Stream Compaction for Deferred Shading

Jared Hoberock*
Victor Lu
Yuntao Jia
John C. Hart

UPCRC
University of Illinois

(*now at NVIDIA)

Our Contribution

- Shading requests from rasterization are spatially coherent
  - Less so when shading is deferred until after rasterization
- Shading requests from ray tracers are spatially incoherent
  - Neighboring processes need to run completely different shaders
- Shading requests can be deferred and batch processed
- SIMD processing of incoherent shading batches suffers from control flow divergence

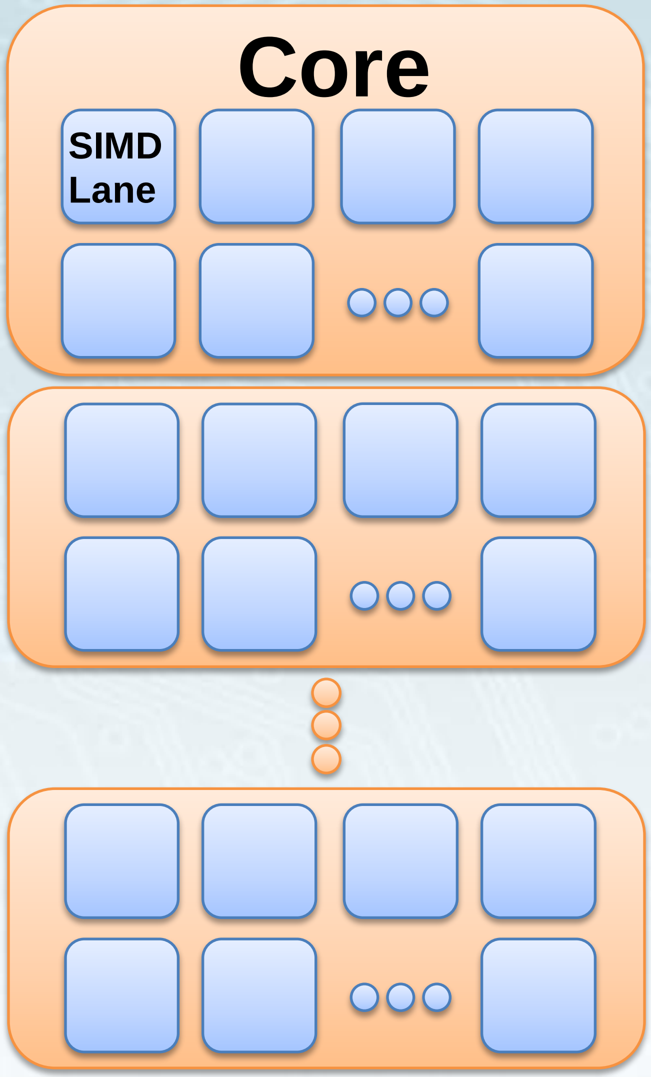
- **Is it worth clustering shading requests into coherent batches to avoid SIMD divergence?**

Previous Work

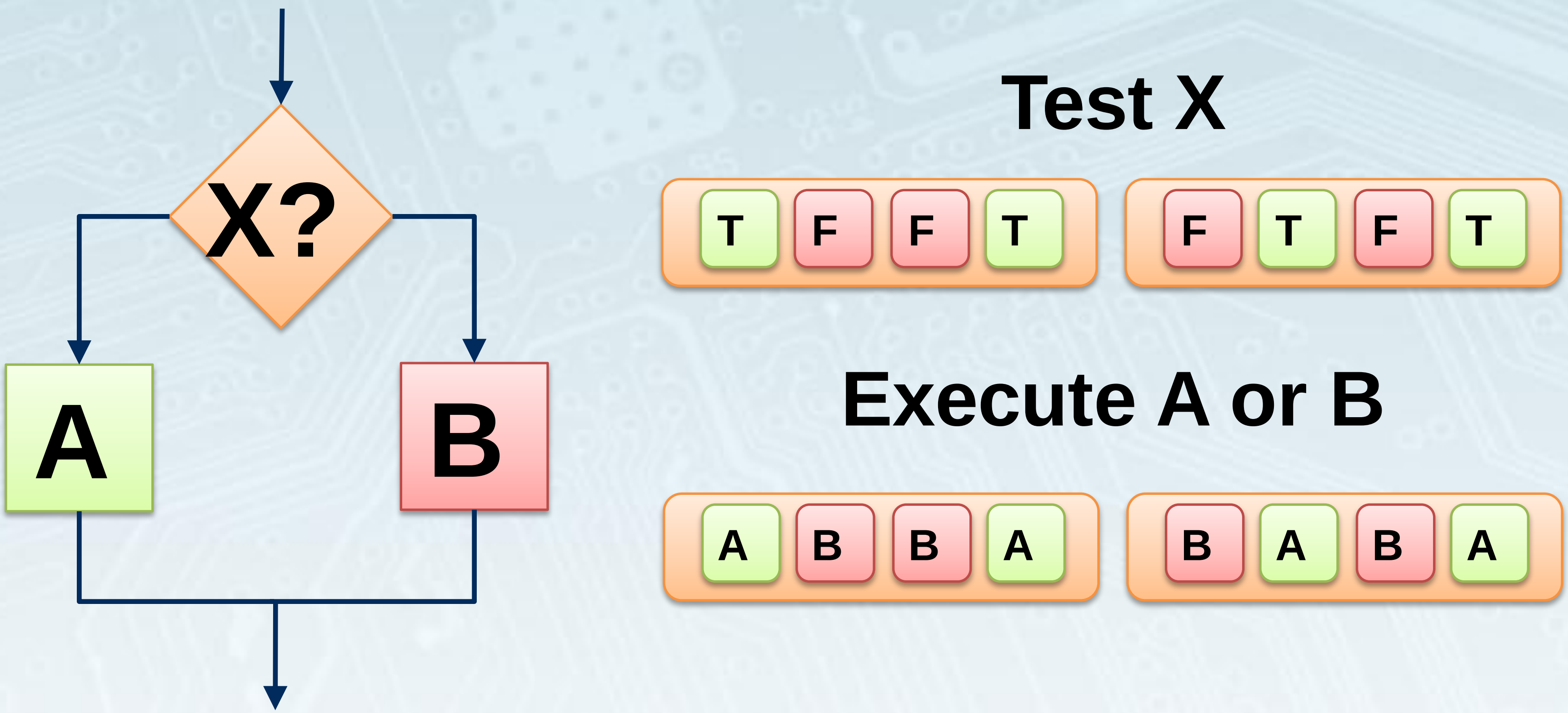
- Memory Coherence for Out-of-Core Processing [Pharr et al. 1997]
  - Encouraged memory coherence within intersection jobs whereas we encourage instruction coherence within shading jobs
- Ray-Hierarchy Traversal
  - Mannson et al. [2007] measured divergence
  - Wald et al. [2007] *simulated* compaction to avoid divergence
- Dynamic Warp Formation [Fung et al. 2007]
  - Local re-ordering hardware v. global re-ordering software
- Load Balancing [Aila & Laine 2009]
  - Ray tracing is a scheduling problem

Data Parallel Architectures

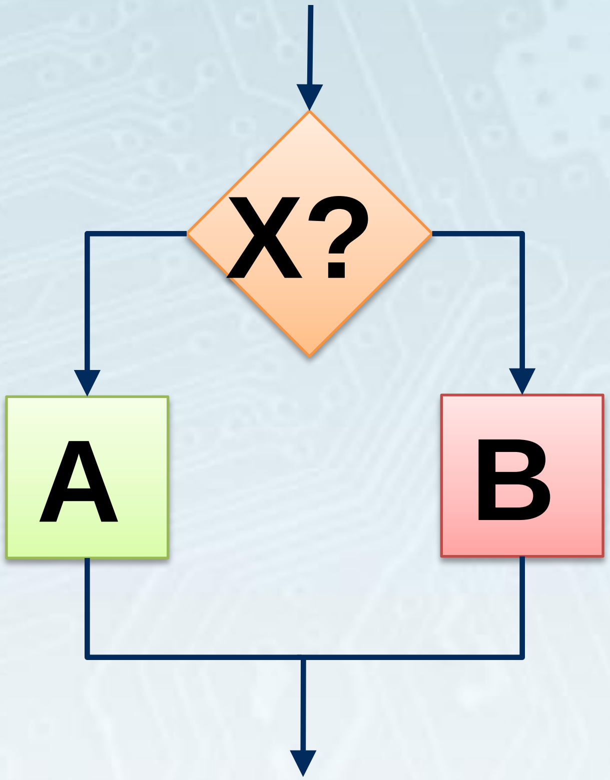
- MIMD "cores"
  - Each core has its own instruction counter
  - Cell:8, GT200:30, LRB:32
- SIMD vector processors
  - Lanes share same instruction counter
  - Cell:4, GT200:8, LRB:16
- Programmer may see even wider degree of SIMD parallelism
  - NVIDIA's 32-wide "warps"



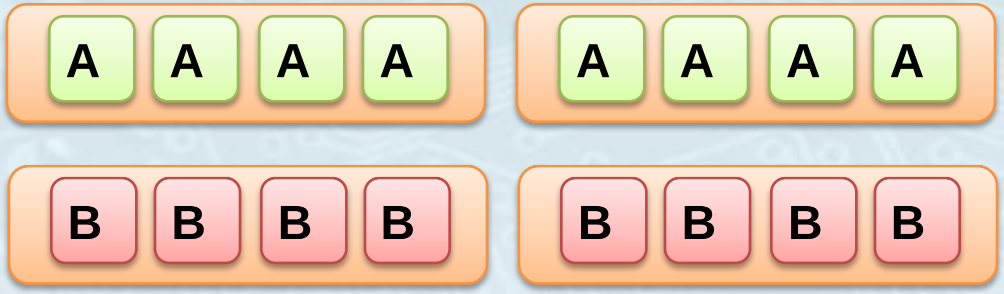
SIMD Divergence: Conceptual



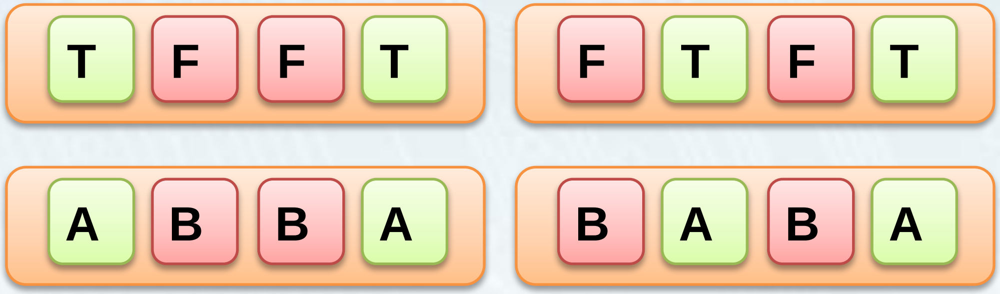
SIMD Divergence: Actual



Execute *Both* A and B



Mask on X



SIMD Divergence: Measurement

$$\text{Efficiency} = \frac{\text{Useful Work}}{\text{Total Effort}}$$

$$= \frac{\#A\,|A| + \#B\,|B|}{(\#A + \#B)(|A|+|B|)}$$

$$= \frac{\sum_{i=1}^{n} n_i \left|A_i\right|}{N \sum_{i=1}^{n} \min(n_i, 1)\left|A_i\right|}$$

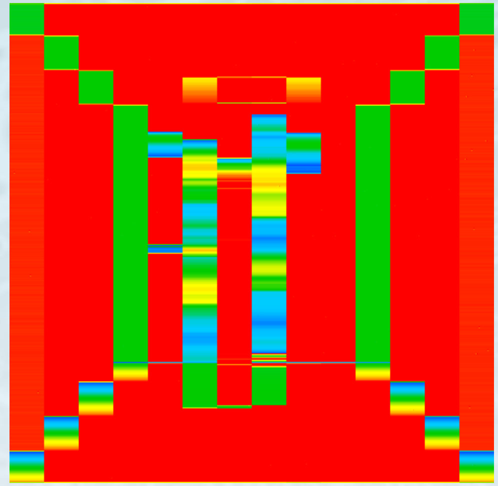
Execute *Both* A and B

A	A	A	A	A	A	A	A
B	B	B	B	B	B	B	B

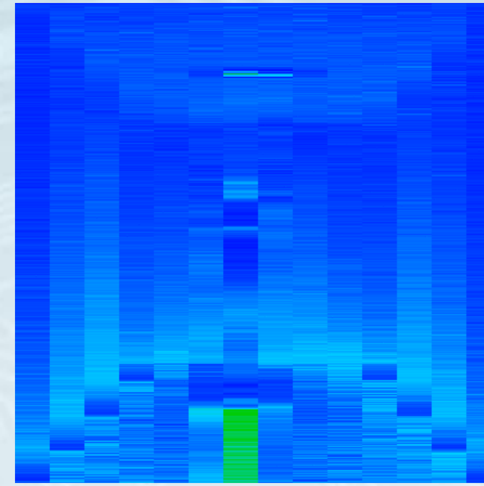
Mask on X

T	F	F	T	F	T	F	T
A	B	B	A	B	A	B	A

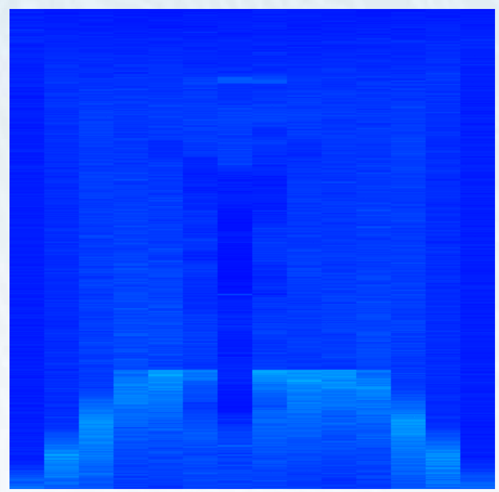
Shading Efficiency in a Path Tracer



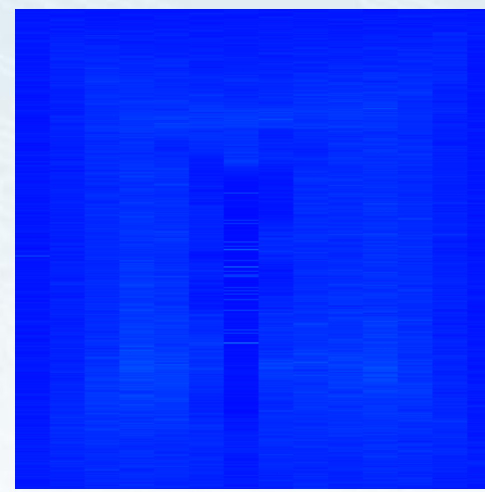
1st Hit: 98% Efficient



2nd Hit: 56% Efficient



3rd Hit: 52% Efficient



4th Hit: 54% Efficient

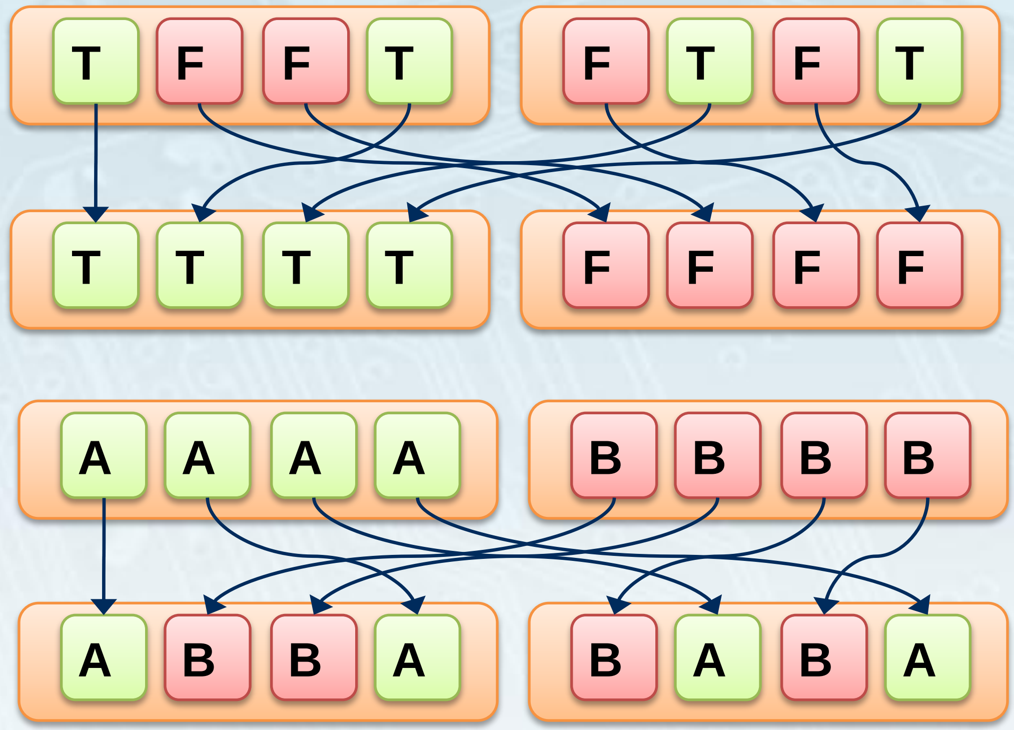
Recovering Coherence

Test X

Compact

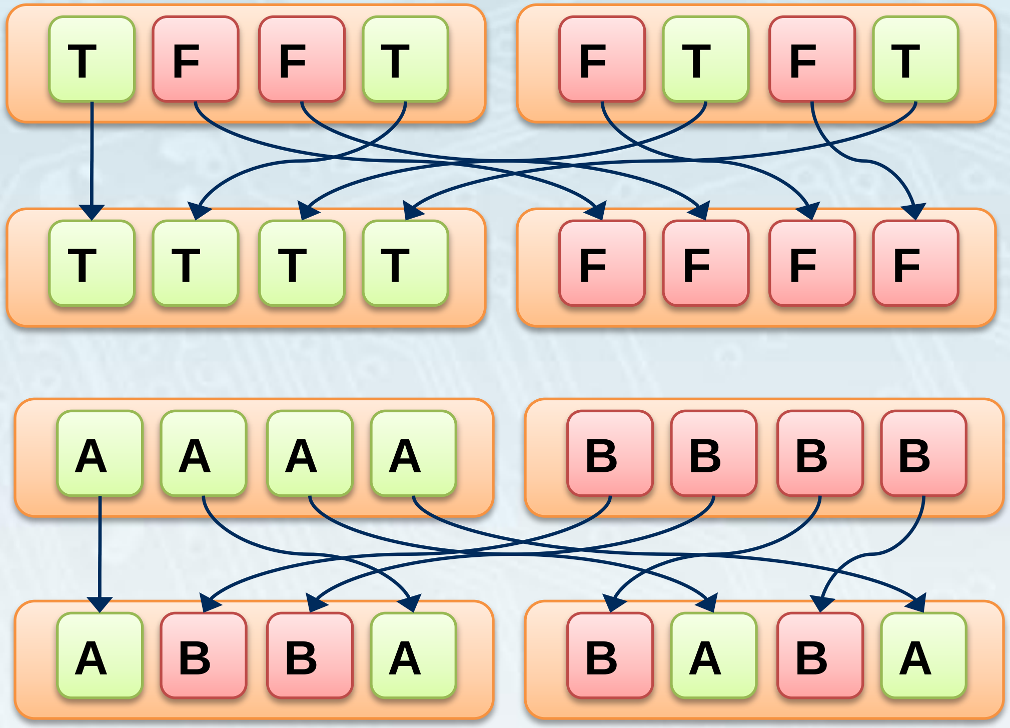
Evaluate A or B

Scatter result



Recovering Coherence

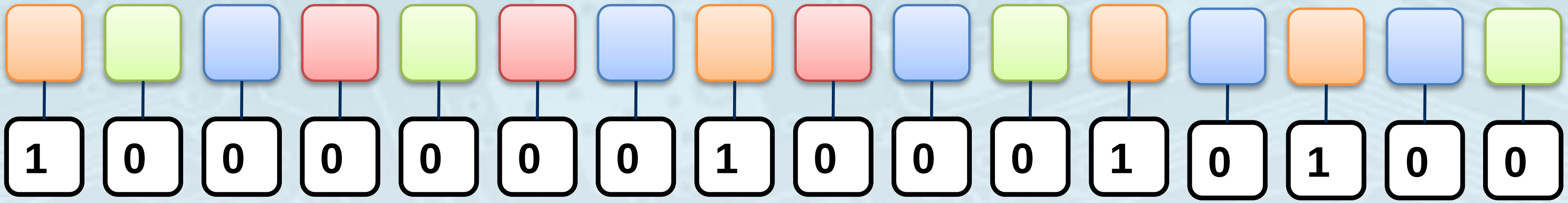
$$\text{Efficiency} = \frac{(A \mid B)}{(A \mid B) + \text{compact}}$$



Stream Compaction

- Compacting disorganized input

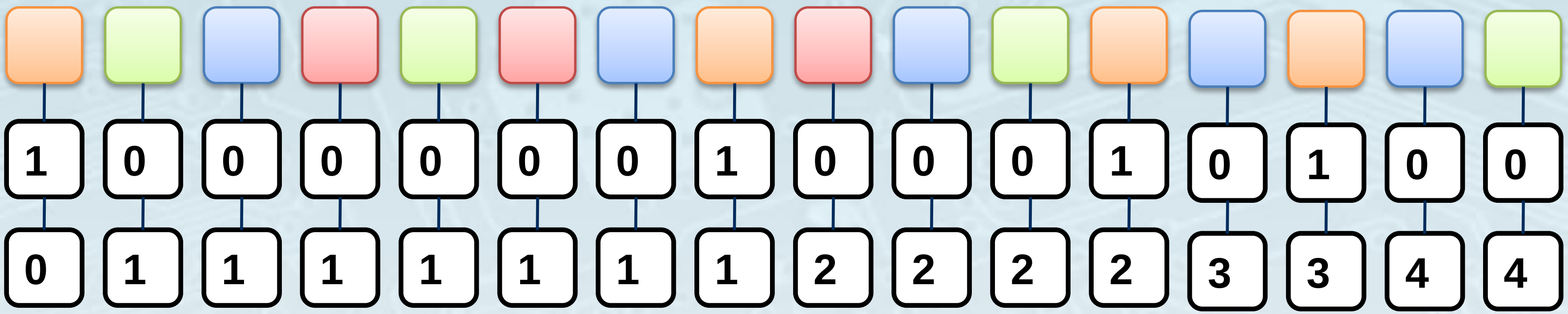
Stream Compaction



- Compacting disorganized input

1. Select orange token

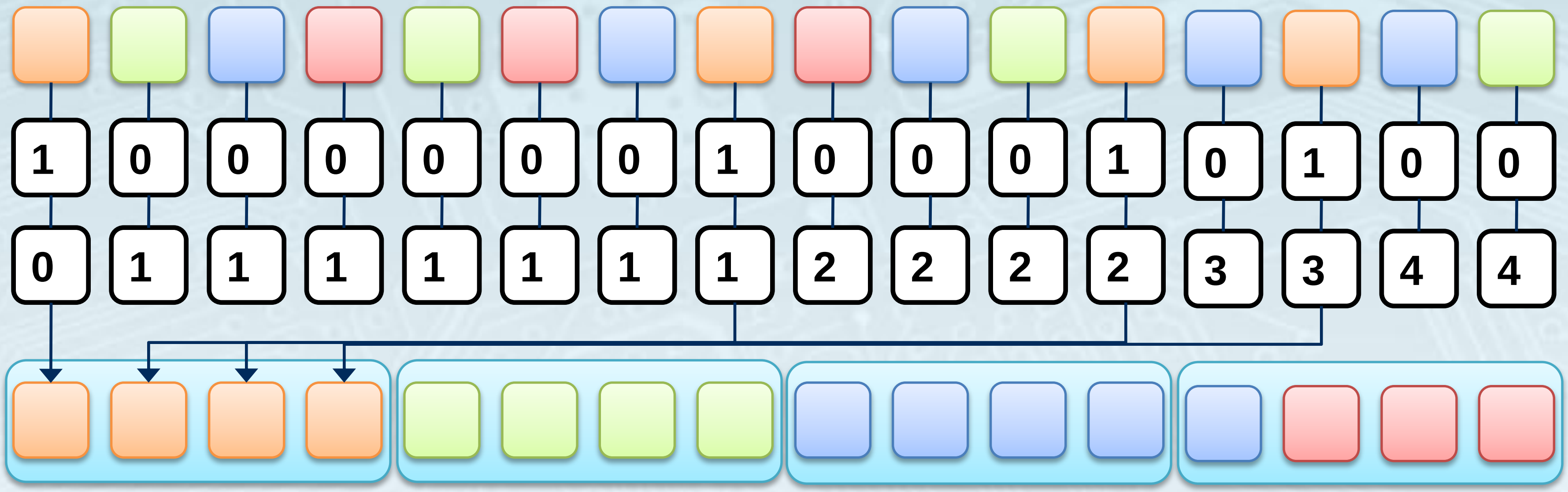
Stream Compaction



- Compacting disorganized input

1. Select orange token
2. Prefix Sum

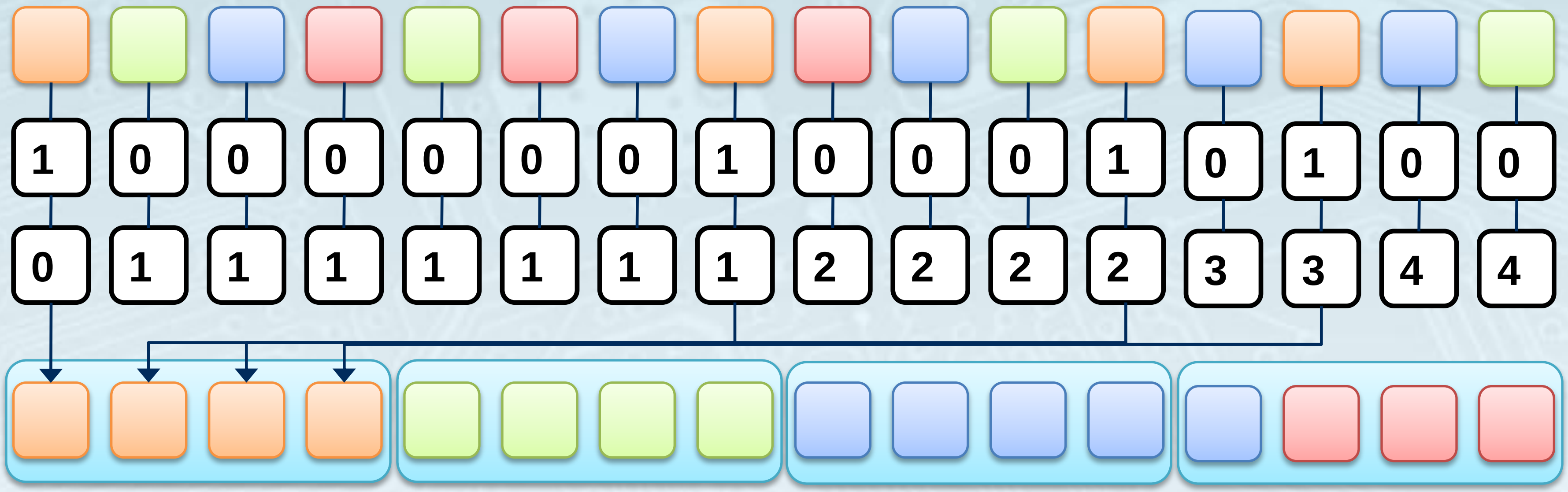
Stream Compaction



- Compacting disorganized input

1. Select orange token
2. Prefix Sum
3. Scatter

Stream Compaction



- Compacting disorganized input

1. Select orange token
2. Prefix Sum
3. Scatter

Scan: M*O(N)
[Sengupta et al. 2007]

Radix Sort: log(M)*O(N)
[Satish et al. 2009]

Shader Scheduling

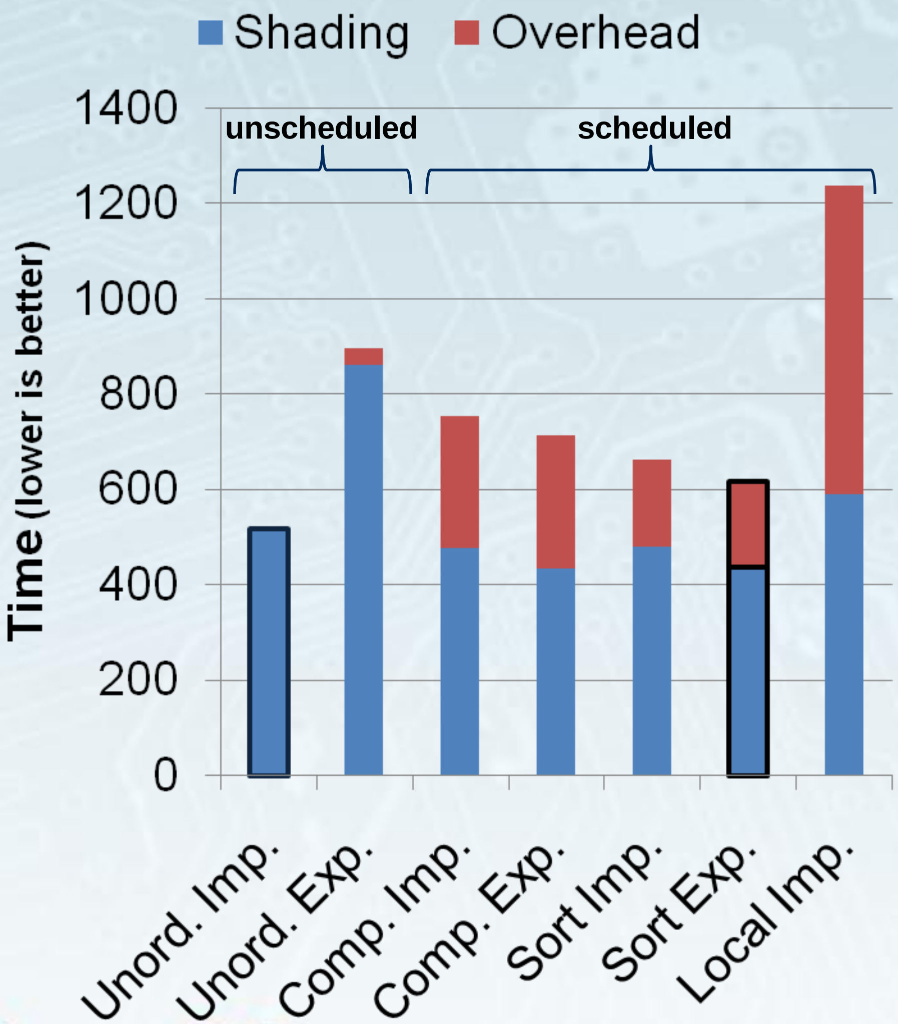
- **Implicit Serialization**
  - (Big Switch)
  - Let hardware schedule
- **Explicit Serialization**
  - Run only jobs w/same shader at a time
- Compact + Imp/Exp Serialization
- Radix Sort + Imp/Exp Serialization
- Local Bitonic Sort + Imp Ser.
  - Local to a CUDA thread block
  - Global loads coalesce

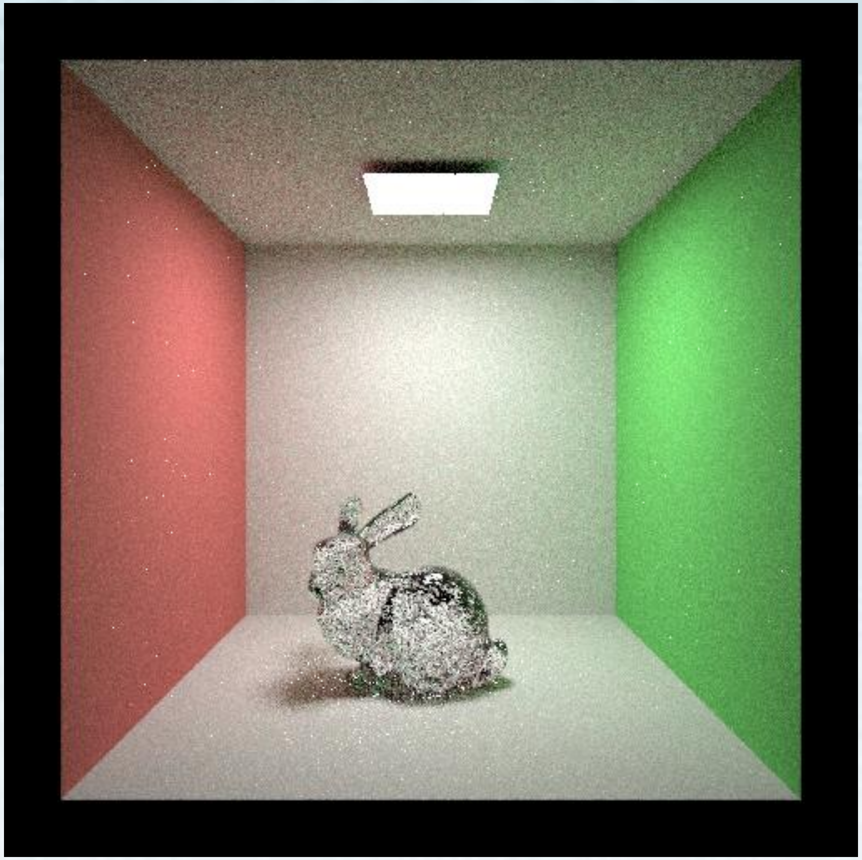
forall *j* **in** *jobs* **in SIMD do**
    switch *j* **do**
        case s_1**:**
            execute(s_1)
        …
        case s_M:
            execute(s_M)

forall *s* **in** *shaders* **do**
    mask = select(*s*,*jobs*)
    forall (*m*,*j*) **in** (*mask*,*jobs*)
                                    in SIMD do
            if *m* **then** execute(*s*)

Implemented in CUDA on G80-class hardware

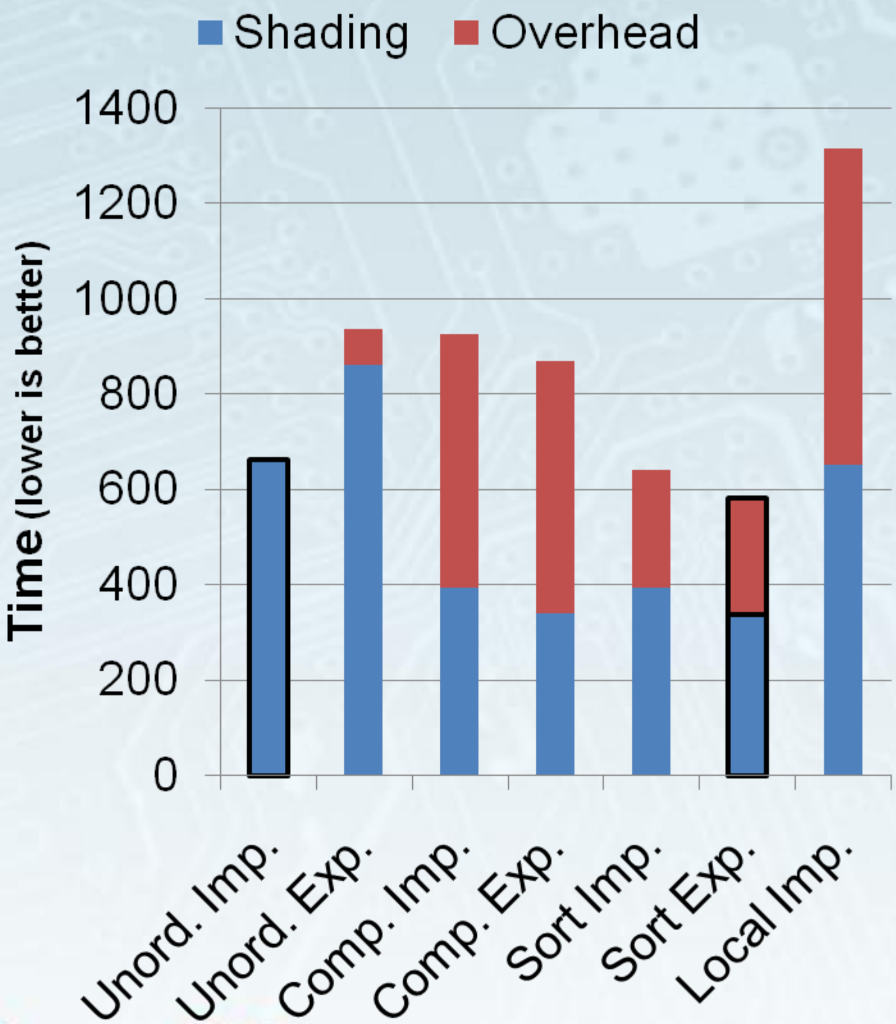
Results

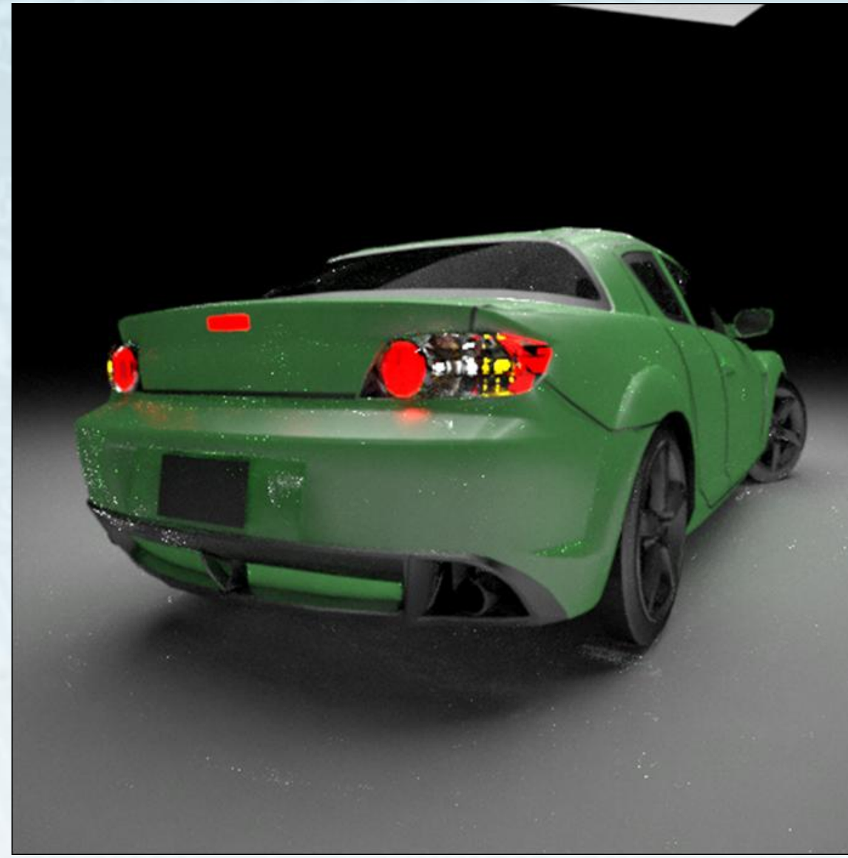




3 simple shaders
19% slower on GX2
5% slower on GTX+

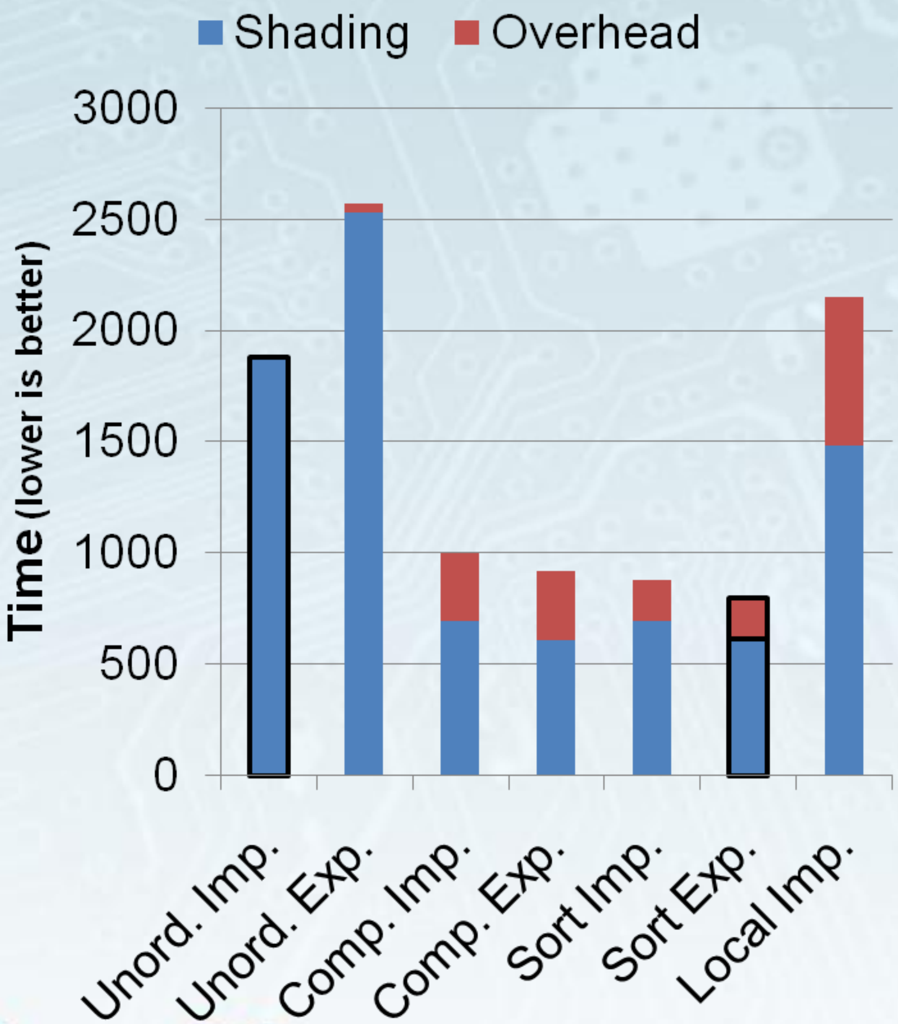
Results





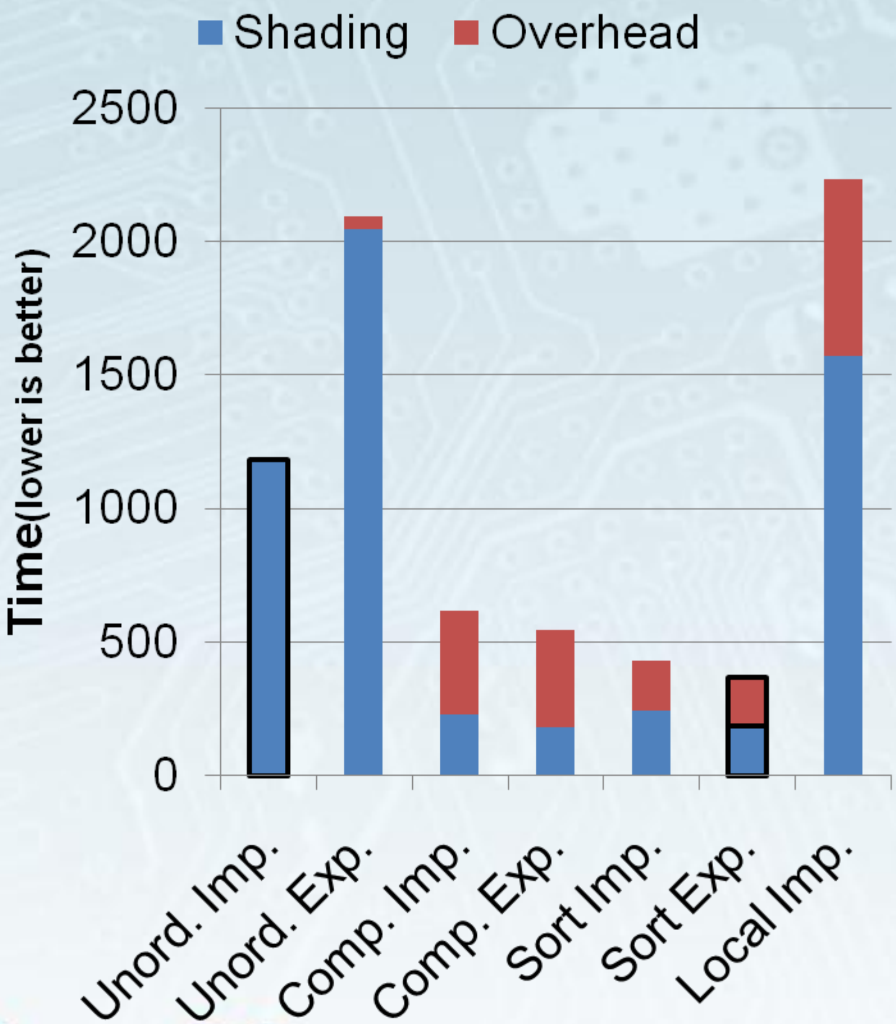
6 simple shaders
14% faster on GX2
38% faster on GTX+

Results



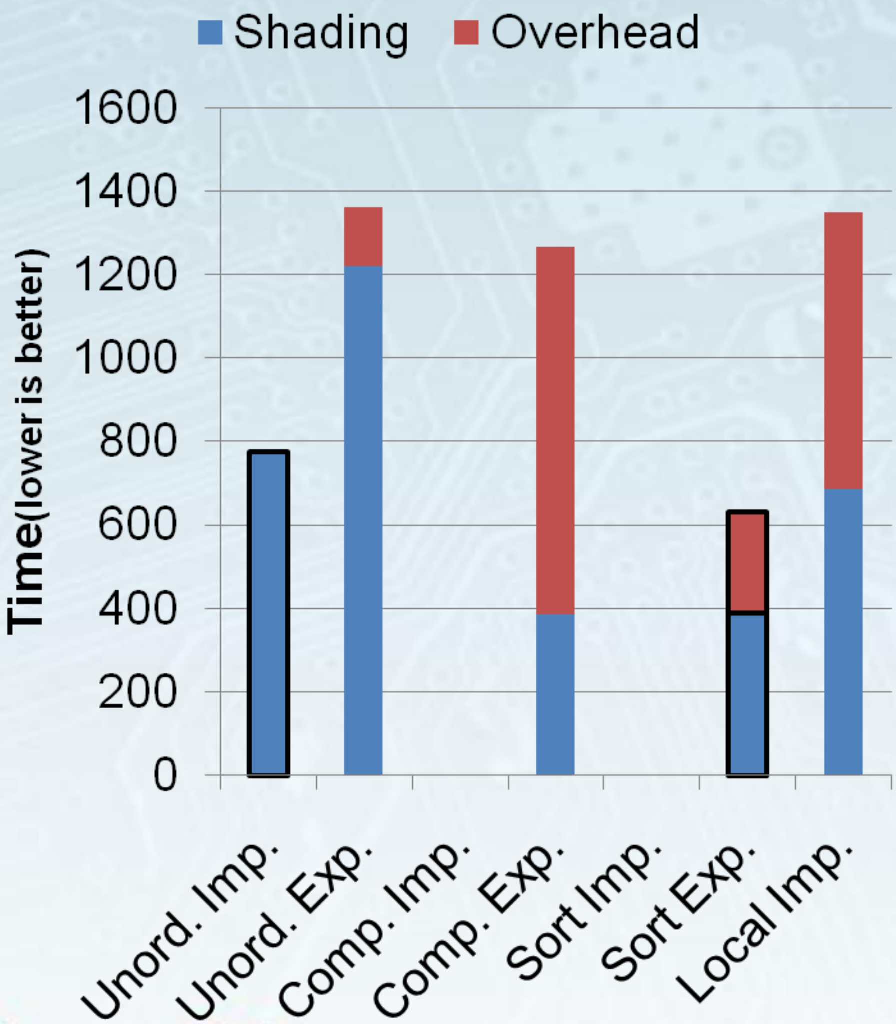
2 simple & 2 proc. shaders
2.4x faster on GX2
2.7x faster on GTX+

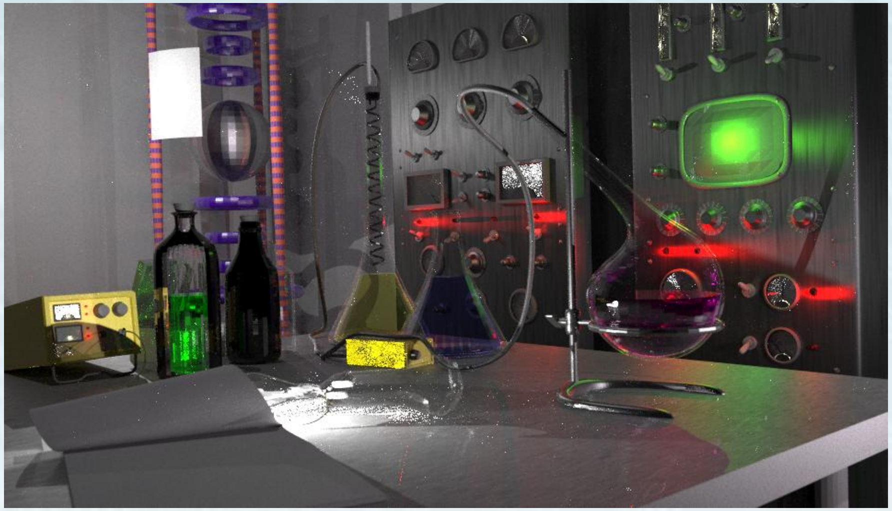
Results



3 simple & 2 proc. shaders
3.2x faster on GX2
3.5x faster on GTX+

Results





11 moderate shaders
23% faster on GX2
51% faster on GTX+

Scaling

	9800GX2	9800GTX+
“Cores”	2x16 (we used 16)	16
Processor Clock	1.50 GHz	1.84 GHz (23%)
Memory Clock	1 GHz	1.1 GHz (10%)
Bandwidth	64 GB/s	70.4 GB/s (10%)
Bus Width	2x256 bit (we used 256)	256 bit

- **Difference between processor clock scaling and memory bandwidth scaling enhances benefits of shader compaction**
- **Compaction further leverages increased processor speed**

Analysis

- Coherent shading time is always smaller
  - But cost of overhead not always worth it for simple cases
- Shader Complexity
  - Simple – No improvement, but little penalty
  - Procedural – Large improvements
- Implicit versus Explicit Serialization
  - With compaction, explicit almost always wins
  - Large penalties for explicit with unordered input
- Local compaction was never successful
  - Too much local data movement
  - Limited working set size

Conclusions

- Global stream compaction is almost always a win
- Surprising positive results for our toy scenes
- Production renderers will require stream compaction to be tractable in large scenes of arbitrary shading complexity

Future work

- Data sensitive scheduling to avoid memory divergence
- Hybrid shader batch approaches
- Scheduling in both space and time

Acknowledgments

- Thanks to Shubho Sengupta & Mark Harris for making their fast CUDA compaction primitives available in CUDPP, and Nathan Bell for GPU radix sort

- This work was funded by the Intel & Microsoft as part of the Illinois Universal Parallel Computing Research Center