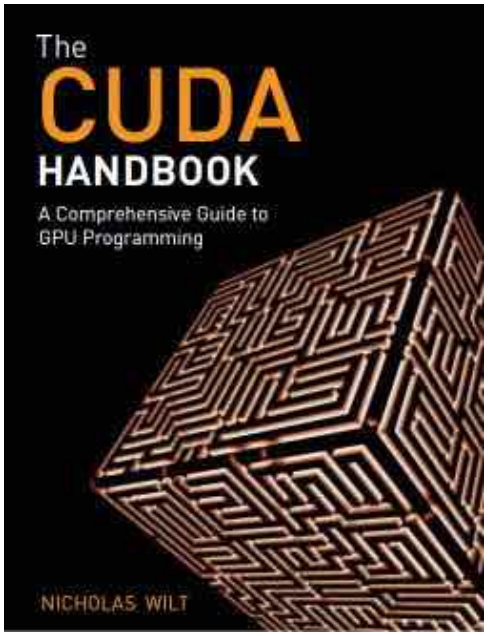


Histograms

Privatized for Fast, Level Performance

Nicholas Wilt

Author, *The CUDA Handbook*

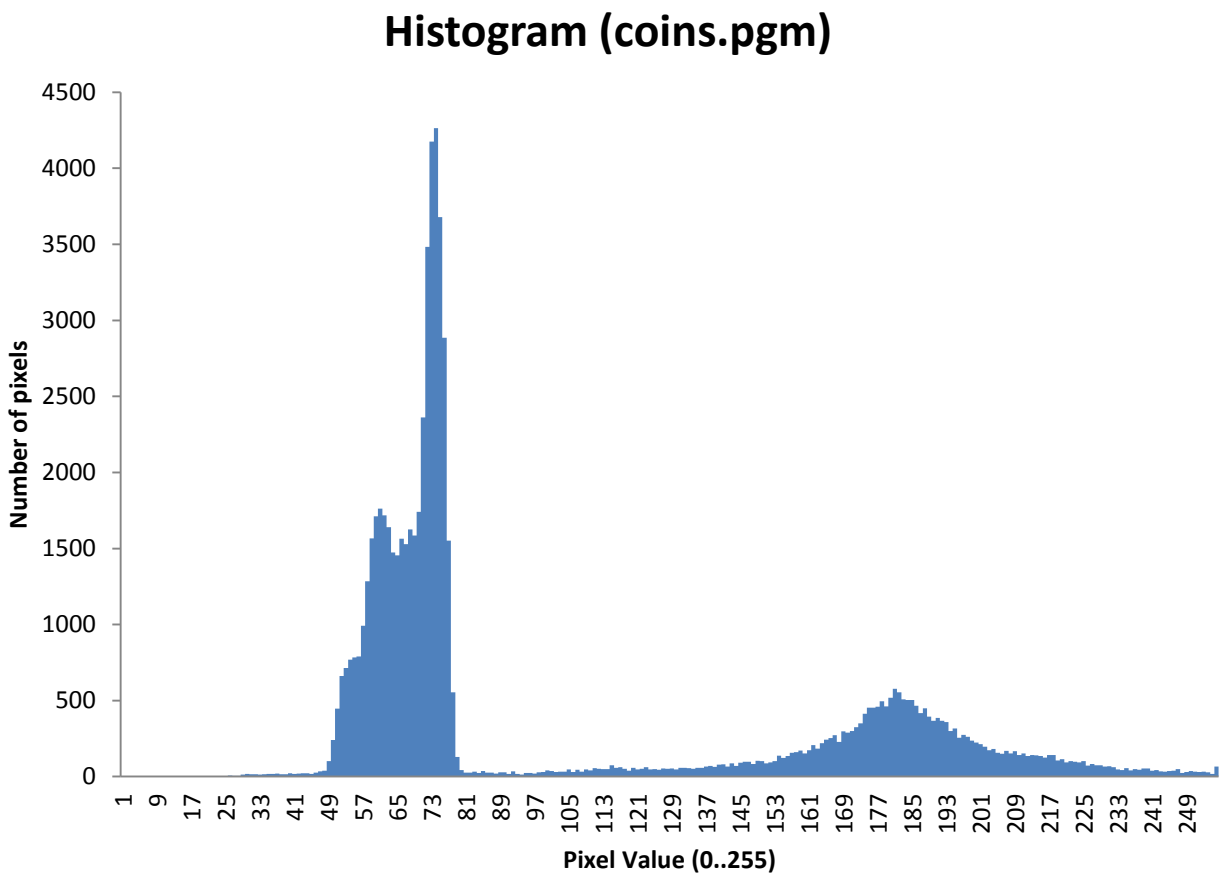
@CUDAHandbook

http://www.facebook.com/cudahandbook

What Is A Histogram?

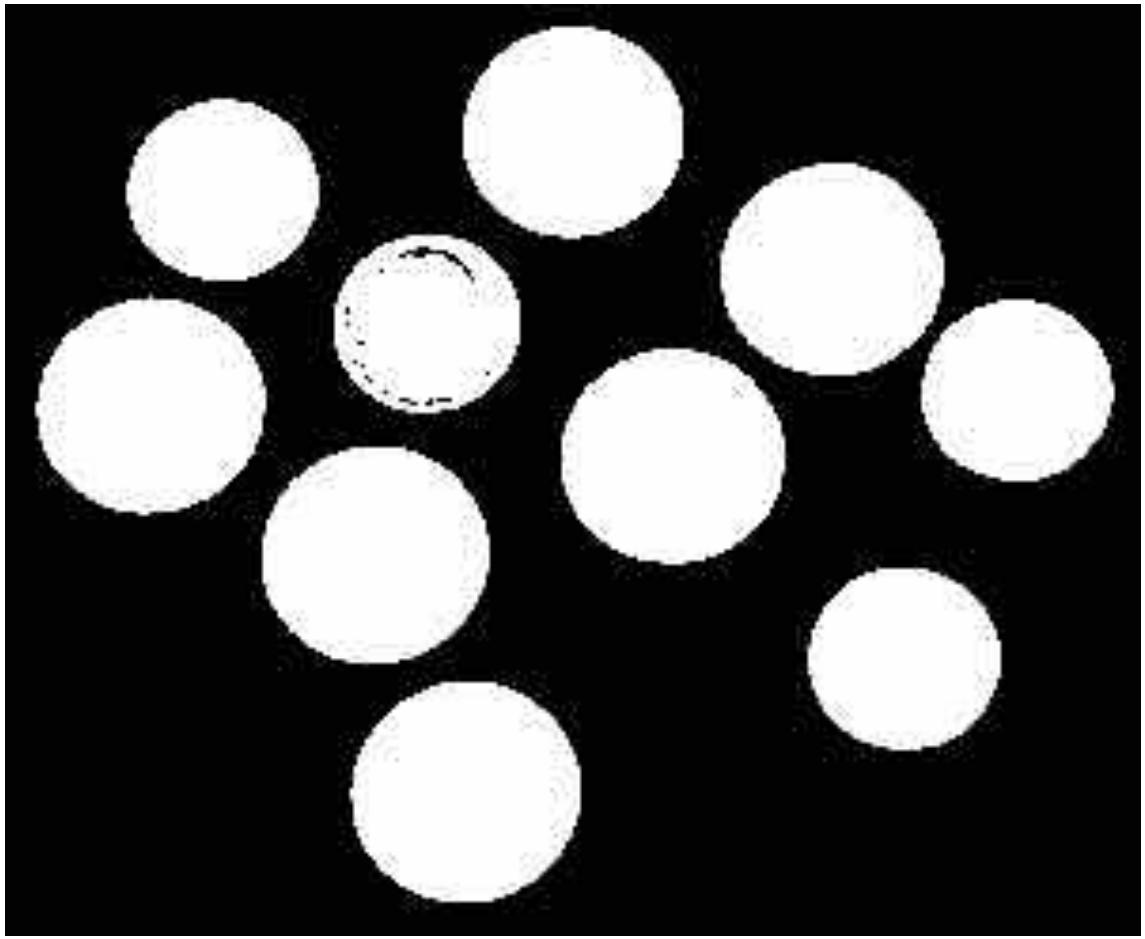
- Probability distribution
- *k* categories and *N* data elements
- Often represented by array of *k* integers
- Many statistics can be inferred from the histogram
  - Min, max, mean, median
- Also a building block (e.g. Radix Sort)

Example Histogram



Source Image

Binarized Image



CPU Code

```
void
hist1DCPU(
    unsigned int pHist[256],
    const unsigned char *p, size_t N )
{
    for ( size_t i = 0; i < N; i++ ) {
        pHist[ p[i] ] += 1;
    }
}
```

Naïve CUDA Code

- One Histogram In Global Memory
  - Use atomic add for correctness

```
__global__ void
histogram1DPerGrid(
    unsigned int *pHist,
    const unsigned char *p, size_t N )
{
    for ( size_t i = blockIdx.x*blockDim.x+threadIdx.x;
                  i < N;
                  i += blockDim.x*gridDim.x ) {
        atomicAdd( &pHist[ p[i] ], 1 );
    }
}
```

Performance

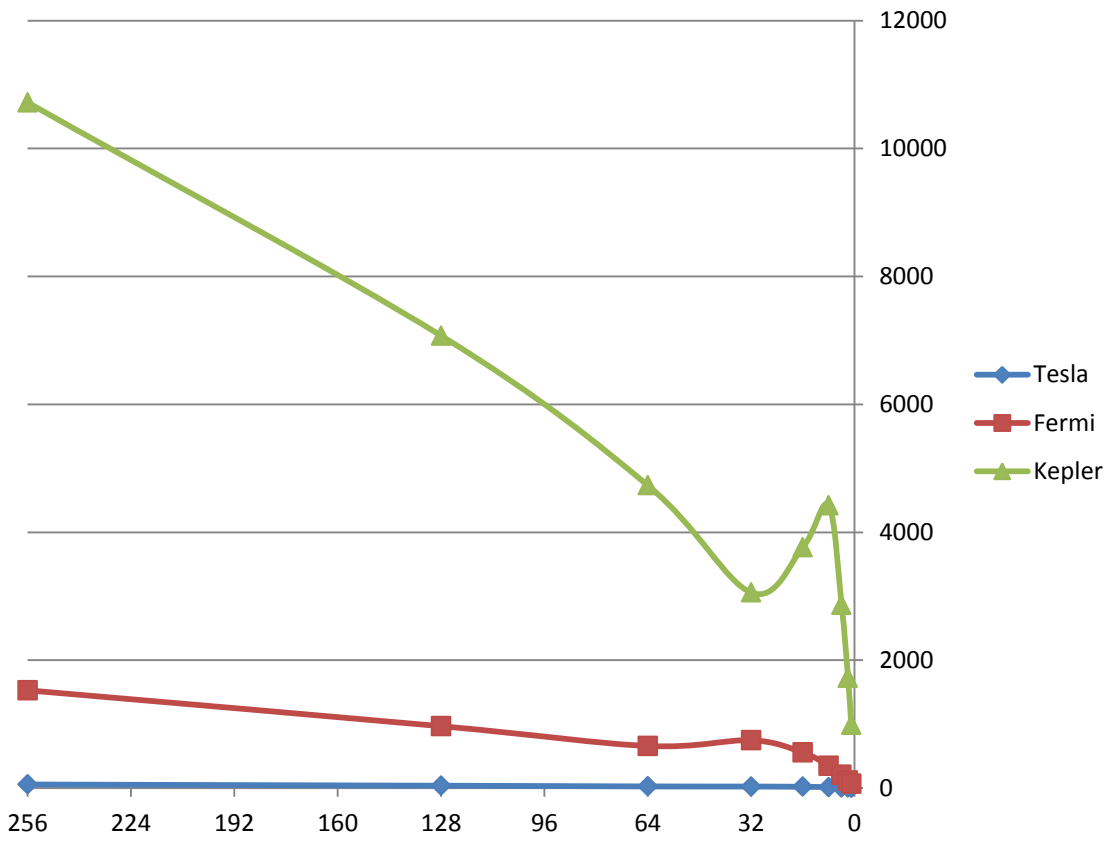
Chip	Speed (Mpix/s)
Tesla (GeForce GTX 280)	58
Fermi (M2050)	1530
Kepler (GeForce GTX 680)	10720

(256 possible pixels in input)

Contention

Values	Tesla	Fermi	Kepler
256	58	1530	10720
128	39	969	7074
64	28	660	4734
32	26	749	3058
16	22	557	3767
8	16	349	4422
4	11	210	2864
2	7	121	1725
1	4	68	988

Contention (Visual)



Anti-Contention Strategies

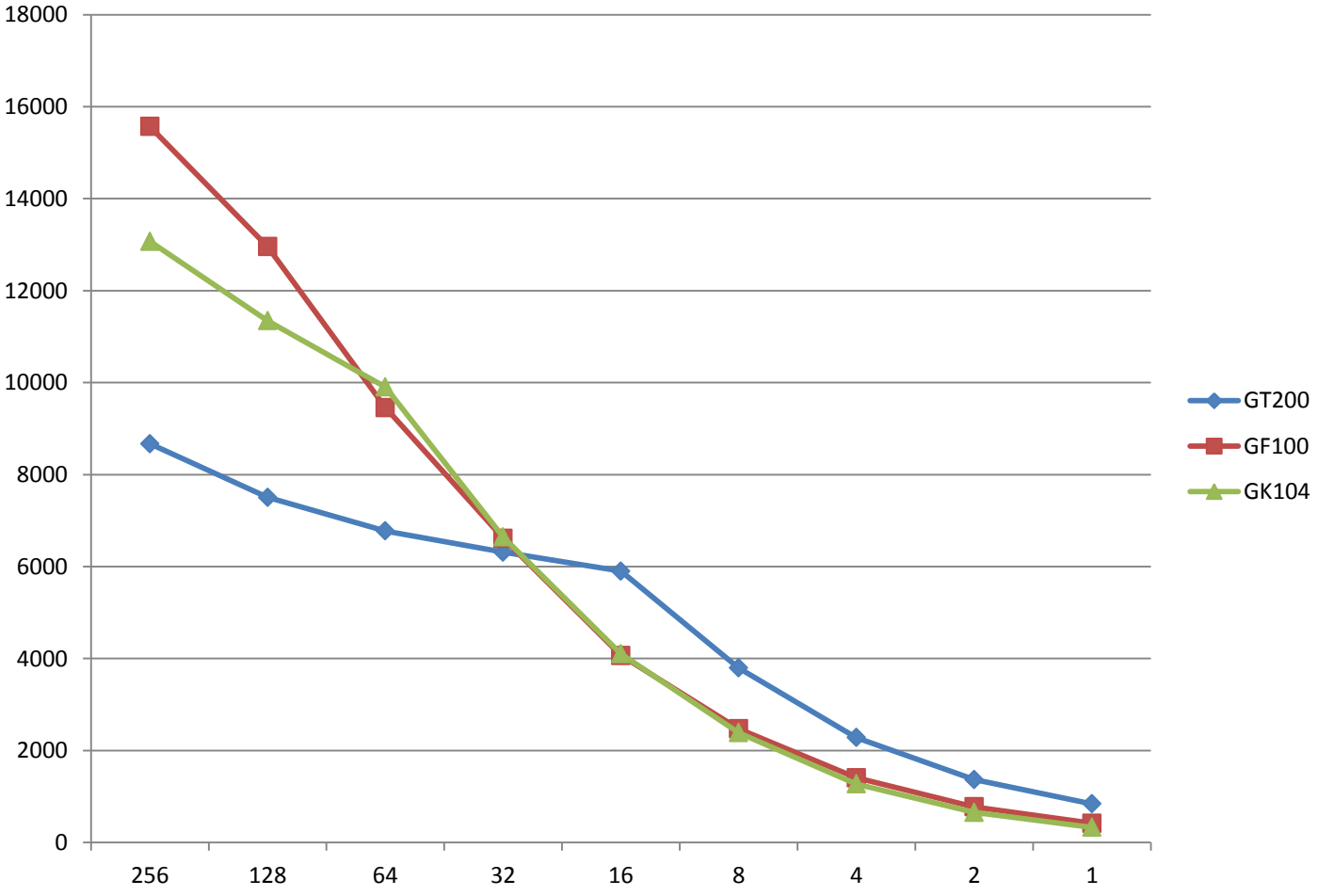
- More histogram arrays!

- Per Block
  - Faster increments (shared memory)
  - Increments still have to be atomic
- Minuses:
  - Have to reduce histograms into final output
  - Threads within block can still contend

Per-Block Code

```
__global__ void
histogram1DPerBlock(
    unsigned int *pHist,
    const unsigned char *base, size_t N )
{
    __shared__ int sHist[256];
    for ( int i = threadIdx.x; i < 256; i += blockDim.x ) {
        sHist[i] = 0;
    }
    __syncthreads();
    for ( int i = blockIdx.x*blockDim.x+threadIdx.x;
              i < N;
              i += blockDim.x*gridDim.x ) {
        atomicAdd( &sHist[ base[i] ], 1 );
    }
    __syncthreads();
    for ( int i = threadIdx.x; i < 256; i += blockDim.x ) {
        atomicAdd( &pHist[i], sHist[ i ] );
    }
}
```

Performance

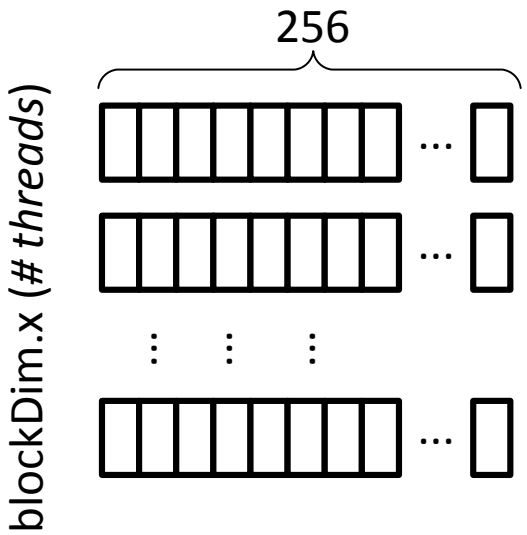


Per-Thread Histograms

- Must use shared memory (addressible)
- 1 byte per element
- 64 threads/block=16K
- Can fit 3 blocks per SM=192 threads

- Many different layout options

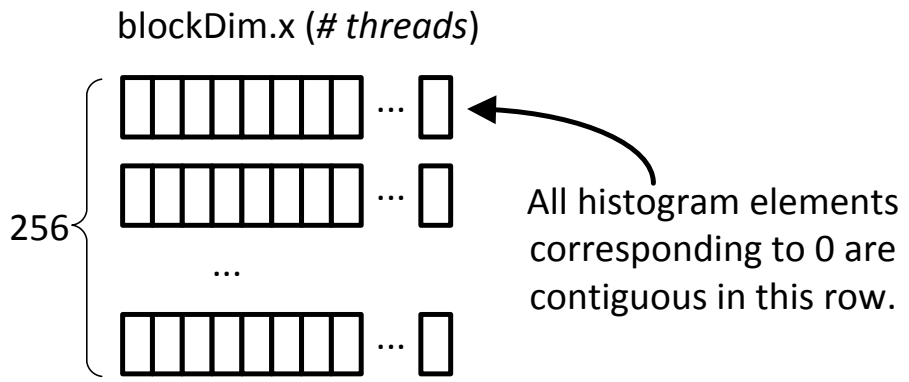
Histogram Per Row?




```
histIndex = threadIdx.x*256+pixval;
```

Problem: For degenerate case, *de facto* contention due to bank conflicts.
And we cannot spend *any* shared memory on padding!

Histogram Per Column?

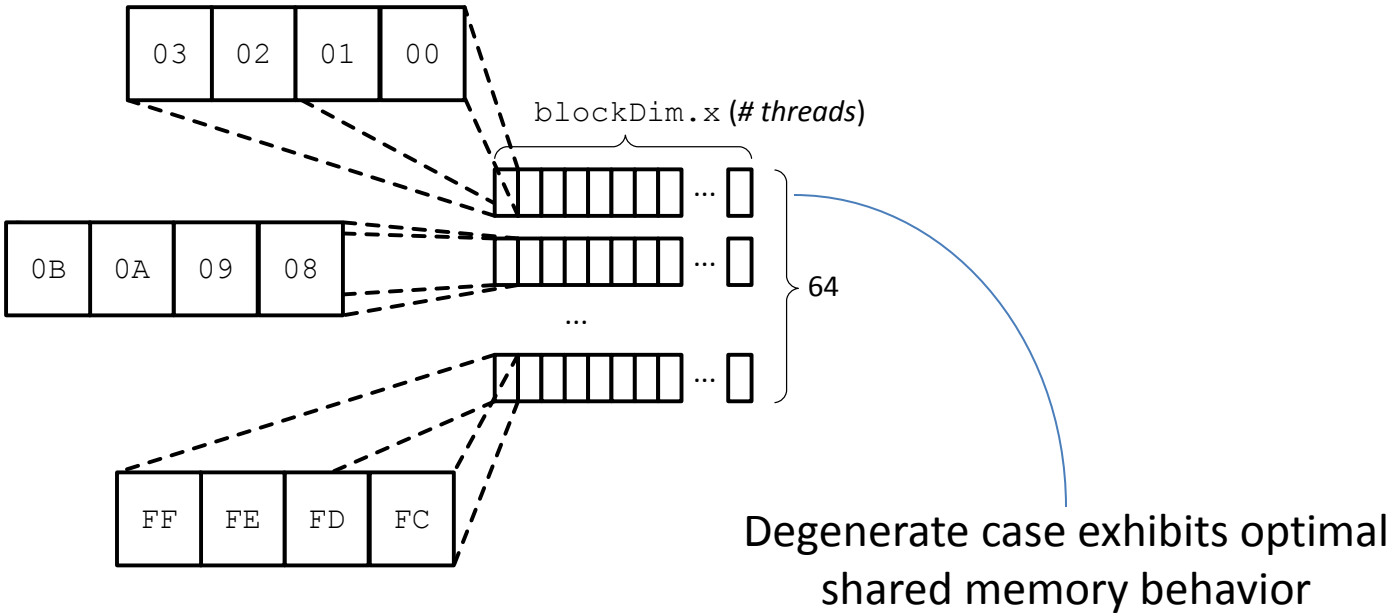



```
histIndex = blockDim.x*pixval+threadIdx.x;
```

Problem: Still prone to bank conflicts (every 4 threads contending for the same 32-bit value in shared memory)

Hybrid Scheme

- 32-bit elements *by column*



32-Bit Increments

- Shared Memory Optimized for 32-bit Accesses
- Rewrite:

```
((unsigned char *) pHist)[i] += 1;
```

- As:

```
((unsigned int *) pHist)[i>>2] += 1<<((i&3)*8);
```

- Fun fact: Kepler compiler translates to byte permute

Resulting Code

- No slower than previous 32-bit increment

*(unsigned int *) pHist)[i>>2] += 1<<((i&3)*8);*

```
inline __device__ void
incPacked32Element( unsigned char pixval )
{
    extern __shared__ unsigned int privHist[];
    const int blockDimx = 64;
    unsigned int increment = 1<<8*(pixval&3);
    int index = pixval>>2;
    privHist[index*blockDimx+threadIdx.x] += increment;
}
```

Gathering Histograms

- Privatized histograms are great! but…
- Now we have 64 histograms per block
  - And multiple blocks→many histograms to reduce.
- And they only contain 8-bit elements
  - need to be gathered frequently to avoid overflow

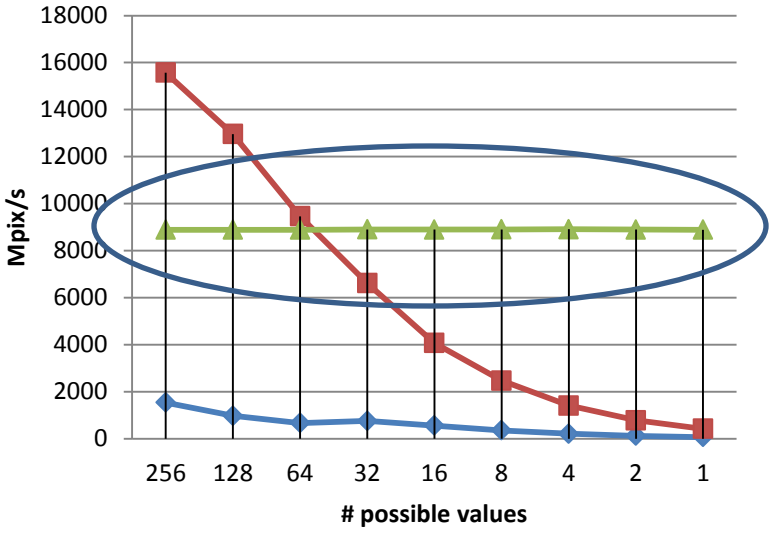
- Performance of this operation surprisingly important!

```
template<bool bClear>
__device__ void
merge64HistogramsToOutput( unsigned int *pHist )
{
    extern __shared__ unsigned int privHist[];

    unsigned int sum02 = 0;
    unsigned int sum13 = 0;
    for ( int i = 0; i < 64; i++ ) {
        int index = (i+threadIdx.x)&63;
        unsigned int myValue = privHist[threadIdx.x*64+index];
        if ( bClear ) privHist[threadIdx.x*64+index] = 0;
        sum02 += myValue & 0xff00ff;
        myValue >>= 8;
        sum13 += myValue & 0xff00ff;
    }

    atomicAdd( &pHist[threadIdx.x*4+0], sum02&0xffff );
    sum02 >>= 16;
    atomicAdd( &pHist[threadIdx.x*4+2], sum02 );

    atomicAdd( &pHist[threadIdx.x*4+1], sum13&0xffff );
    sum13 >>= 16;
    atomicAdd( &pHist[threadIdx.x*4+3], sum13 );
}
```

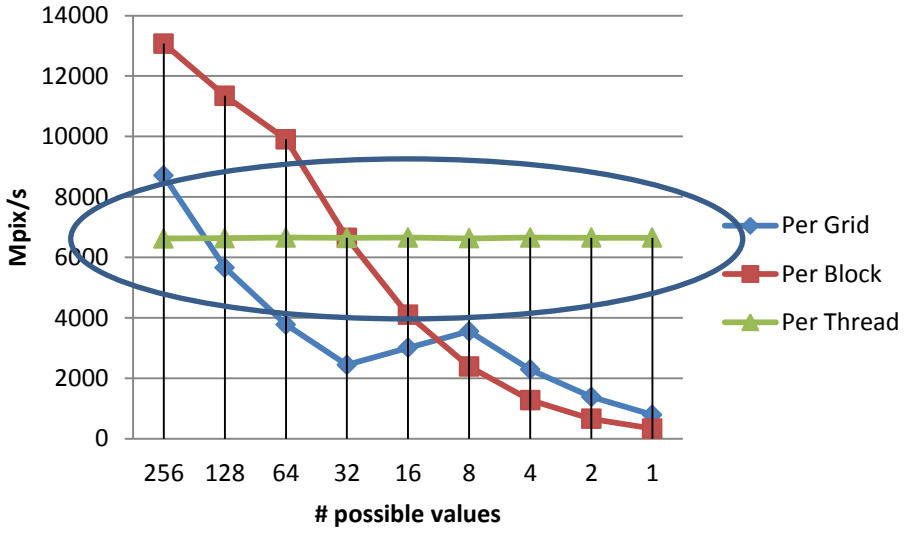
Using thread ID to avoid bank conflicts

64 threads, so exactly 256 global atomic adds per invocation

Result: Level Performance



Fermi (GF100)



Kepler (GK104)

CPU Comparison (Haswell)

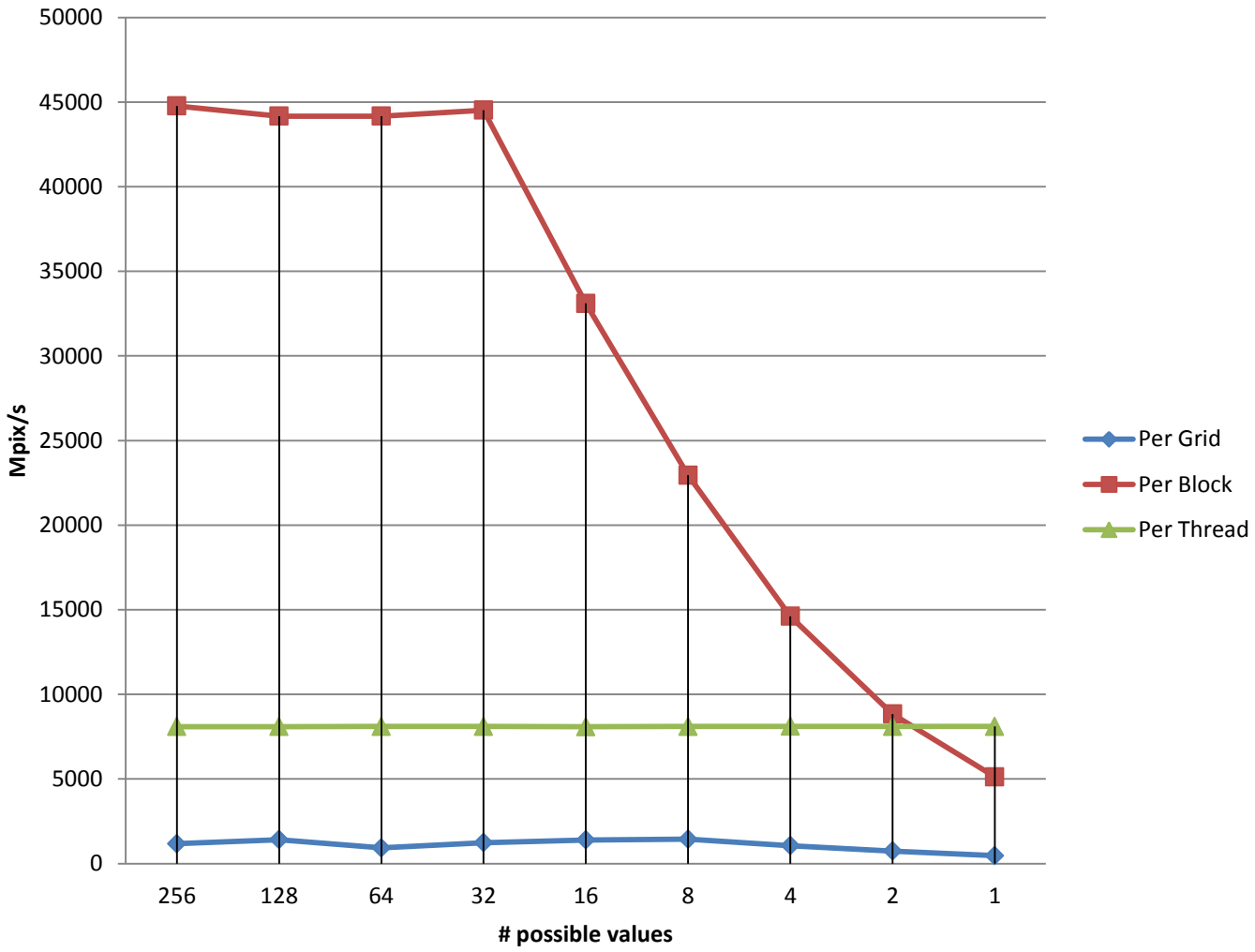
- Per-thread privatized histograms
- 2 GB/s/core, and very level
- GF100 is only 9GB/s, GK104 is only 6.6 GB/s

- So this is a workload where GPUs don't "pwn" CPUs. Best done on data already in the GPU.

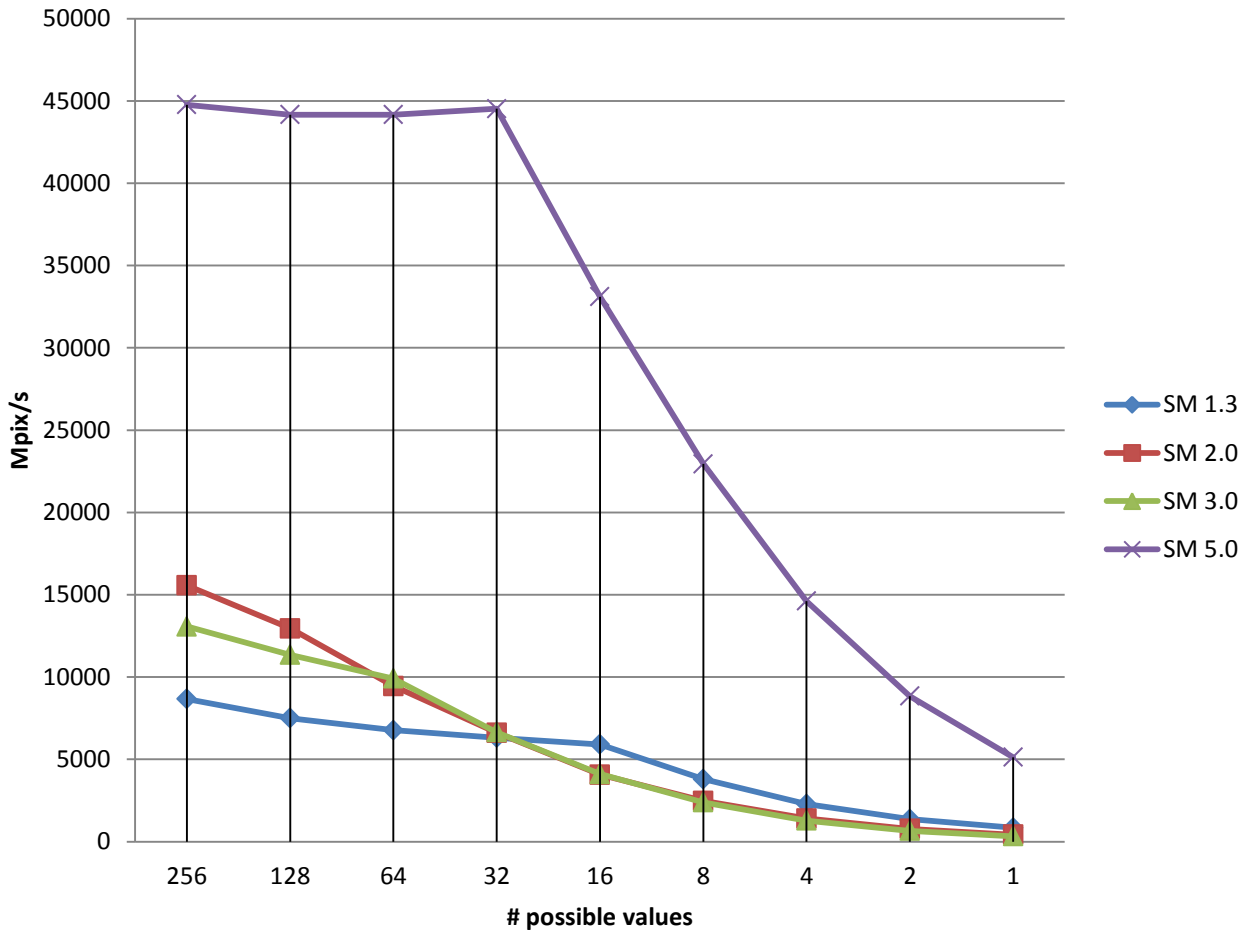
Epilogue

- *To improve performance of this and similar workloads, NVIDIA could add native hardware support for shared memory reductions. –November 2013*
- Maxwell
  - Hardware support for smem atomics!
    - Lower-latency increments for per-block formulation!
  - 64K shared memory (not shared with L1)!
    - More occupancy!

Maxwell Performance



Per-Block Performance



Questions?

- Twitter handle: @CUDAHandbook
- http://www.facebook.com/cudahandbook