| | |
|---|---|
| Title: | Parallel Algorithms and Patterns |
| Author(s): | Robey, Robert W. |
| Intended for: | Report |
| Issued: | 2016-06-16 |

# Parallel Algorithms and Patterns

Bob Robey

June 16th, 2016

## Parallel Algorithms and Patterns

- **Parallel Algorithms**
  - A well-defined, step-by-step computational procedure that emphasizes concurrency to solve a problem
  - Examples of problems include: Sorting, searching, optimization, matrix operations
- **Parallel Patterns**
  - A computational step in a sequence of independent, potentially concurrent operations that occurs in diverse scenarios with some frequency
  - Examples are: Reductions, prefix scans, ghost cell updates

**We only touch on parallel patterns in this presentation. It really deserves its own detailed discussion which Gabe Rockefeller would like to develop. See references, McCool, et al. and Mattson, et al. (on last slides).**

## Future of Algorithm Research

- Algorithm research still focuses on serial operation, algorithmic complexity, and work efficiency
- Lets look at the flaws in this traditional approach to algorithm development, with an emphasis on parallel architectures, including next-generation architectures
- The new considerations, in no particular order, follow in the next slides

## New Considerations for Algorithms

- **Exposing Concurrency**
  - We have a partnership with the hardware/system developers. Our highest mission is to expose concurrency in every way possible.
- **Fine-grained Parallelism – Forall loops concept**
- **Step Efficiency**
  - I can do eight operations for the cost of one – how do I can I exploit this? I may do more work, but fewer steps
- **Cache friendly**
  - A linear search can be just as fast as a much more “efficient” algorithm, just because it uses cache better
  - Cache-oblivious algorithms are a hot topic replacing earlier auto-tuning approaches such as Automatically Tuned Linear Algebra Software, ATLAS
- **Load Balancing**
- **Reproducibility**

## New Considerations for Algorithms(continued)

- **Branch Free**
  - Branching imposes a high performance penalty and in particular for single instruction, multiple data (SIMD) architectures
  - Also related, comparison free or lower thread divergence
- **Recursion free**
- **Numerical Intensity – loop fusion, blocking, tiling, functional**
- **Reduce Data Movement**
- **Energy Efficiency**
- **Locality**
  - Space-Filling Curves
  - Cache-oblivious Data Structures
- **Asynchronous or Overlapping Compute/[Communication, I/O, Memory]**

## Three Case Studies

- Parallel Global Sum
- Spatial Hashing
- Prefix Scan

## Parallel Global Sum

- **Running on different numbers of processors yields different summation results, e.g. total mass**
- **Gets worse as problem size gets larger**
- **Problem is due to finite precision arithmetic not being associative**
  - Sorting data or always summing in a particular order would solve this, but the cost is too high to be practical
  - So we assumed we would just have to live with the problem

***But, it can be solved by viewing it as a precision problem***

- **Breakthrough approaches emerged in about 2010**
- **Use double-double sums such as Kahan or Knuth Sums and custom data types and operations in MPI_Allreduce**
- **In Search of Numerical Consistency, Robey, Robey, Aulwes, Parallel Computing, 2010 and following**
- **UC Berkeley research – reproBlas library and rigorous mathematical proofs for edge cases – see P. Ahrens**
- **High Precision software Libraries -- D. Bailey**
- **exBLAS library by Roman Iakymchuk**

## Spatial Hashing

- **Comparison Sort**
  - Sort a room by having each person compare to their neighbor and move left if their last name is earlier in the alphabet and right if later.
  - Continue for log n steps, where n is the number of people in the room
  - When you reach the end of the row, it is like you have reached the GPU workgroup and you have to exit the kernel to do a comparison with the next row.
- **Hash Sort**
  - Place a table for each letter at the front of the room
  - Each person goes to the table with the first letter of their last name (approaches 1 step for enough tables and additional letters, 2nd, 3rd, etc.)

# Hash Work Acknowledgements

- Remaps and Hierachical Hashing (Breadcrumbs)
  - ➢ Gerald Collom and Colin Redman
- Compact Hash
  - ➢ Rebecka Tumblin, Peter Ahrens, and Sara Hartse
- Perfect Hash and Unstructured Mesh Hashing
  - ➢ Rachel Robey and David Nicholaeff
- CLAMR mini-app, Cell-based Adaptive Mesh Refinement (AMR) on GPUs
  - ➢ Neal Davis, David Nicholaeff and Dennis Trujillo
- Open Source:
  - ➢ Github.com/losalamos/PerfectHash
  - ➢ Github.com/losalamos/CompactHash
  - ➢ Github.com/losalamos/CompactHashRemap
  - ➢ Github.com/losalamos/CLAMR

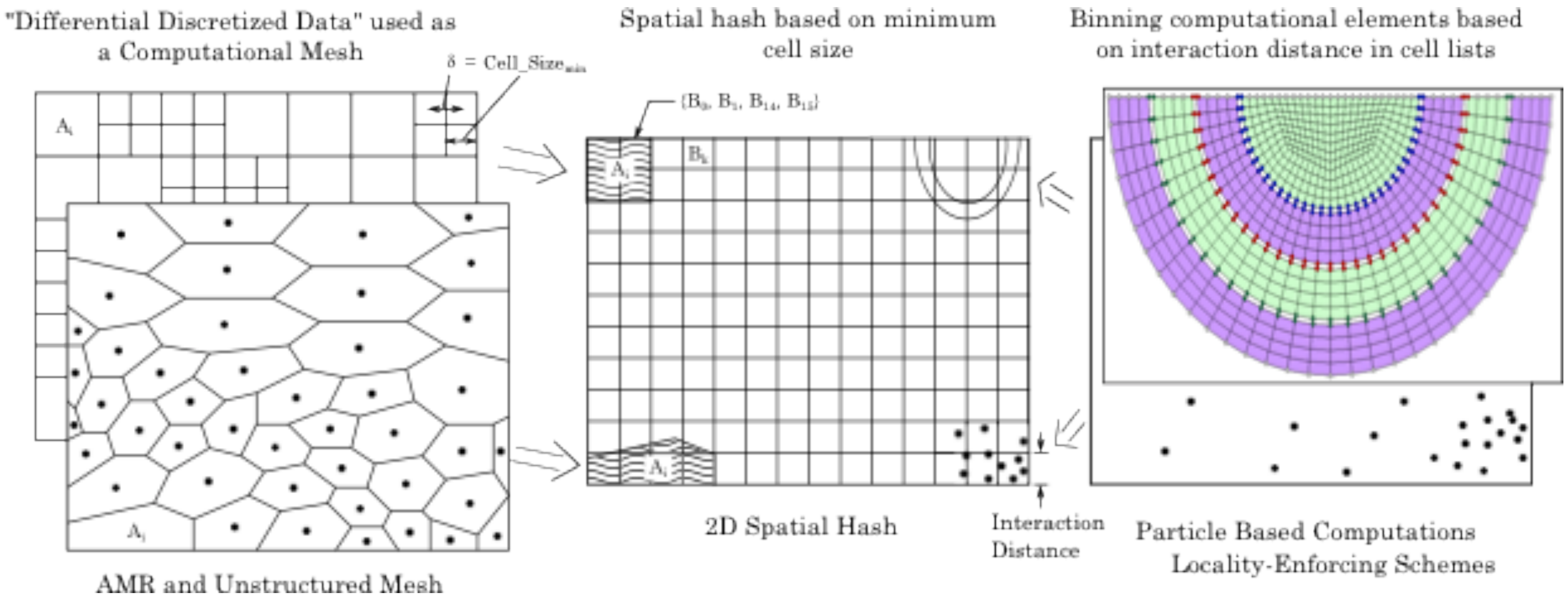
Slide 9

## Spatial Hashing

- Exploiting the properties of a computational mesh -- only one cell at each physical location in space and well-spread out
- *Perfect Hash* -- if we can guarantee that "bins" only contain one item
- *Compact Hash* – if we might have more than one item in a bin, maybe because it is more complex or we want to use a smaller hash table
- Spatial operations – sort, neighbor finding, remap and table lookup all show significant speedup and are easier to port to the GPU

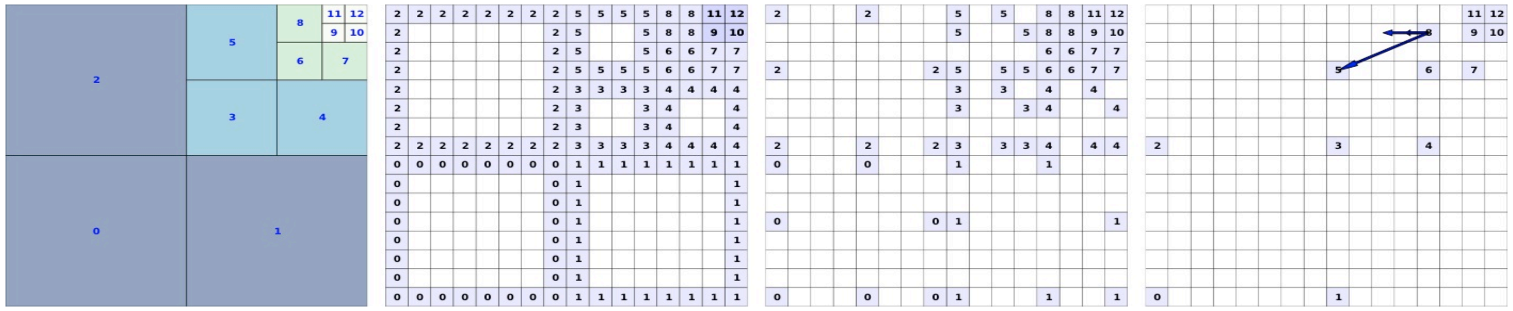
- With a few months work, our hash sort beat the fastest sort (quicksort) on the CPU by a factor of 4 and the fastest GPU sort (radix) by a factor of 3.

# Project 1D or 2D, Particles, Objects, AMR or Unstructured Mesh to Hash


"Differential Discretized Data" used as a Computational Mesh
$\delta$ = Cell_Size$_{min}$
$A_i$
AMR and Unstructured Mesh
Spatial hash based on minimum cell size
{$B_0$, $B_1$, $B_{14}$, $B_{15}$}
$B_k$
2D Spatial Hash
Interaction Distance
Binning computational elements based on interaction distance in cell lists
Particle Based Computations
Locality-Enforcing Schemes

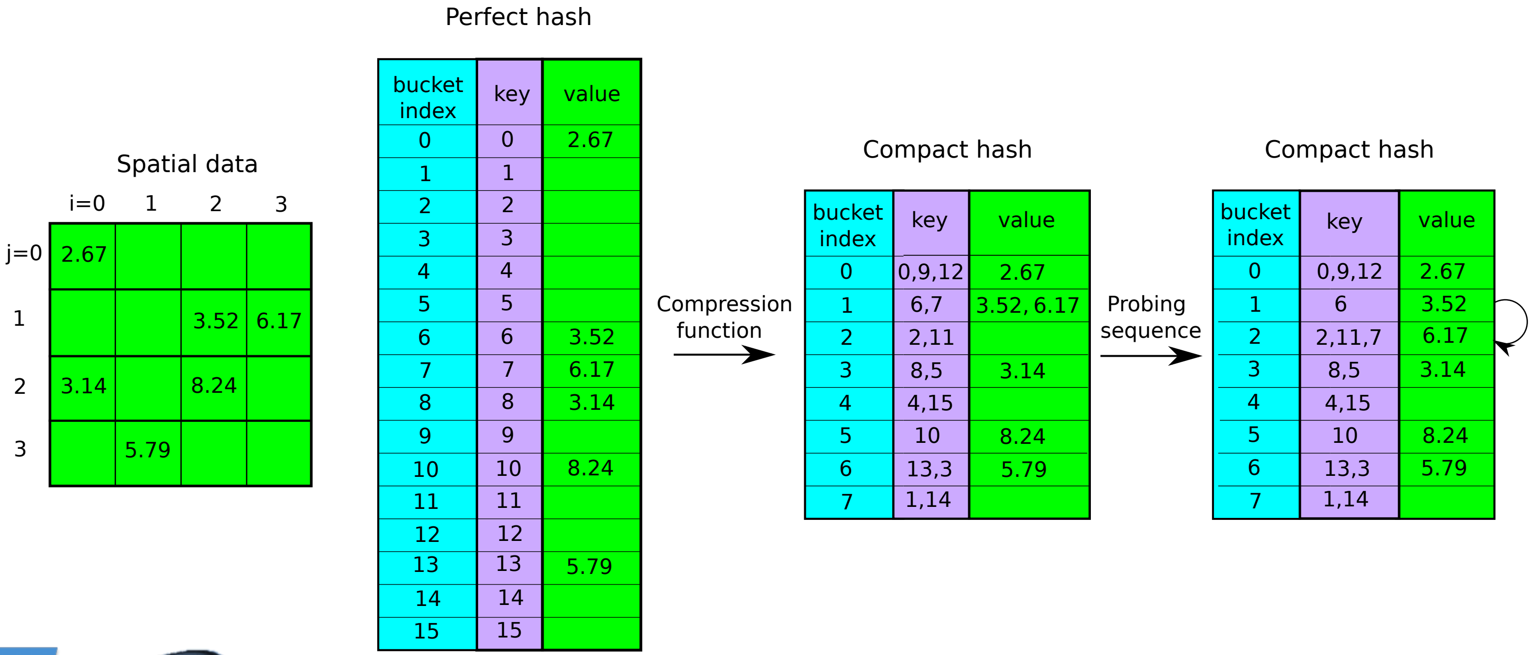
## Hashing Methodology: Reducing Reads & Writes

- Mesh cells project their topological information (i.e. cell index) into a bin in a “hash table” based on spatial information. Other cells than look spatially nearby.
- Cells do not need to communicate directly with other cells for any topological information, i.e. no comparison operations… all operations take place in the hash table
  - Writing and reading to the hash table, are analytic operations
- Initial implementation wrote to all underlying cells. Shown left to write are implementations that reduce the number of reads and writes



# Hashing Methodology – Compact Hashing

We take the sparse hash table (a linear array) and compress it. If a value is already in the bin, we move down to the next empty bucket and put it there. Both the key and the value must be placed in the bin so that the queries can check for the proper key and continue looking until it is found.



Spatial data

| | i=0 | 1 | 2 | 3 |
|---|---|---|---|---|
| j=0 | 2.67 | | | |
| 1 | | | 3.52 | 6.17 |
| 2 | 3.14 | | 8.24 | |
| 3 | | 5.79 | | |

Perfect hash

| bucket index | key | value |
|---|---|---|
| 0 | 0 | 2.67 |
| 1 | 1 | |
| 2 | 2 | |
| 3 | 3 | |
| 4 | 4 | |
| 5 | 5 | |
| 6 | 6 | 3.52 |
| 7 | 7 | 6.17 |
| 8 | 8 | 3.14 |
| 9 | 9 | |
| 10 | 10 | 8.24 |
| 11 | 11 | |
| 12 | 12 | |
| 13 | 13 | 5.79 |
| 14 | 14 | |
| 15 | 15 | |

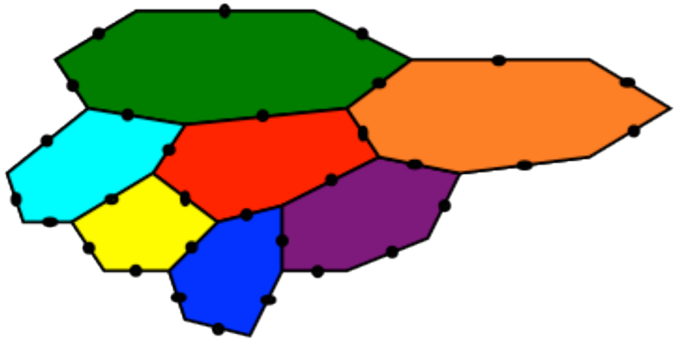
Compact hash

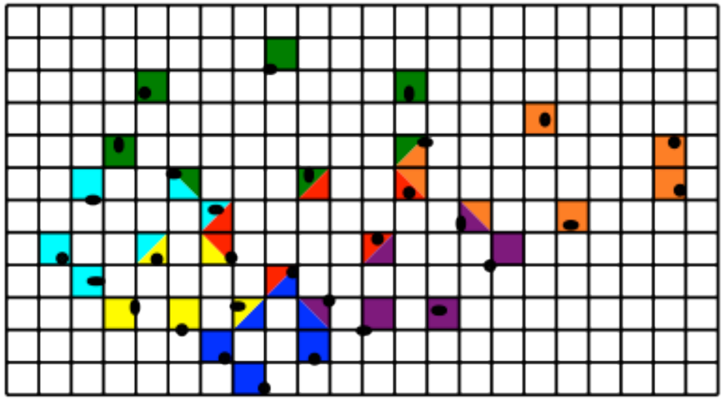
| bucket index | key | value |
|---|---|---|
| 0 | 0,9,12 | 2.67 |
| 1 | 6,7 | 3.52, 6.17 |
| 2 | 2,11 | |
| 3 | 8,5 | 3.14 |
| 4 | 4,15 | |
| 5 | 10 | 8.24 |
| 6 | 13,3 | 5.79 |
| 7 | 1,14 | |

Compact hash

| bucket index | key | value |
|---|---|---|
| 0 | 0,9,12 | 2.67 |
| 1 | 6 | 3.52 |
| 2 | 2,11,7 | 6.17 |
| 3 | 8,5 | 3.14 |
| 4 | 4,15 | |
| 5 | 10 | 8.24 |
| 6 | 13,3 | 5.79 |
| 7 | 1,14 | |

## Hashing for Unstructured Meshes





1. Every cell writes its cell number into the bin at the center of each face. If the face is to the left and up from the center it writes its index to the first of two places in the bin, else it writes to the second place.

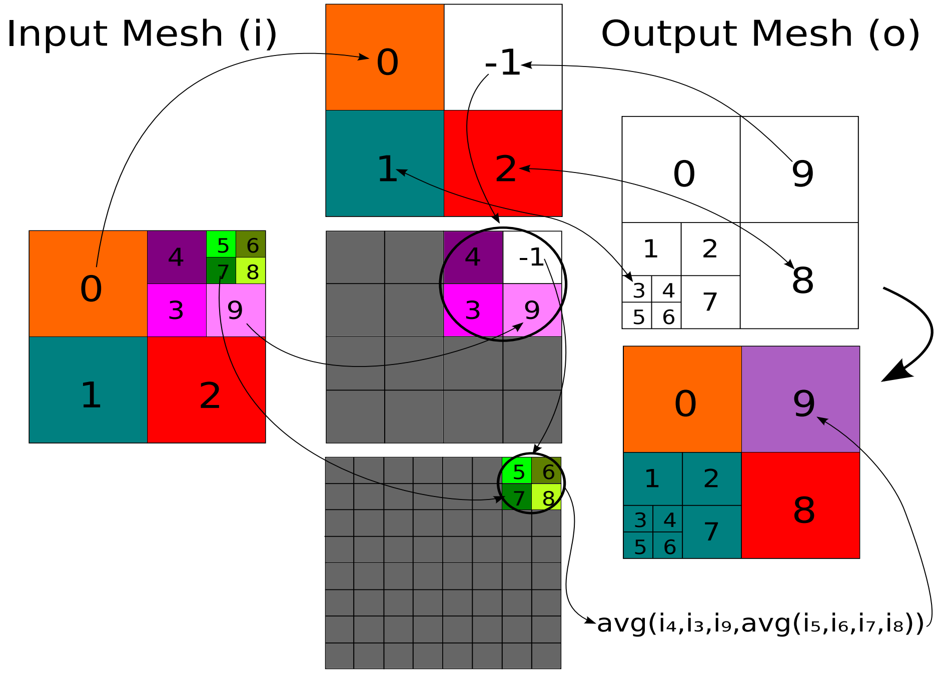
2. Every cell checks for each face if there is a number in the other bucket. If there is, it is the neighbor cell. If not, it is an external face with no neighbor.

→ We have found our neighbors in a single write and single read!

# Hashing Methodology: Higher Order Structures

**Take a sequence of hash tables at each level of refinement, connect them into a hierarchy using breadcrumbs…**

Cell 0 writes to the coarsest hash table. Cell 7 writes to the fine hash, but leaves -1 in the coarser hashes along the way as breadcrumbs.



Cell 8 finds its match of cell 2 in the coarse hash. Cell 9 follows the bread crumbs and gets cells -1 → (3,4,9, -1→ (5, 6, 7, 8) ) and weighted averages them for its result.

## N-body and particle codes

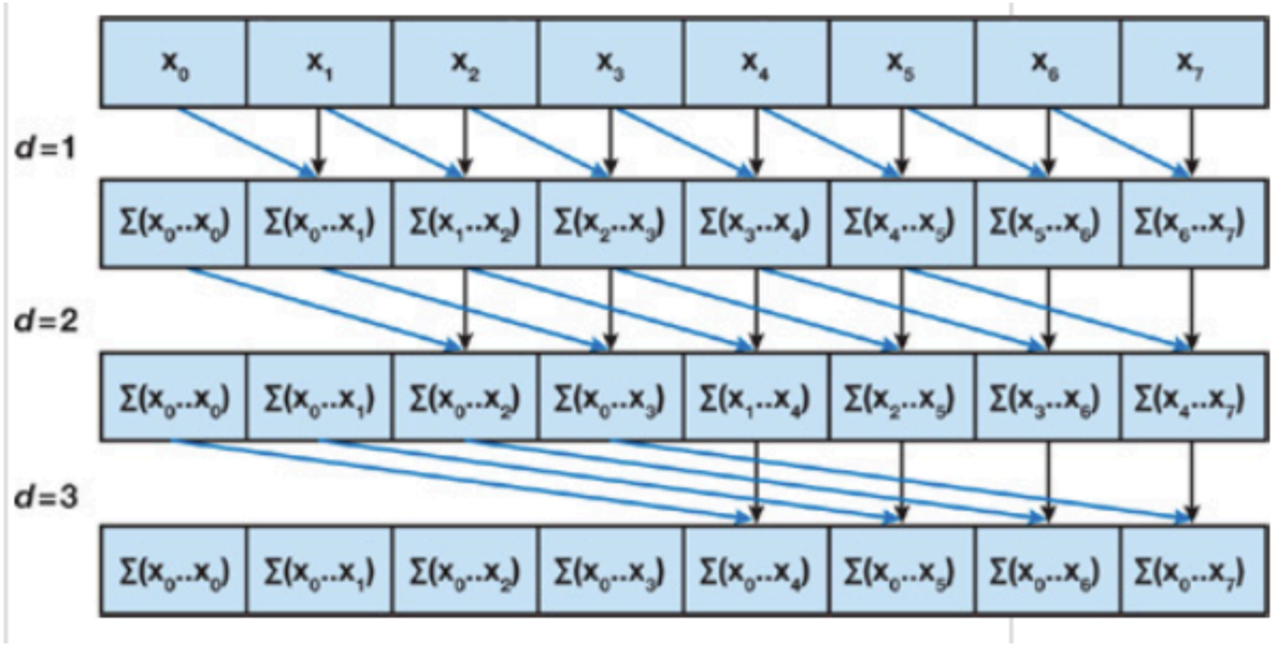
- **Hashed oct-tree (HOT), Warren, M., LANL**
  - Exploits the oct-tree by using a z-order curve. Operations then involve bit arithmetic to find neighbors and parent cells.
  - Stores data using a hash key for each position in the oct-tree
- **Smooth Particle Hydrodynamics – only look at interactions within a certain distance**
  - Method of cells (and other names) – bin particles by interaction distance and only have to look at adjacent bins.

- **These are early examples of spatial hashing.**

## Prefix Scan Exercise

- **Each person write 4 numbers from 1 – 100 on a piece of paper.**
- **These are the number of cells of 4 different types that you have**
- **Compute the starting point in a contiguous array for your section of data**
  - People before me – 120, 220, 80, so I start at 421. I have 160, so I end at 580 and pass that on to the next person
- **This seems like a serial algorithm (actually a pattern). Hillis and Steele developed a parallel version of the scan (1986). Blelloch came up with a way to do it with a work efficient approach (1990). Sengupta developed a hybrid work-efficient step-efficient version, recognizing that on the GPU, the number of steps in the Hillis-Steele version is more important the number of flops.**

## A Step-Efficient Prefix Scan – Hillis and Steele, 1986 Revived by Horn, 2005, for the GPU



$x_0$ $x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$
d=1
$\Sigma(x_0..x_0)$ $\Sigma(x_0..x_1)$ $\Sigma(x_1..x_2)$ $\Sigma(x_2..x_3)$ $\Sigma(x_3..x_4)$ $\Sigma(x_4..x_5)$ $\Sigma(x_5..x_6)$ $\Sigma(x_6..x_7)$
d=2
$\Sigma(x_0..x_0)$ $\Sigma(x_0..x_1)$ $\Sigma(x_0..x_2)$ $\Sigma(x_0..x_3)$ $\Sigma(x_1..x_4)$ $\Sigma(x_2..x_5)$ $\Sigma(x_3..x_6)$ $\Sigma(x_4..x_7)$
d=3
$\Sigma(x_0..x_0)$ $\Sigma(x_0..x_1)$ $\Sigma(x_0..x_2)$ $\Sigma(x_0..x_3)$ $\Sigma(x_0..x_4)$ $\Sigma(x_0..x_5)$ $\Sigma(x_0..x_6)$ $\Sigma(x_0..x_7)$

## Work efficient scan (upsweep)

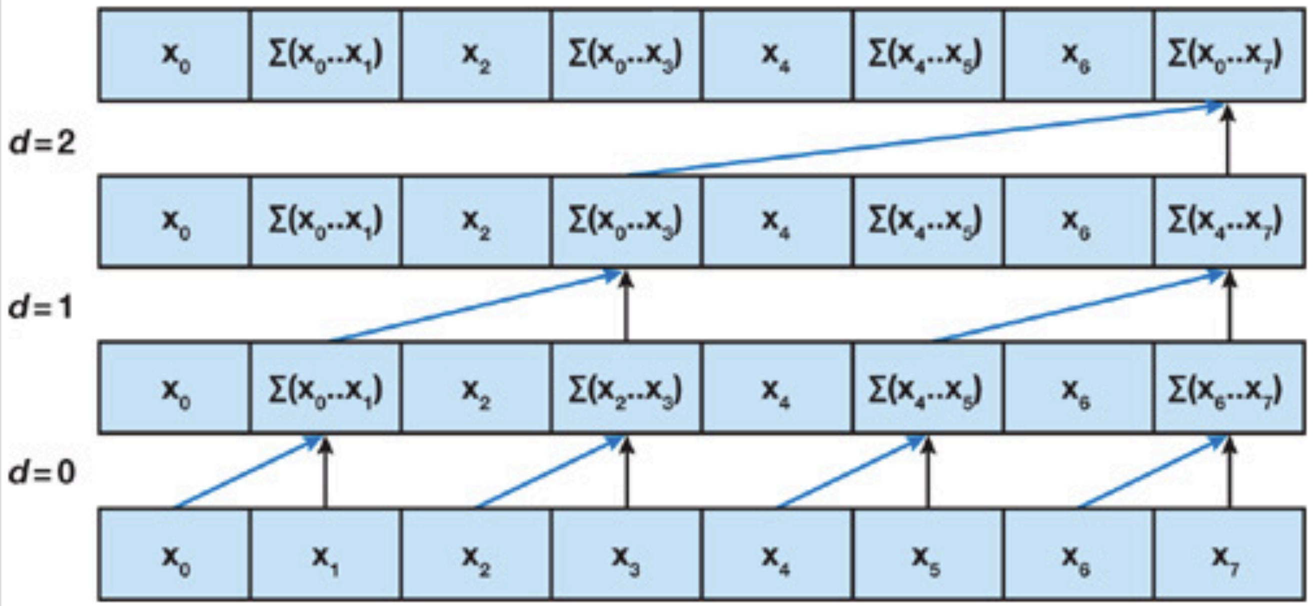


Figure 39-3 An Illustration of the Up-Sweep, or Reduce, Phase of a Work-Efficient Sum Scan Algorithm

**Example 3. The Up-Sweep (Reduce) Phase of a Work-Efficient Sum Scan Algorithm (After Blelloch 1990)**

```
1: for $d$ = 0 to $\log_2 n - 1$ do
2:     for all $k$ = 0 to $n - 1$ by $2^{d+1}$ in parallel do
3:         $x[k + 2^{d+1} - 1] = x[k + 2^{d} - 1] + x[k + 2^{d} + 1 - 1]$
```

From GPU Gems 3, Nvidia

## Work efficient scan (downsweep)

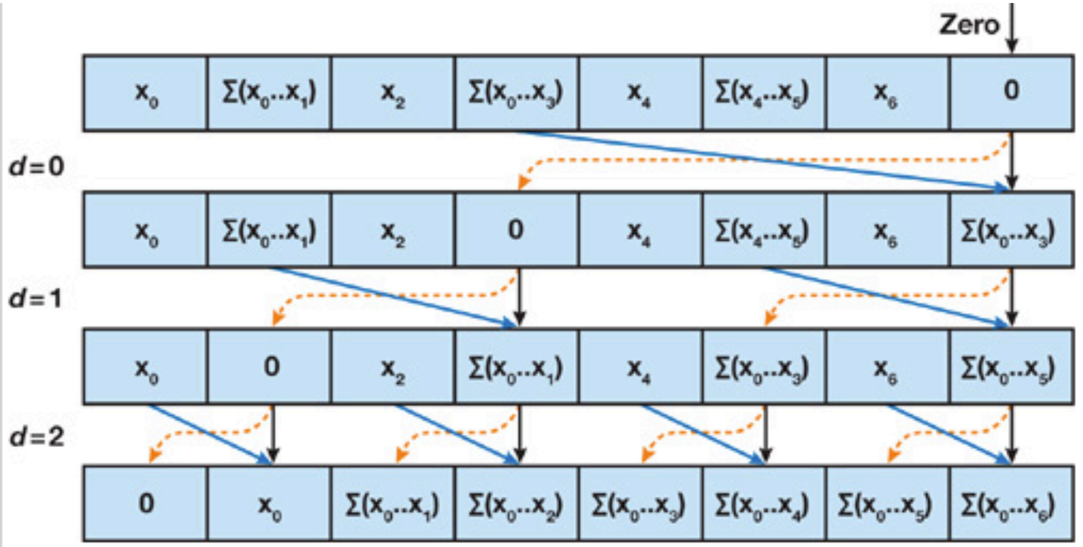

Zero
$x_0$ | $\Sigma(x_0..x_1)$ | $x_2$ | $\Sigma(x_0..x_3)$ | $x_4$ | $\Sigma(x_4..x_5)$ | $x_6$ | 0
$d=0$
$x_0$ | $\Sigma(x_0..x_1)$ | $x_2$ | 0 | $x_4$ | $\Sigma(x_4..x_5)$ | $x_6$ | $\Sigma(x_0..x_3)$
$d=1$
$x_0$ | 0 | $x_2$ | $\Sigma(x_0..x_1)$ | $x_4$ | $\Sigma(x_0..x_3)$ | $x_6$ | $\Sigma(x_0..x_5)$
$d=2$
0 | $x_0$ | $\Sigma(x_0..x_1)$ | $\Sigma(x_0..x_2)$ | $\Sigma(x_0..x_3)$ | $\Sigma(x_0..x_4)$ | $\Sigma(x_0..x_5)$ | $\Sigma(x_0..x_6)$

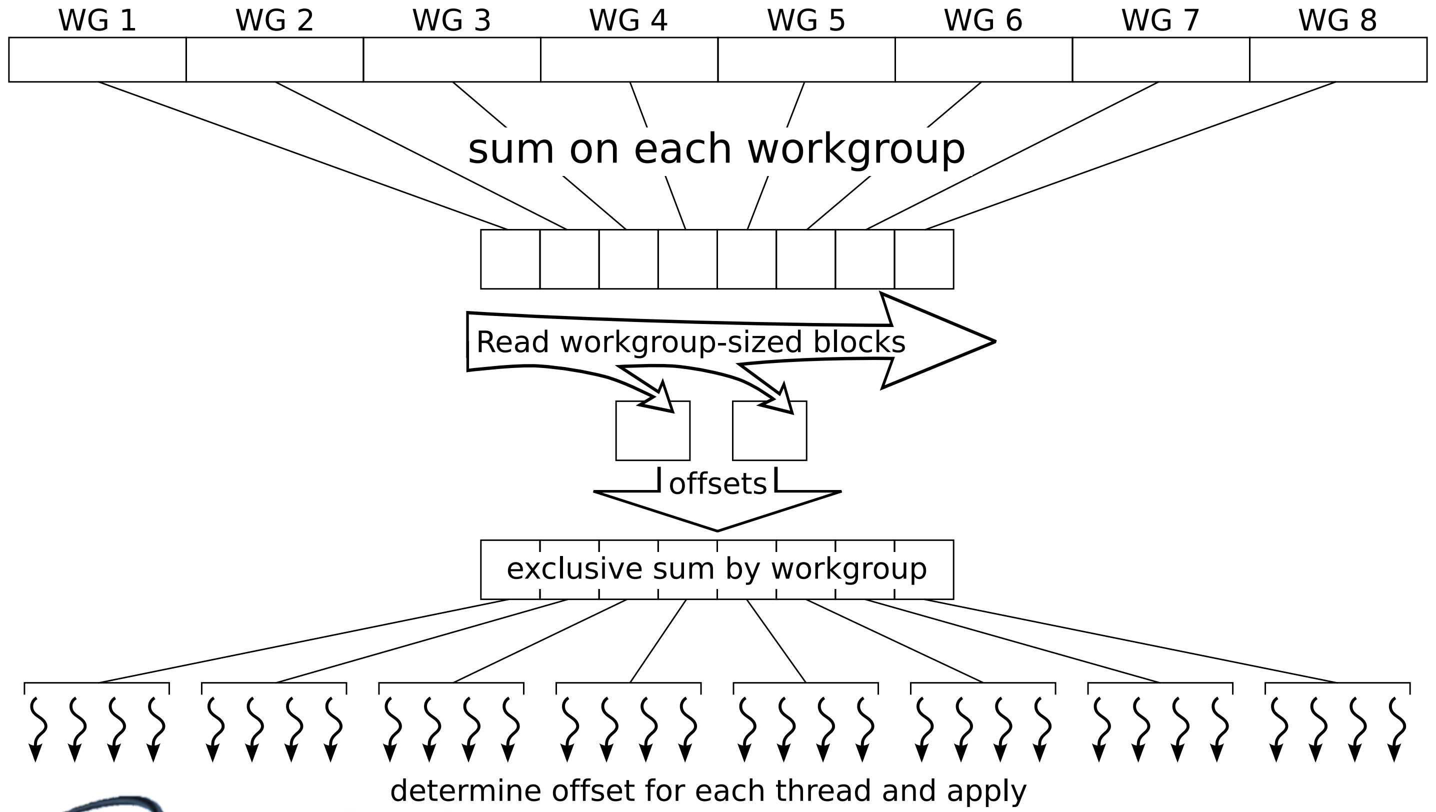
Figure 39-4 An Illustration of the Down-Sweep Phase of the Work-Efficient Parallel Sum Scan Algorithm

**Example 4. The Down-Sweep Phase of a Work-Efficient Parallel Sum Scan Algorithm (After Blelloch 1990)**

```
1: $x[n - 1] \leftarrow 0$
2: for $d = \log_2 n - 1$ down to 0 do
3:       for all $k = 0$ to $n - 1$ by $2^d + 1$ in parallel do
```

From GPU Gems 3, Nvidia

## Prefix scan algorithm on the GPU for large datasets Hybrid step efficient, work efficient

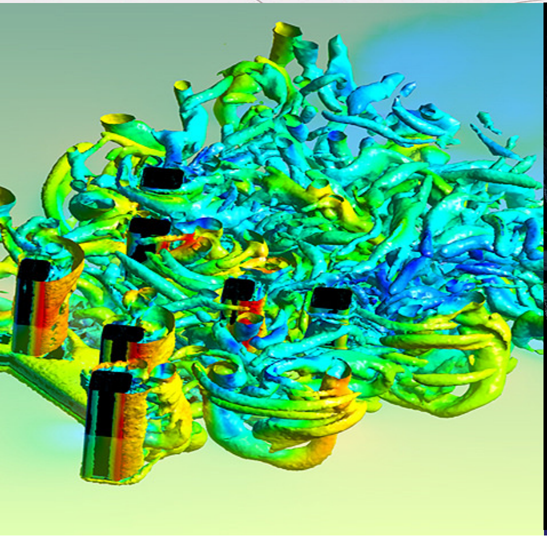

WG 1
WG 2
WG 3
WG 4
WG 5
WG 6
WG 7
WG 8
sum on each workgroup
Read workgroup-sized blocks
offsets
exclusive sum by workgroup
determine offset for each thread and apply

## References

- McCool, M., Robison, A., Reinders, J., "Structured Parallel Programming: Patterns for Efficient Computation", Elsevier, 2012
- Mattson, T., Sanders, B., Massingill, B., "Patterns for Parallel Programming", Pearson Education, 2005.
- D. Nicholaeff, N. Davis, D.P. Trujillo, R. W. Robey, "Cell-based Adaptive Mesh Refinement Implemented with General Purpose Graphics Processing Units", Tech. Rep. LA-UR-13-20165, Los Alamos National Laboratory, 2013.
- R. N. Robey, D. Nicholaeff, and R. W. Robey, "Hash-based Algorithms for Discretized Data," *SIAM Journal on Scientific Computing*, vol. 35, no. 4, pp. C346–C368, 2013.
- R. Tumblin, P. Ahrens, S. Hartse, and R. W. Robey, "Parallel Compact Hash Algorithms for Computational Meshes," *SIAM Journal on Scientific Computing*, Feb. 2015.
- D. Nicholaeff, R. Tumblin, I. Karlin, R.W. Robey, P. Ahrens, J. Sauer, R.N. Robey, S. Hartse, "A Survey of Hash-based Algorithms for Scalable Computational Mesh Management across Heterogeneous Architectures, 2014, LLNL-CONF-653580-DRAFT, LA-UR-14-22667

## More References

- G. Blelloch, “Vector Models for Data-Parallel Computing”, MIT Press, 1990
- G. Blelloch, “Prefix Sums and their Applications, 1990, Carnegie Mellon University, Tech Report, CMU-CS-90-190
- Kahan, William (January 1965), "Further remarks on reducing truncation errors", Communications of the ACM 8 (1): 40
- D.E. Knuth, The Art of Computer Programming, vol. 2, Addison-Wesley Press, 1969. chap. 4.
- Roman Iakymchuk, et al. ExBLAS: Reproducible and Accurate BLAS Library. *NRE: Numerical Reproducibility at Exascale*, Nov 2015, Austin, TX, United States. 2015.
- A Fortran-90 double-double library. Bailey, D., 2001, **http://www.nersc/gov/~dhbailey/mpdist/mpdist.html**



Los Alamos
NATIONAL LABORATORY
EST. 1943

Delivering science and technology
to protect our nation
and promote world stability

Parallel Computing SRI

ISTI INFORMATION SCIENCE & TECHNOLOGY INSTITUTE

NSEC National Security Education Center

NNSA National Nuclear Security Administration