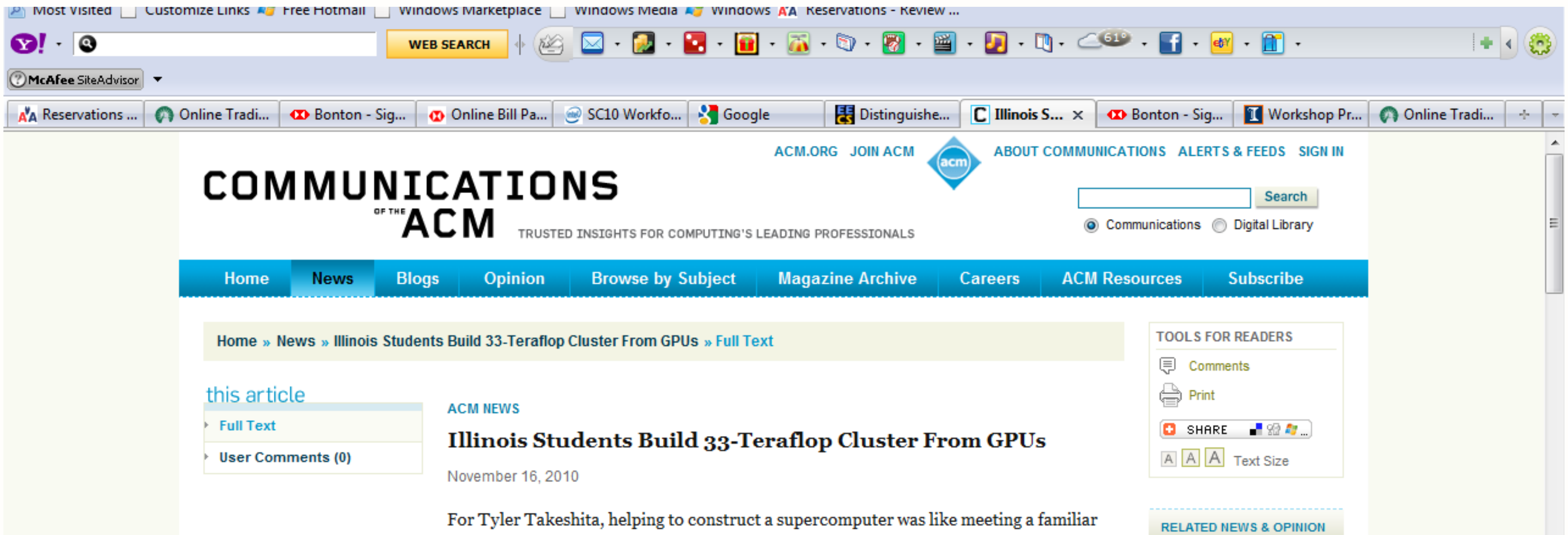
Extreme Scale Heterogeneous Computing

Wen-mei Hwu
University of Illinois, Urbana-Champaign

Extreme Scale Heterogeneous Computing Gaining Momentum

- Chinese (Tienhe and Nebula) and Japenese (Tsubame 2.0) machines rose to top of Top500
- Two of the top 3 of Green500 are Heterogeneous Computing Clusters



GPU computing is catching on.

- Financial Analysis
- Scientific Simulation
- Engineering Simulation
- Data Intensive Analytics
- Medical Imaging
- Digital Audio Processing
- Digital Video Processing
- Computer Vision
- Biomedical Informatics
- Electronic Design Automation
- Statistical Modeling
- Ray Tracing Rendering
- Interactive Physics
- Numerical Methods

- 280 submissions to GPU Computing Gems
  - 110 articles included in two volumes

Samples of UIUC Heterogeneous Computing Application Efforts

- NAMD/VMD (Phillips/Stone)
- MILC (Gotlieb/Shi)
- QMC (Kim/Ceperley)
- De Novo Gene Assembly (Ma/Chen/Hwu)
- IMPATIENT (Sutton/Liang/Hwu)
- …

A Common GPU Usage Pattern

- A desirable approach considered impractical
  - Due to excessive computational requirement
  - But demonstrated to achieve domain benefit
  - Convolution filtering (e.g. bilateral Gaussian filters), De Novo gene assembly, etc.

- Use GPUs to accelerate the most time-consuming aspects of the approach
  - Kernels in CUDA or OpenCL
  - Refactor host code to better support kernels

- Rethink the domain problem

AVAILABLE KERNELS

Library Kernels

- CUBLAS
  - Basic Linear Algebra
  - CUDA SDK
- CULA, Magma
  - Linear Algebra Solvers
  - www.culatools.com
  - http:/icl.cs.utk.edu/magma
- CUSP
  - Sparse data structures and algorithms
  - SpMV, CG, …
- Graph algorithms
  - BFS kernels exist
  - Need graph partitioning kernels
- Unstructured grid algorithms
  - 3D surface mesh generation/ refinement
  - Need 3D volume mesh generation (e.g. CGAL)/ refinement
- Add your favorite library here

Four Challenges

- Computations with no known scalable parallel algorithms
  - Shortest path, Delaunay triangulation, …
- Data distributions that cause catastrophical load imbalance in parallel algorithms
  - Free-form graphs, MRI spiral scan
- Computations that do not have data reuse
  - Matrix vector multiplication, …
- Algorithm optimizations that are hard and labor intensive
  - Locality and regularization transformations

Kernel development for GPUs is heavy lifting.

Each kernel is typically a 3-month job but very few developers benefit from advanced compiler technology today.

Little code reuse due to kernel sensitivity to memory access patterns and work distribution.

KERNEL DEVELOPMENT

Many-core Kernel Development

- Many-core programming is about performance and scalability.
  - Scalability is also key to power efficiency.
  - Performance and scalability for many-cores requires largely the same techniques.
    - To regularize work and data for massively parallel execution.
    - To localize data for conserving memory bandwidth

- There is a gap between what the programmers need and what the tools provide today.

Key to Massive Parallelism - Regularity and Locality

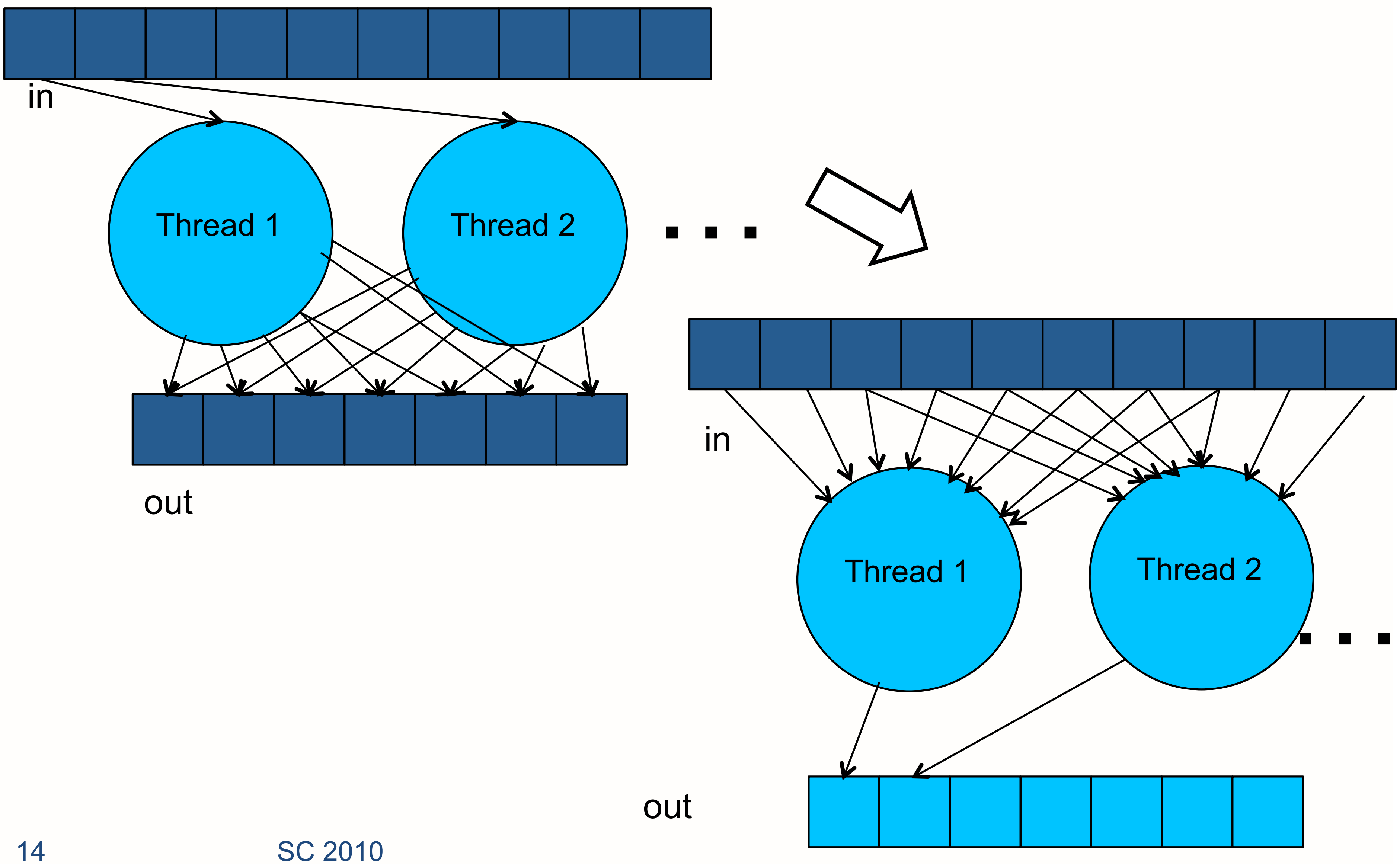
Eight Optimization Patterns for Algorithms (so far)

Technique	Contention	Bandwidth	Locality	Efficiency	Load Imbalance	CPU Leveraging
Tiling		X	X			
Privatization	X		X			
Regularization				X	X	X
Compaction		X				
Binning		X	X	X		X
Data Layout Transformation	X		X			
Thread Coarsening	X	X	X	X		
Scatter to Gather Conversion	X					

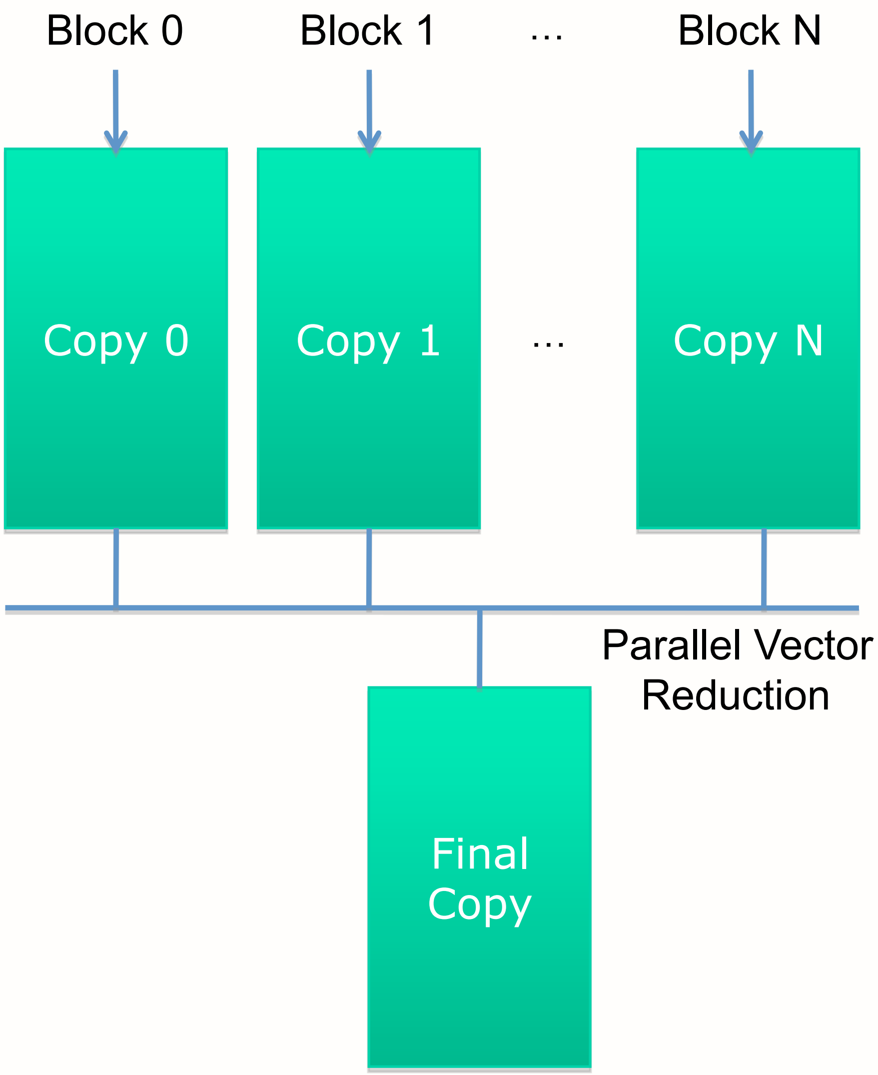
http://courses.engr.illinois.edu/ece598/hk/

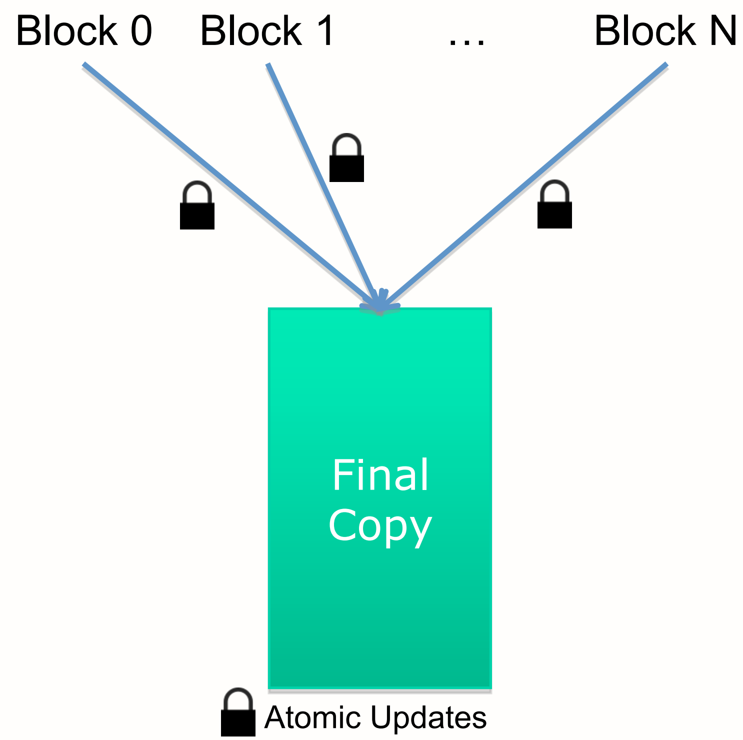
GPU Computing Gems, Vol. 1 and 2

1: Scatter to Gather Transformation



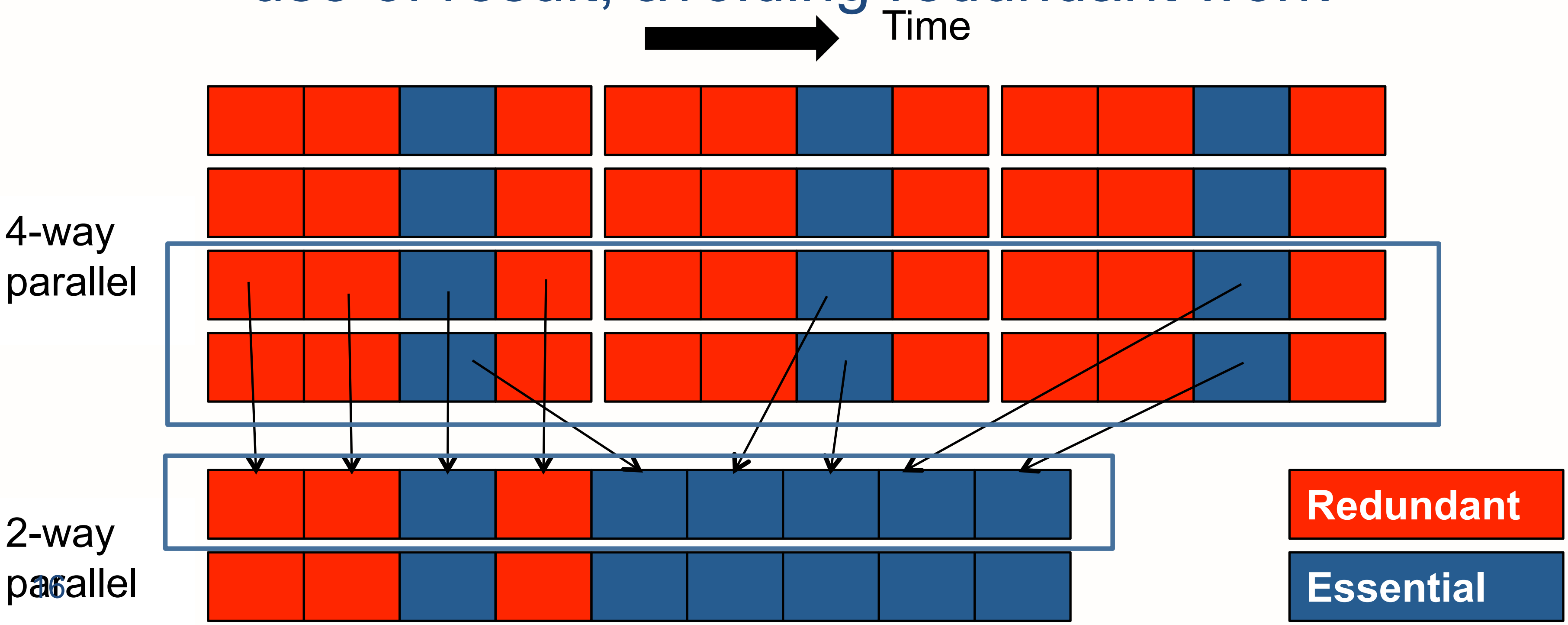
2. Privatization



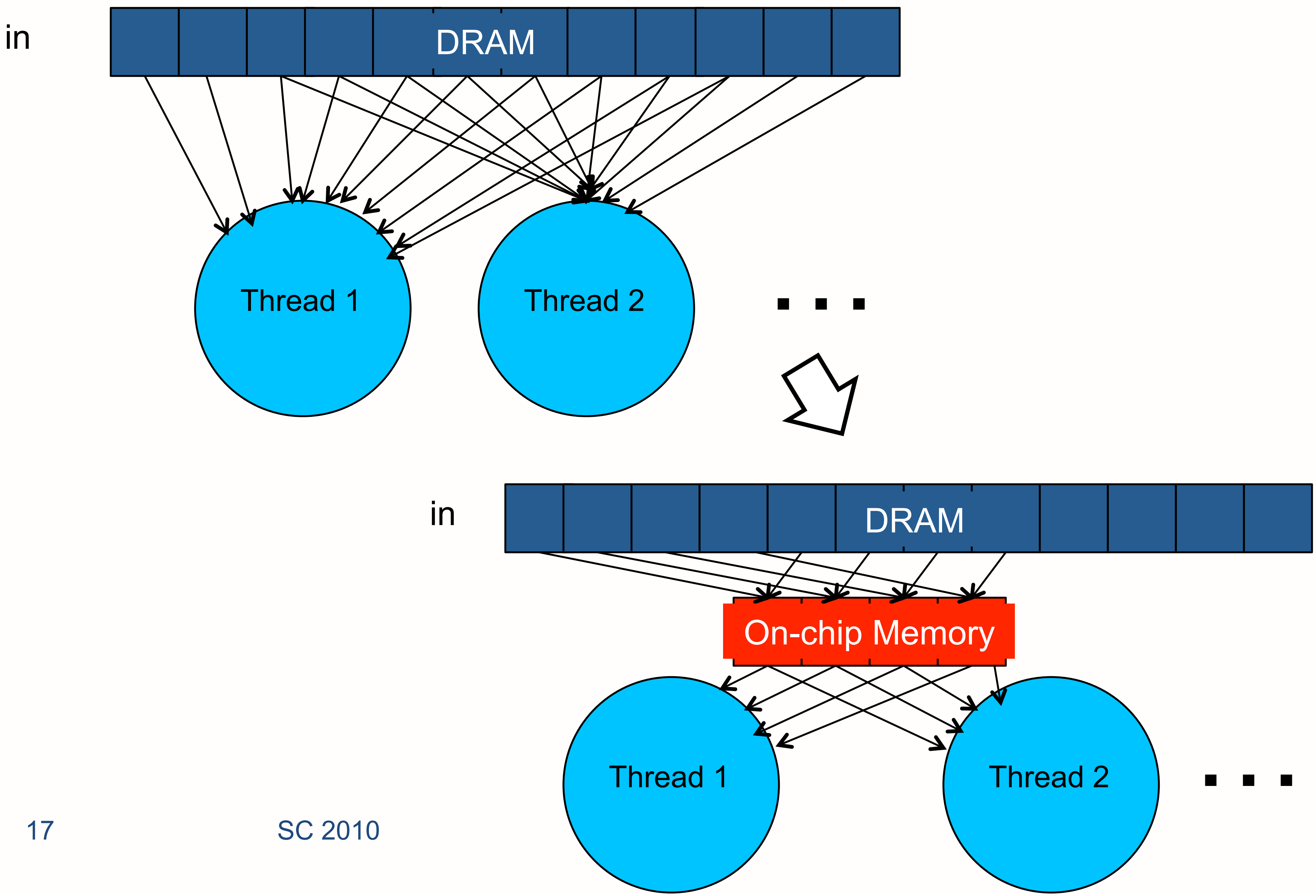


3. Granularity Coarsening and Register Data Reuse

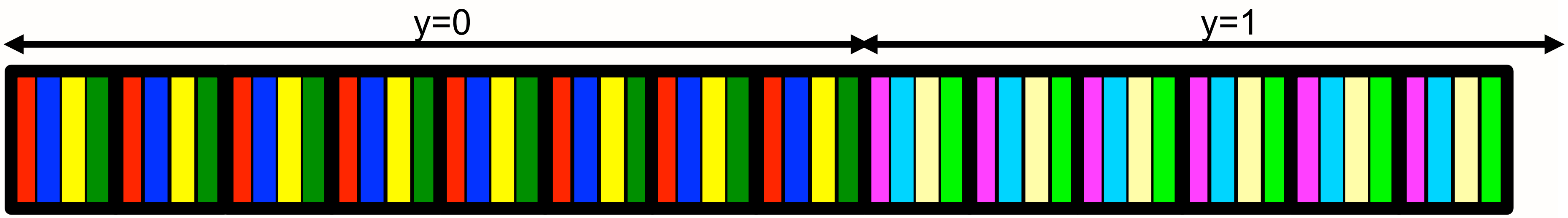
- Parallel execution often requires redundant and coordination work
  - Merging multiple threads into one allows re-use of result, avoiding redundant work



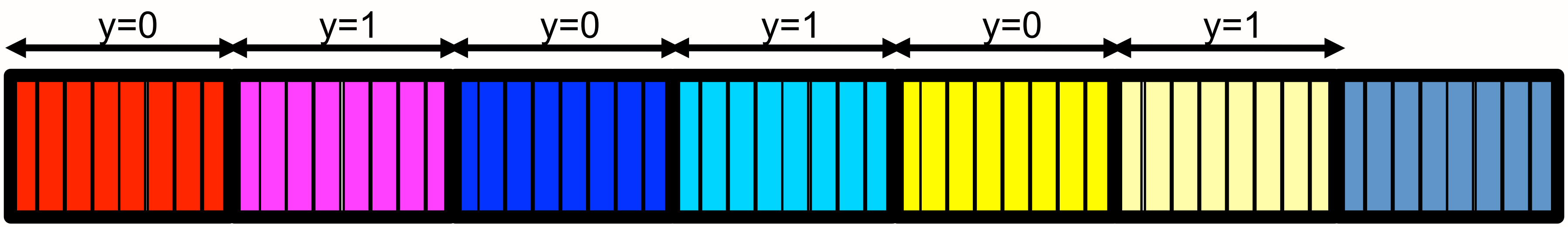
4: Data Access Tiling



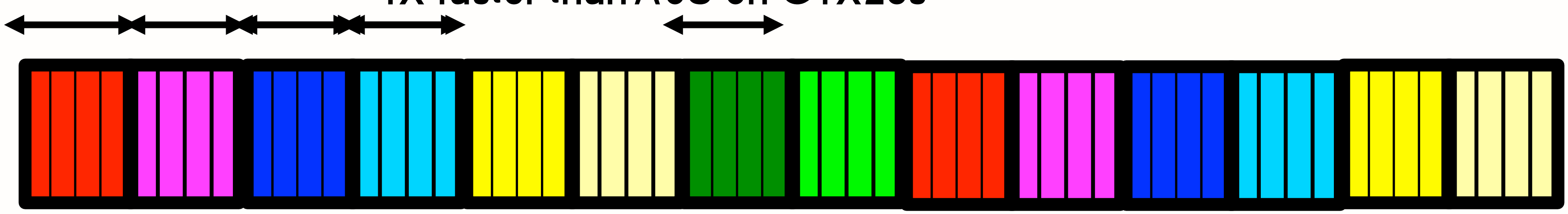
5. Data Layout Transformation



Array of Structure: [z][y][x][e]

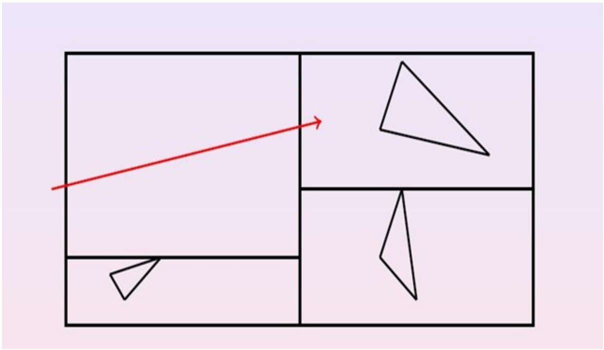


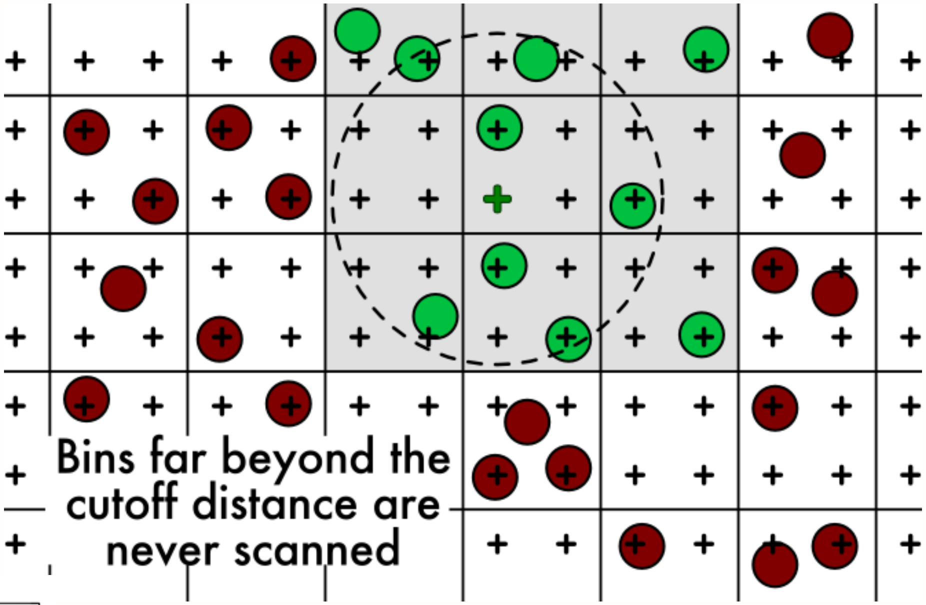
Structure of Array: [e][z][y][x]
4X faster than AoS on GTX280

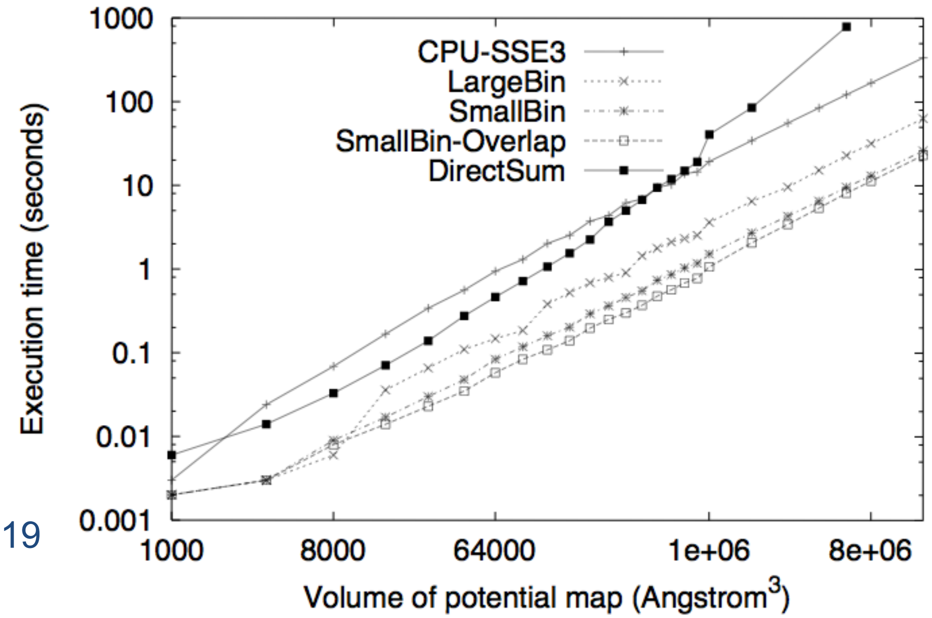


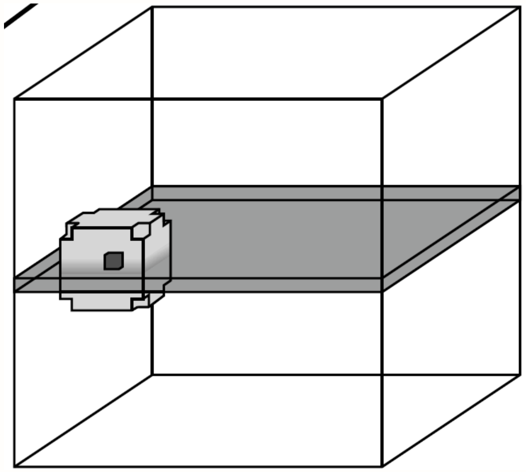
$[z][y_{31:4}][x_{31:4}][e]$**$[y_{3:0}][x_{3:0}]$**, 6.6 X faster than AoS

6: Input Data Binning

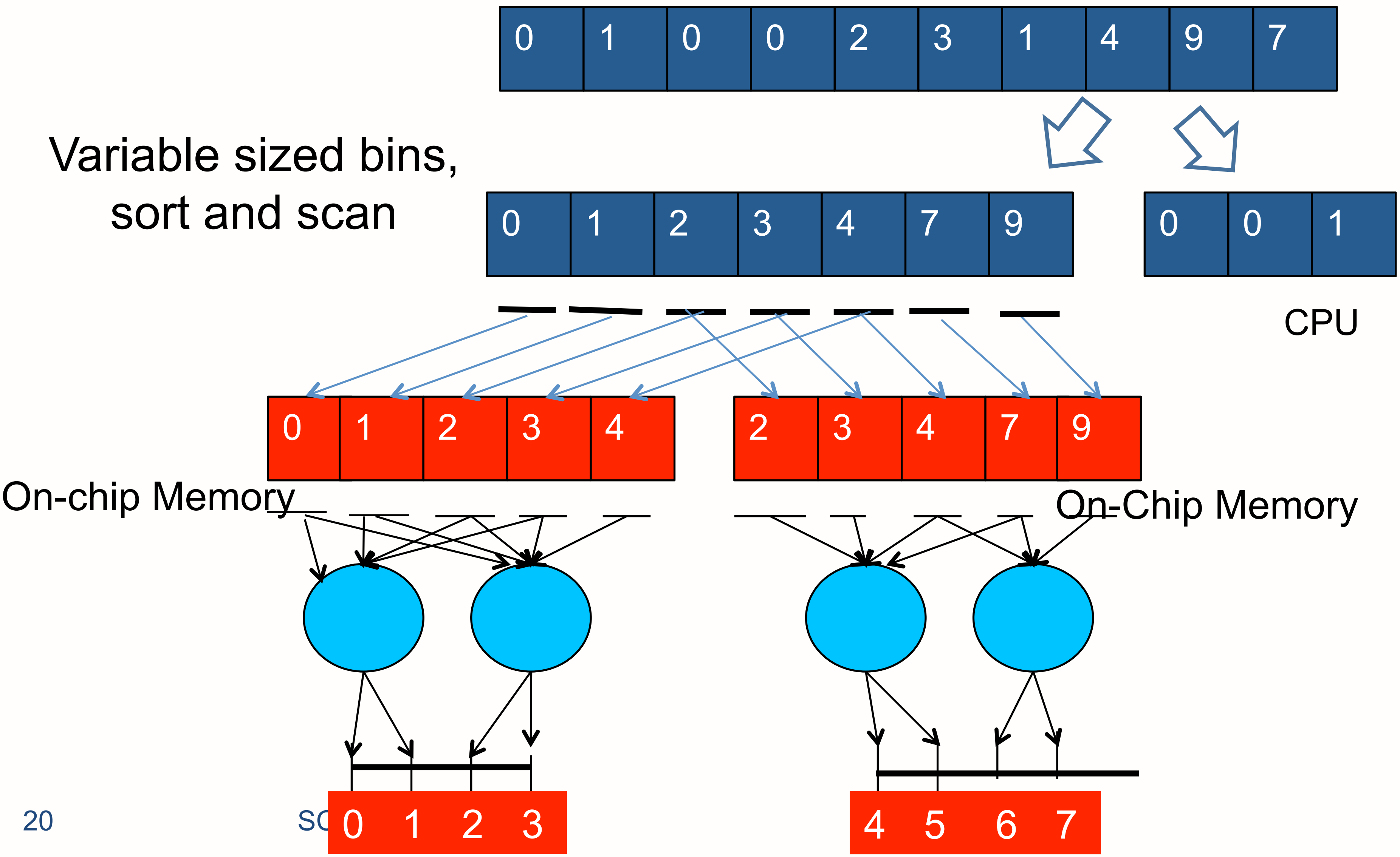




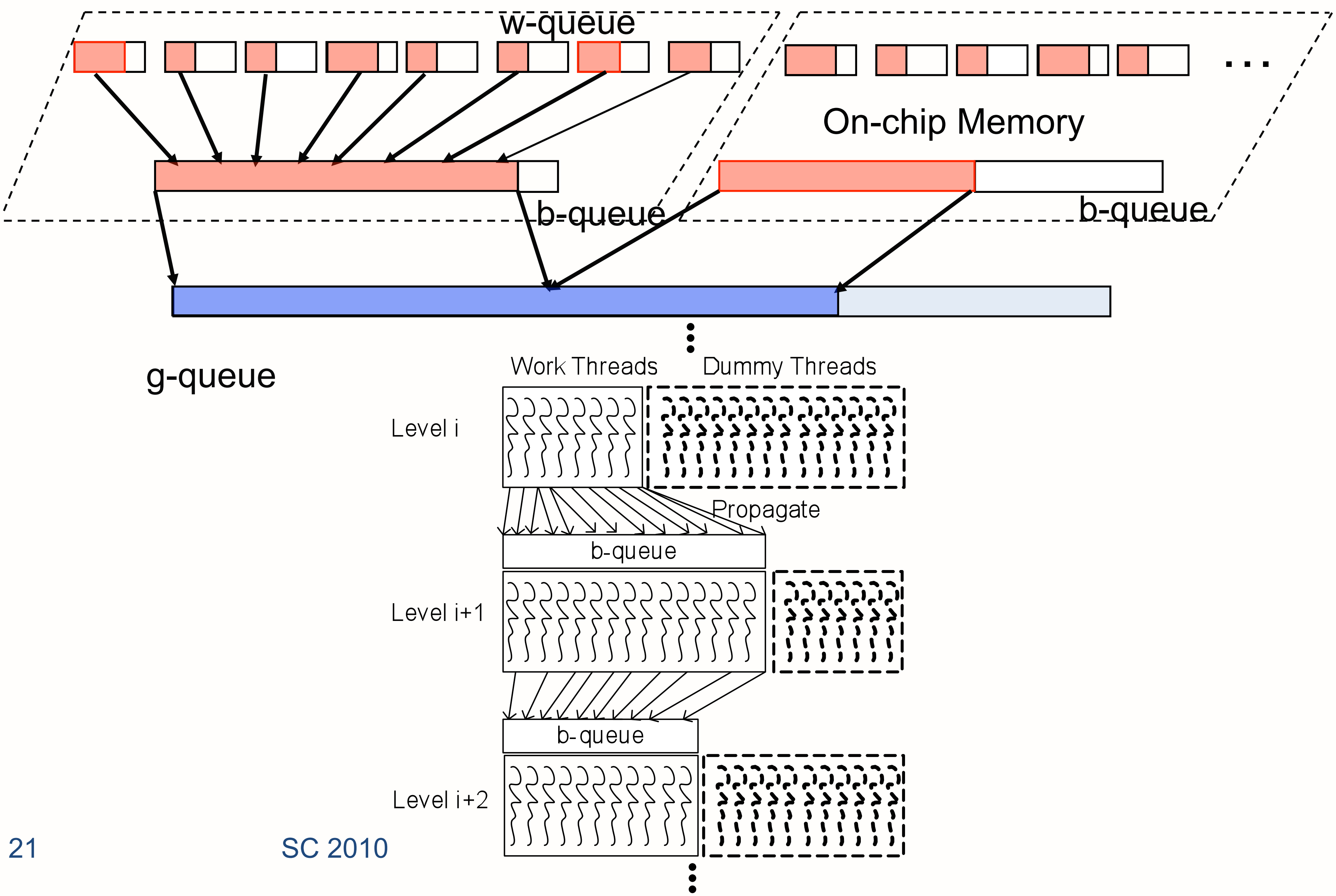




7. Compaction



8. Regularization

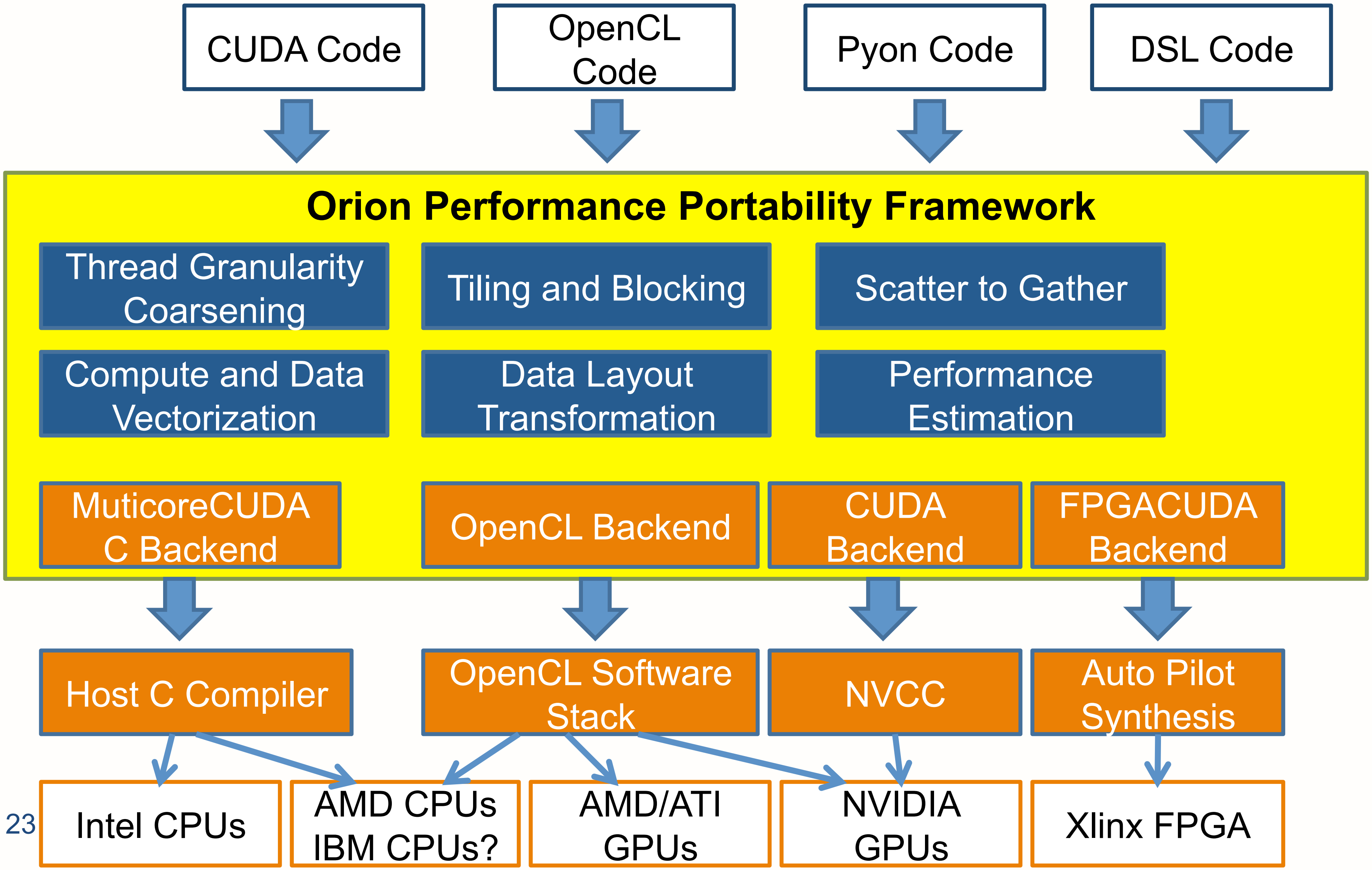


Tools go with techniques.

- Tools should facilitate key techniques
  – Programmers should write code "for others to understand instead of for computers to execute" - Dijkstra

- Techniques vary in their potential for automation
  – Scatter-to-gather, granularity coarsening, data access tiling, and memory layout quite amenable
    - Need clear performance guidance
  – Input binning, bin sorting, and hierarchical queues are much harder
    - Need to provide APIs understood by compilers/tools
    - Developer feedback critical to success

Orion: Reducing Performance Cost of Heterogeneous Parallelism

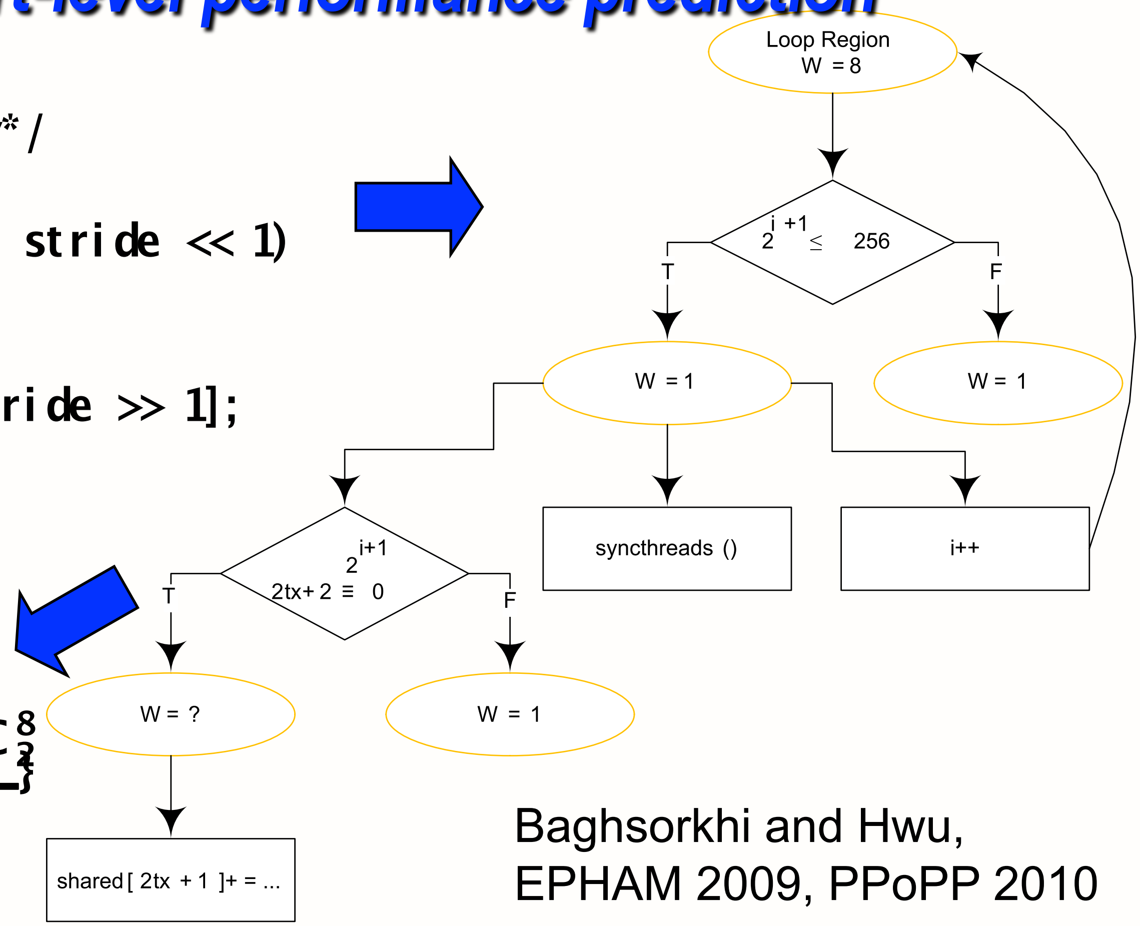


ADAPT: Example of Advanced Compiler Technique in kernel performance prediction

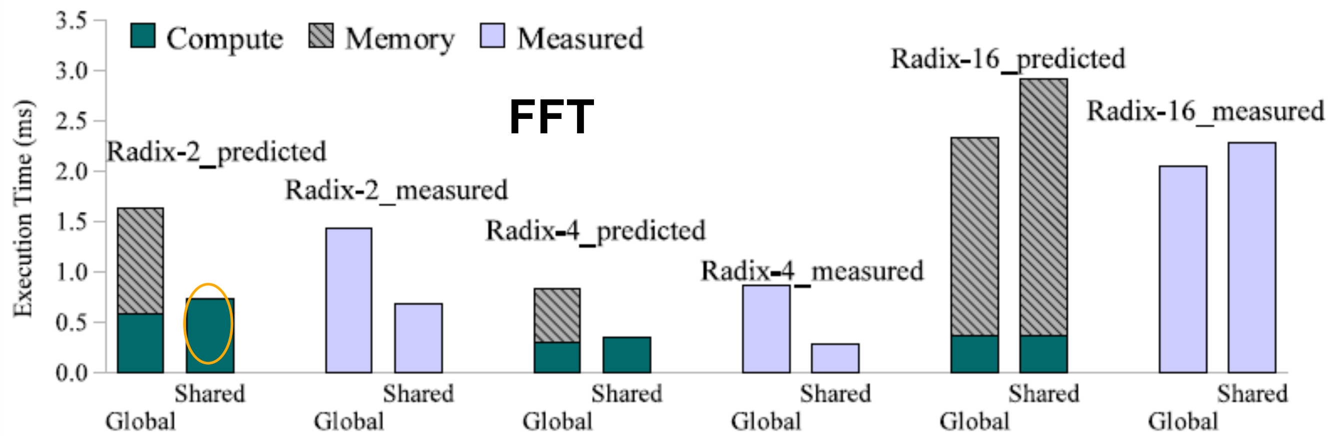
HW constraints enable efficient abstract interpretation to emulate expert-level performance prediction

```
n = 2 * tx + 1;
/*Load data into shared memory*/
...
for(stride = 2; stride <= 256; stride << 1)
f
  if( ((n+1) % stride  == 0)
      shared[n] +=shared[n - stride >> 1];
  __syncthreads();
g
```

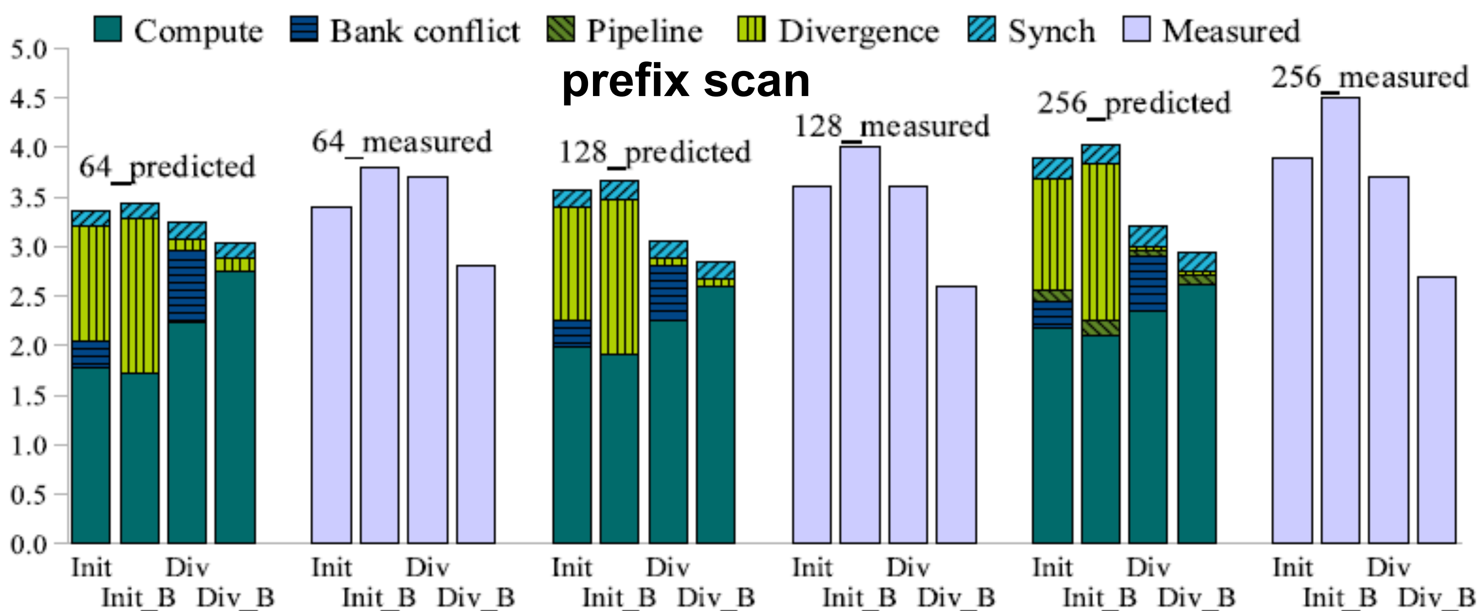
$$\underbrace{C_6^{24} S^{cs} C_4^{16 cc} S^{cs} C_1^{4 cc} S^{cs}}_{8\ \text{times}} B^4 C_2^8$$



Baghsorkhi and Hwu,
EPHAM 2009, PPoPP 2010



Compute
Memory
Measured
FFT
Execution Time (ms)
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Radix-2_predicted
Radix-2_measured
Radix-4_predicted
Radix-4_measured
Radix-16_predicted
Radix-16_measured
Shared
Global

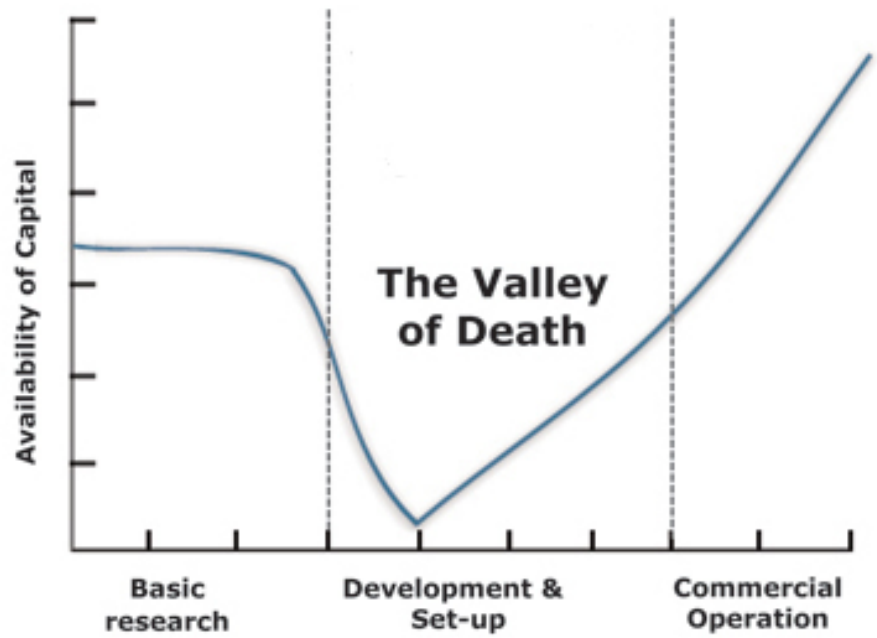


Compute
Bank conflict
Pipeline
Divergence
Synch
Measured
prefix scan
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
64_predicted
64_measured
128_predicted
128_measured
256_predicted
256_measured
Init
Div
Init_B
Div_B

Invitation for Collaboration

- Development and Validation of Scalable Kernel Libraries for Heterogeneous Computing
  – Linear algebra, graph algorithms, PDE solvers, Fourier methods, …
  – New methods/algorithms/implementations
  – Performance portability tools
  – Validation methodology and tools
  – Usable libraries

Crossing the Valley of Death



We can make it through the valley by collaborating with each other.

THANK YOU!