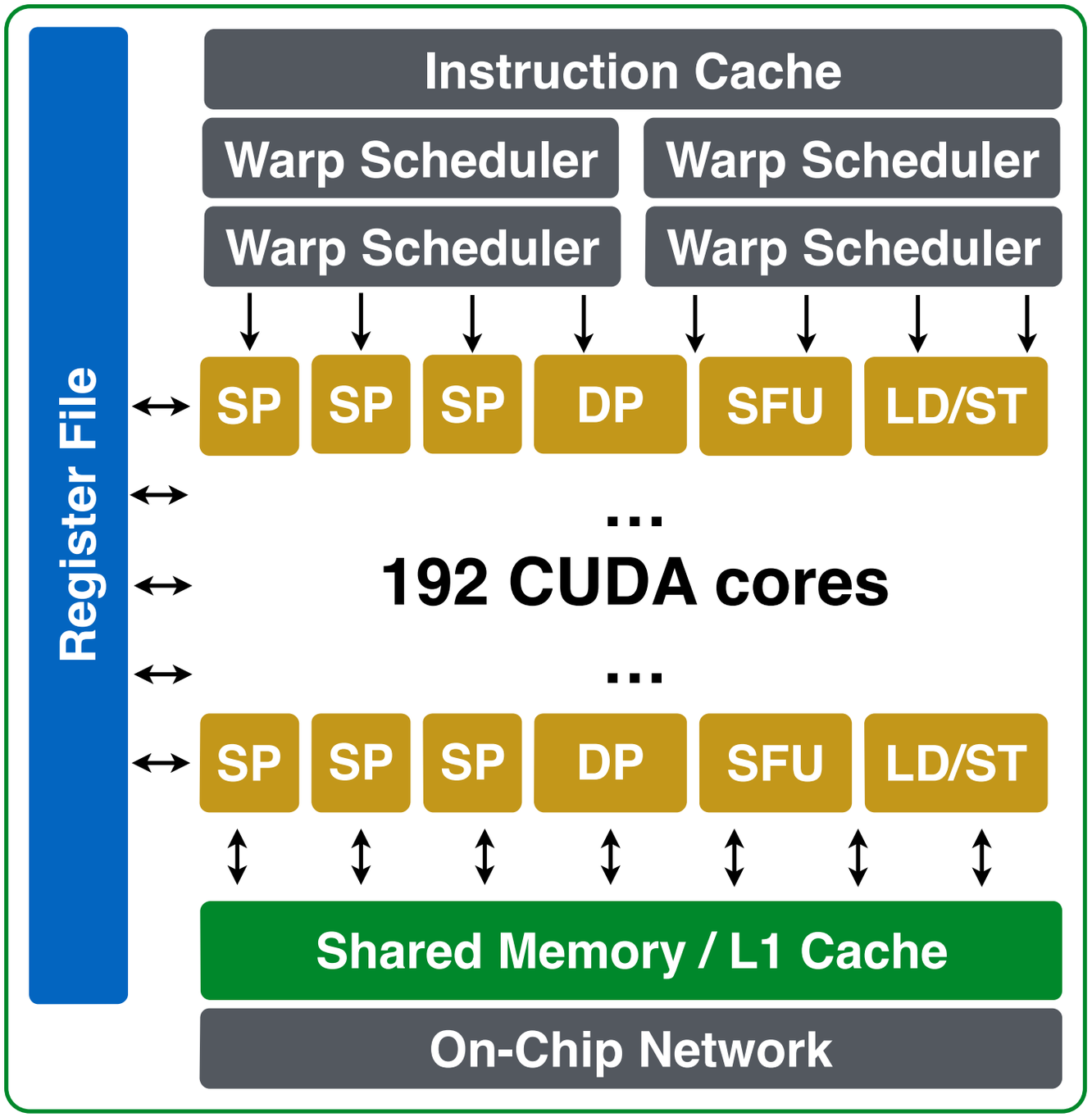
Agenda

- **New in Maxwell**
- Shared-memory atomics
  - histograms, filtering
- PANDA Triplet Finder

New in Maxwell

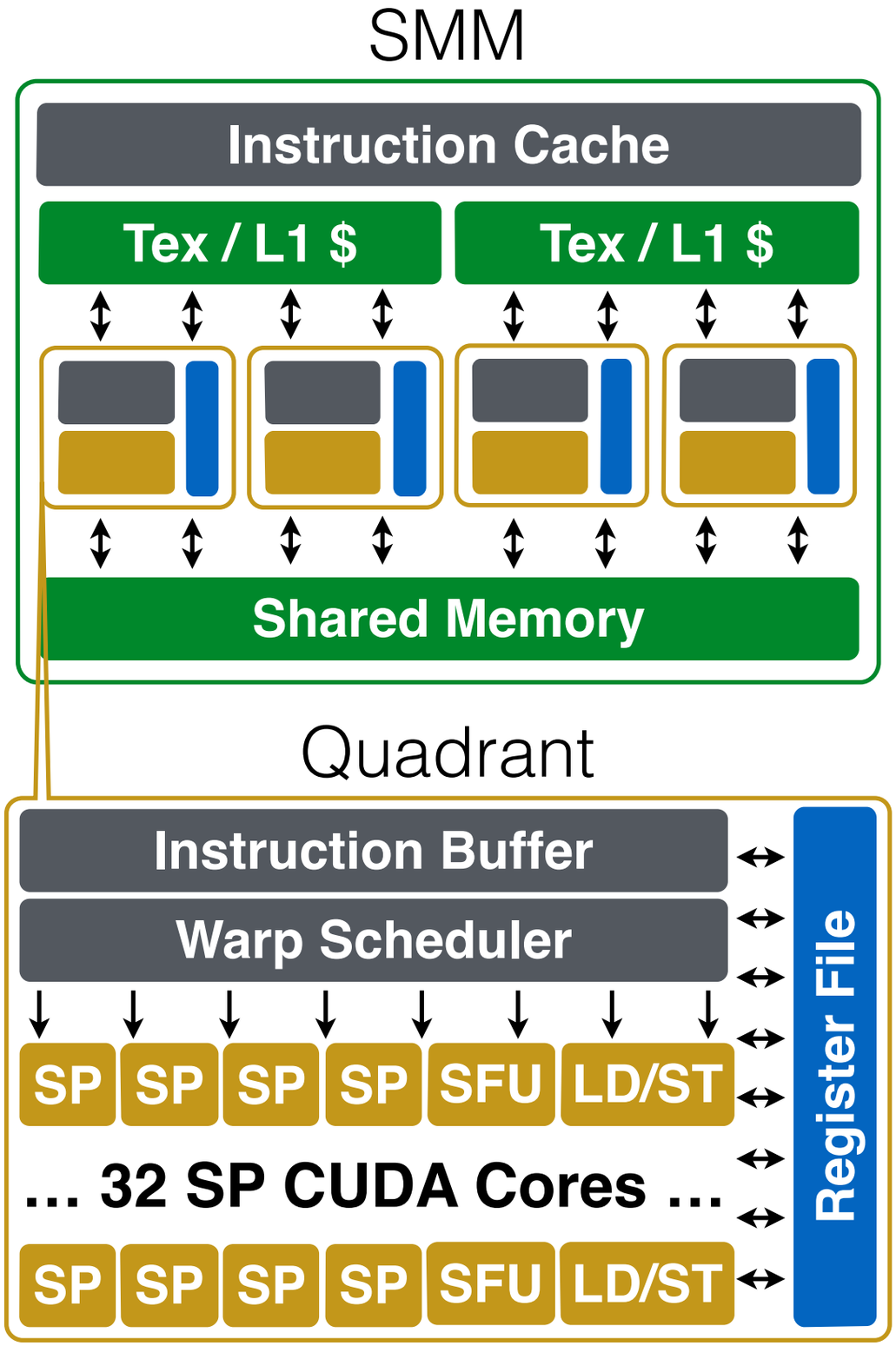
- Functionally same as Kepler (K20)
- Dynamic parallelism is mainstream
- Architectural improvements
  - new SM design (SMM)
  - native shared-memory 32-bit atomics
  - other improvements
- CC 5.x (5.0 for GM 207, 5.2 for GM 204)
  - `nvcc -arch=sm_50 …`

Kepler SM (SMX)



- Scheduler not tied to cores
- Double issue for max utilization

Maxwell SM (SMM)



- Simplified design
  - power-of-two, quadrant-based
  - scheduler tied to cores
- Better utilization
  - single issue sufficient
  - lower instruction latency
- Efficiency
  - <10% difference from SMX
  - ~50% SMX chip area

Shared-Memory Atomics

```
__shared__ int l_n;
// . . .
atomicAdd(&l_n, 1);
```

pre-Maxwell (e.g. Kepler)

`nvcc -arch=sm_35 …`

```
/*00c8*/        LDSLK P0, R2, [RZ];
/*00d0*/   @P0  IADD R2, R2, 0x1;
/*00d8*/   @P0  STSCUL P1, [RZ], R2;
/*00e0*/  @!P1  BRA 0xc0;
```

expensive lock-modify-unlock in a loop

Maxwell

`nvcc -arch=sm_50 …`

```
/*00f0*/   @!P0 ATOMS.ADD RZ, [RZ], R4;
```

single instruction

Other Maxwell Features

- More thread blocks/SM
  - 32 (vs. 16), full occupancy with 64-thread TBs
- Shared memory
  - 64 KiB/SM in GM 207, 96 KiB/SM in GM 204
  - 4-byte mode only
- L1 cache
  - read-only (same as texture cache)
  - not sharable with shared memory
- Larger L2 cache
  - 2 MiB vs. 1.5 MiB on Kepler

Parameters of Test GPUs

	Kepler K20X	**Maxwell GTX 750 Ti**
#SMs	14	5
#CUDA cores	2688	640
Frequency, MHz	732	1110
Peak SP GFlop/s	3935	1305.6
Memory BW, GiB/s	169.5	68.9

All results are preliminary

Agenda

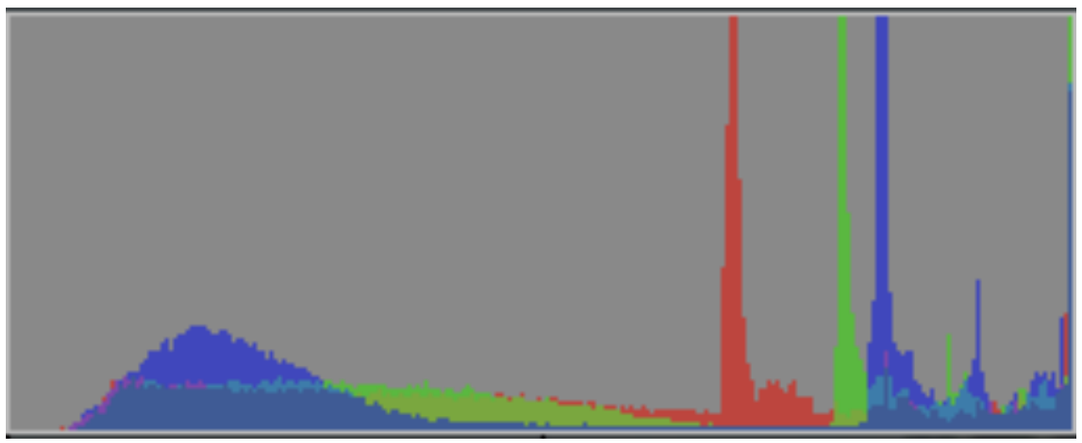
- New in Maxwell
- **Shared-memory atomics**
  - histograms, filtering
- PANDA Triplet Finder

Histogram

Common operation in signal and image processing

on CPU:

```
for(i = 0; i < n; i++)
  histo[data[i]]++;
```



on GPU:

```
__global__ void histo_k
(int *histo, uchar *data, int n) {
  int i = threadIdx.x +
    blockIdx.x * blockDim.x;
  if(i >= n)
    return;
  atomicAdd(&histo[data[i]], 1);
}
```

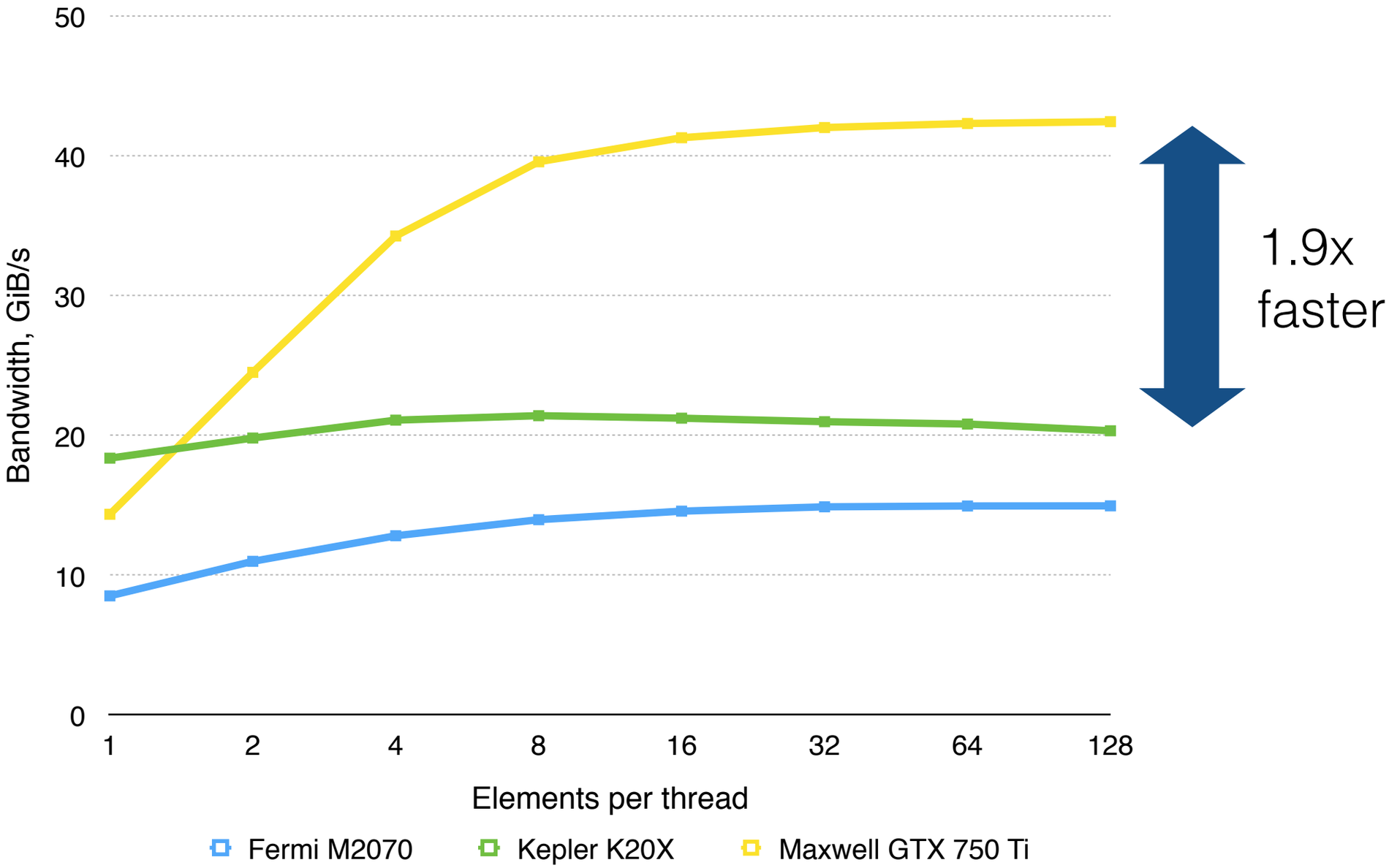
Global atomics are the bottleneck

Shared-Memory Histogram

Avoid global atomics bottleneck

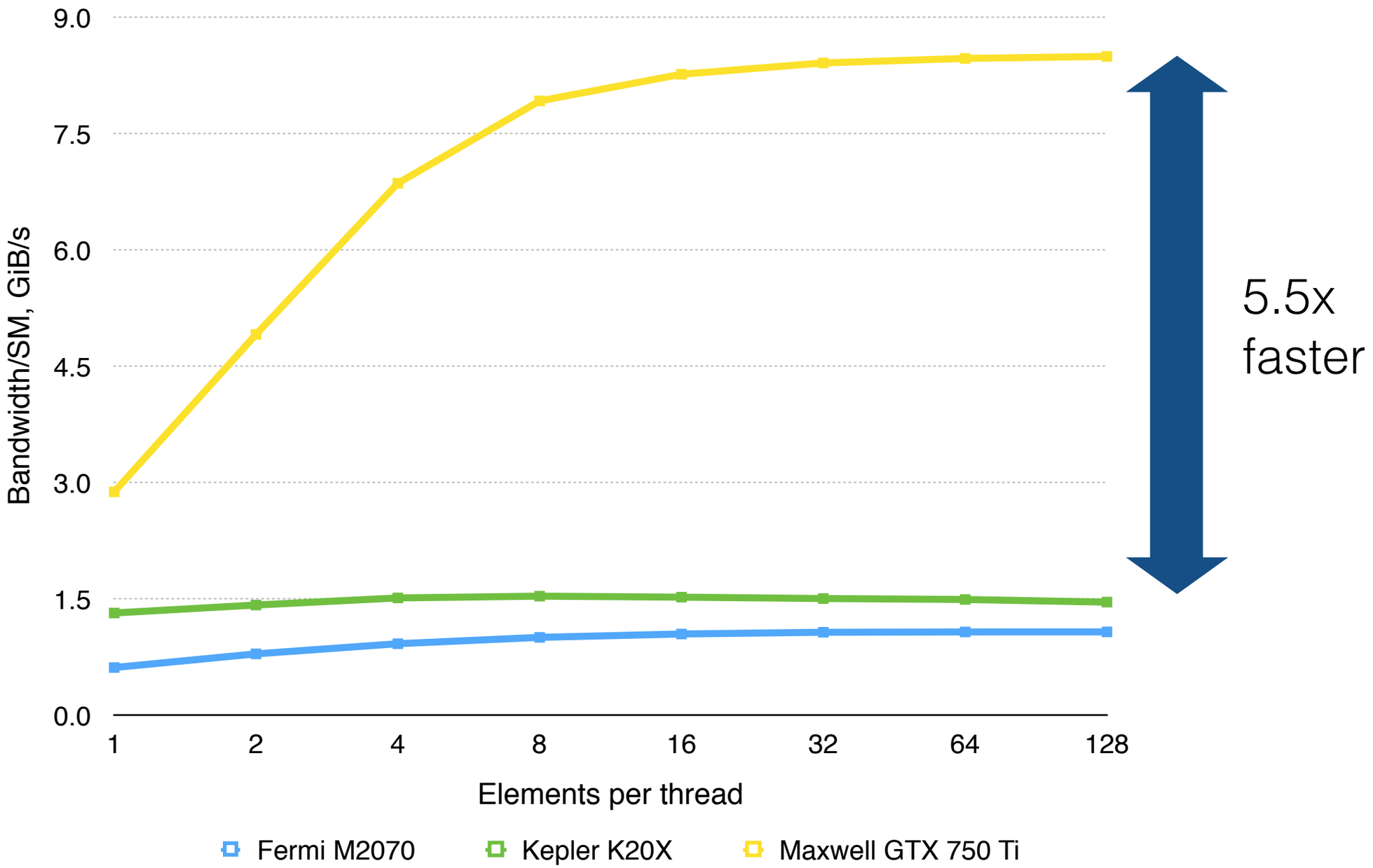
```
#define NCLASSES 256
#define BS 256
#define PER_THREAD 32

__global__ void histo_k(int *histo, const unsigned* data, int n) {
  // init per-block histogram
  __shared__ int lhisto[NCLASSES];
  for(int i = threadIdx.x; i < NCLASSES; i += BS)
    lhisto[i] = 0;
  __syncthreads();
  // compute per-block histogram
  int istart = blockIdx.x * (BS * PER_THREAD) + threadIdx.x;
  int iend = min(istart + BS * PER_THREAD, n);
  for(int i = istart; i < iend; i += BS) {
    union { unsigned char c[sizeof(unsigned)]; unsigned i; } el;
    el.i = data[i];
    for(int j = 0; j < sizeof(unsigned); j++)
      atomicAdd(&lhisto[el.c[j]], 1); // shared-memory atomic
  }
  __syncthreads();
  // accumulate histogram to global storage
  for(int i = threadIdx.x; i < NCLASSES; i += BS)
    atomicAdd(&histo[i], lhisto[i]);  // global atomics
}  // histo_kernel
```

Histogram256 Performance



- Maxwell clearly wins (against sever-grade GPUs!)
- 60% memory bandwidth achieved

Performance per SM



Higher performance expected with larger GPUs (more SMs)

Filtering

Copying elements based on predicate (e.g., > 0)

1	-4	2	-3	3	-2	5	0	7

3	1	2	7	5

on CPU:

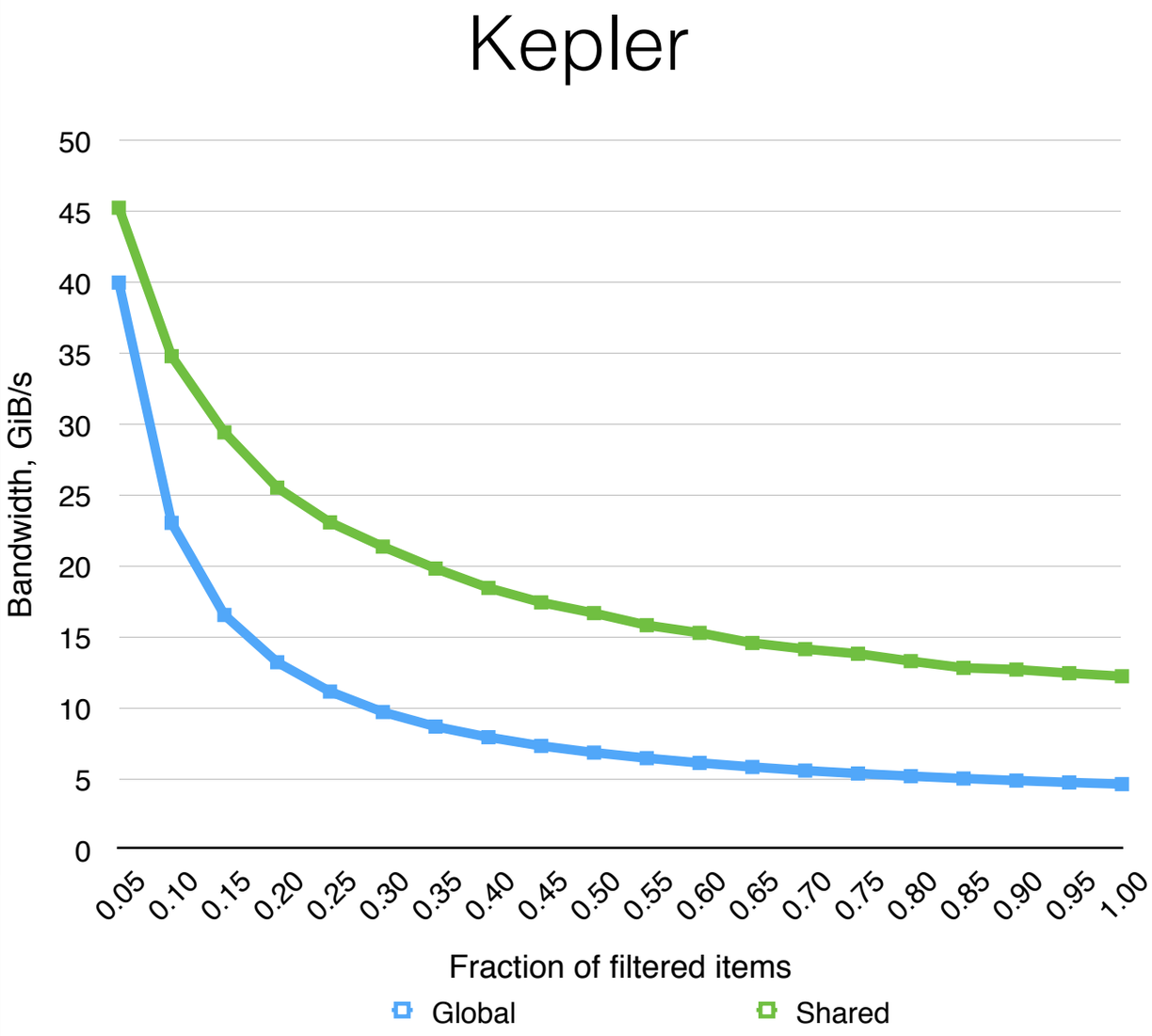
```
nres = 0;
for(i = 0; i < n; i++)
  if(src[i] > 0)
    tgt[nres++] = src[i];
```

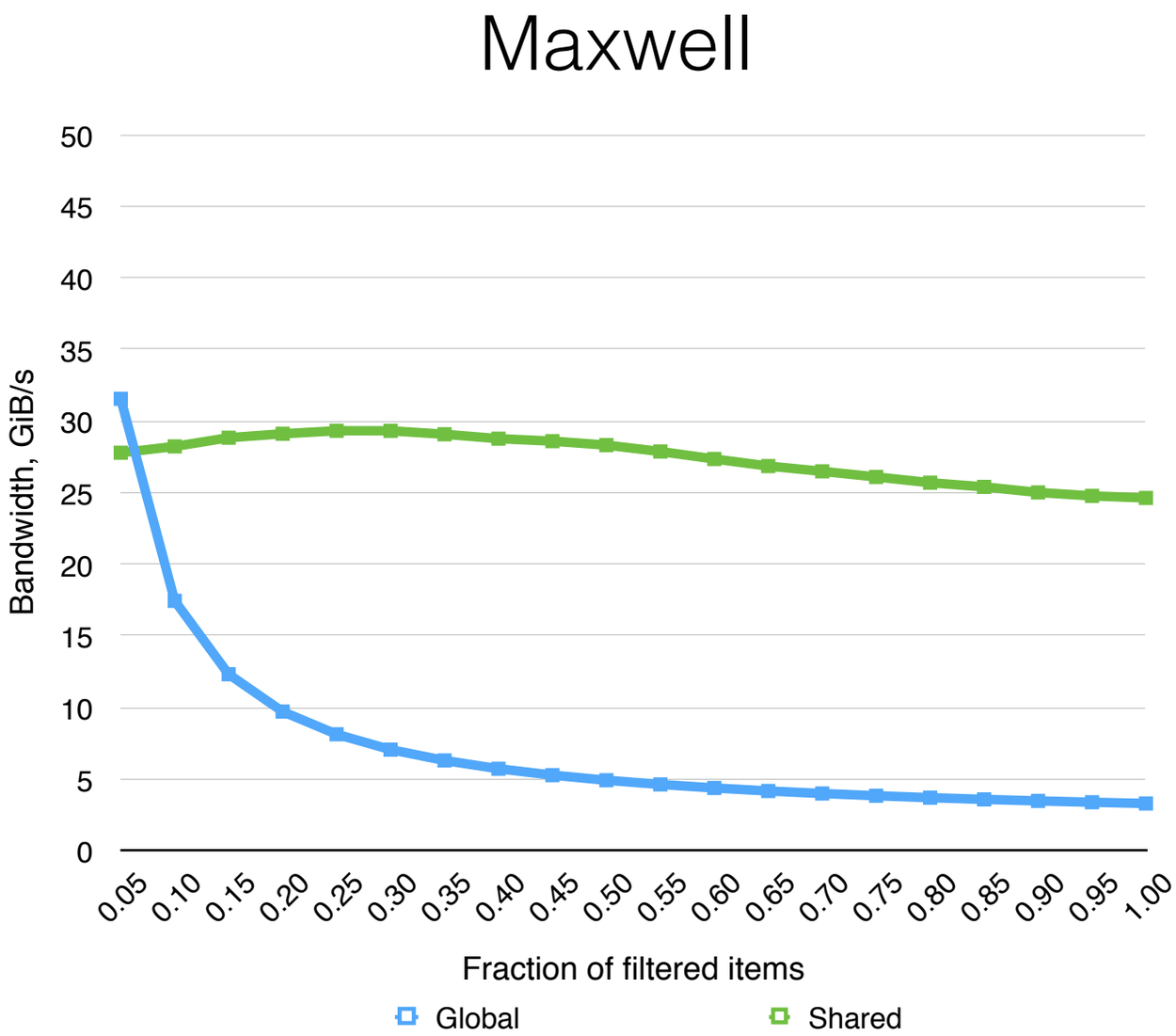
on GPU:

```
__global__ void filter_k(int *tgt, int *nres, int *src, int n) {
  int i = threadIdx.x + blockIdx.x * blockDim.x;
  if(i >= n)
    return;
  if(src[i] > 0)
    tgt[atomicAdd(nres, 1)] = src[i];
}
```

Global atomics are the bottleneck

Shared vs Global Atomics

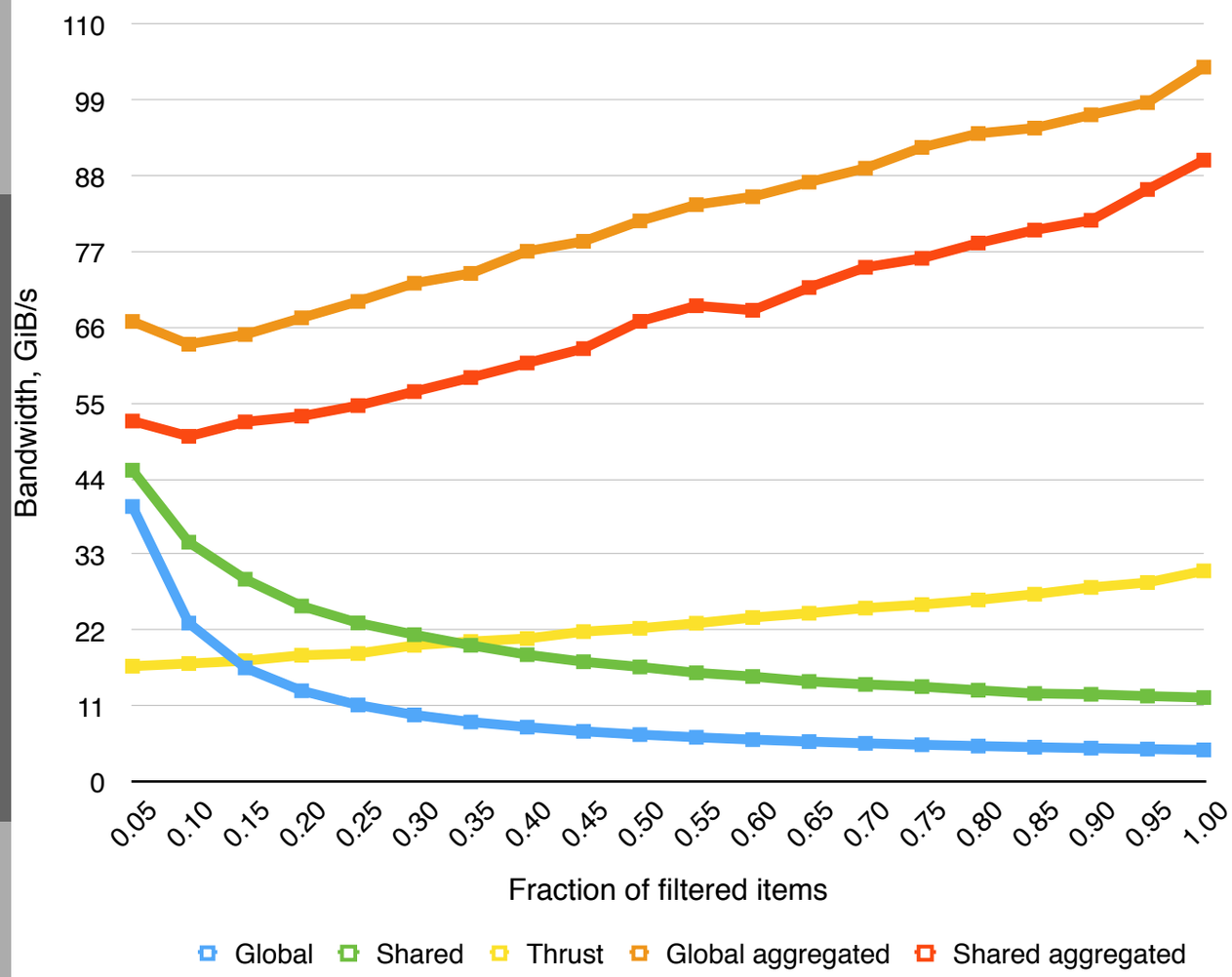




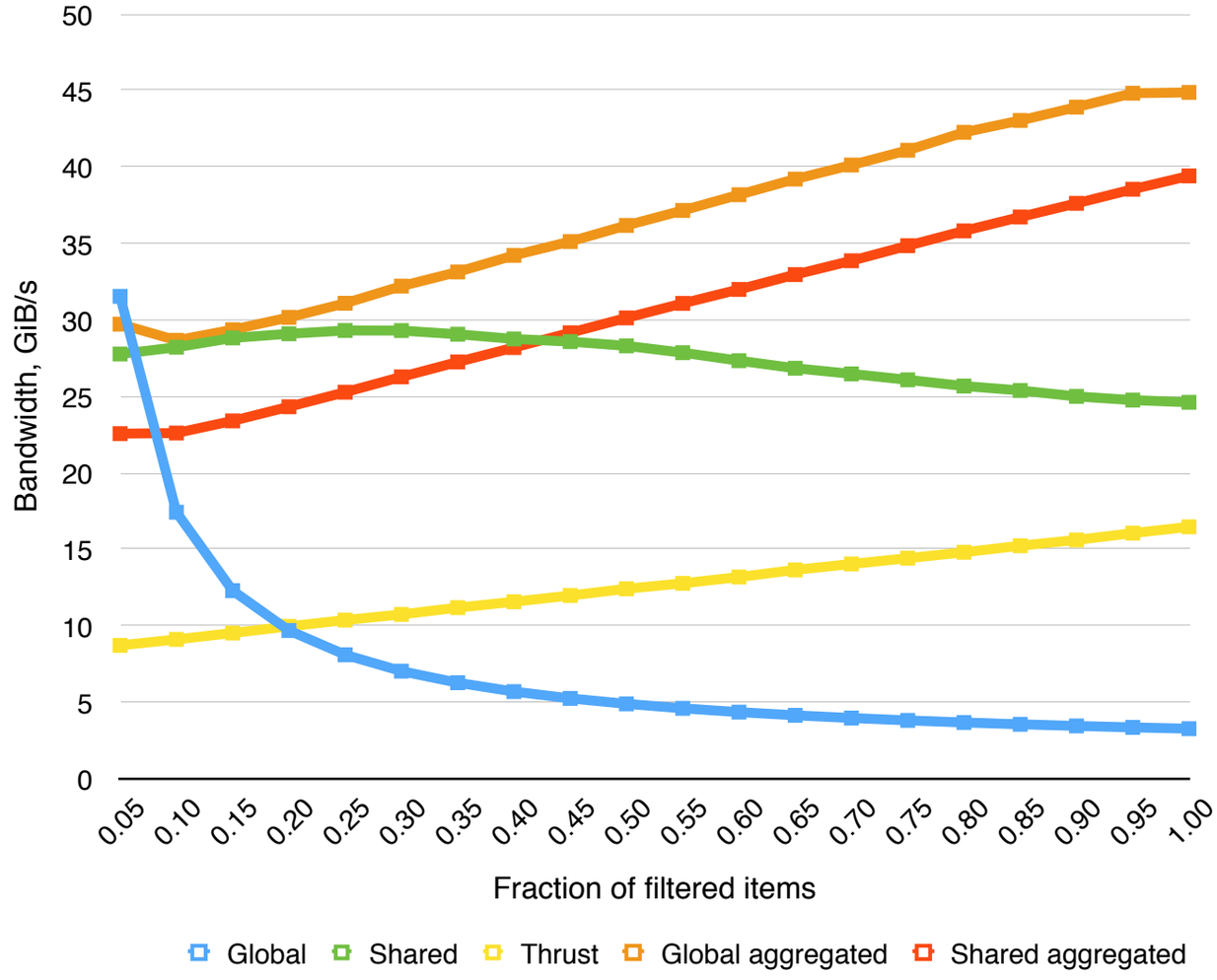
Qualitatively different behavior on Maxwell

Warp-Aggregated Atomics

Kepler



Maxwell



http://devblogs.nvidia.com/parallelforall/cuda-pro-tip-optimized-filtering-warp-aggregated-atomics/

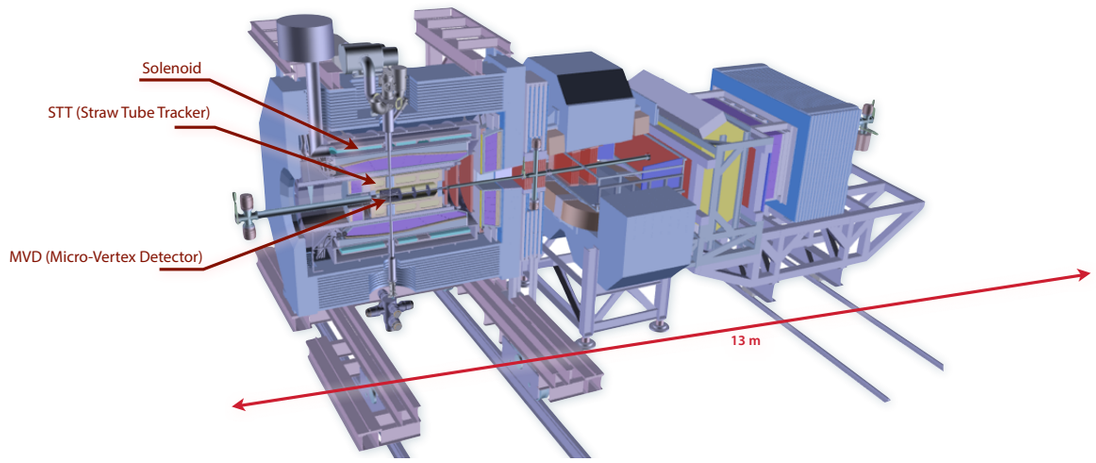
Higher shared-memory performance with more SMs expected

Agenda

- New in Maxwell
- Shared-memory atomics
  - histograms, filtering
- **PANDA Triplet Finder**

PANDA Experiment

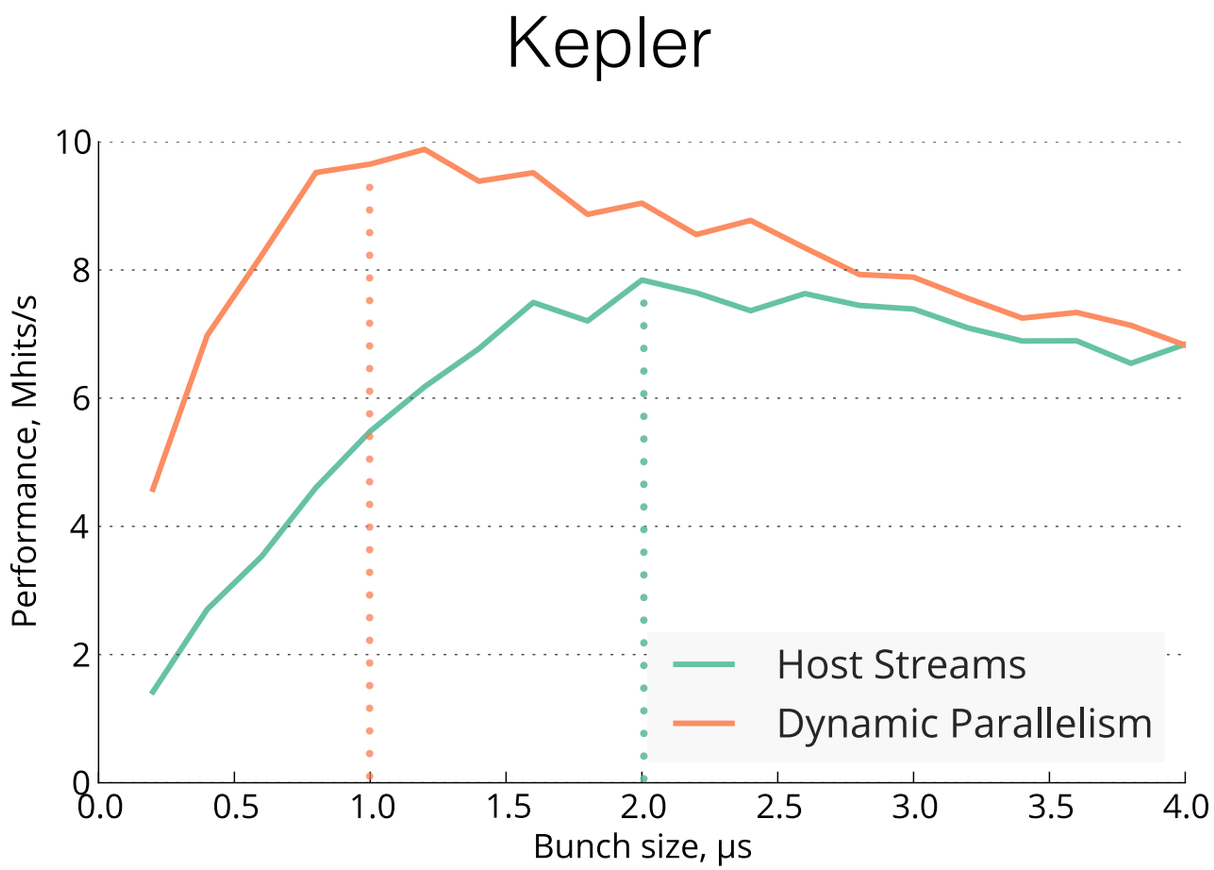
- state-of-the-art triggerless HEP experiment
- starts in 2019
- latest GPUs are of interest (e.g., Maxwell)
  - single precision OK

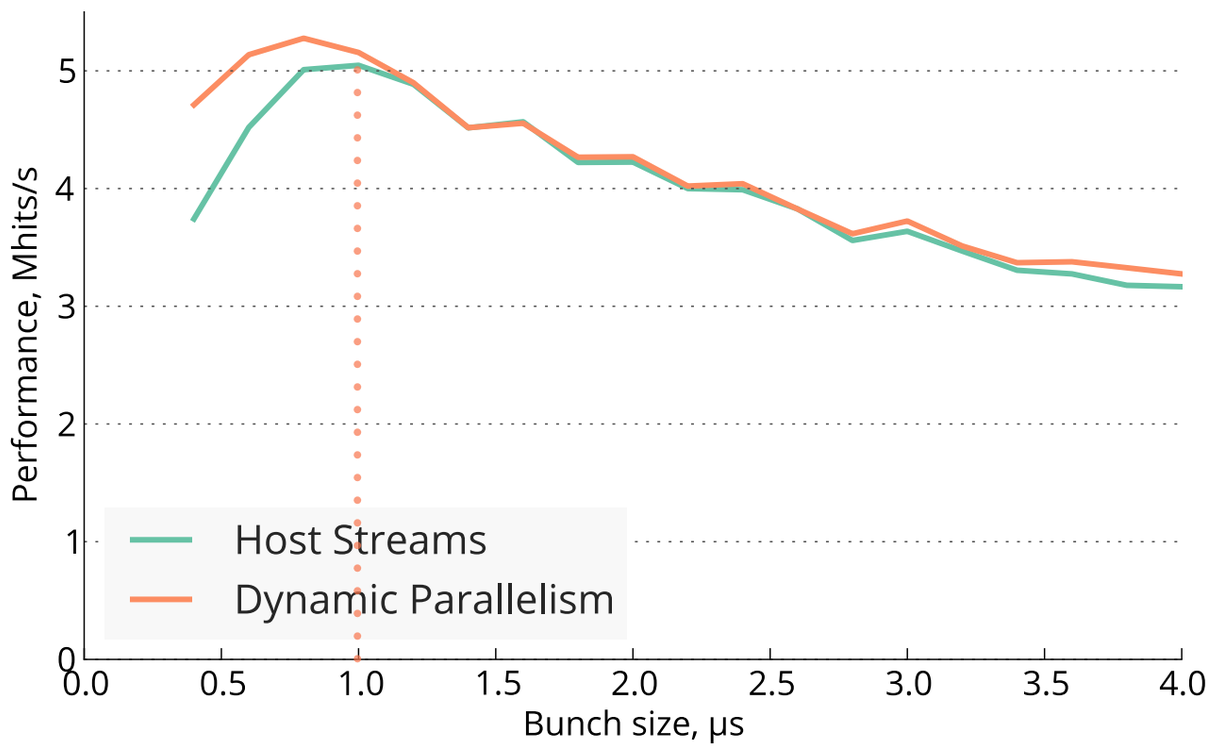


Triplet Finder Maxwell Tests

- Optimal bunch size
- Bunching strategies
  - dynamic parallelism vs. host streams
- Specific optimizations
  - SoA vs. AoS
  - all-tube testing vs. sector-rows
  - all-skewlet testing vs. binning
  - optimal #connections
- Maxwell vs. Kepler comparison

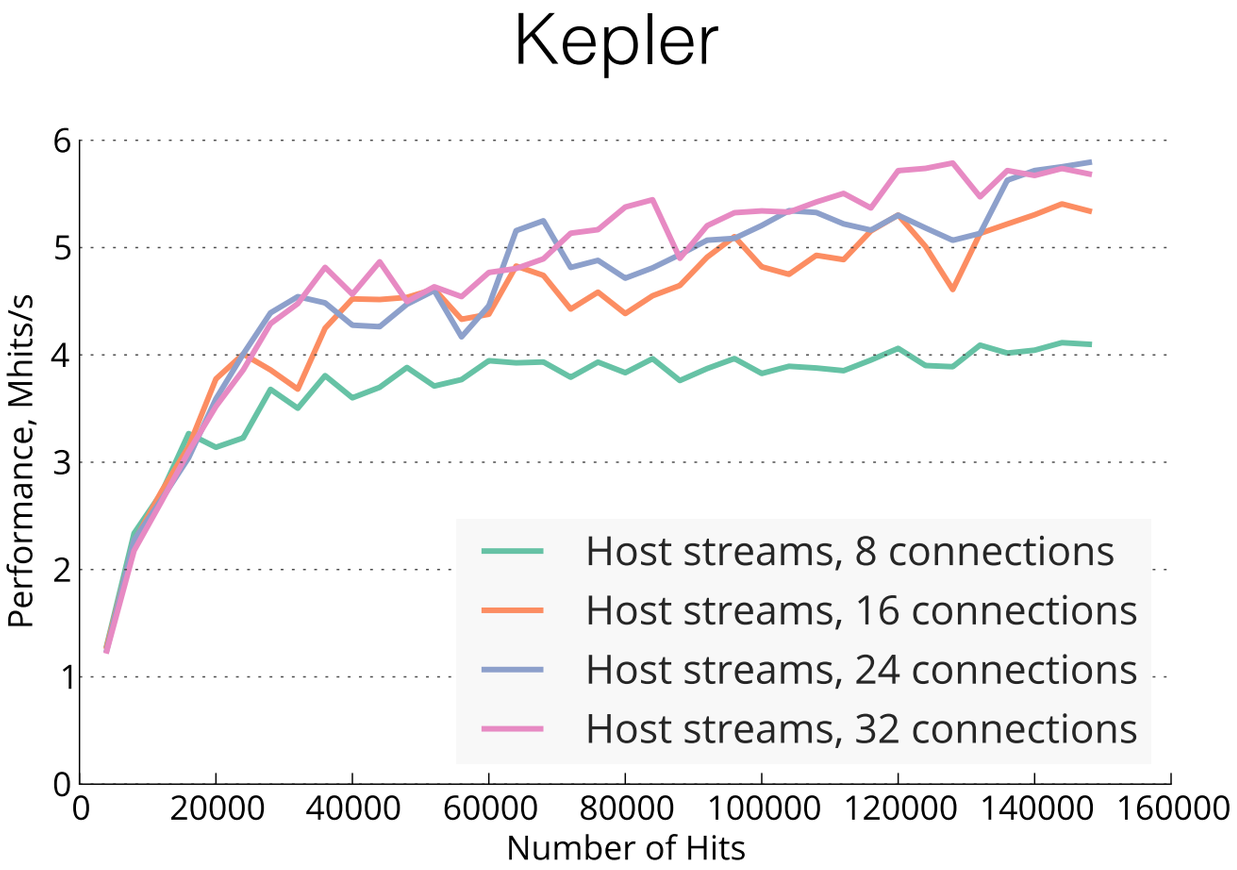
Optimal Bunch Size

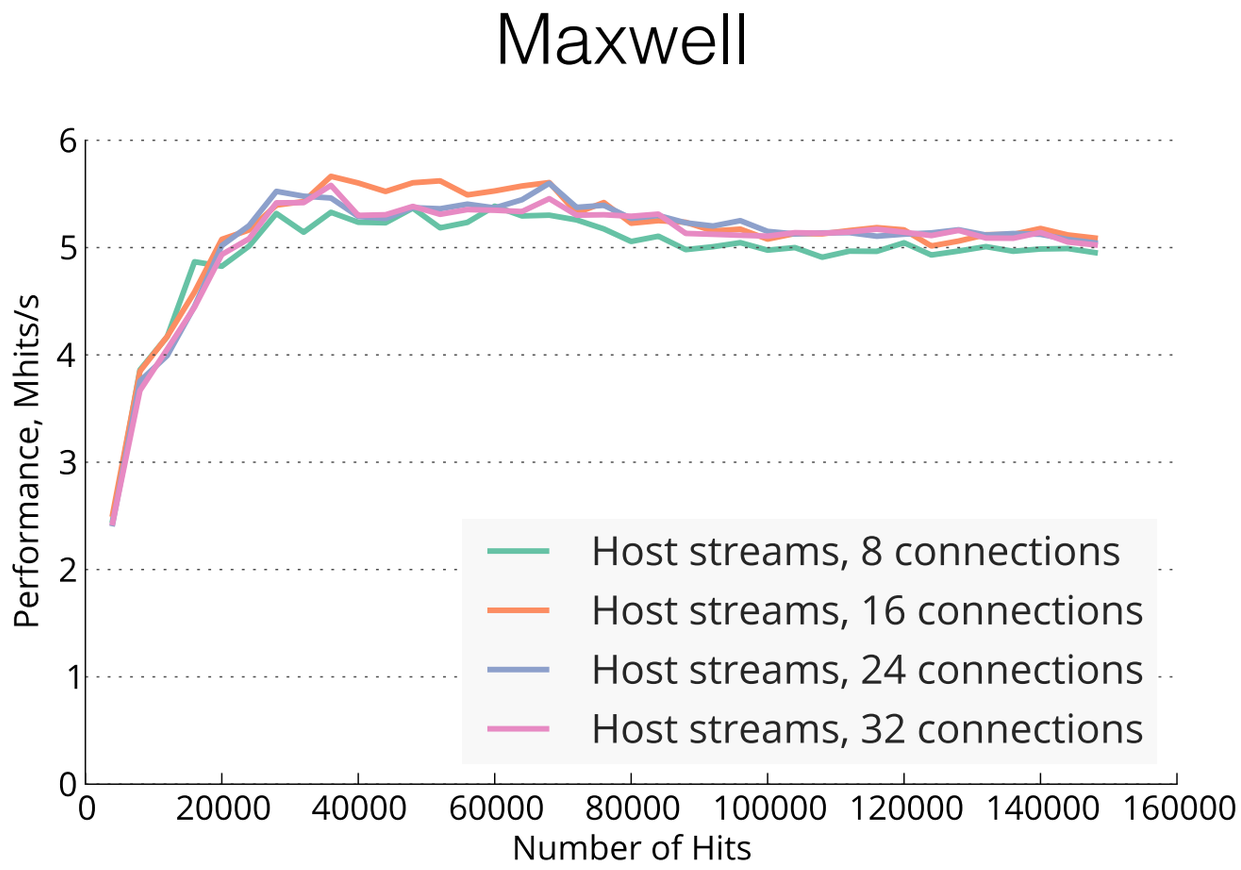




- Kepler: 1 (dynamic parallelism) and 2 µs
- more similar behavior between approaches
- possibly due to smaller GPUs

Number of Connections

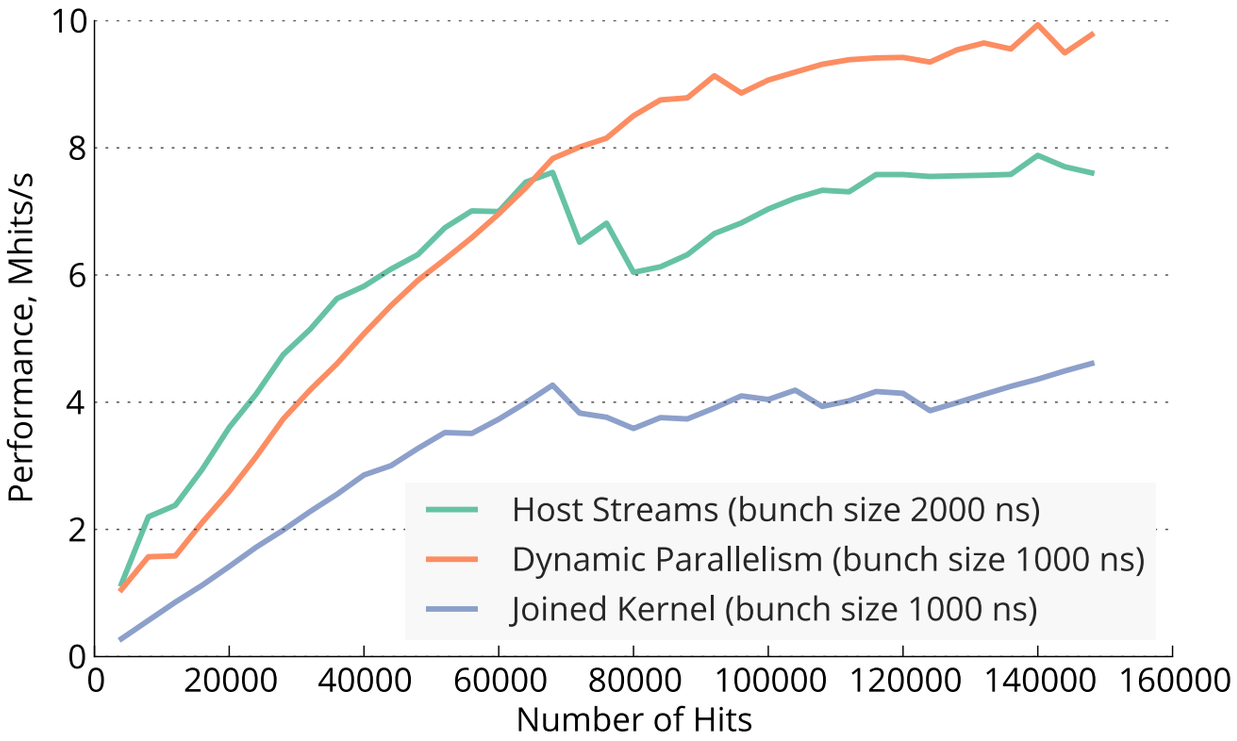




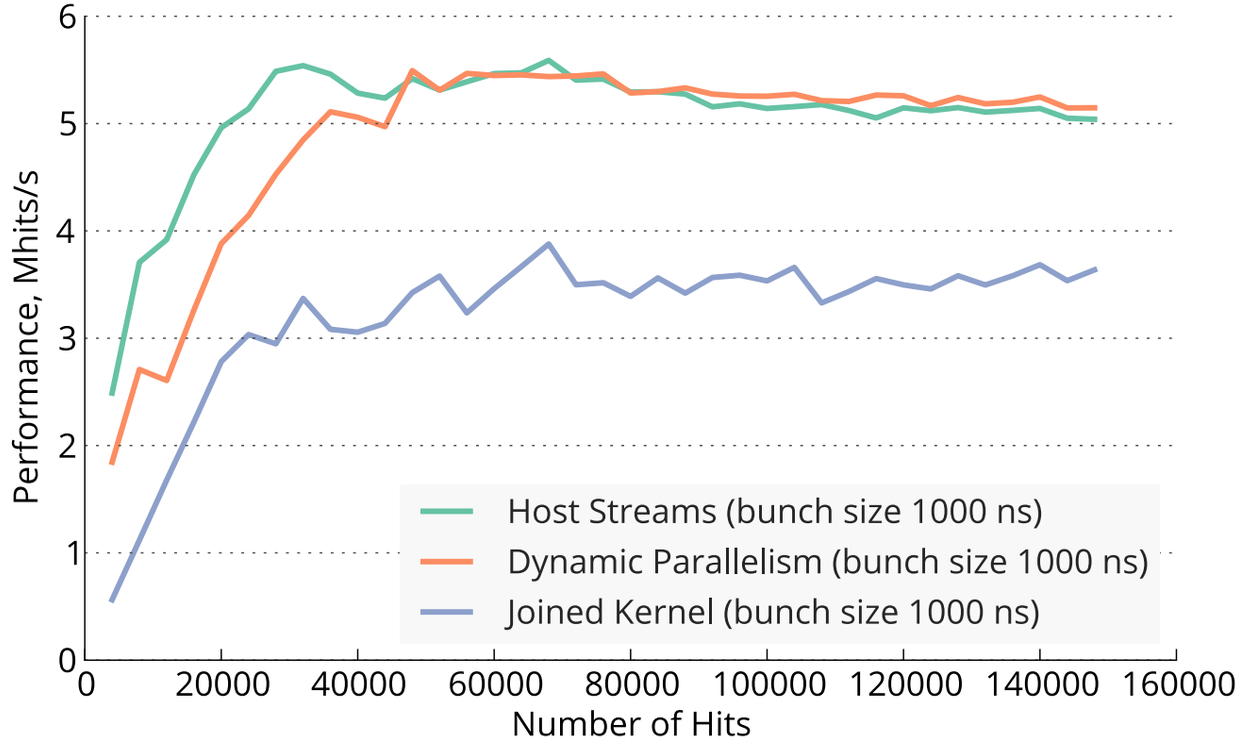
- more similar behavior between #connections
- possibly due to smaller GPUs
- 24 connections used (as for Kepler)

Bunching Strategy

Kepler



Maxwell



- more similar behavior between approaches
- possibly due to smaller GPUs

SoA vs AoS

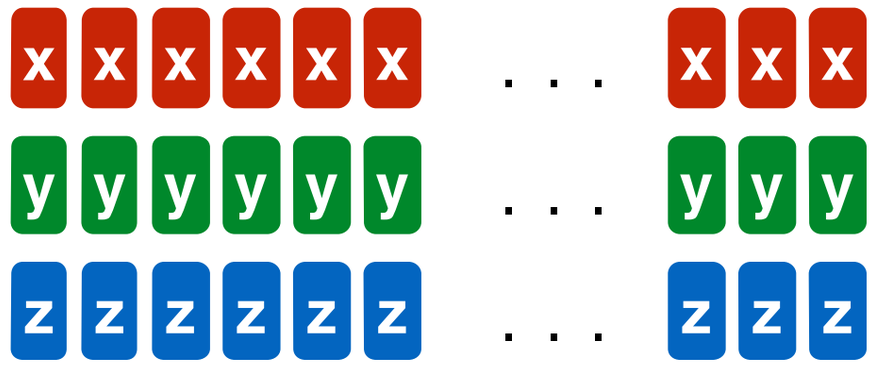
array-of-structures (AoS)

```
struct fvec3 {
  float x;
  float y;
  float z;
} a[N];
```

- better language support

structure-of-arrays (SoA)

```
float ax[N];
float ay[N];
float az[N];
```



- better coalescing
- better performance

Automating SoA

- R. Strzodka, http://people.mpi-inf.mpg.de/~strzodka/projects/layout/

```
template<class Tl> struct fvec3_flex {
  typedef laygroup_t<Tl, float> group_t;
  union {float x; group_t dummy_x;};
  union {float y; group_t dummy_y;};
  union {float z; group_t dummy_z;};
};
typedef fvec3_flex<aos> fvec3;
```

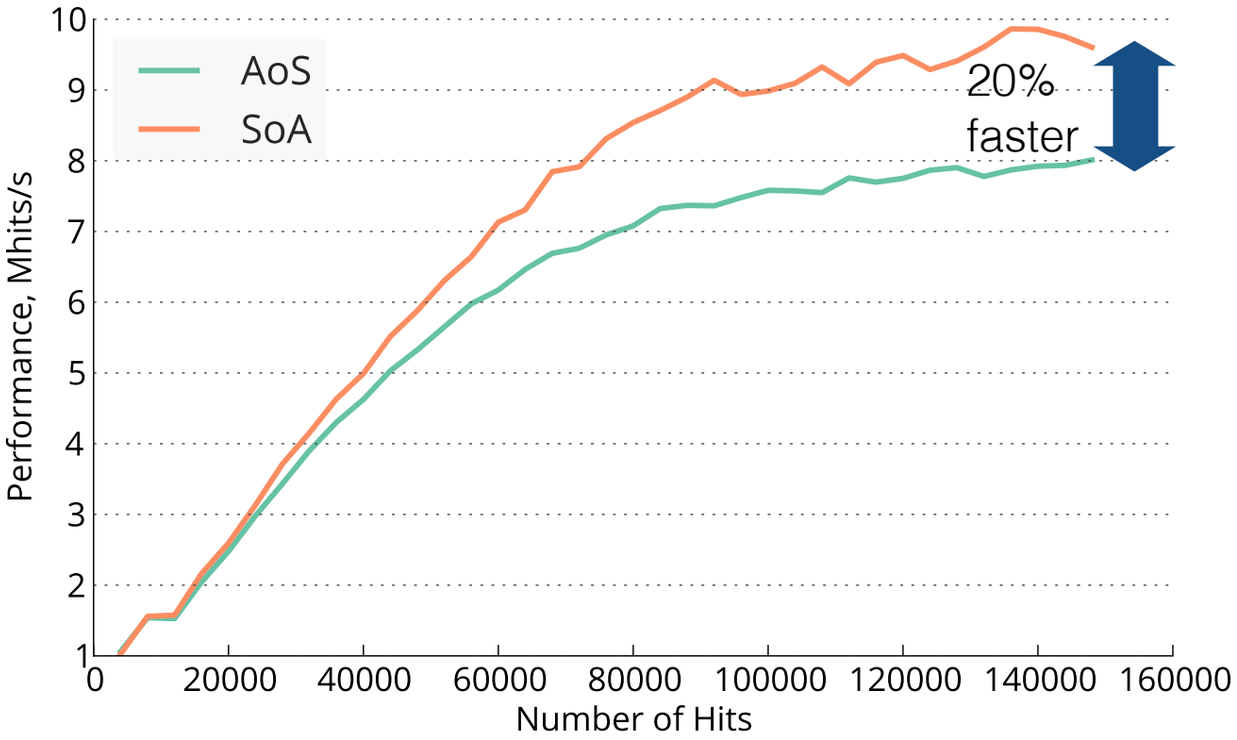
Tl = aos

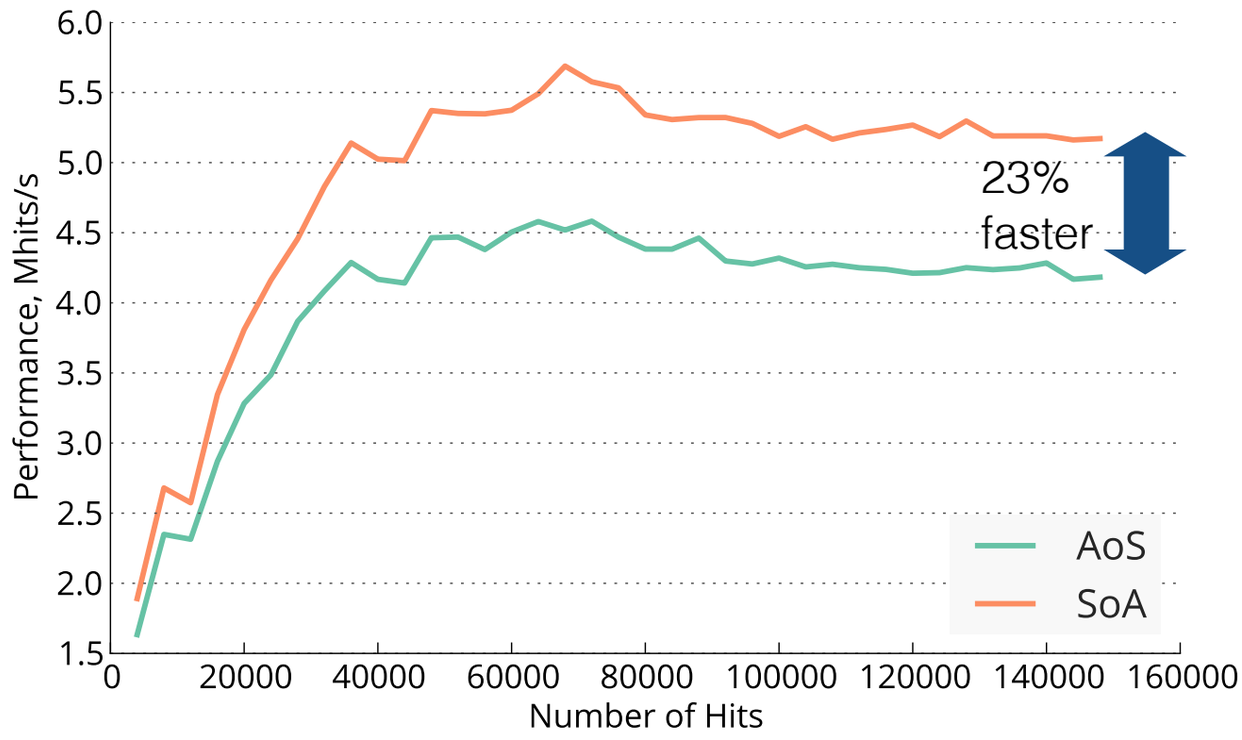
Tl = soa

- clever [] overload
- (Ao)SoA layout
- same field/element access

(really groups of 128 elements)

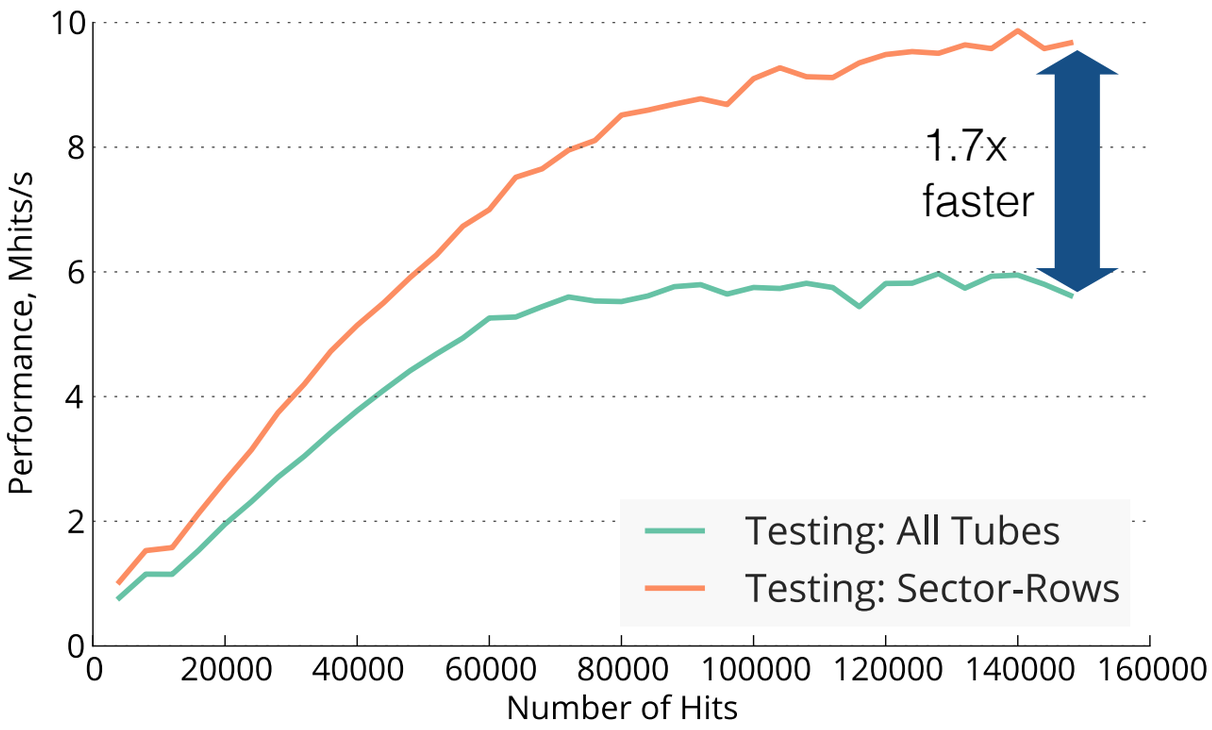
SoA vs AoS Performance

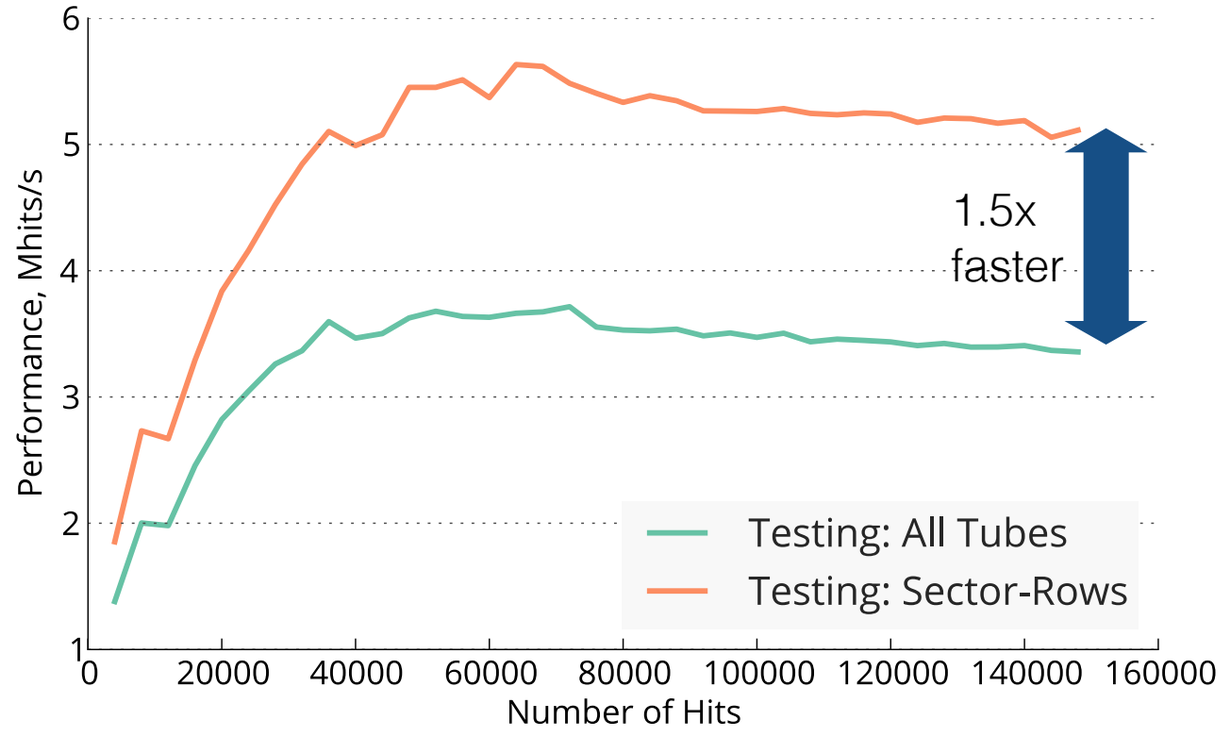




~ 20% improvement with SoA, similar to Kepler

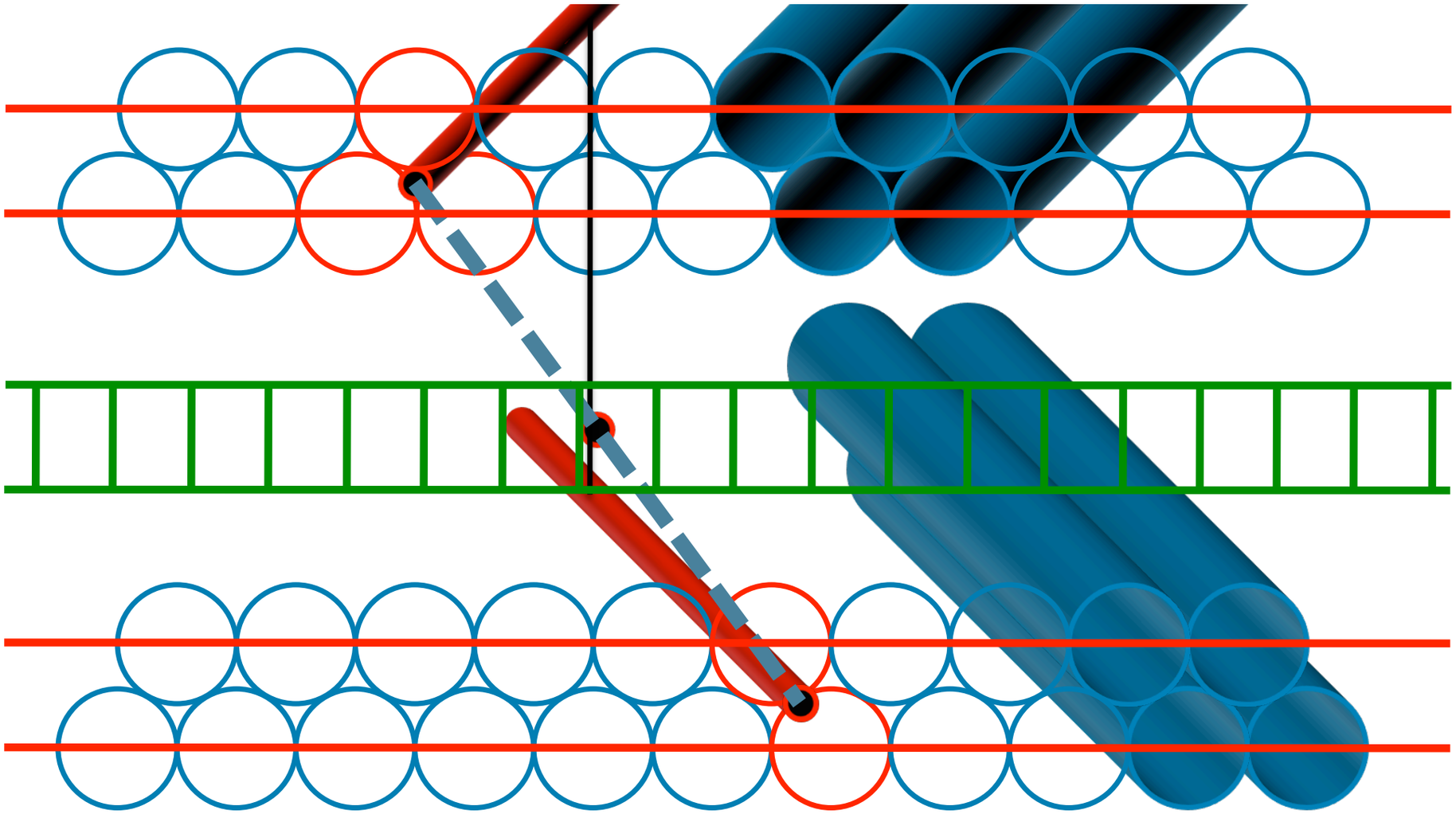
Tube Testing





1.5x improvement, less than Kepler (1.7x)

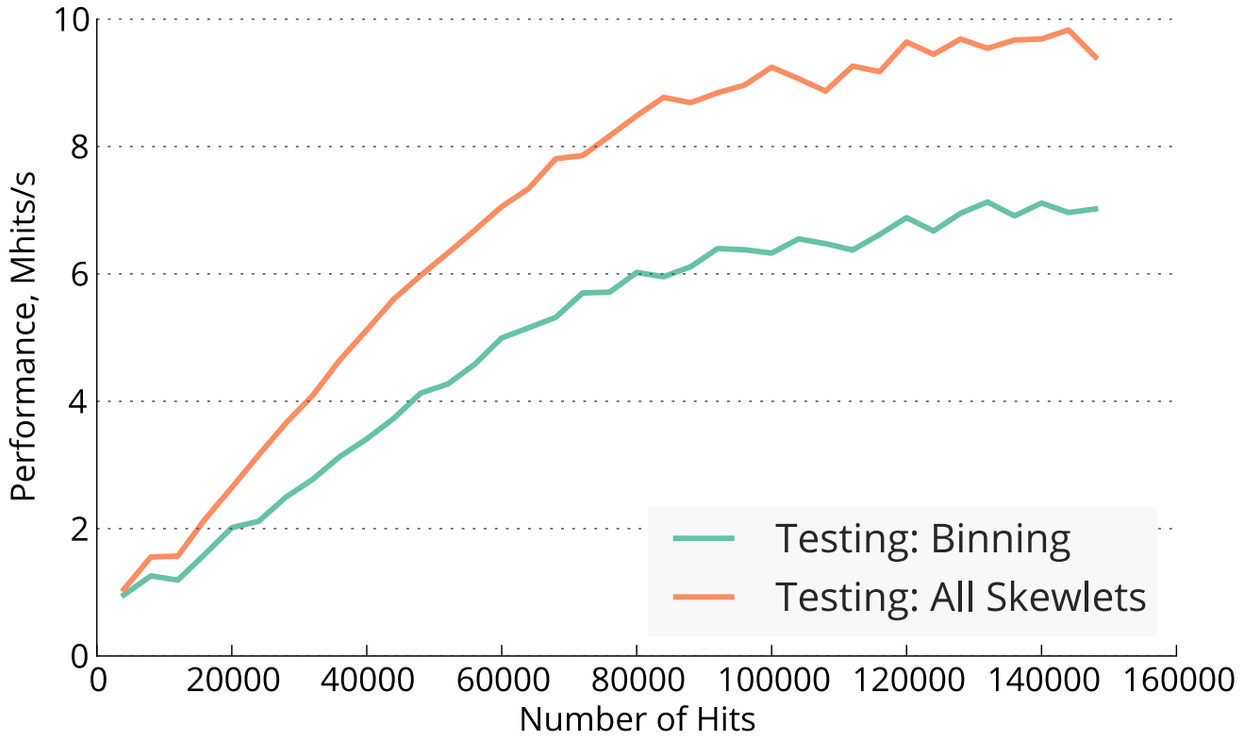
Skewlet Binning



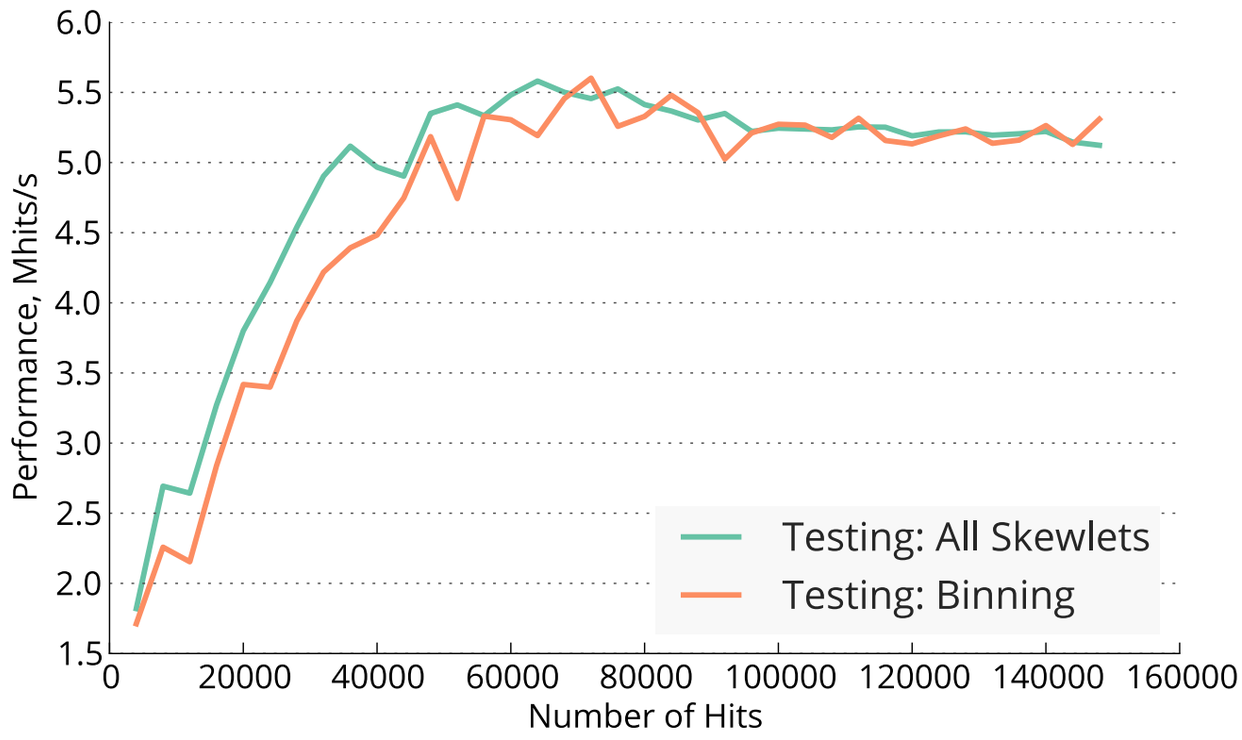
- >30% time taken by skewlet testing
- like sector-rows, but with skewlets
- bin assignment required, different statistics

Skewlet Binning Performance

Kepler

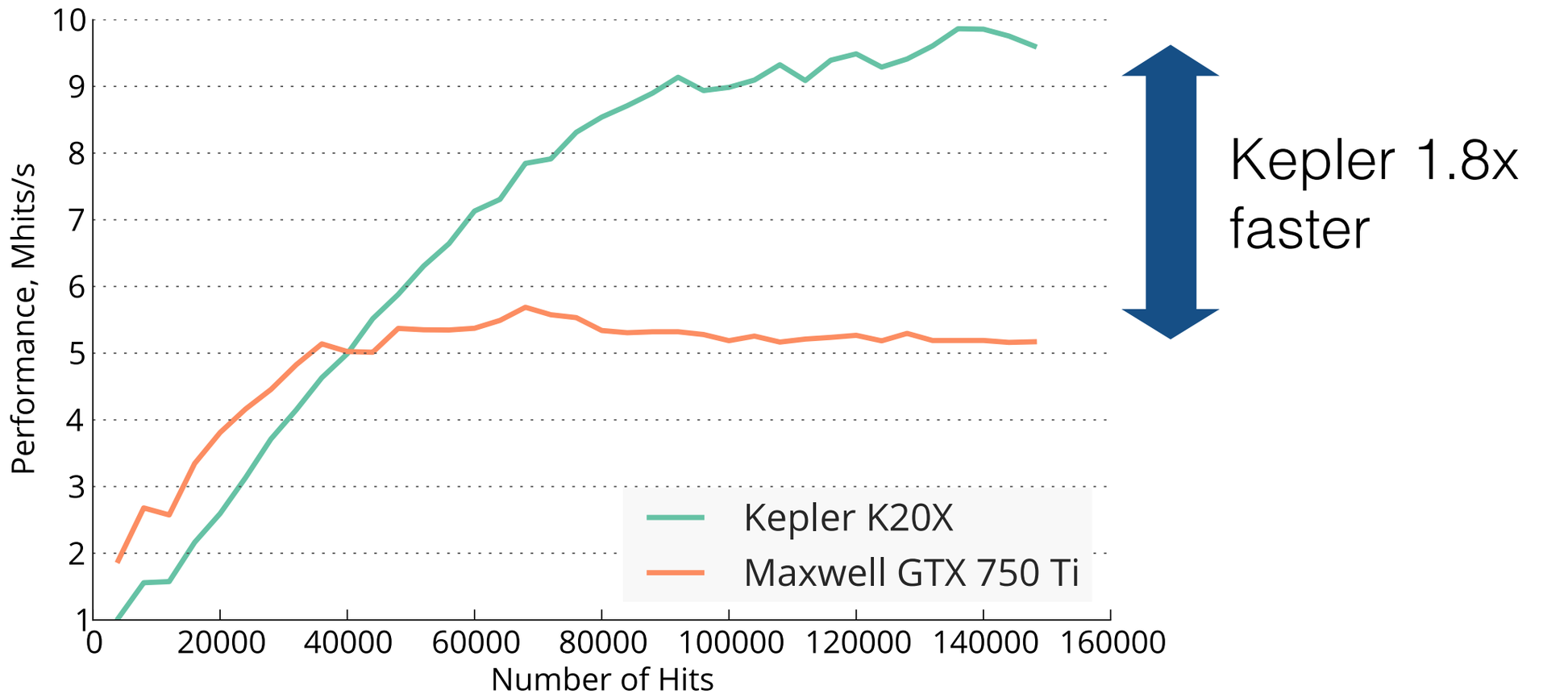


Maxwell



- similar performance
- Kepler: binning **2x slower**

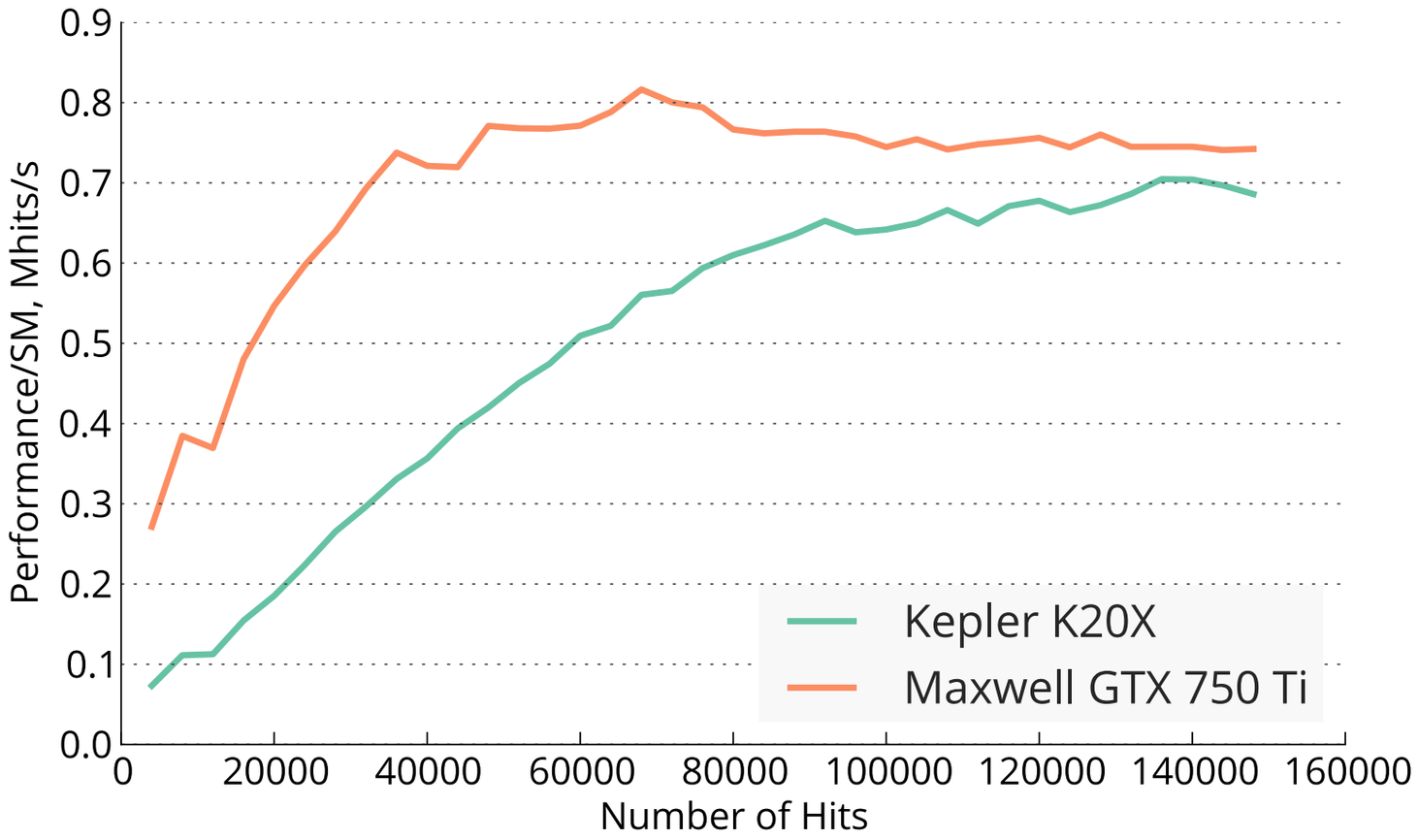
Maxwell vs. Kepler



- Kepler faster when saturated
- Maxwell faster on smaller #hits (< 40K)
  - higher frequency?

Maxwell vs. Kepler (per-SM)

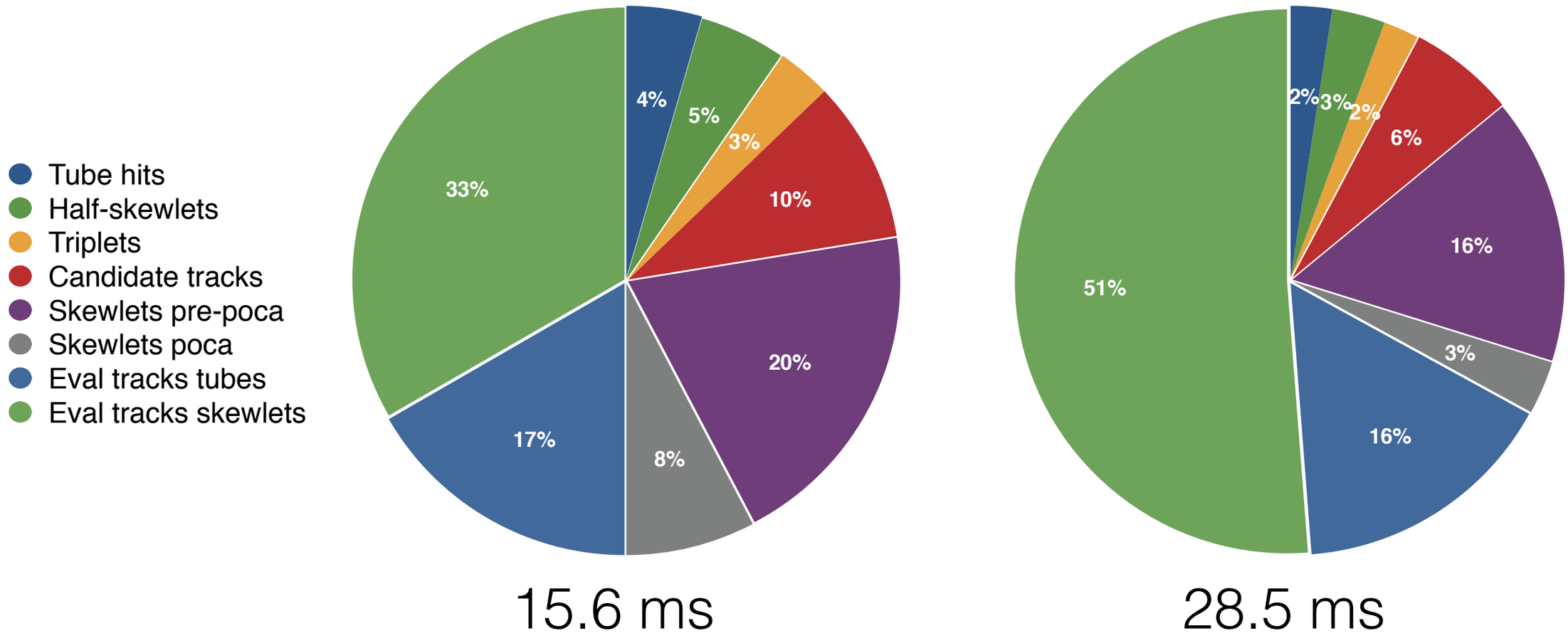
Adjusted for frequency



- SMM vs. SMX performance difference within 10%
  - ~20% error range for Maxwell

Runtime Breakdown

148 556 hits



15.6 ms

28.5 ms

- similar runtimes before track evaluation
- skewlet testing dominates even more for Maxwell

Conclusions

- Very promising first experiences
  - histogram faster than on K20X
  - PANDA: good performance of some stages
- Larger Maxwell GPU wanted
  - smaller GPUs: not all effects are clear
  - = GM 204 or server-grade

Questions?

- NVLab at FZJ: http://fz-juelich.de/ias/jsc/nvlab