GPU TECHNOLOGY CONFERENCE

ACCURATE FLOATING-POINT SUMMATION IN CUB

URI VERNER

Technion Israel Institute of Technology

NVIDIA Summer intern

OUTLINE

- Who needs accurate floating-point summation?!
- Round-off error: source and recovery
- A new method for accurate FP summation on a GPU
  - Added as a function to the open-source CUB library
- How fast is it?
- Download link

NORMAL FLOATING-POINT SUMMATION

INPUT	1.000000×10^{0}	-3.333333×10^{-1}	•••	$-2.467579 \times 10^{-19}$
RESULT	0.74999988	0.75000000		0.75000024

INACCURATE RESULTS

NON-DETERMINISTIC RESULTS!

ACCURATE FP SUMMATION

INPUT: 1.000000×10^{0} | -3.333333×10^{-1} | ... | $-2.467579 \times 10^{-19}$

EXACT SUM: 0.74999999082897... ✓

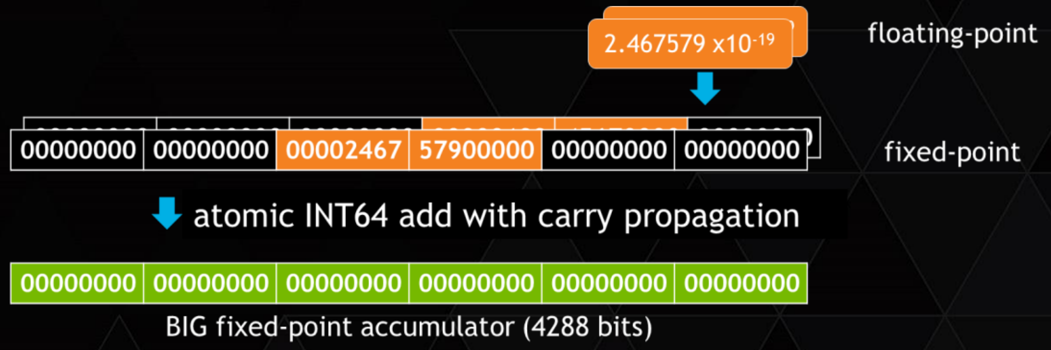
ACCURATE SUM AS FLOATING-POINT: 0.7500000 ✓

Our method computes both!

GPU TECHNOLOGY CONFERENCE

EXISTING WORK: EXBLAS (OPENCL)

- By Iakymchuk, Collange, et al.
- Uses Kulisch accumulators (very wide fixed-precision variables)
- Our method uses a different approach



WHERE IS THIS USEFUL?

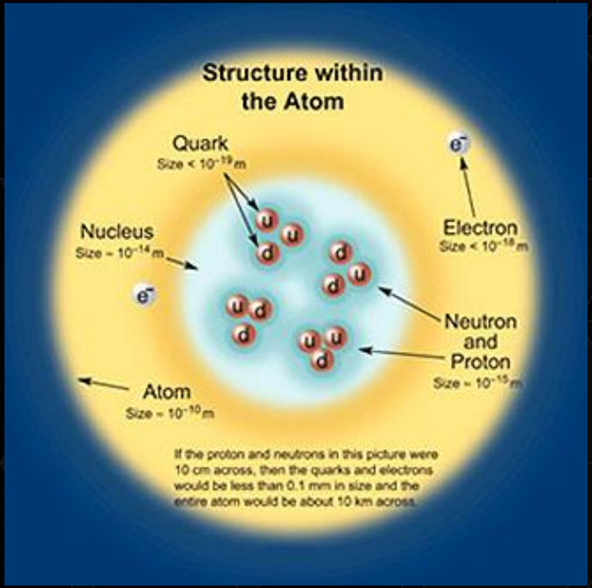
- High Performance Computing applications
  - An example coming next
- Cross-platform applications
- Debugging: bit-exact results for floating point!

```
if (d_result != d_reference)
    error("wrong answer!");
```

EXAMPLE: LATTICE QCD COMPUTATIONS

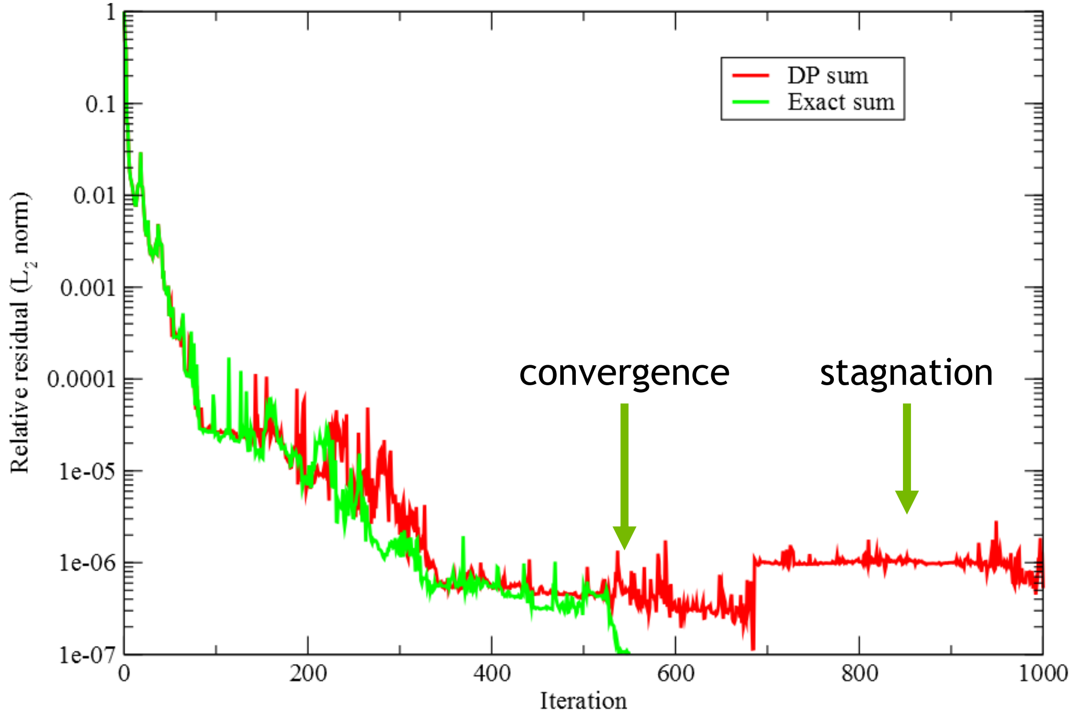
- QCD - Quantum Chromodynamics
- Describes the strong force that binds quarks and gluons

- GPU accelerated QUDA library
  lattice.github.io/quda
- Accurate summation can potentially improve convergence and reduce computation time



CONVERGENCE OF ITERATIVE ALGORITHM

BiCGstab Algorithm: Dirac equation solver



ROUND-OFF ERROR: SOURCE AND RECOVERY

IEEE-754 FLOATING-POINT STANDARD

32/64 bits

S	EXP	SIGNIFICANT DIGITS

$$x = (-1)^{s} \times 2^{EXP} \times 1.D_b$$

	single-precision	double-precision
Format width	32 bits	64 bits
Exponent range	-126 .. 127 (8 bits)	-1022 .. 1023 (11 bits)
Significant digits	23 (+1 implicit)	52 (+1 implicit)

- Special cases:

  +/-NaN, +/-Inf, +/-0, subnormals

SOURCE OF NON-REPRODUCIBILITY

Non-associative operations

Order of operations matters:

1,000,000 + (0.4 + 0.4) -> 1,000,001

(1,000,000 + 0.4) + 0.4 -> 1,000,000

different implementations return different sum values

SOURCE OF ACCURACY LOSS

Round-off error in compute operations

Computer sum: 1234.567 + 1.234567

	accurate	actual
	1235.801567	1235.802

bigger difference in magnitude => more digits lost

TWO-SUM ALGORITHM (KNUTH)

TwoSum(a,b)

Round(a+b)

round-off error

6 FP operations

```
[s,r] = TwoSum(a,b)
 s <- a+b
 z <- s-b
 r <- (b-(s-z)) + (a-z)
```

FAST TWO-SUM (DEKKER)

FastTwoSum(a,b)

Round(a+b) round-off error

3 FP operations

Requires EXP(a) $\geq$ EXP(b)

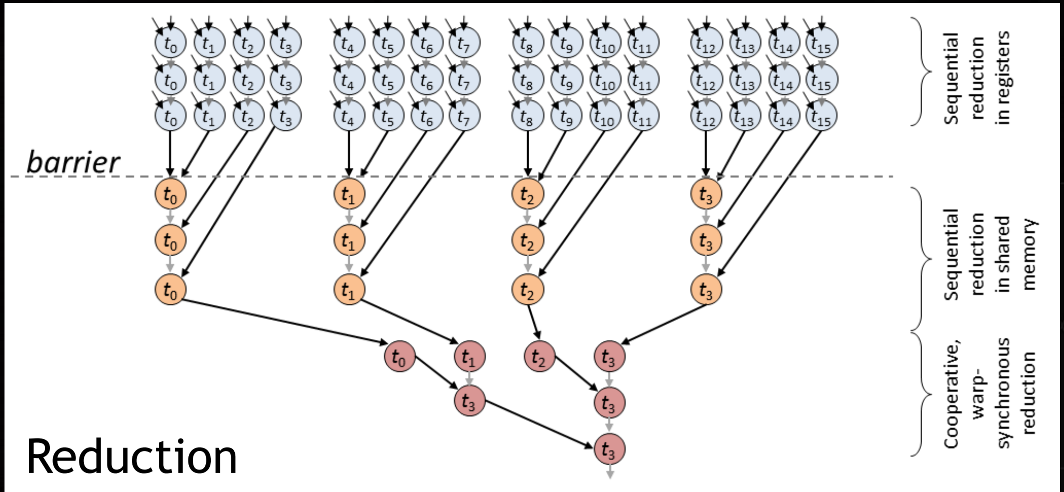
```
[s,r] = FastTwoSum(a,b)
 s <- a+b
 z <- s-a
 r <- b-z
```

ERROR-FREE PARALLEL SUMMATION

GPU TECHNOLOGY CONFERENCE

INTEGRATION INTO CUB LIBRARY

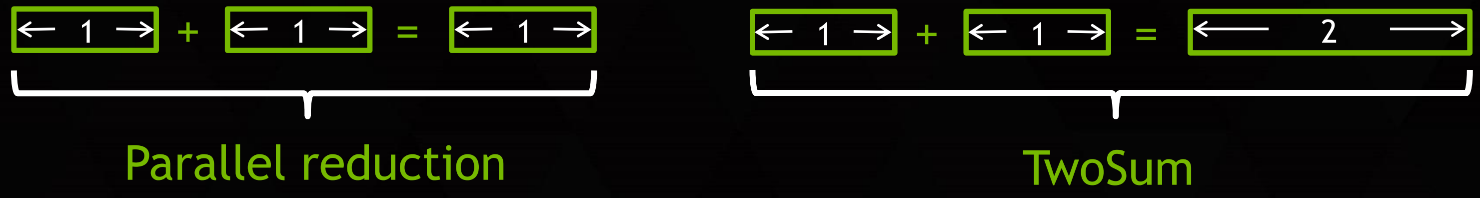
- CUB: Parallel primitives in CUDA
- Includes parallel primitives like Sum, Scan, Sort, etc.
- Performance tuned for every NVIDIA GPU architecture



Aim: use Reduction with TwoSum() for an error-free sum

REDUCTION+TwoSum: PROBLEM #1

The output of TwoSum is two FPs, instead of one!

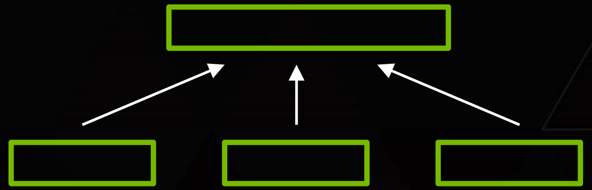



```
(x1,x2)=2Sum(s1,s2)
(y1,y2)=2Sum(r1,r2)
x2=x2+y1
(x1,x2)=fast(x1,x2)
x2=x2+y2
(s3,r3)=fast(x1,x2)
```

1. Convert x → (x, 0.0)
2. Define (s1,r1) + (s2,r2) → (s3,r3)

REDUCTION+TwoSum: PROBLEM #2

- Limited accuracy
- E.g.: $10^{100} + 10^{0} + 10^{-100}$



Multiple values with similar exponents can be added without overflow

DIVIDE THE EXPONENT RANGE INTO BINS

Number of EXP values: 2048 (double)							
0	1	2	3	4	5	6	7
s r	s r	s r	s r	s r	s r	s r	s r

add to bin #3

3 (bin id)

$$\text{floor}\left(1023 \,/\, \frac{2048}{8}\right) = 3$$

0	1023	0101010101010101

HOW MANY EXPONENT VALUES PER BIN?

1. 000000000000000000000 1 1 00000000000000000000000

106 binary digits

- Suppose we add n numbers to a bin: $a_i = 2^{e_i} \cdot m_i$, where $e_l \leq e_i < e_h$.
- Our budget is 106 digits
- $106 \geq 53 + \lceil log_2 n \rceil + (e_h - e_l)$
- For $n = 2^{20}$, $(e_h - e_l) = 32$ different exponents!

$a_l =$ 1 00000000000111111111

$a_h =$1 11111111111111111111 000000000

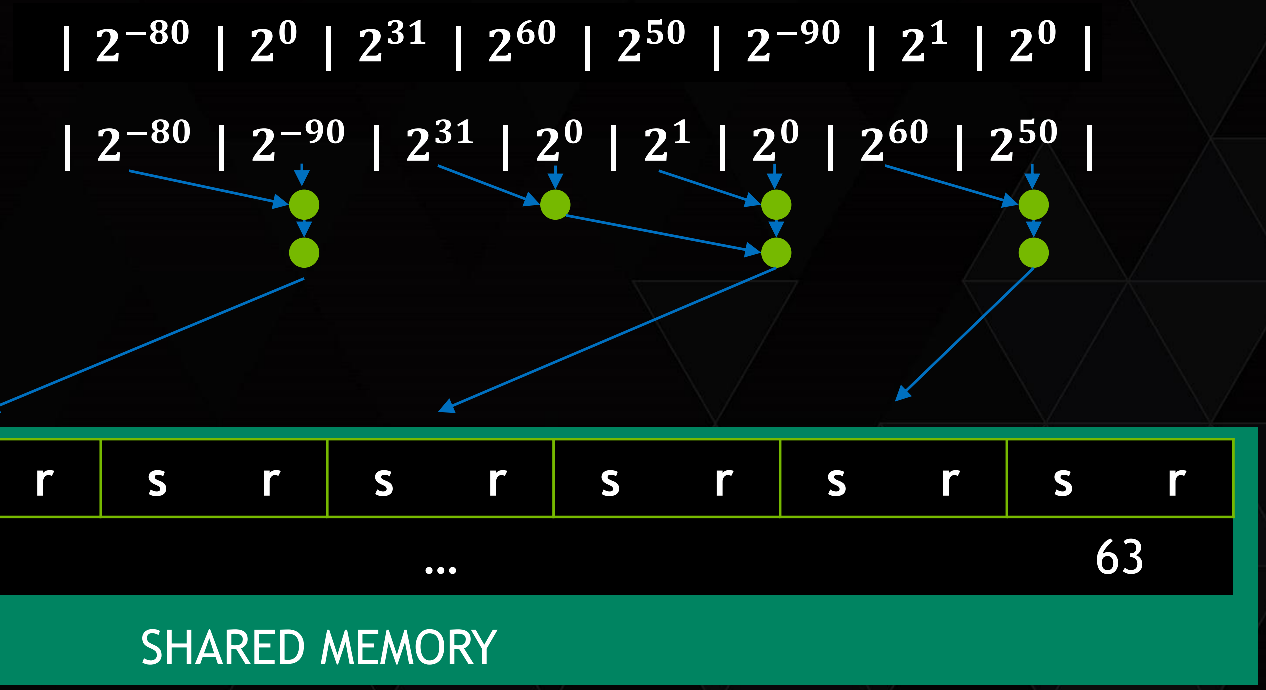
$e_h - e_l$

ALGORITHM: ERROR-FREE SUMMATION ON GPUS

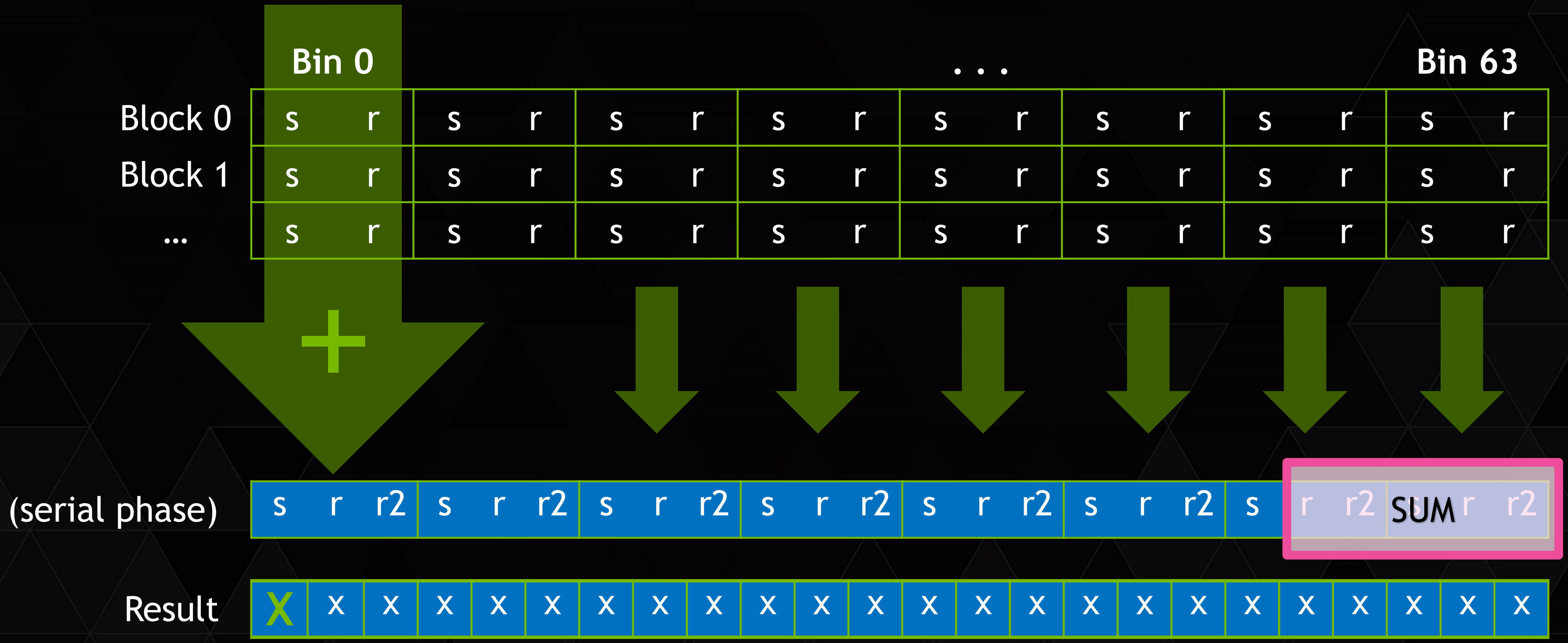
EACH THREAD DOES THE FOLLOWING

EXPONENT=11bit	
binidx=6bit	bin=5bit

- read input
- radix-sort by binidx
- reduce-by-key binidx
- update smem bins



FINAL SUMMATION PHASE



ALGORITHM SUMMARY

1. For each thread block:

    Repeat:

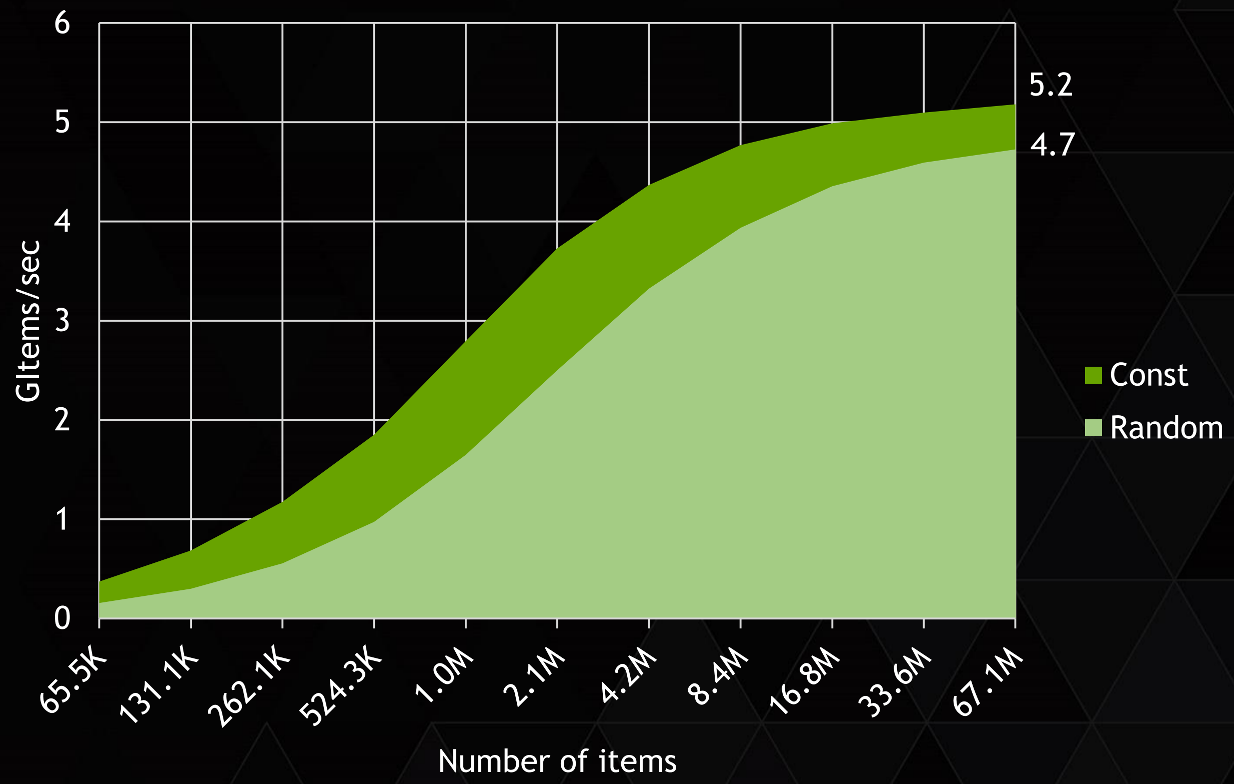
Read input tile	in registers
Radix-sort items by bin ID	in registers (+ temp buffer in shared)
Compute sum for each bin with Reduce-by-key	in registers (+ temp buffer in shared)
Update bins in shared memory	in shared memory
Save bins to global memory	in global memory

2. Merge bins with the same bin ID
3. “Normalize” bins by adding them from low to high
4. Rounded result is in the highest word

GPU TECHNOLOGY CONFERENCE

PERFORMANCE (K40)

5 Billion Items per second



Normal summation is ~6 times faster

DOWNLOAD AND CONTRIBUTE

- Get it at:

  https://github.com/uriv/accusum

- Usage instructions in README.ACCUSUM

- It's open source. Use it, improve it!

GPU TECHNOLOGY CONFERENCE

THANK YOU

JOIN THE CONVERSATION

#GTC15

BACKUP