AFOSR BRI: Codifying and Applying a Methodology for Manual Co-Design and Developing an Accelerated CFD Library

Synergy@VT Collaborators: **Paul Sathre**, Sriram Chivukula, Kaixi Hou, Tom Scogland, Harold Trease, Hao Wang, Wu Feng

VirginiaTech
Invent the Future

SyNeRG
synergy.cs.vt.edu

Forecast

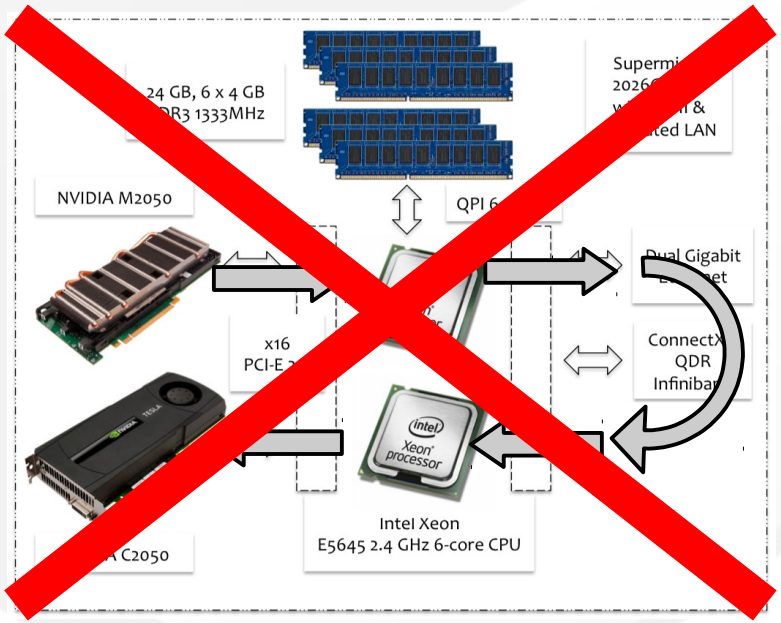
- Collaborative Efforts
- An Iterative Refinement Co-Design Methodology
  - Windfalls from Manual Co-Design
  - Manual Co-Design in GenIDLEST: Dot Product Reduction
- Towards a Library of Accelerated CFD Primitives
- Conclusions, Publications, and Questions

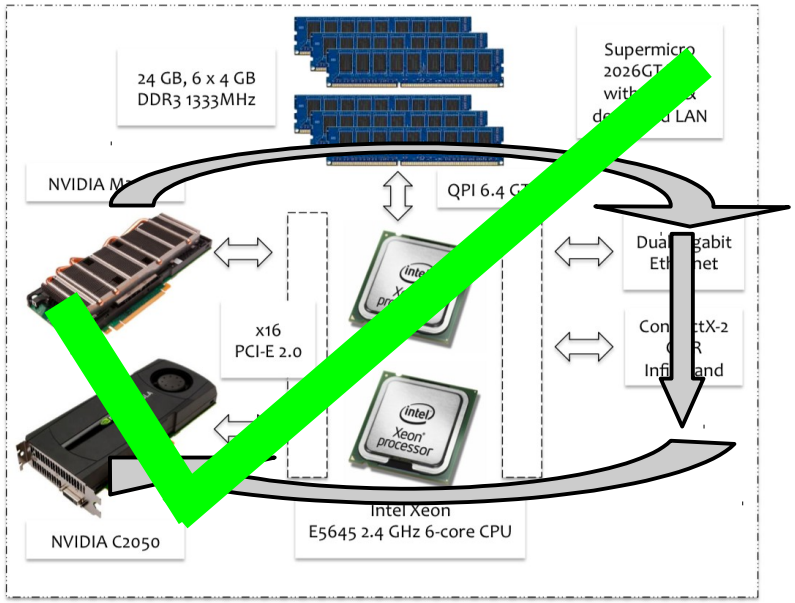
Collaborative Efforts

- General Knowledge Transfer
  - Tool Chain for Co-Design (Scogland)
    - Example: PGI Compiler/OpenACC collaboration
    - More in his talk, up next
  - Efficient GPU → GPU Communication/MVAPICH (Wang)
  - Unstructured Grids (Trease)
  - CUDA Portability via Automated OpenCL Translation (Sathre)
  - CUDA/OpenCL/GPU Computing (All)
  - Manual Co-Design Development (All)

Efficient GPU → GPU Communication/MVAPICH

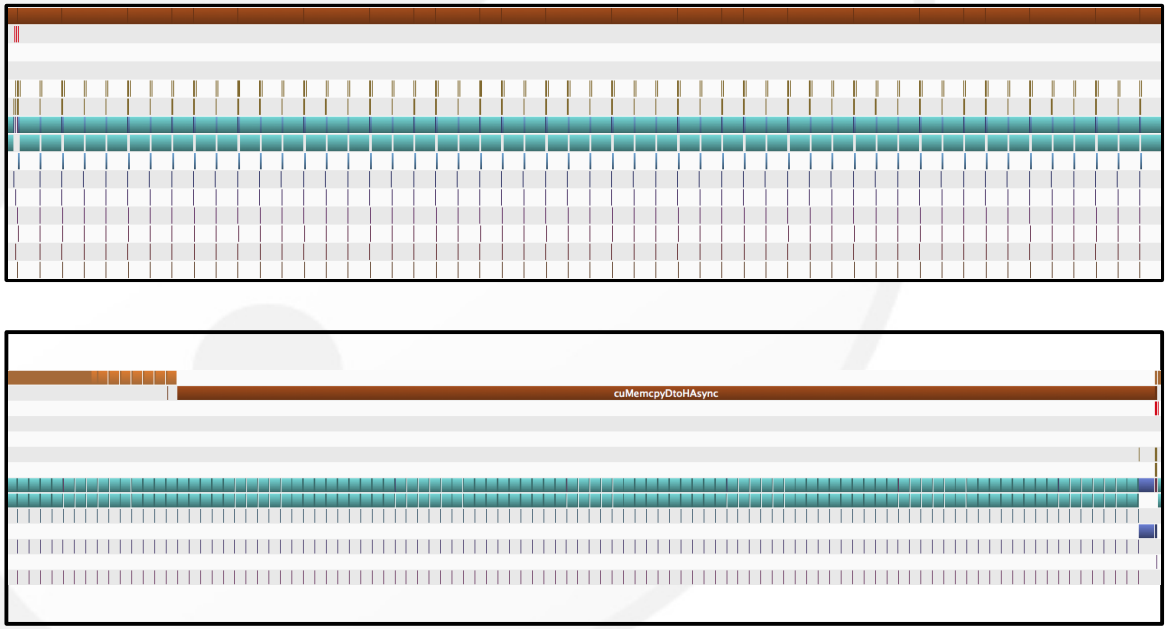
- GPU-Aware MPI for INCOMP3D and GenIDLEST (Wang)
  - Overlap communication stages for improved performance
  - Support GPU-Aware MPI in OpenACC Fortran via CUDA Fortran wrapper functions (INCOMP3D w/ Luo)
  - Performance evaluation of new MVAPIH2-GDR support for "GPU-Direct-Remote DMA → direct GPU-to-GPU data transfer" on HokieSpeed





Tool Chain for Co-Design

- Lid-Driven Cavity/Sensei-Lite OpenACC Development (Scogland)
  - Preliminary Optimization of ILUT/BILUT preconditioners w/ de Sturler's group (Math)
  - OpenACC Performance Tuning of SENSEI-Lite w/ Roy's group (CFD)
  - OpenACC Performance Analysis on AMD GPUs
  - Reduced Synchronization Overhead/Frequency of Convergence Tests

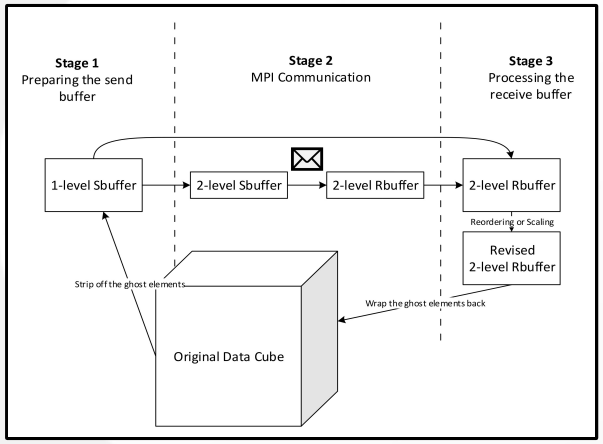


Check every iteration

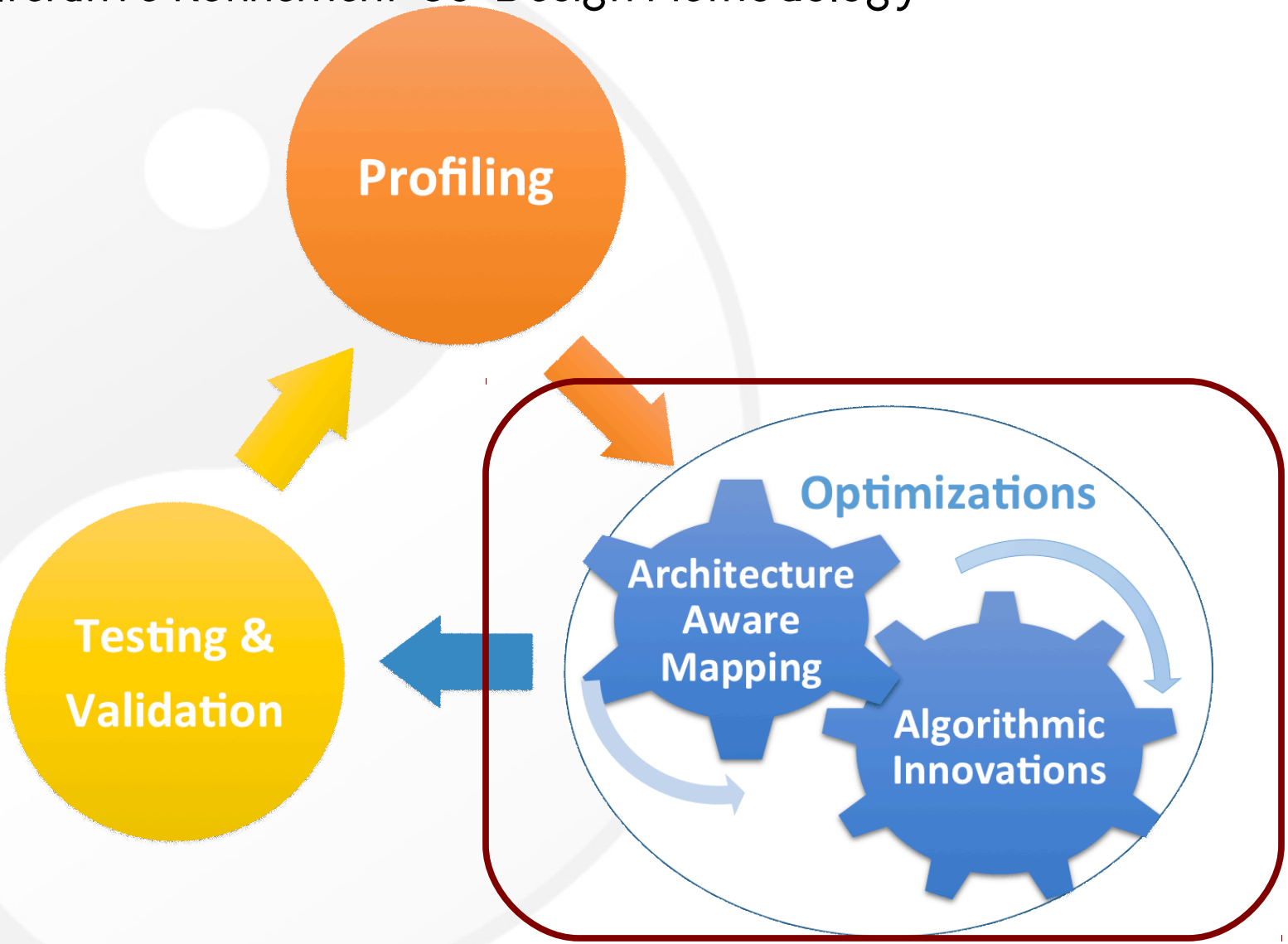
Check every 100th iteration

GPU Computing (CUDA Fortran)

- Manual Optimizations for GenIDLEST (Chivukula, Hou, Sathre)
  - Program-wide remapping of data structures in GPU to improve performance of GPU memory accesses
  - Restructured algorithms for increased parallelism/throughput
    - Boundary-avoiding 3D Dot Products and Reductions
    - Symmetric Successive Over-Relaxation Preconditioner
    - 7-point (Orthogonal) Jacobi Preconditioner
  - Reduced CPU/GPU synchronization overhead
  - Bottleneck analysis of data exchange methods



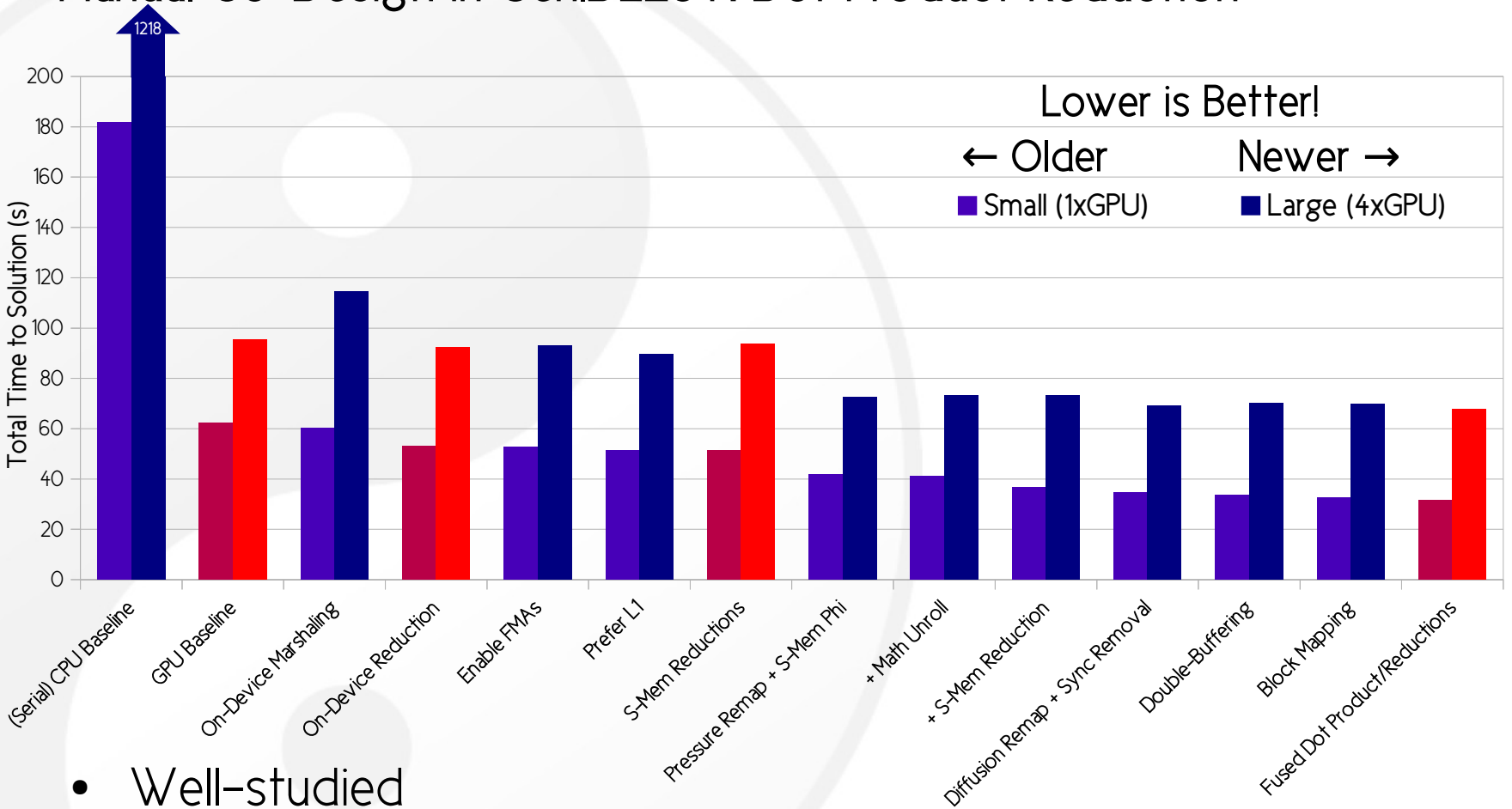
An Iterative Refinement Co-Design Methodology



Products of Manual Co-Design

- Not just a few faster applications …
- Cross-disciplinary and cross-code exchange of knowledge
  - OpenACC best practices
  - Efficient GPU → Network → GPU communication methods
- Identifying common elements of CFD codes
  - e.g. reductions, data exchanges, Matrix/vector operations
  - Accelerated methods can be extracted, generalized, and reused
- Identifying common elements of Co-design efforts
  - Foundational experiences for reasoning about a recipe for manual co-design

Manual Co-Design in GenIDLEST: Dot Product Reduction



- Well-studied
- 1000s of calls per execution, even on small problems

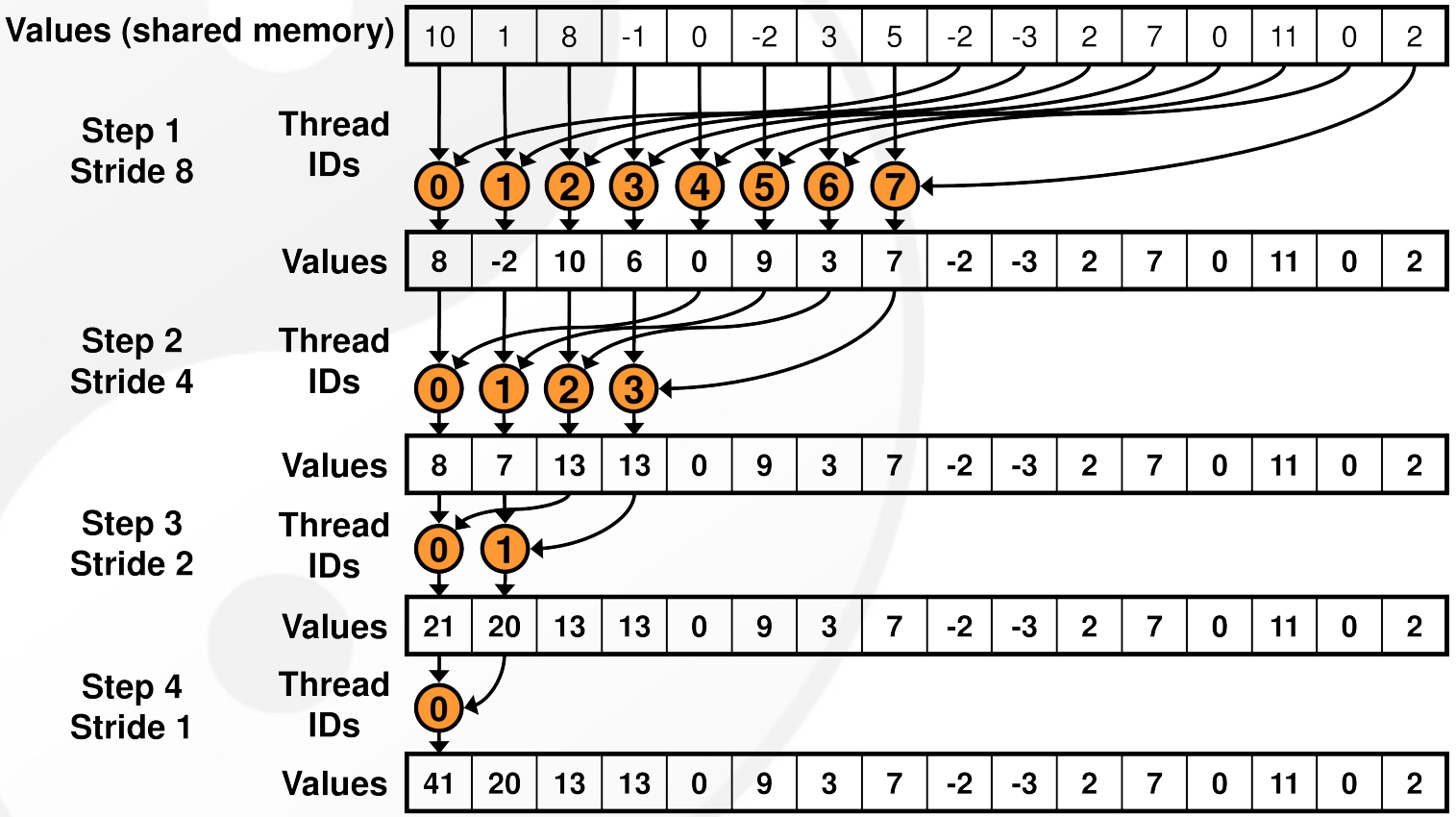
"Towards a Methodology for Co-Design of Accelerator-Aware CFD Applications", Paul Sathre, Amit Amritkar, Sriram Chivukula, Kaixi Hou, Danesh Tafti, Wu-chun Feng (IN PREPARATION)

CPU-Side Reductions

- In original GPU port, parallel multiply for dot-product was on GPU, serial summation loop was on CPU
  - Quick to implement: vector multiply on GPU is trivial, efficient reduction is not
  - Con: Had to transfer a big vector from GPU to CPU 1000s of times per execution, causing a communication bottleneck
    - Profiling data from micro test showed each transfer took ~5.3x more time than the multiply itself! (400us vs. 75us)

GPU Reduction Thread Layout

Parallel Reduction: Sequential Addressing

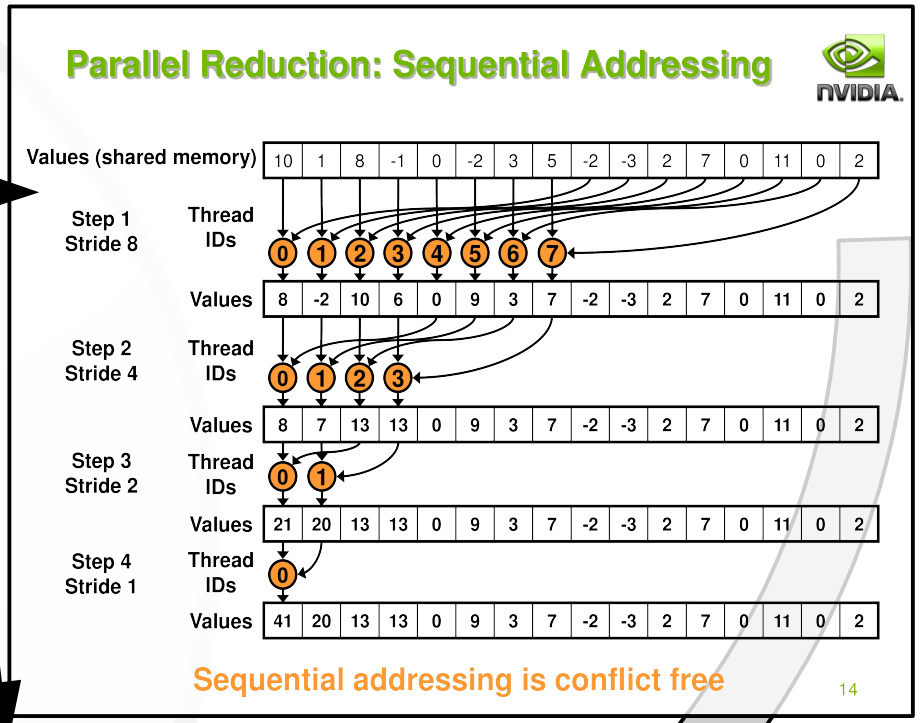


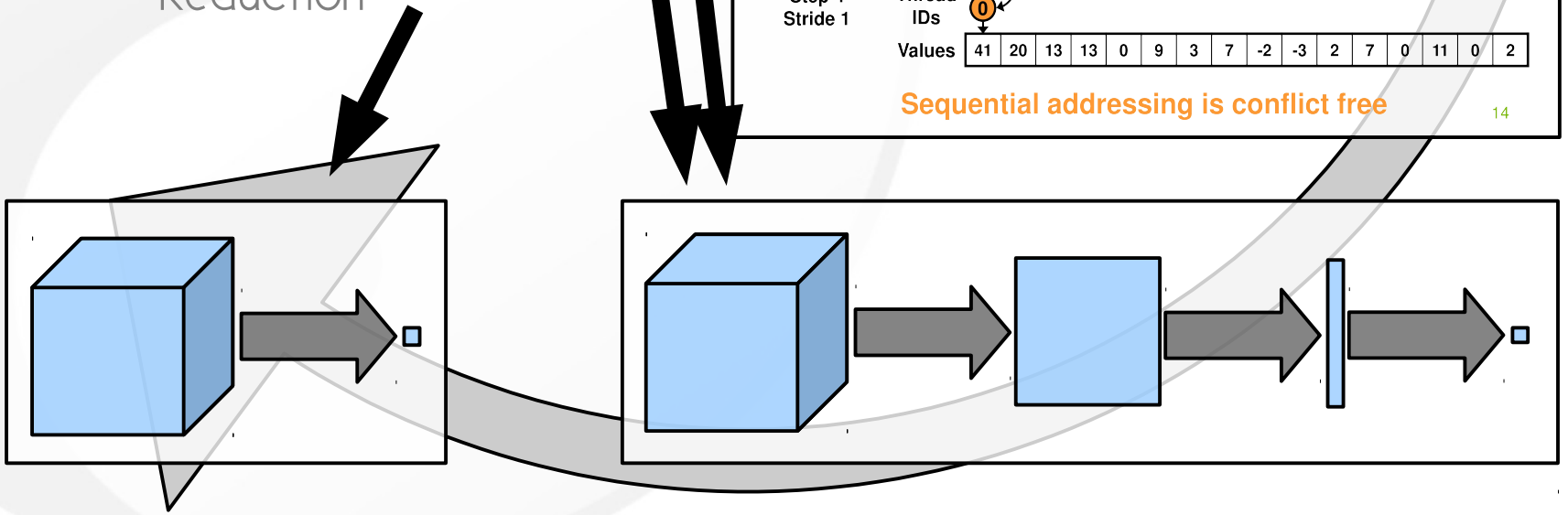
Sequential addressing is conflict free

From: developer.download.nvidia.com/assets/cuda/files/reduction.pdf

Successive Optimization of GPU Reduction

- Algorithmic innovation of sequential addressing underlies three GPU versions:
  - 3-stage, Global Memory
  - 3-stage, Shared Memory
  - Fused Dot Product + Reduction





3-Stage GPU Global Memory Reductions

- In original GPU port, parallel multiply for dot-product was on GPU, serial summation loop was on CPU
  - Quick to implement: vector multiply on GPU is trivial, efficient reduction is not
  - Con: Had to transfer a big vector from GPU to CPU 1000s of times per execution, causing a communication bottleneck
    - Profiling data from micro test showed each transfer took ~5.3x more time than the multiply itself! (400us vs. 75us)
- Moving reductions to GPU required an **Algorithmic Innovation**
  - 3-stage tree-based reduction in global memory (3D → 2D → 1D → scalar)
    - Each GPU block collaboratively sums values along the reduction dimension
      - Sub-results exchanged via global memory when each thread finishes
  - Pro: Eliminates GPU to CPU transfer bottleneck by only transferring a scalar each time
    - Micro profiling data shows each transfer is **160x** faster (2.5us vs. 400us)
    - And the entire dot product is **1.37x** faster than *just the original multiply and copy,* not counting the CPU summation!
      - (248us+11us+10us + 2.5us +75us) = 346.5us vs. 475us
      - (Stage1+Stage2+Stage3 + D2Hcopy + multiply)
    - And the **total time to solution** over the previous GPU version on a small test case was 1.13x faster (53.03s vs. 60.29s)

3-Stage GPU Shared Memory Reductions

- GPU Devices provide a region of fast "shared" memory that threads in the same group (block) can all access and share data through
  - Goal: improve performance of three stage reduction by accumulating and sharing data through shared memory rather than global
  - Architecture-Aware Mapping of the algorithm
- Improves reduction performance by **~2.6x** in micro profiling data
  - 105us for stages 1-3, vs. 269us
  - Con: Could no longer permanently swap 48KB Shared memory for 16KB L1 Cache on each GPU core, slowing other cache-heavy kernels
    - Caused slight net *increase* in total time to solution
    - Kept because shared memory optimizations were already planned for other portions of the code and resulted in further significant performance gains
    - May be able to temporarily swap L1/shared memory to eliminate this con, but we have not evaluated any additional overhead that may be incurred.

Fused Dot Product + Reduction

- In GPU-GenIDLEST, on-device reductions are only used for dot products, and GPU kernel launches have high overhead
  - Algorithmic Innovation of *Kernel Fusion* can reduce total launches from 4 to 1
    - Also provided new opportunities to reduce global memory traffic by communicating common values through shared memory
  - Used a Fortran translation of Mark Harris' Reduction #5 from his "Optimizing Parallel Reduction in CUDA" Webinar
    - developer.download.nvidia.com/assets/cuda/files/reduction.pdf
    - Modified to take an extra input array and perform the dot product's vector multiply
    - Translation and last-warp unrolling were tricky to implement
      - Fortran indexing is 1+offset, C is 0+offset
      - Had to add extra `__syncthreads()` calls to synchronize shared memory properly, required `volatile` keyword for warp-implicit sync didn't appear to be respected by PGI CUDA Fortran compiler
- Dot product + reduction performance observed in profiling data improved by **~2.1x** over previous version

So why build a library?

6 months of 1 domain expert's and 3 architecture experts' time for <2x and <6x speedup over original GPU and (serial) CPU, is not efficient software development!

Code becomes a maintainability nightmare!

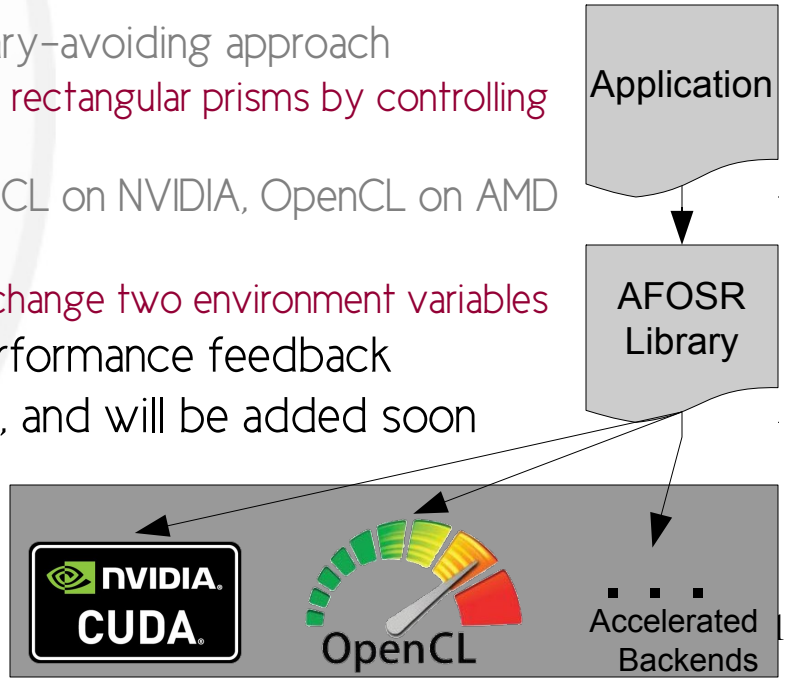
We want to enable domain scientists to write fast codes faster, and be able to maintain it them themselves.

Towards a Library of Accelerated CFD Primitives

- Accelerated implementations from manual co-design
  - Extract, generalize, and reuse
  - Focus on methods shared across at least two of the codes
- Library users shouldn't need to hand-implement much accelerator code, if any
  - Provide (optional) performance metrics
  - Expose relevant tuning parameters
- Library users should focus on the application, not waste time managing devices and mixed accelerator models
  - Generalize multiple accelerator models between a single, unified interface
  - Support retargeting code to alternative devices without recompiling

The Library So Far

- Generic API wrapper around CUDA and OpenCL implementations
  - "Library of Libraries"
    - Modular construction should ease adding support for new accelerator models
  - OpenMP support considered, but CPUs already supported by OpenCL
  - Still requires explicit accelerator allocations, copies, and invocations, but agnostic of what backend is used
  - C API for now, Fortran compatibility coming soon
- Started with Reductions
  - Inspired by GenIDLEST's accelerated, boundary-avoiding approach
    - 3D by design, but supports arbitrary-shaped rectangular prisms by controlling start/end indices in X, Y, and Z dimensions
  - Correct results from: CUDA on NVIDIA, OpenCL on NVIDIA, OpenCL on AMD GPU/CPU, OpenCL on Intel CPU
    - Without code changes or recompiling, just change two environment variables
- Adding opt-in, automatic timers for easy performance feedback
- Data exchange primitives have been isolated, and will be added soon
- 7-point stencil to follow



Conclusion

- Manual co-design can be ad-hoc by nature, but intentional introspection can help identify common design/behavior patterns.
- Cross-disciplinary and cross-codebase knowledge transfer can be facilitated via co-location and intelligent use of all-hands meetings.
- Focusing on generalizing and packaging methods for reuse during manual co-design helps get more mileage out of hand-optimized codes.

Publications and Submissions

- P. Sathre, A. Amritkar, S. Chivukula, K. Hou, D. Tafti, W. Feng, "Towards a Methodology for Co-Design of Accelerator-Aware CFD Applications". (in preparation)
- A. Amritkar, D. Tafti, P. Sathre, K. Hou, S. Chivukula, W. Feng, "Accelerating Bio-Inspired MAV Computations using GPUs", AIAA Aviation and Aeronautics Forum and Exposition 2014, 16 – 20 June 2014, Atlanta, Georgia. (under review)
- B. P. Pickering, C. W. Jackson, T. R. W. Scogland, W.-C. Feng, and C. J. Roy, "Directive-Based GPU Programming for Computational Fluid Dynamics," AIAA Paper 2014-1131, 52nd Aerospace Sciences Meeting, National Harbor, MD, January, 2014.

Questions?