NVIDIA®

UC DAVIS

IDAV Institute for Data Analysis and Visualization

CUDPP
CUDA Data-Parallel Primitives Library

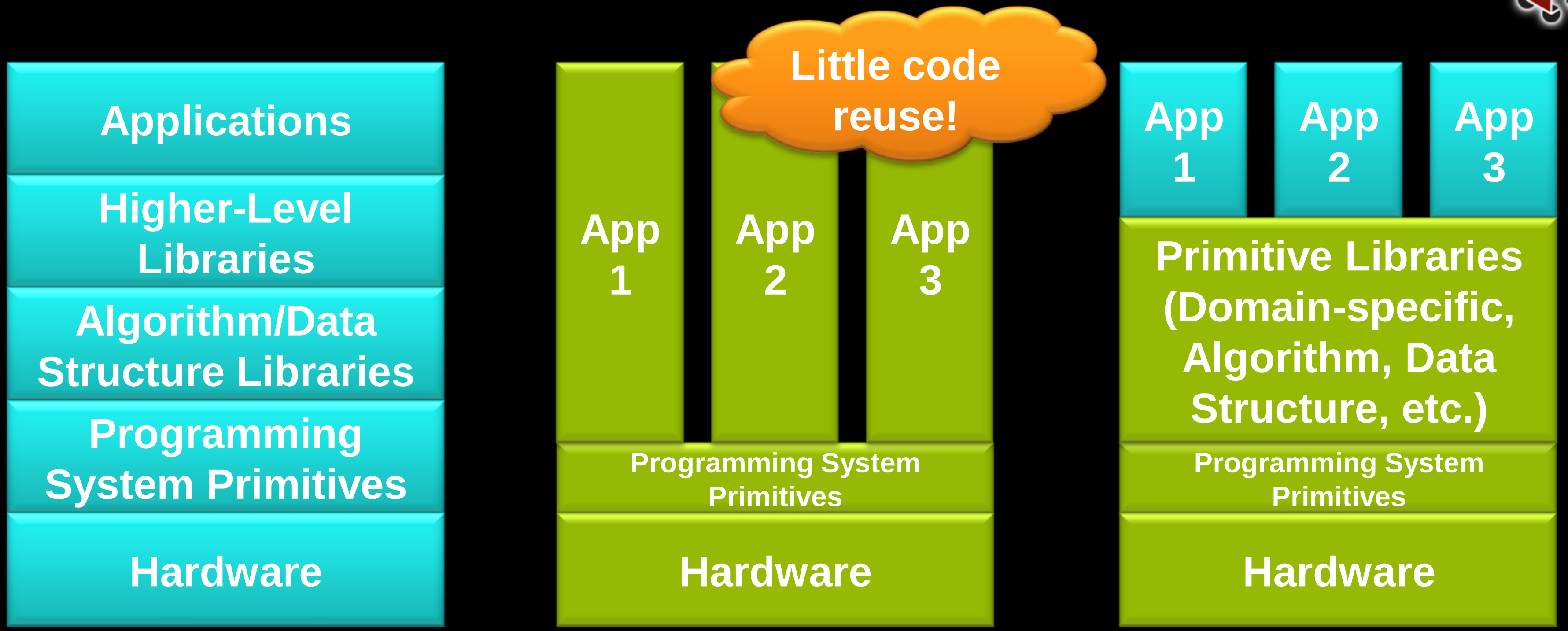
Data Parallel

- The GPU is a data-parallel processor
  - Many cores, thousands of parallel threads
  - Thousands of data elements to process
  - All data processed by the same program
    - SIMT computation model (i.e. threads may diverge)
  - Contrast with task parallelism and ILP
- Best results when you “Think Data Parallel”
  - Design algorithms for data parallelism
  - Understand parallel algorithmic complexity and efficiency
  - Use data-parallel algorithmic primitives as building blocks: CUDPP

Challenge: Libraries

- What are the…
  - …fundamental parallel algorithms?
  - …fundamental parallel data structures?
  - …methods to bring them together?
- Goal: library of fundamental parallel primitives and algorithms
  - With best-in-class performance and efficiency
  - For data-parallel (many-core) GPUs
- Result: CUDPP

Horizontal vs. Vertical Development



CPU

GPU (Historical)

GPU (Our Goal)

CUDPP

- Library of high-performance parallel primitives for GPUs
  - Written in C for CUDA
  - Runs on all CUDA-capable GPUs (100M+ shipped)
  - Support for Windows, Linux, and OS X
- Collaboration between UC Davis and NVIDIA
  - John Owens (UC Davis)
    - Shubho Sengupta, Yao Zhang, Andrew Davidson, Stanley Tzeng
  - Mark Harris (NVIDIA)
- http://code.google.com/p/cudpp

CUDPP

- Current in CUDPP:
  - (Segmented) scan, stream compact
  - Radix sort, sparse matrix-vector multiply
  - Random number generation
- In progress:
  - Parallel reduction, more sorts, graphs, trees
- Open Source under BSD License
- http://code.google.com/p/cudpp

CUDPP Design Principles

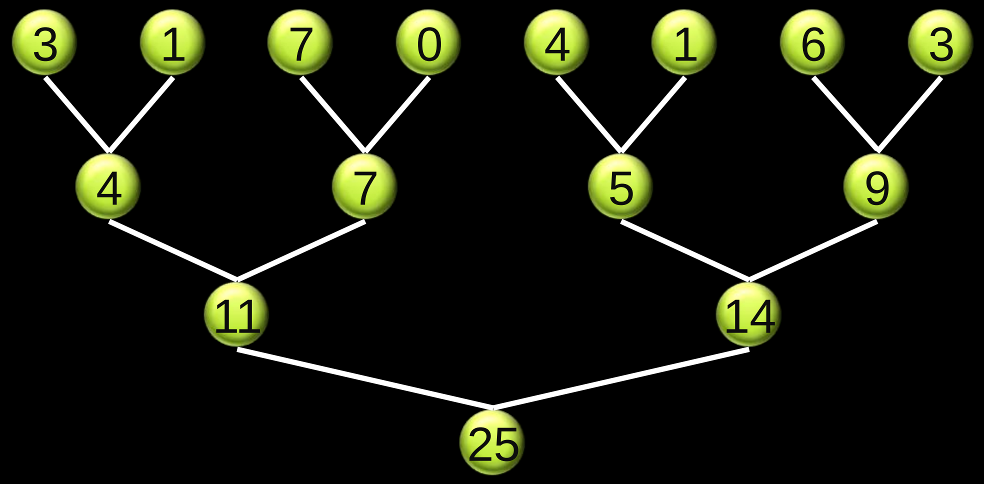
- Performance
  - Provide fundamental primitives with best-of-class performance
- CUDPP functions run on the GPU on GPU data
  - CUDPP doesn’t handle allocation or data transfers
- Modularity
  - Easily include primitives in applications
  - Library can be linked to other applications
  - Code from the multiple abstraction levels can be re-used (e.g. kernels, or cta-level __device__ functions, in addition to library-level calls)

Common Situations in Parallel Computation

- Many parallel threads need to generate a single result value
  - Reduce
- Many parallel threads that need to partition data
  - Split
- Many parallel threads and variable output per thread
  - Compact / Expand / Allocate

Parallel Reductions

- Common Data Parallel Operation
- **Reduce** vector to a single value
- Operator: +, *, min/max, AND/OR
  - Binary associative operators
- Tree-based implementation



Split Operation

- Given an array of true and false elements (and payloads)

Flag	T	F	F	T	F	F	T	F
Payload	3	1	7	0	4	1	6	3

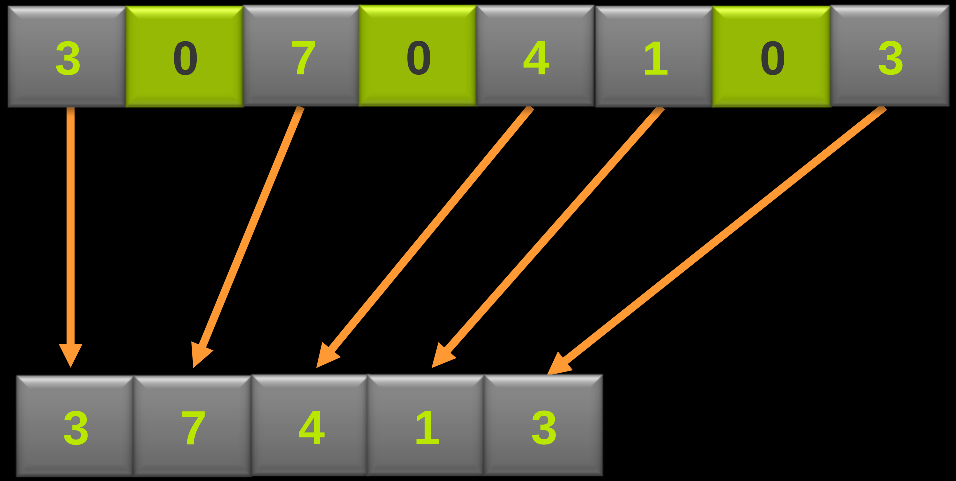
- Return an array with all true elements at the beginning

T	T	T	F	F	F	F	F
3	0	6	1	7	4	1	3

- Examples: sorting, building trees

Variable Output Per Thread: Compact

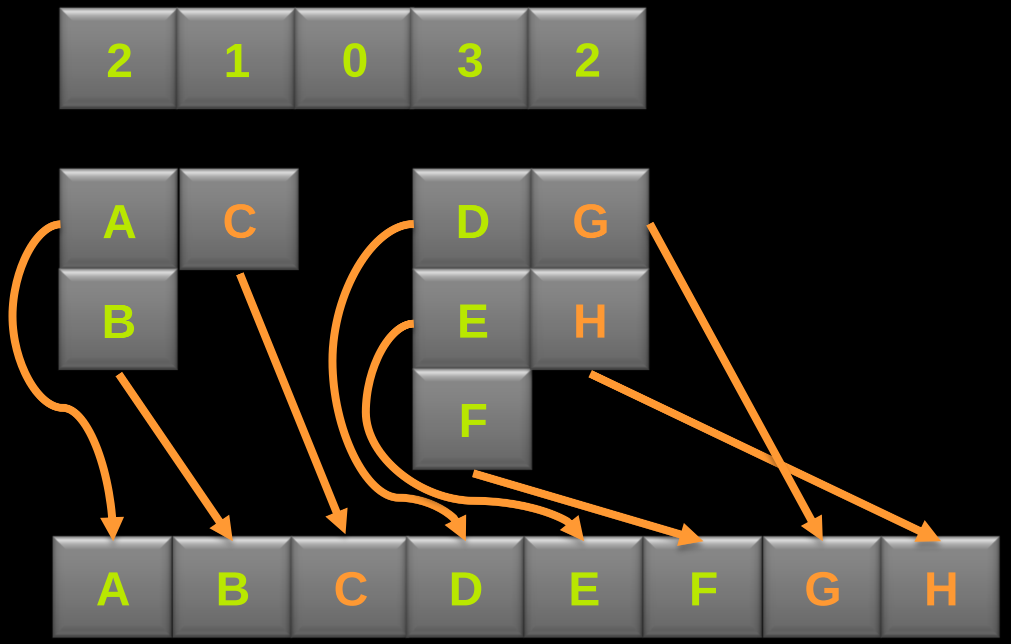
- Remove null elements



- Example: collision detection

Variable Output Per Thread: General Case

- Allocate Variable Storage Per Thread



- Examples: marching cubes, geometry generation

“Where do I write my output?”

- In all of these situations, each thread must answer that simple question
- The answer is:

  “That depends (on how much the other threads need to write)!”
- “Scan” is an efficient way to answer this question in parallel

Parallel Prefix Sum (Scan)

- Given an array $A = [a_0, a_1, \ldots, a_{n-1}]$ and a binary associative operator $\oplus$ with identity I,

$$\text{scan}(A) = [I, a_0, (a_0 \oplus a_1), \ldots, (a_0 \oplus a_1 \oplus \ldots \oplus a_{n-2})]$$

- Example: if $\oplus$ is addition, then scan on the set

[3 1 7 0 4 1 6 3]

returns the set

[0 3 4 11 11 15 16 22]

Scan Literature

Pre-GPU

- First proposed in APL by Iverson (1962)
- Used as a data parallel primitive in the Connection Machine (1990)
  - Feature of C* and CM-Lisp
- Guy Blelloch used scan as a primitive for various parallel algorithms
  - *Blelloch, 1990, "Prefix Sums and Their Applications"*

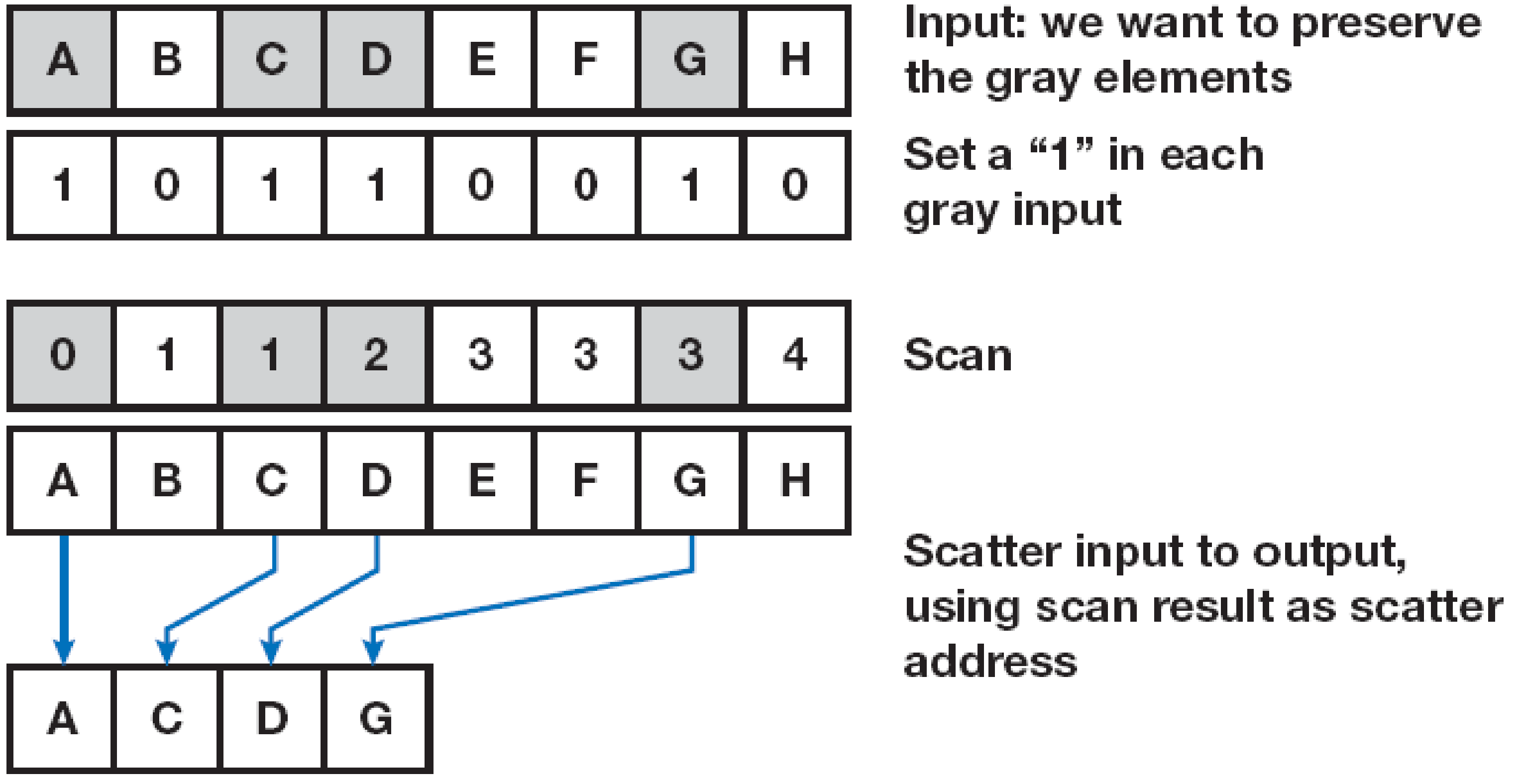
Post-GPU

- O(n log n) work GPU implementation by Daniel Horn (GPU Gems 2)
  - Applied to Summed Area Tables by Hensley et al. (EG05)
- O(n) work GPU scan by Sengupta et al. (EDGE06) and Greß et al. (EG06)
- O(n) work & space GPU implementation by Harris et al. (2007)
- Scan and segmented scan by Sengupta et al. (GH07)
- Vector-based (segmented) scan by Dotsenko et al. (ICS08)
- Warp-based (segmented) scan by Sengupta et al. (NV Tech Report 08 – used in CUDPP)

Applications of Scan

- Scan is a simple and useful parallel building block for many parallel algorithms:
  - radix sort
  - quicksort (segmented scan)
  - String comparison
  - Lexical analysis
  - Stream compaction
  - Run-length encoding
  - Polynomial evaluation
  - Solving recurrences
  - Tree operations
  - Histograms
  - Allocation
  - Etc.
- Fascinating, since scan is unnecessary in sequential computing!

Application: Stream Compaction



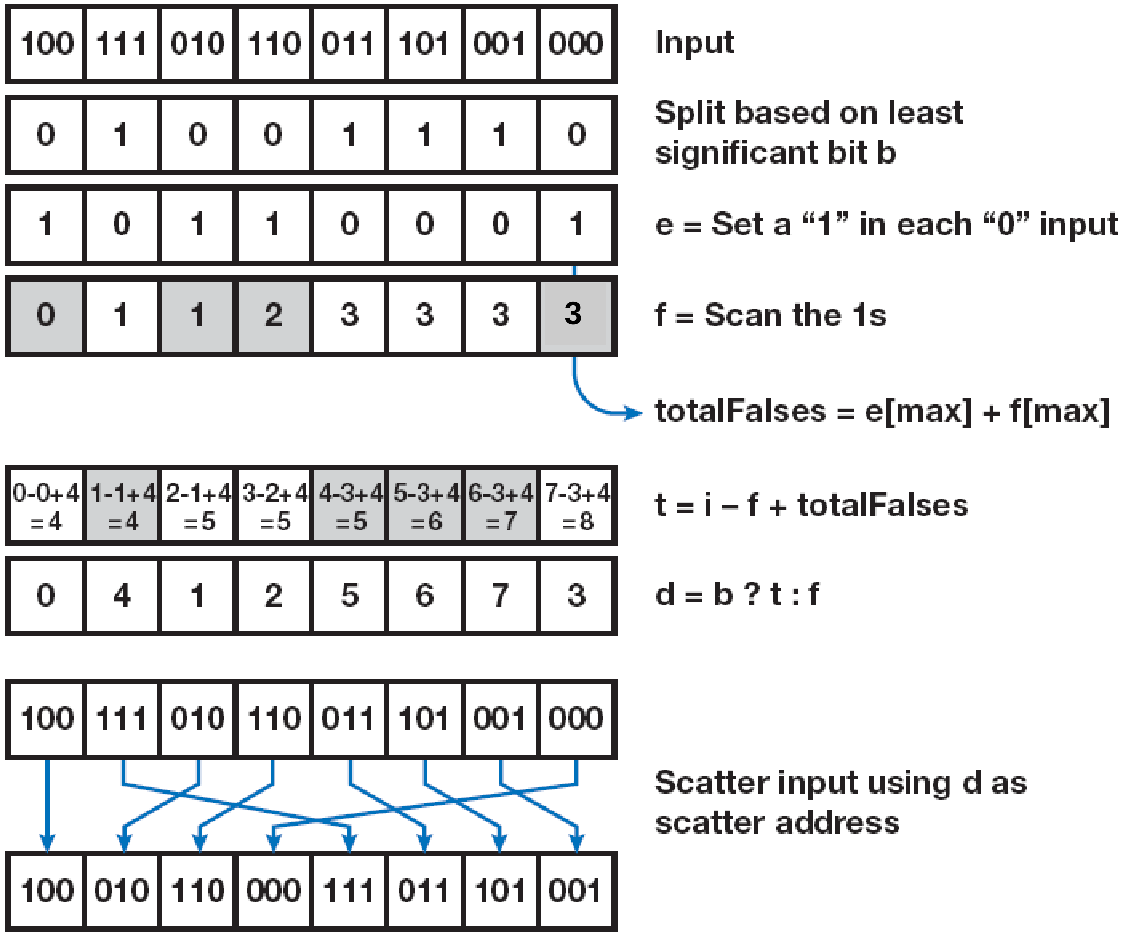
1M elements:
~0.6-1.3ms

16M elements:
~8-20ms

Perf depends on # elements retained

Harris, M., S. Sengupta, and J.D. Owens. "Parallel Prefix Sum (Scan) in CUDA". *GPU Gems 3*

Application: Radix Sort



- Perform split operation on each bit using scan
- Can also sort each block and merge
  - Slower due to cost of merge
- CUDPP Radix sort similar, but more sophisticated & efficient
  - See Satish et al. 2009

CUDPP Radix Sort Perf (from Satish et al. 09)

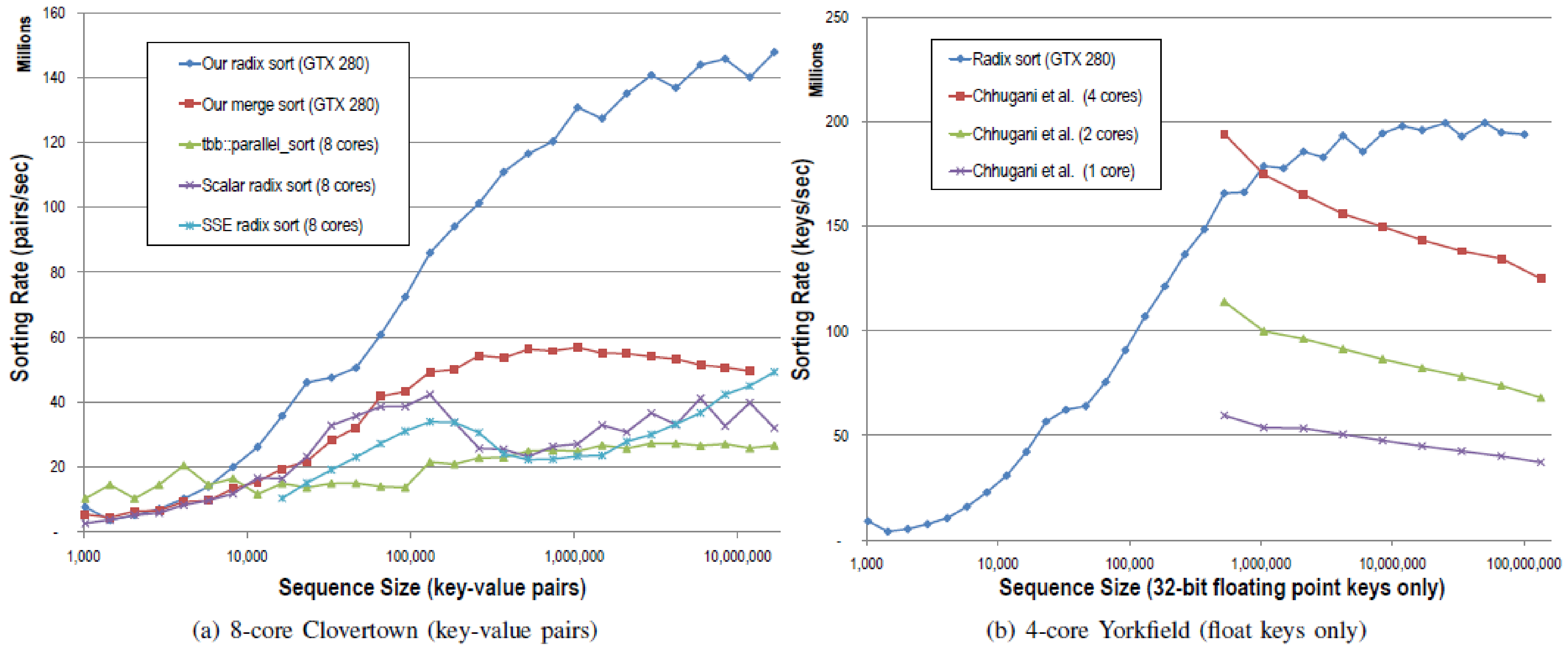


Fig. 8. Performance comparison with efficient multicore sort implementations.

N. Satish, M. Harris, and M. Garland. "Designing Efficient Sorting Algorithms for Manycore GPUs". Proc. IPDPS 2009

CUDPP Radix Sort Performance

- **Fastest published GPU sorting algorithm**
  - **"CUDPP radix sort" here is *old* radix sort**

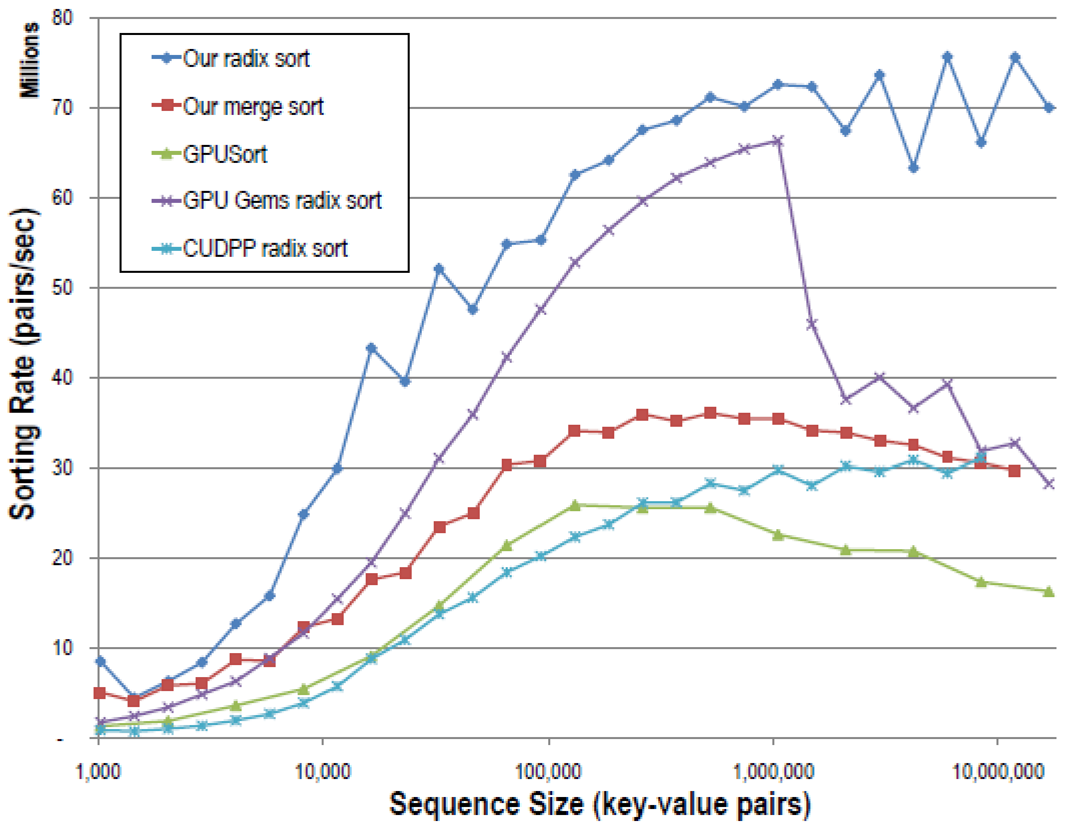


Fig. 7. Sorting rates for several GPU-based methods on an 8800 Ultra.

N. Satish, M. Harris, and M. Garland. "Designing Efficient Sorting Algorithms for Manycore GPUs". Proc. IPDPS 2009

Application: Summed Area Tables

- Each pixel in SAT is the sum of all pixels below and to the left
- Can be used to perform box filter of arbitrary radius per pixel in constant time
  - Crow, 1984
  - Hensley, 2006 (O(n log n) scan)

- Easy to compute with scan
  - Scan all rows, then all columns
  - Transpose in between and scan only rows
  - GPU can scan all rows in parallel

- Scan all rows of 1024x1024 image in 0.85 ms
  - Build summed area table in 3.5 ms
  - 6 scans, transpose, (de)interleave RGBA

Segmented Scan

Segment Head Flags	0	0	1	0	0	1	0	0
Input Data Array	3	1	7	0	4	1	6	3
Segmented scan	3	1	7	0	4	1	6	3

- **Segmented scan enables another class of parallel algorithms**
  - **Parallel quicksort**
  - **Parallel sparse matrix-vector multiply in CSR format**

- **Sengupta, S., M. Harris, Y. Zhang, and J.D. Owens.**
  "Scan Primitives for GPU Computing". ***Proceedings of Graphics Hardware 2007***
- **Sengupta, S., M. Harris, M. Garland. "Efficient parallel scan algorithms for GPUs".**
  NVIDIA Technical Report NVR-2008-003, December 2008

CUDPP Impact

- CUDPP used for multiple research projects
  - At UC Davis, NVIDIA, and elsewhere
- 15+ research papers (and counting) published that use CUDPP
  - http://cudpp.googlecode.com/svn/trunk/cudpp/doc/html/cudpp_refs.html
  - Increasing number of papers using CUDPP that CUDPP developers didn’t know about until publication
- Provides template for what good libraries should provide
  - Not just code but documentation, examples, unit tests, performance tests, etc.
- CUDPP 1.1 2000+ downloads

Related Libraries: Thrust

- Thrust: CUDA parallel algorithms C++ template library
  - Many of the same algorithms included in Thrust and CUDPP
  - Different design goals:
    - Thrust designed for programmer productivity
    - CUDPP designed for high performance
  - Code using Thrust must be compiled with NVCC
    - CUDPP functions can be called from code compiled by other compilers, and even code written in other languages
  - Thrust has many container classes that ease handling of CPU-GPU shared data

- http://code.google.com/p/thrust

Related Libraries: cusp

- A CUDA library for sparse linear algebra graph computations
- CUSP uses highly optimized sparse matrix-vector multiplication code
  - Likely more efficient than CUDPP for these operations

- http://code.google.com/p/cusp-library/

UC Davis Sponsors

- UC Davis CUDPP efforts supported by:
  - Department of Energy
    - Early Career Principal Investigator Award DE-FG02-04ER25609
    - SciDAC Institute for Ultrascale Visualization
    - Los Alamos National Laboratory
  - National Science Foundation (grant 0541448)
  - Hardware donations from NVIDIA