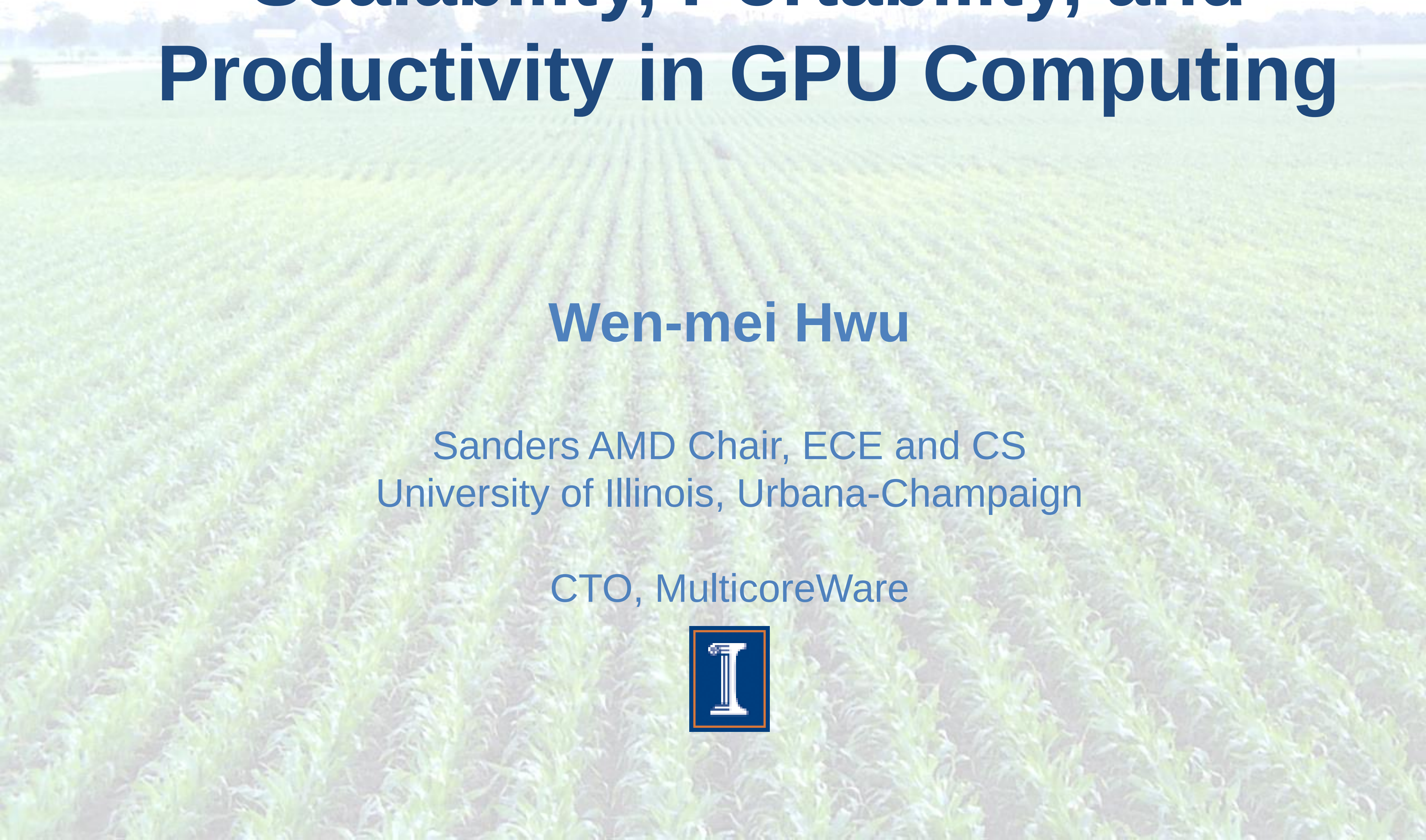
Scalability, Portability, and Productivity in GPU Computing

Wen-mei Hwu

Sanders AMD Chair, ECE and CS
University of Illinois, Urbana-Champaign

CTO, MulticoreWare



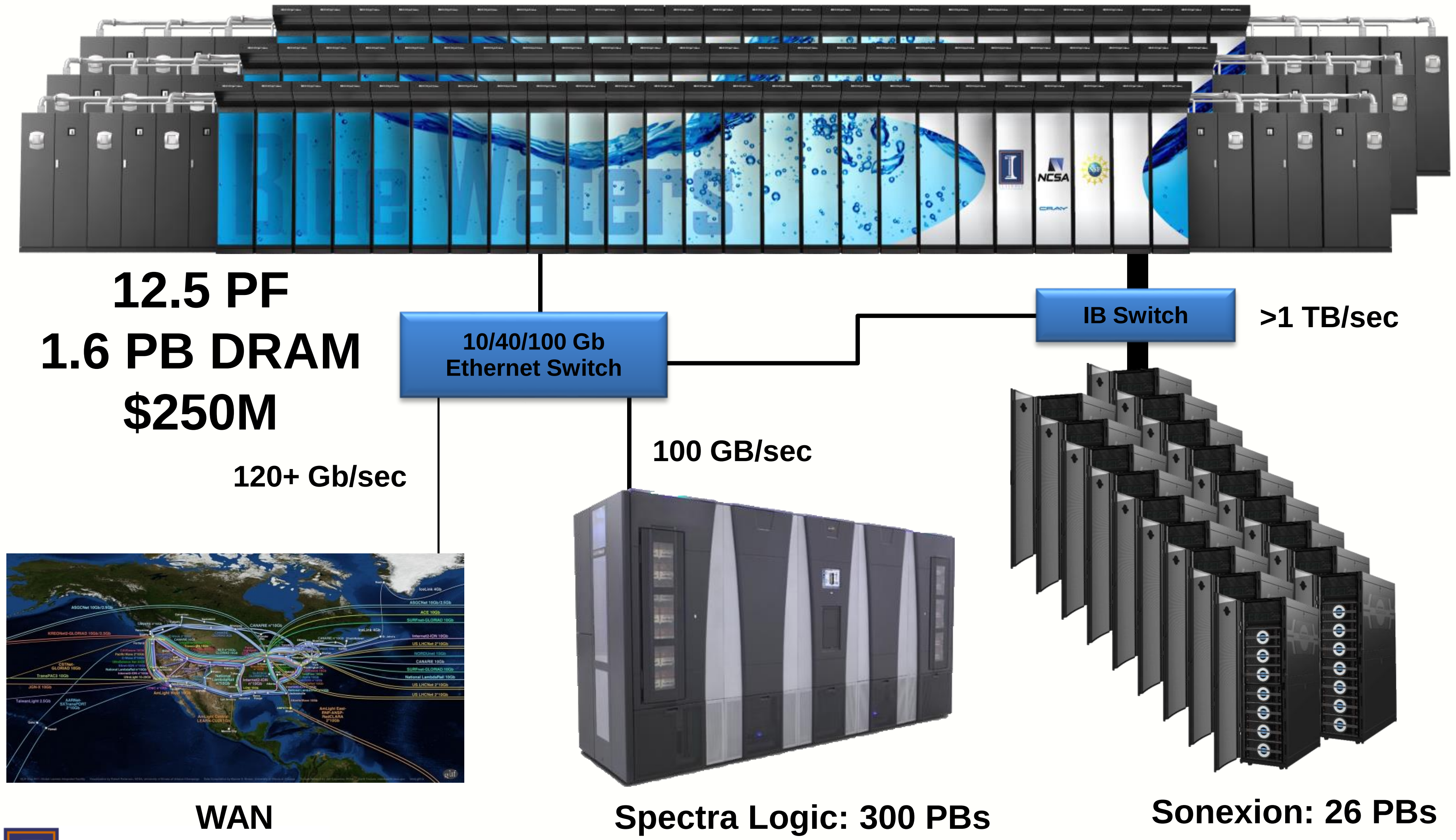
Agenda

- 4,224 Kepler GPUs in Blue Waters
- Programming Interfaces and Tools
- Conclusion and Outlook

Blue Waters Computing System

Operational at Illinois since 3/2013

12.5 PF
1.6 PB DRAM
$250M



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Blue Waters and Titan Computing Systems

System Attribute	NCSA Blue Waters	ORNL Titan
Vendors	Cray/AMD/NVIDIA	Cray/AMD/NVIDIA
Processors	Interlagos/Kepler	Interlagos/Kepler
Total Peak Performance (PF)	12.5	27.1
Total Peak Performance (CPU/GPU)	7.1/5.4	2.6/24.5
Number of CPU Chips	49,504	18,688
Number of GPU Chips	4,224	18,688
Amount of CPU Memory (TB)	1600	584
Interconnect	3D Torus	3D Torus
Amount of On-line Disk Storage (PB)	26	13.6
Sustained Disk Transfer (TB/sec)	>1	0.4-0.7
Amount of Archival Storage	300	15-30
Sustained Tape Transfer (GB/sec)	100	7

Why did we have only 4,224 GPUs in Blue Waters?

- Blue Waters will be the only Petascale machine for the NSF community for at least two years
  - Must minimize risk for petasacle application teams

- The NSF review panel was very concerned about the usability of GPUs in 2011
  - Small DRAM – up to 6GB
  - **Hard to program for application teams**
  - Lack of at-scale experience
  - Lack of begin-to-end production use experience

APPLICATIONS

At Scale, Begin-to-end execution including I/O

Science Area	Number of Teams	Codes	Struct Grids	Unstruct Grids	Dense Matrix	Sparse Matrix	N-Body	Monte Carlo	FFT	PIC	Sig I/O
Climate and Weather	3	CESM, GCRM, CM1/WRF, HOMME	X	X		X		X			X
Plasmas/ Magnetosphere	2	H3D(M),VPIC, OSIRIS, Magtail/UPIC	X				X		X		X
Stellar Atmospheres and Supernovae	5	PPM, MAESTRO, CASTRO, SEDONA, ChaNGa, MS-FLUKSS	X			X	X	X		X	X
Cosmology	2	Enzo, pGADGET	X			X	X				
Combustion/ Turbulence	2	PSDNS, DISTUF	X						X		
General Relativity	2	Cactus, Harm3D, LazEV	X			X					
Molecular Dynamics	4	AMBER, Gromacs, **NAMD**, LAMMPS				X	X		X		
Quantum Chemistry	2	SIAL, **GAMESS**, NWChem			X	X	X	X			X
Material Science	3	NEMOS, OMEN, GW, **QMCPACK**			X	X	X	X			
Earthquakes/ Seismology	2	AWP-ODC, HERCULES, PLSQR, SPECFEM3D	X	X			X				X
Quantum Chromo Dynamics	1	**Chroma,** MILC, USQCD	X		X	X					
Social Networks	1	EPISIMDEMICS									
Evolution	1	Eve									
Engineering/System of Systems	1	GRIPS,Revisit						X			
Computer Science	1			X	X	X			X		X

Current Science Team Use of GPUs

- ***About 1/3 of PRAC projects have active GPU efforts, including***
  - AMBER
  - LAMMPS
  - USQCD/**Chroma**/MILC
  - **GAMESS**
  - **NAMD**
  - **QMCPACK**
  - PLSQR/SPECFEM3D
  - PHOTONPLASMA
  - AWP-ODC
- Others are investigating use of GPUs (e.g., Cactus, PPM,, MS-FLUKSS)

Initial Production Use Results

- NAMD
  - 100 million atom benchmark with Langevin dynamics and PME once every 4 steps, from launch to finish, all I/O included
  - 768 nodes, Kepler+Interlagos is 3.9X faster over Interlagos-only
  - 768 nodes, XK7 is 1.8X XE6
- Chroma
  - Lattice QCD parameters: grid size of 48^3 x 512 running at the physical values of the quark masses
  - 768 nodes, Kepler+Interlagos is 4.9X faster over Interlagos-only
  - 768 nodes, XK7 is 2.4X XE6
- QMCPACK
  - Full run Graphite 4x4x1 (256 electrons), QMC followed by VMC
  - 700 nodes, Kepler+Interlagos is 4.9X faster over Interlagos-only
  - 700 nodes, XK7 is 2.7X XE6

Eight Techniques for Scalable Kernels

	Memory Bandwidth	Update Contention	Load Balance	Regularity	Efficiency
Scatter to Gather		X			
Privatization		X			
Tiling	X				X
Coarsening	X	X			X
Data Layout	X	X			X
Input Binning	X				X
Regularization			X	X	X
Compaction	X		X	X	X

Stratton, et al, IEEE Computer, 8/2012

Numerical Error and Stability

(case study: tridiagonal solver)

Relative Backward Error

Matrix type	SPIKE-diag_pivoting	SPIKE-Thomas	CUSPARSE	MKL	Intel SPIKE	Matlab
1	1.82E-14	1.97E-14	7.14E-12	1.88E-14	1.39E-15	1.96E-14
2	1.27E-16	1.27E-16	1.69E-16	1.03E-16	1.02E-16	1.03E-16
3	1.55E-16	1.52E-16	2.57E-16	1.35E-16	1.29E-16	1.35E-16
4	1.37E-14	1.22E-14	1.39E-12	3.10E-15	1.69E-15	2.78E-15
5	1.07E-14	1.13E-14	1.82E-14	1.56E-14	4.62E-15	2.93E-14
6	1.05E-16	1.06E-16	1.57E-16	9.34E-17	9.51E-17	9.34E-17
7	2.42E-16	2.46E-16	5.13E-16	2.52E-16	2.55E-16	2.27E-16
8	2.14E-04	2.14E-04	1.50E+10	3.76E-04	2.32E-16	2.14E-04
9	2.32E-05	3.90E-04	1.93E+08	3.15E-05	9.07E-16	1.19E-05
10	4.27E-05	4.83E-05	2.74E+05	3.21E-05	4.72E-16	3.21E-05
11	7.52E-04	6.59E-02	4.54E+11	2.99E-04	2.20E-15	2.28E-04
12	5.58E-05	7.95E-05	5.55E-04	2.24E-05	5.52E-05	2.24E-05
13	5.51E-01	5.45E-01	1.12E+16	3.34E-01	3.92E-15	3.08E-01
14	2.86E+49	4.49E+49	2.92E+51	1.77E+48	3.86E+54	1.77E+48
15	2.09E+60	Nan	Nan	1.47E+59	Fail	3.69E+58
16	Inf	Nan	Nan	Inf	Fail	4.68E+171

ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Chang, et al, SC2012, new NVIDIA CUSPARSE

PROGRAMMING INTERFACES AND TOOLS

Writing efficient parallel code is complicated.

Planning how to execute an algorithm

- Distribute computation across
  - cores,
  - hardware threads, and
  - vector processing elements
- Distribute data across
  - discrete GPUs or
  - clusters
- Orchestrate communication for
  - reductions,
  - variable-size list creation,
  - stencils, etc.

Implementing the plan

- Rearrange data for locality
- Fuse or split loops
- Map loop iterations onto hardware

- Allocate memory
- Partition data
- Insert data movement code

- Reduction trees
- Array packing
- Boundary cell communication

Levels of GPU Programming Interfaces

Prototype & in development X10, Chapel, Nesl, Delite, Par4all, Triolet/Tangram...

Implementation manages GPU threading and synchronization invisibly to user

Next generation OpenACC, C++AMP, Thrust, Bolt

Simplifies data movement, kernel details and kernel launch

Same GPU execution model (but less boilerplate)

Current generation CUDA, OpenCL, DirectCompute

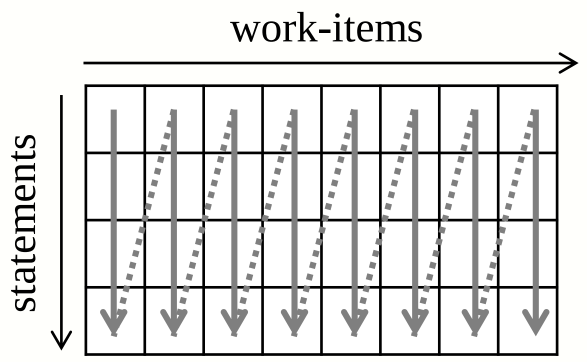
ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

LOW-LEVEL INTERFACE

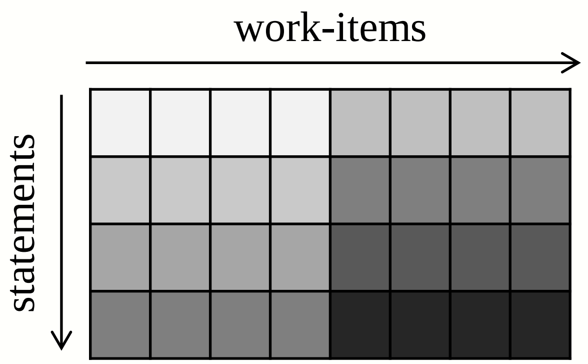
CPU vs. GPU Code Versions

- Maintaining multiple code versions is extremely expensive
- Most CUDA/OpenCL developers maintain original CPU version
- Many developers report that when they back ported the CUDA/OpenCL algorithms to CPU, they got better performing code
  - Locality, SIMD, multicore
- MxPA is designed to automate this process *(John Stratton, Hee-Seok Kim, Izzat El Hajj)*

Treatment of Work-Items



time goes in direction of arrows



time goes from lighter to darker

```
i = get_global_id(0);
statement1(i);
statement2(i);
...
statementM(i);
```

Original OpenCL kernel

```
for(i=0; i<N; ++i){
  statement1(i);
  statement2(i);
  ...
  statementM(i);
}
```

Serialization-based work-item treatment

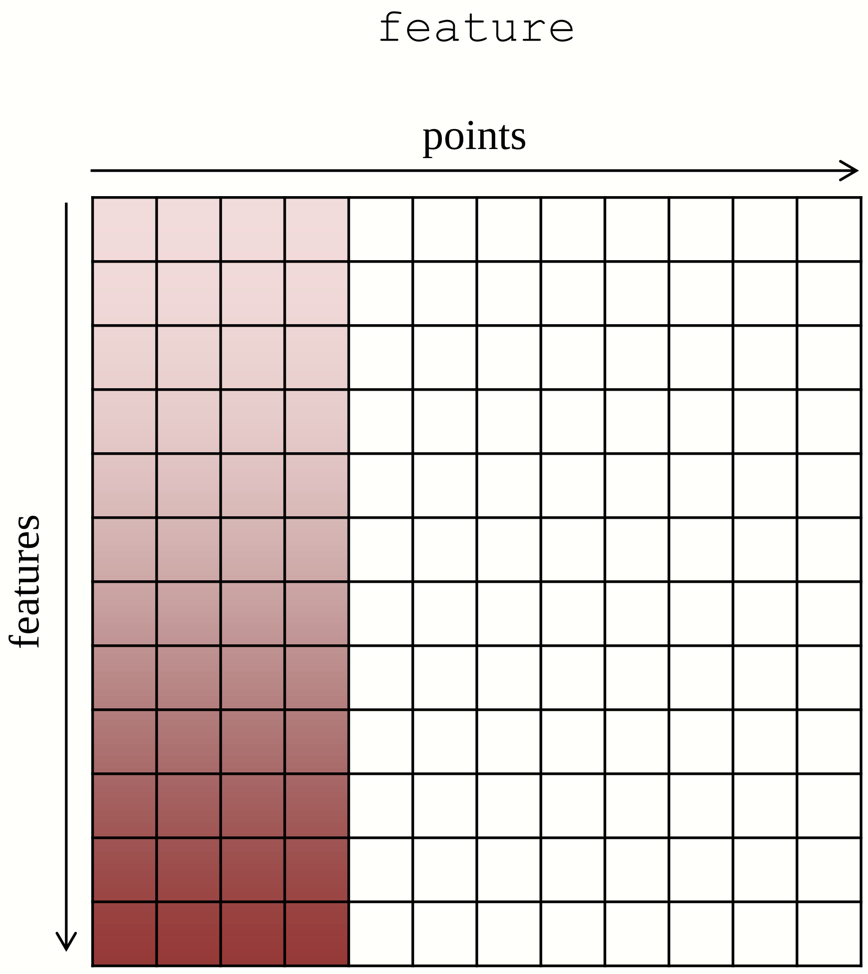
```
for(x=0; x<N; x+=S){
  statement1(i:S);
  statement2(i:S);
  ...
  statementM(i:S);
}
```

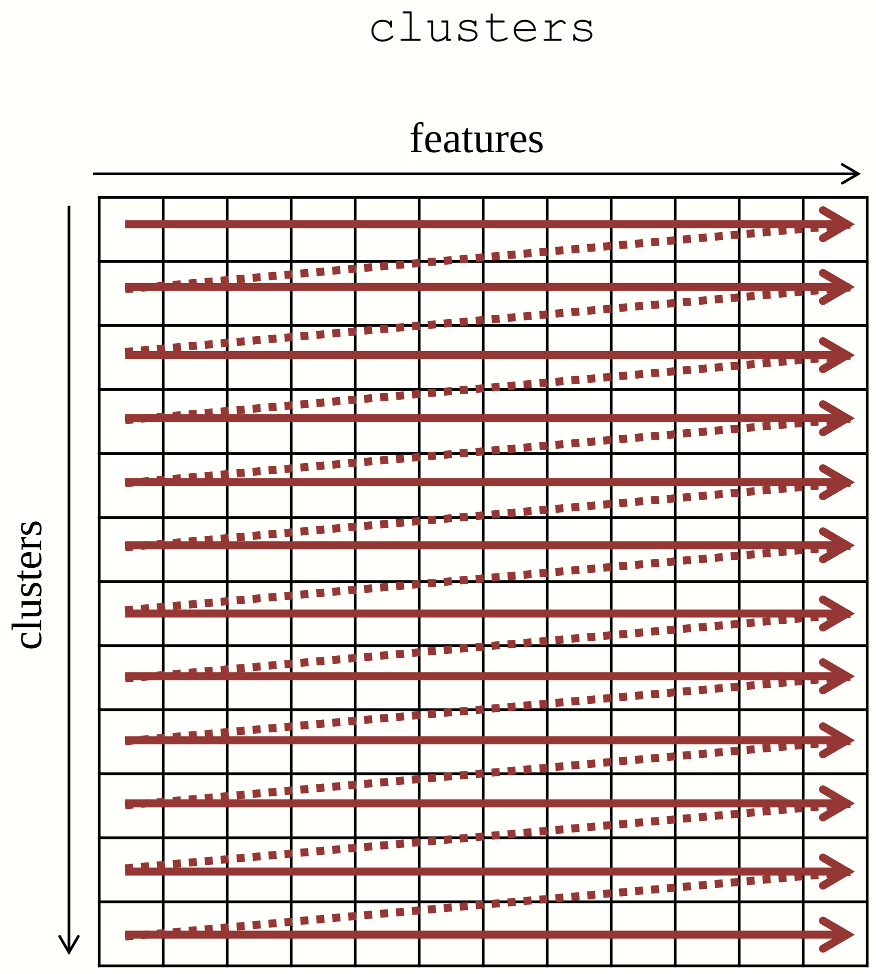
Vectorization-based work-item treatment

Example: K-means (Rodinia)

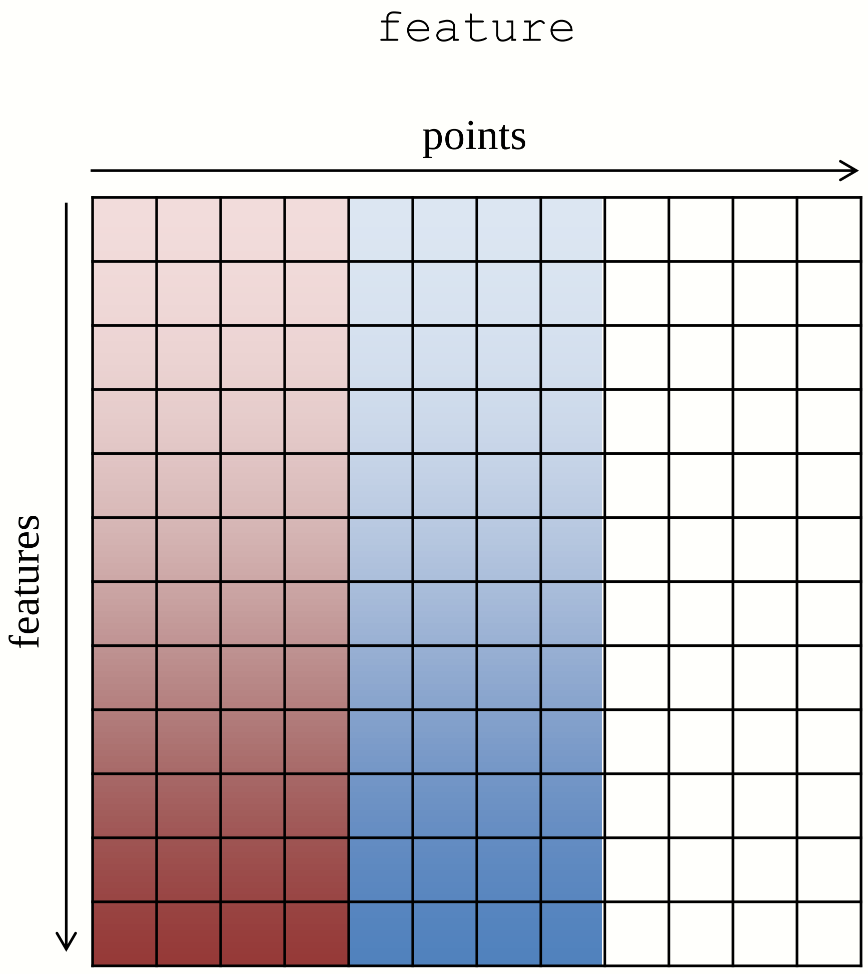
```
if (point_id < npoints) {
  float min_dist=FLT_MAX;
  for (int i=0; i < nclusters; i++) {
    float dist = 0;
    float ans  = 0;
    for (int l=0; l<nfeatures; l++) {
      ans +=
        (feature[l*npoints+point_id] - clusters[i*nfeatures+l])
        *(feature[l*npoints+point_id] - clusters[i*nfeatures+l]);
    }
    dist = ans;
    if (dist < min_dist) {
      min_dist = dist;
      index = i;
    }
  }
  membership[point_id] = index;
}
```

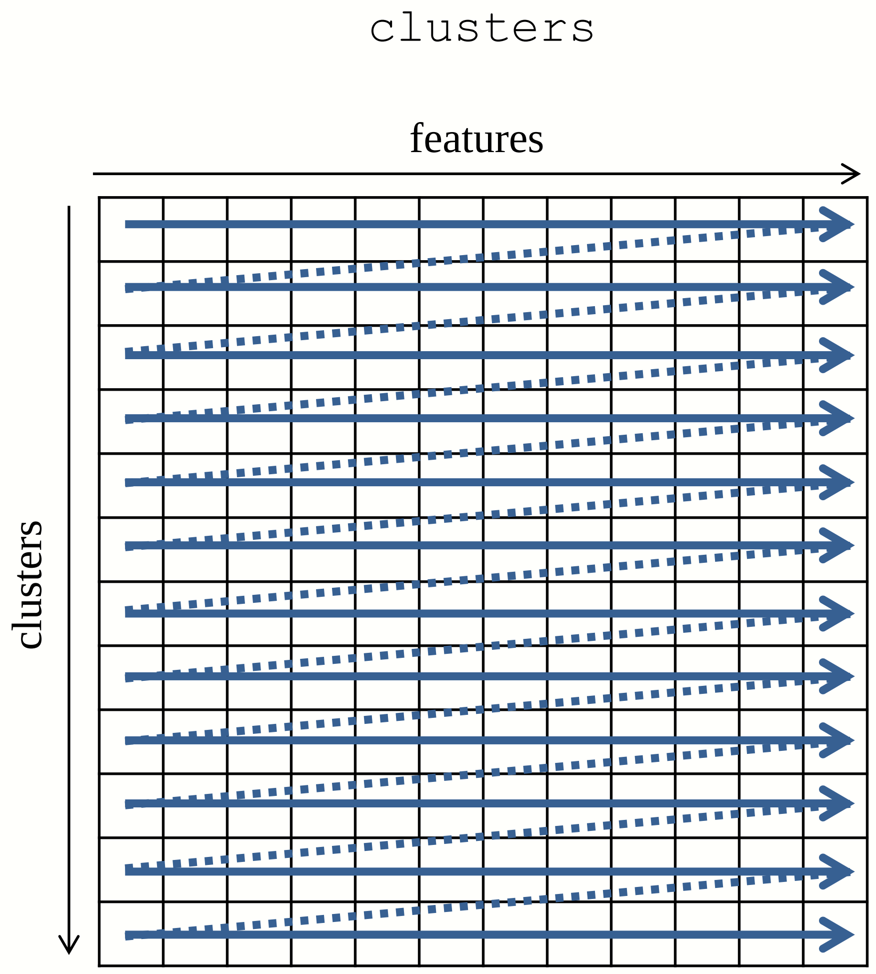
Example: K-means (Rodinia)



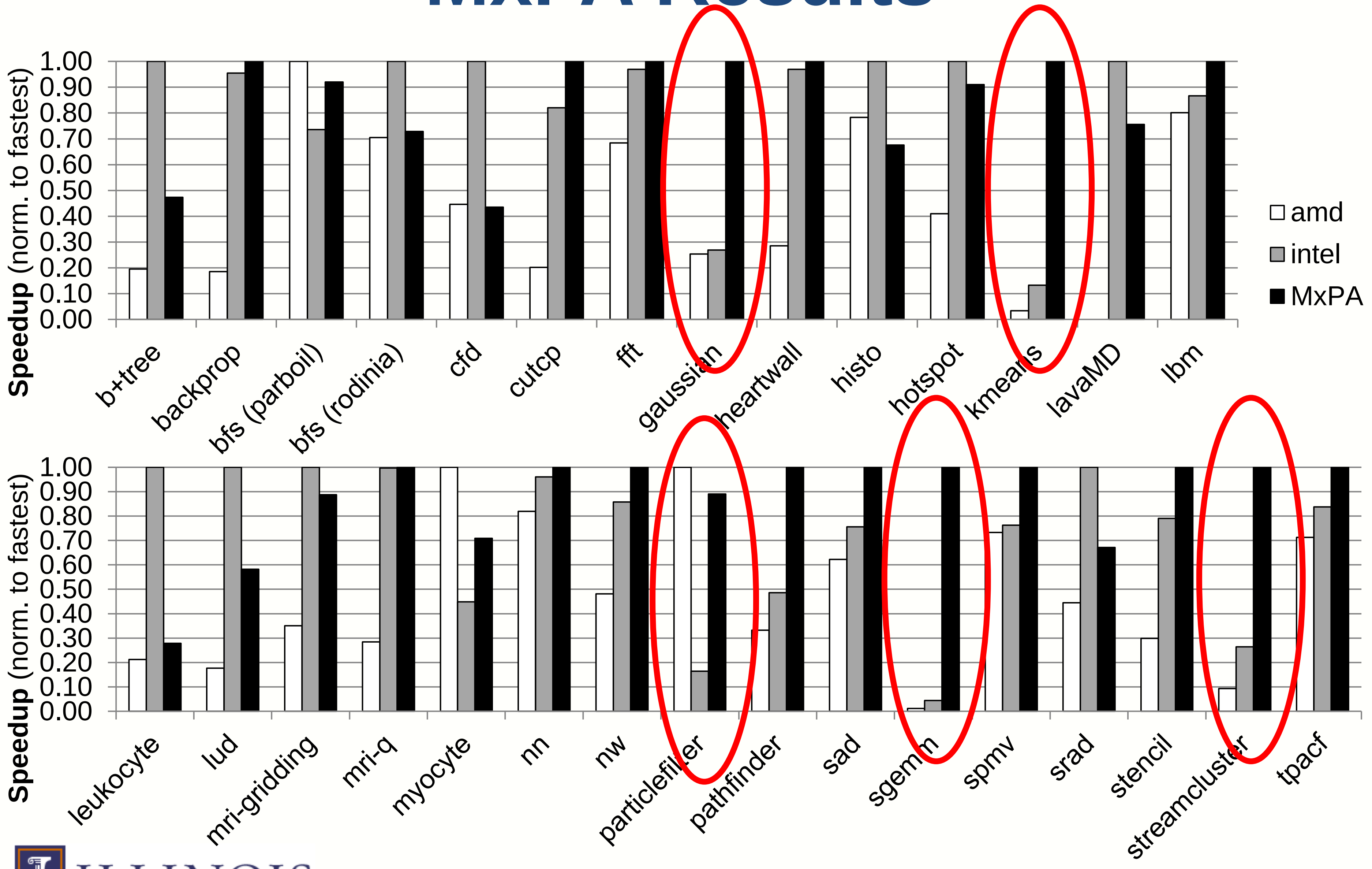


Example: K-means (Rodinia)



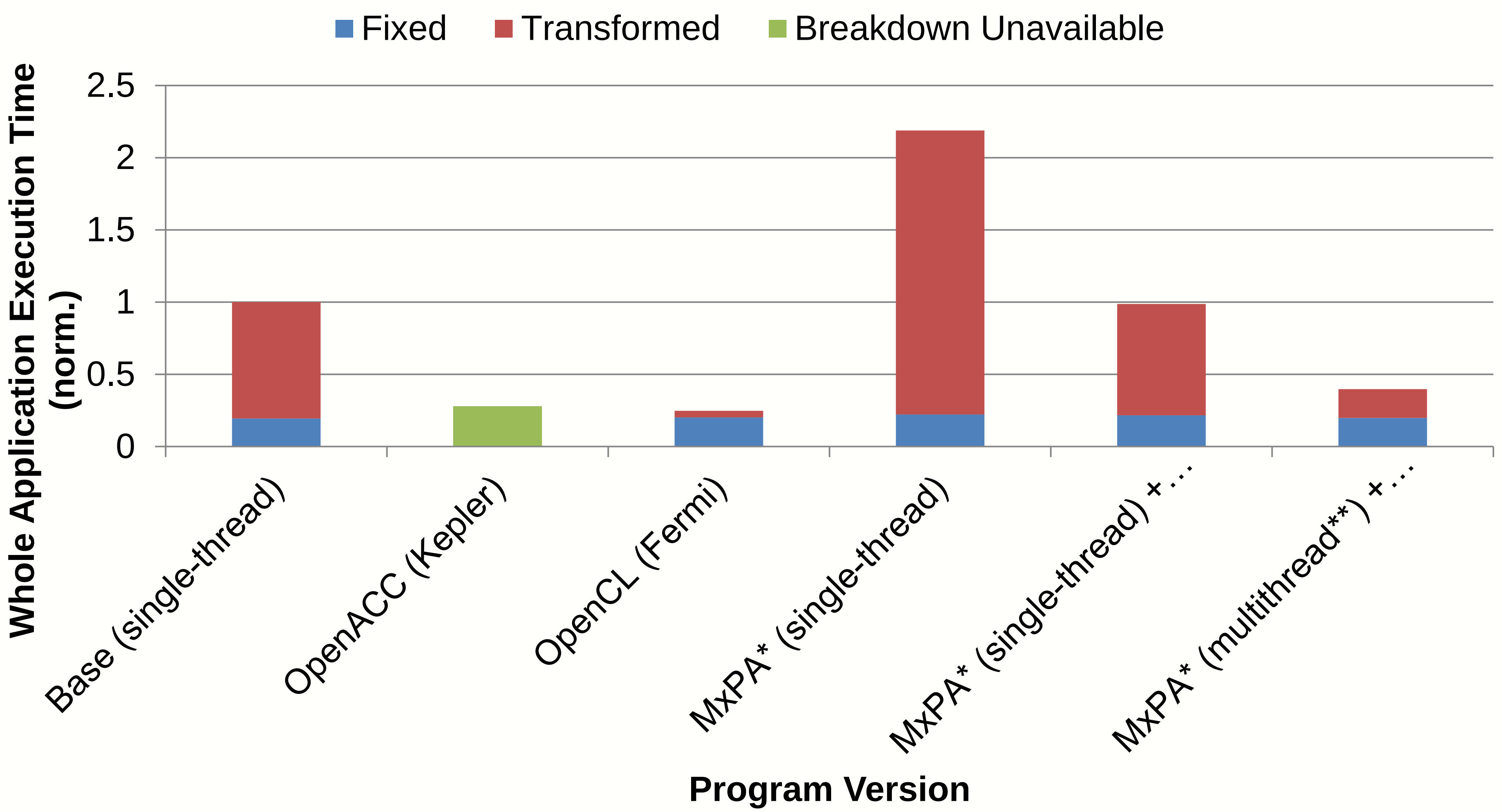


MxPA Results



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

MxPA MOCFE-Bone Results



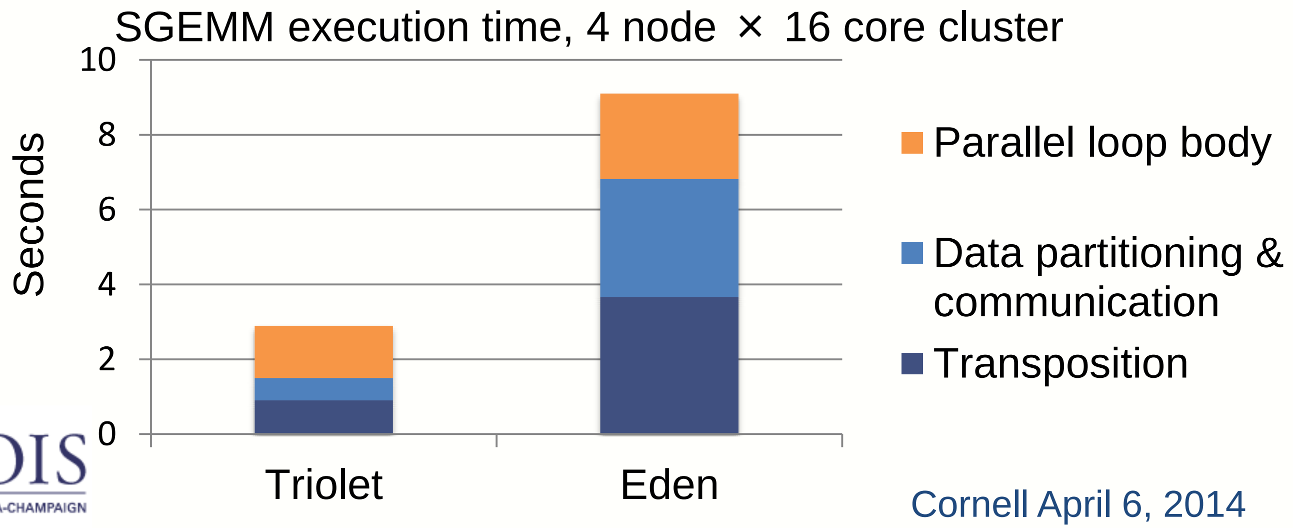
Configurations: Nodes = 1, Groups = 25, Angles = 128, MeshScale=10 (Elements=10^3)

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

HIGH-LEVEL INTERFACE

High-Level Languages: Does relinquishing control mean giving up performance?

- High-level tools are making strides in usability, generality, and performance
- Typically designed to be effective on a small problem domain
- Performance lost from
  - Optimization heuristics outside programmer's control (e.g., vectors vs. threads)
  - Requiring programmers to reformulate algorithm in ways that add computation
- Need flexible languages that accept programmer hints for performance-critical decisions



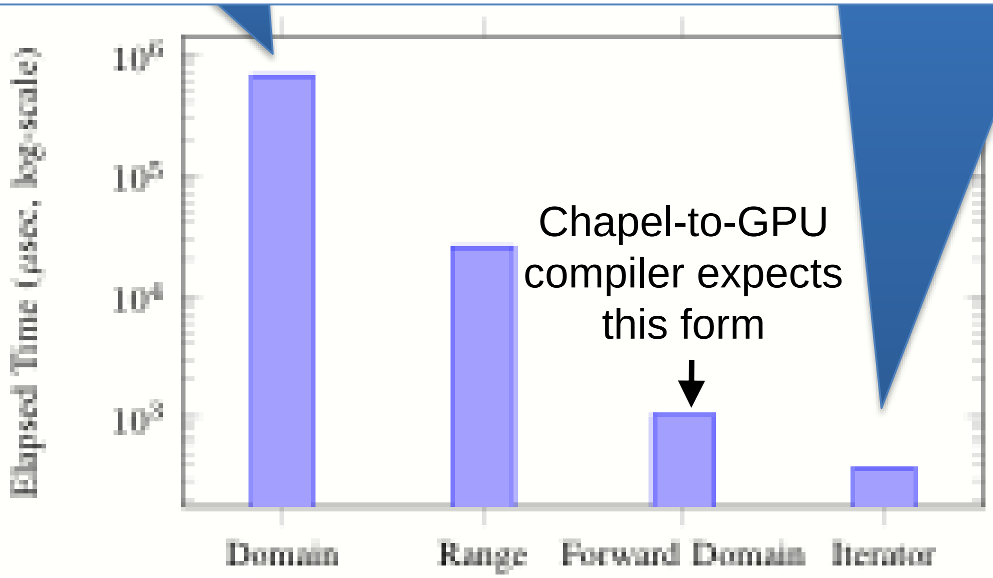
Huge overhead is often caused by the generality of the interface.

- Many performance pitfalls are fixable, but will still cause problems for novices

```
for j in [1..N] do ...;
```

```
iter myIter(min:int, max:int, step:int=1) {
  while min <= max {
    yield min;
    min += step;
}}
for j in myIter(1,N) do ...;
```

Time to execute a one-iteration loop on CPU in Chapel



Source: Dun and Taura, IPDPSW 2012

Who does the hard work in parallelization?

- General-purpose language + parallelizing compiler
  - Requires a very intelligent compiler
  - Limited success outside of regular array algorithms
- Delite - Domain-specific language + domain-specific compiler
  - Simplify compiler's job with language restrictions and extensions
  - Requires customizing a compiler for each domain
- Triolet - Parallel library + general-purpose compiler
  - Library makes parallelization decisions
  - Uses a general-purpose, rich transformation compiler
  - Extensible—just add library functions

Who does the hard work in parallelization?

- Triolet - Parallel library + general-purpose compiler
  - Library makes parallelization decisions
  - Uses a general-purpose, rich transformation compiler
  - Extensible—just add library functions

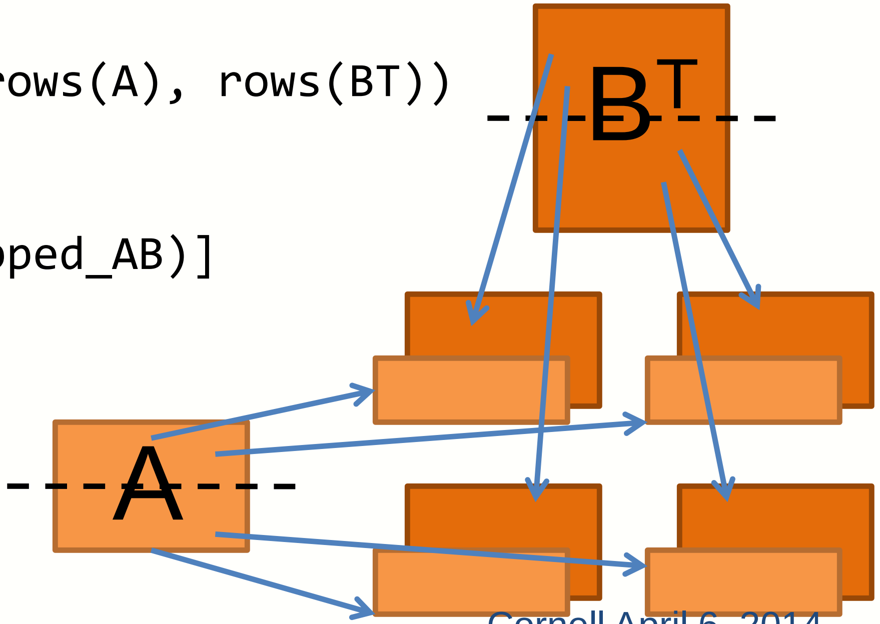
ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Example: 2D Data Distribution on a Cluster

- Matrix multiplication has a block-based data decomposition
- Difficult to write manually, but still a simple and regular pattern
- Triolet library provides functions for **looping over a data structure**
  - Expresses parallelism and access pattern together
- This is the entire algorithm:

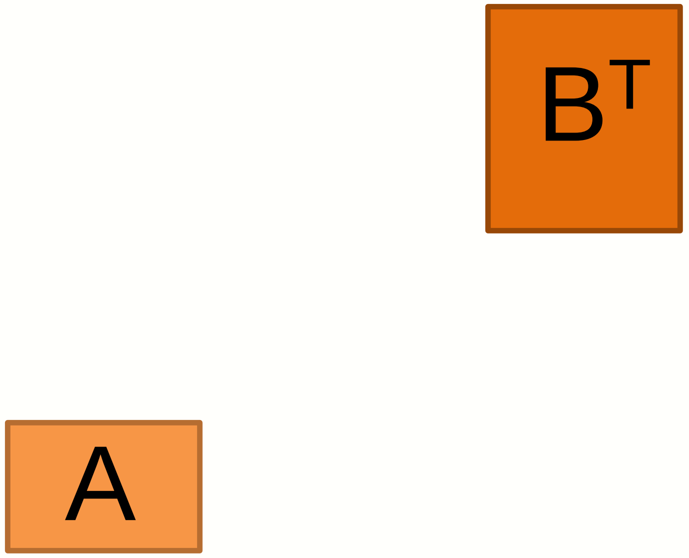
```
# Matrix multiplication in Triolet

zipped_AB = outerproduct(rows(A), rows(BT))

C = [dotproduct(a, b)
     for (a, b) in par(zipped_AB)]
```

2D blocked matrix multiplication



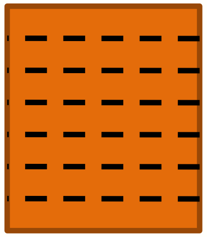
Data Decomposition in Triolet

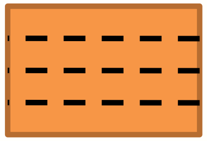
```
#  Matrix multiplication in Triolet
```



Data Decomposition in Triolet

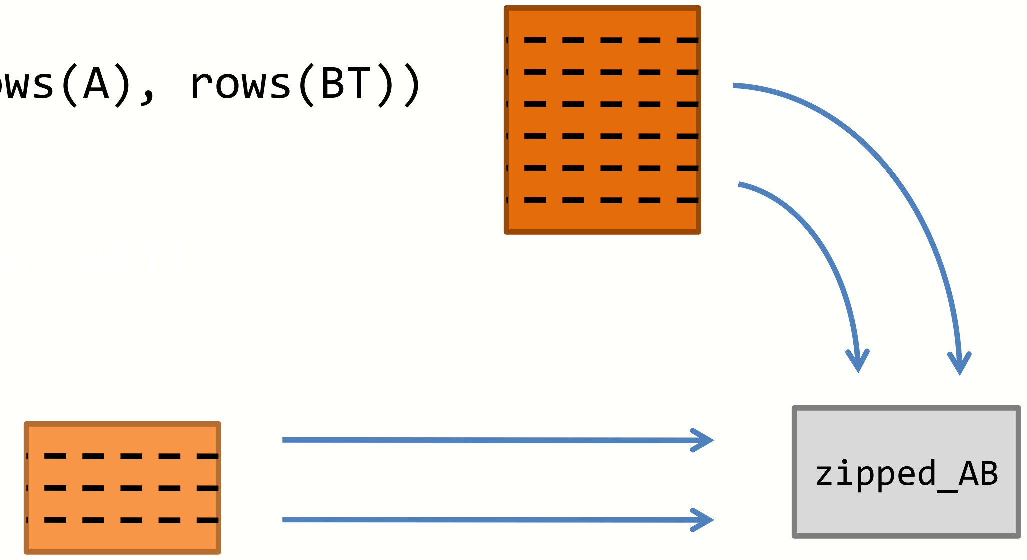
```
# Matrix multiplication in Triolet

                         rows(A), rows(BT)
```





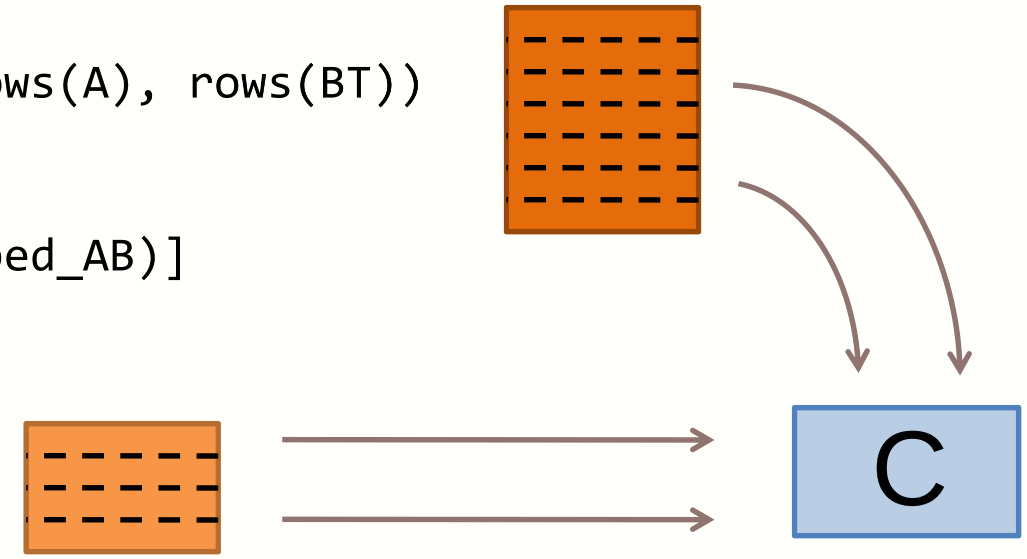
Data Decomposition in Triolet

```
#  Matrix multiplication in Triolet

zipped_AB = outerproduct(rows(A), rows(BT))
```



Data Decomposition in Triolet

```
# Matrix multiplication in Triolet

zipped_AB = outerproduct(rows(A), rows(BT))

C = [dotproduct(a, b)
     for (a, b) in par(zipped_AB)]
```

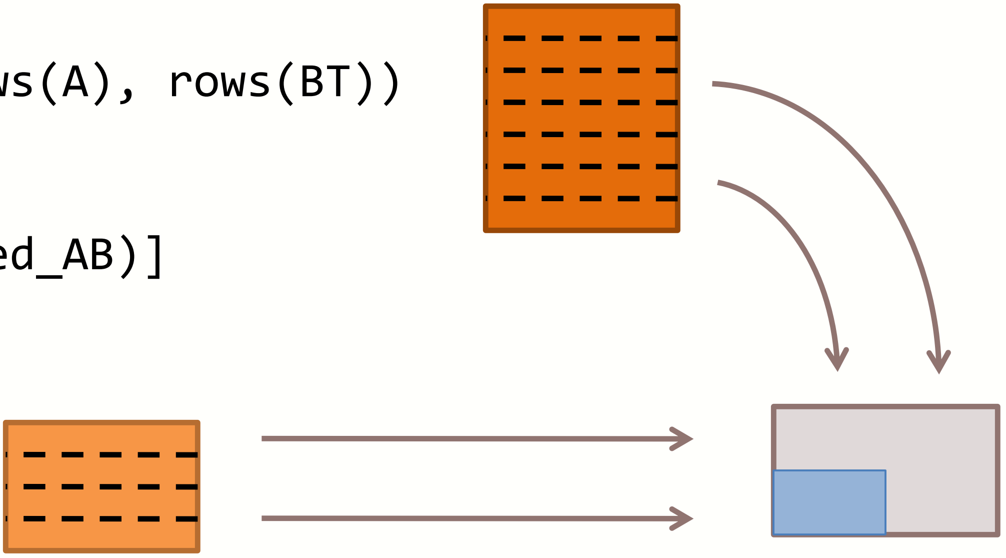


Data Decomposition in Triolet

- Parallel loop assigns a range of output to each cluster node

```
# Matrix multiplication in Triolet

zipped_AB = outerproduct(rows(A), rows(BT))

C = [dotproduct(a, b)
     for (a, b) in par(zipped_AB)]
```

Give me input range (0,100)–(99, 199)



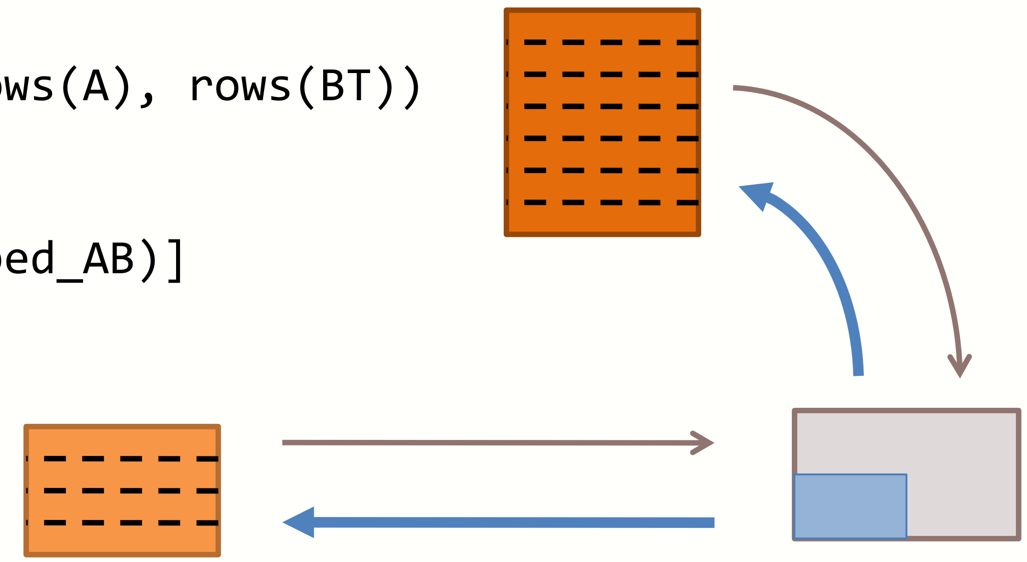
Data Decomposition in Triolet

- Parallel loop assigns a range of output to each cluster node
- Library functions translate output ranges into input ranges

Give me input range (0,99) from rows(A)
Give me input range (100, 199) from rows(BT)

```
# Matrix multiplication i

zipped_AB = outerproduct(rows(A), rows(BT))

C = [dotproduct(a, b)
     for (a, b) in par(zipped_AB)]
```

Give me input range (0,100)–(99, 199)



Data Decomposition in Triolet

- Parallel loop assigns a range of output to each cluster node
- Library functions translate output ranges into input ranges
- and find the subarrays to send to each node

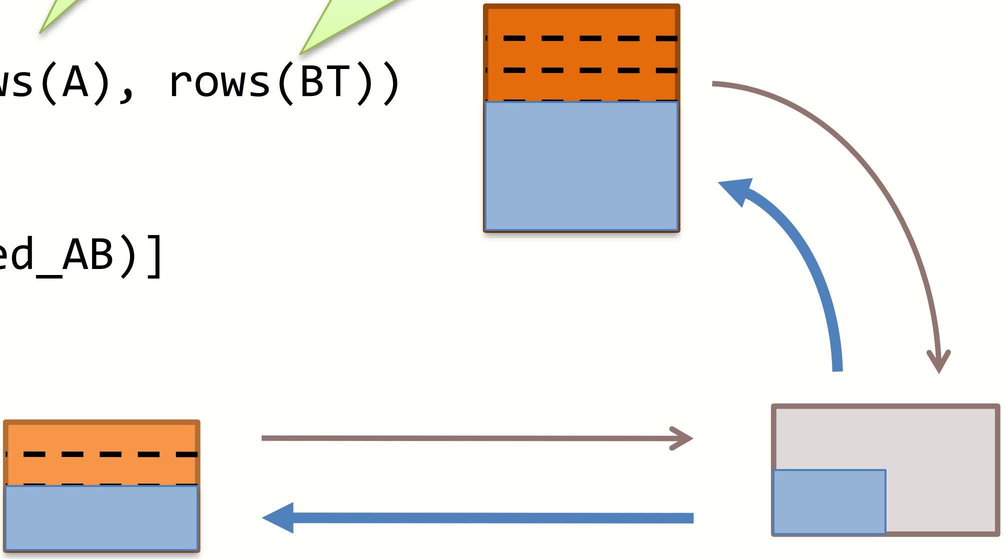
Take rows 0–99 of A

Give me input range (0,99) from rows(A)
Give me input range (100, 199) from rows(BT)

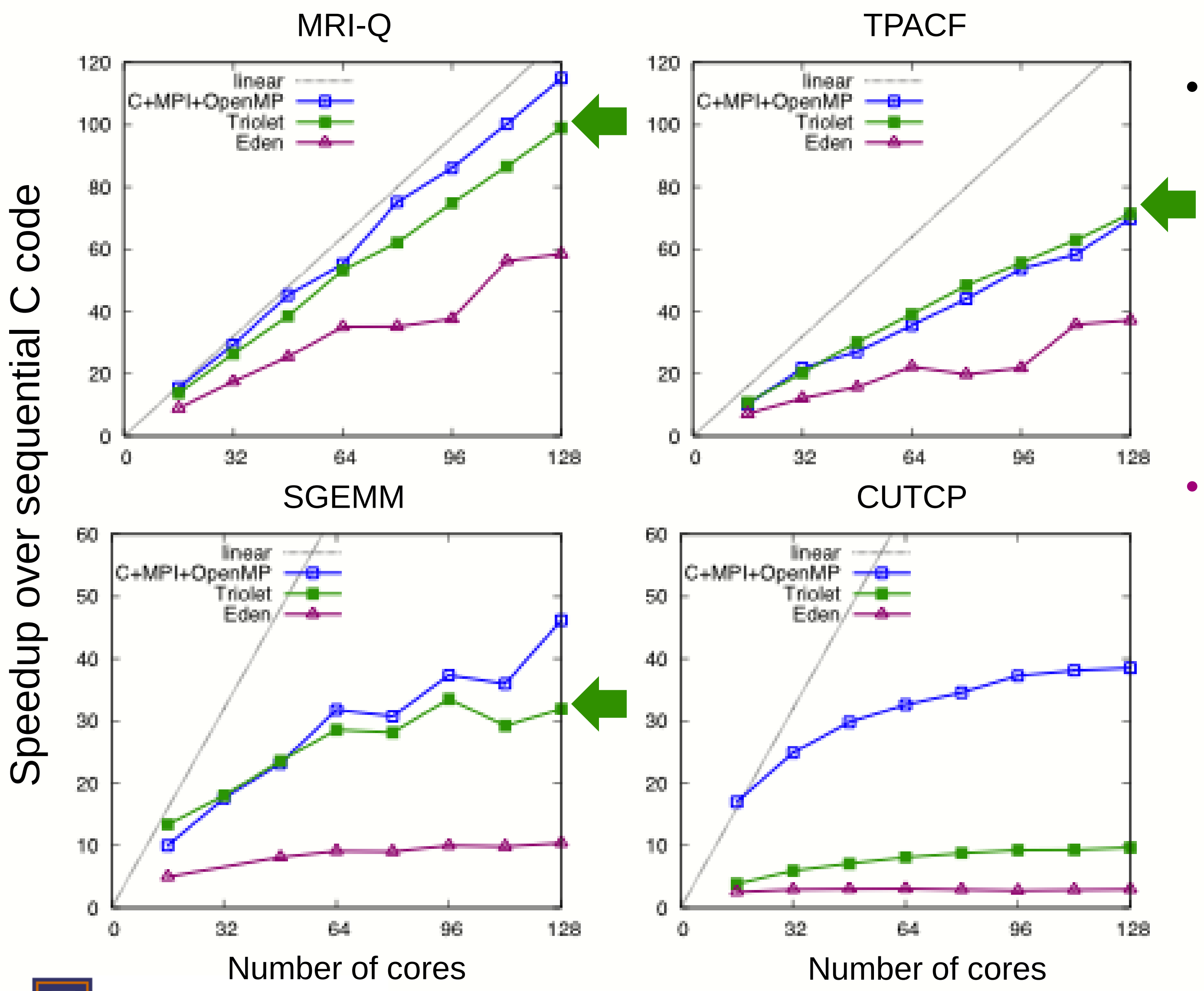
Take rows 100–199 of B

```
# Matrix multiplication i

zipped_AB = outerproduct(rows(A), rows(BT))

C = [dotproduct(a, b)
     for (a, b) in par(zipped_AB)]
```

Give me input range (0,100)–(99, 199)



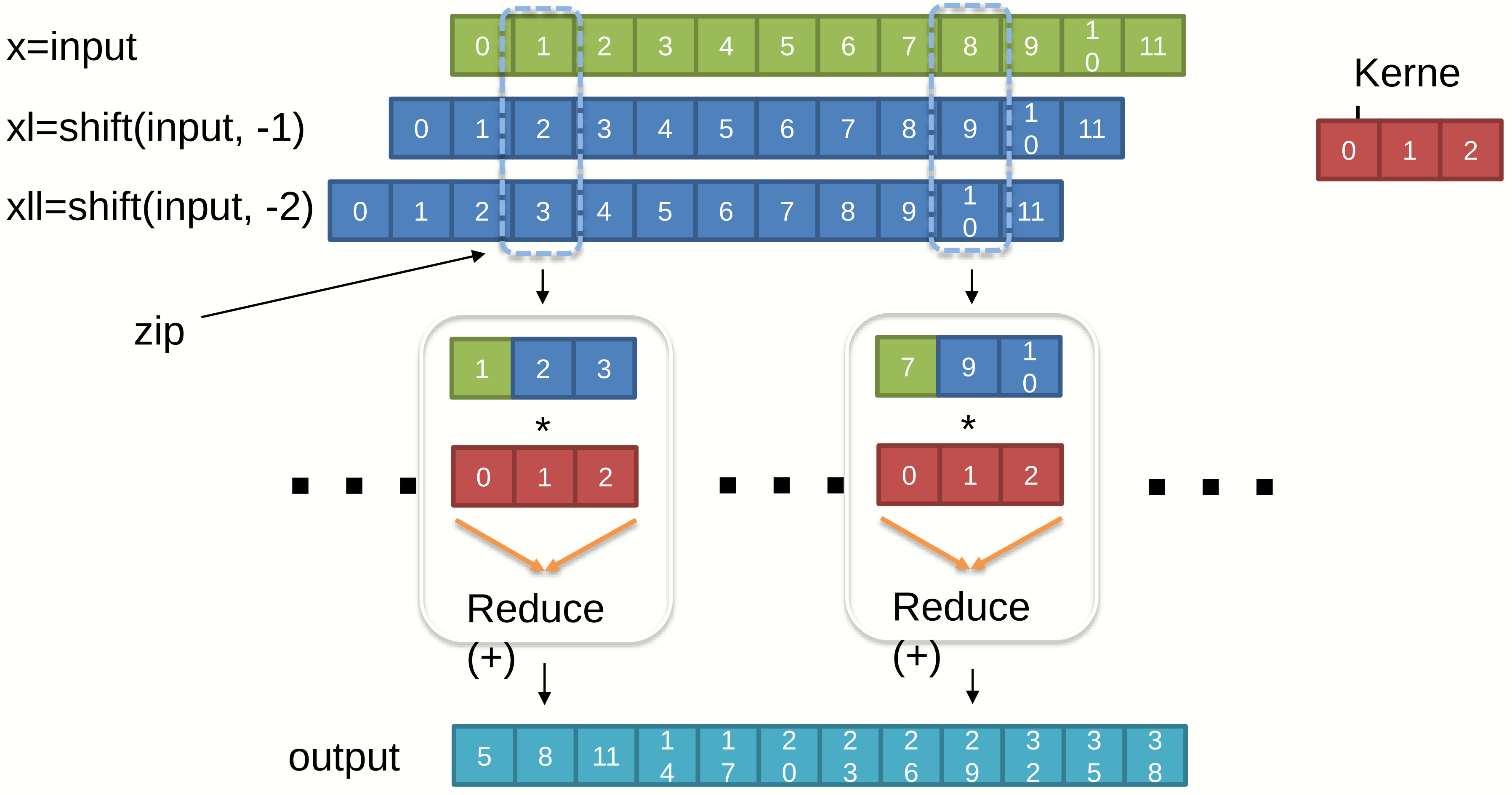
Cluster-Parallel Performance and Scalability



- Triolet delivers large speedup over sequential C
  - On par with manually parallelized C
  - Except in CUTCP; needs better GC policy for large arrays
- Similar high-level interfaces incur additional overhead
  - Message passing
  - Array split/merge
  - Run time variability

Rodrigues, et al PPoPP 2014

Map/Reduce Example: 1D convolution

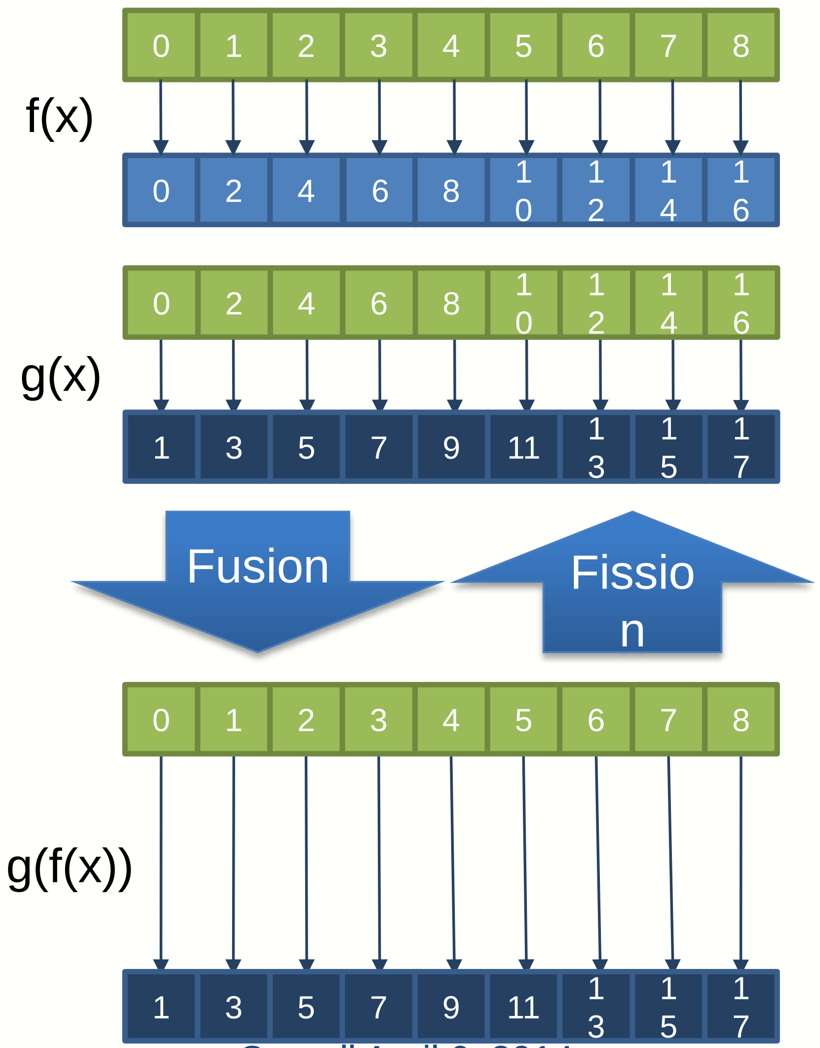


output = map(compute_kernel, zip(x, xl, xll)[:])

compute_kernel = reduce(+, map(*, kernel[:], zipped_elements[:]))

Loop Fusion of Map

- Fused
  - Increased locality
  - More execution resources per element
- Fission
  - f() and g() may have different dependencies -> expose parallelism by splitting them



An Opportunity of a Lifetime

- Scalable and portable software lasts through many hardware generations

Scalable algorithms and libraries could be the best legacy we can leave behind from this era

Conclusion and Outlook

- We have enjoyed some victories
  - Good set of applications and kernels
  - Good low-level interface in major languages
  - Good initial results, educated developers
- We will face more challenges
  - Potential fragmentation of programming interfaces
  - Widen the set of applications, algorithms and kernels
    - Analytics and machine learning
  - Productive, robust programming interfaces and tools

Acknowledgements

- D. August (Princeton), S. Baghsorkhi (Illinois), N. Bell (NVIDIA), D. Callahan (Microsoft), J. Cohen (NVIDIA), B. Dally (Stanford), J. Demmel (Berkeley), P. Dubey (Intel), M. Frank (Intel), M. Garland (NVIDIA), Isaac Gelado (BSC), M. Gschwind (IBM), R. Hank (Google), J. Hennessy (Stanford), P. Hanrahan (Stanford), M. Houston (AMD), T. Huang (Illinois), D. Kaeli (NEU), K. Keutzer (Berkeley), I. Gelado (UPC), B. Gropp (Illinois), D. Kirk (NVIDIA), D. Kuck (Intel), S. Mahlke (Michigan), T. Mattson (Intel), N. Navarro (UPC), J. Owens (Davis), D. Padua (Illinois), S. Patel (Illinois), Y. Patt (Texas), D. Patterson (Berkeley), C. Rodrigues (Illinois), S. Ryoo (ZeroSoft), K. Schulten (Illinois), B. Smith (Microsoft), M. Snir (Illinois), I. Sung (Illinois), P. Stenstrom (Chalmers), J. Stone (Illinois), S. Stone (Harvard) J. Stratton (Illinois), H. Takizawa (Tohoku), M. Valero (UPC)

- And many others!

There is always hope.

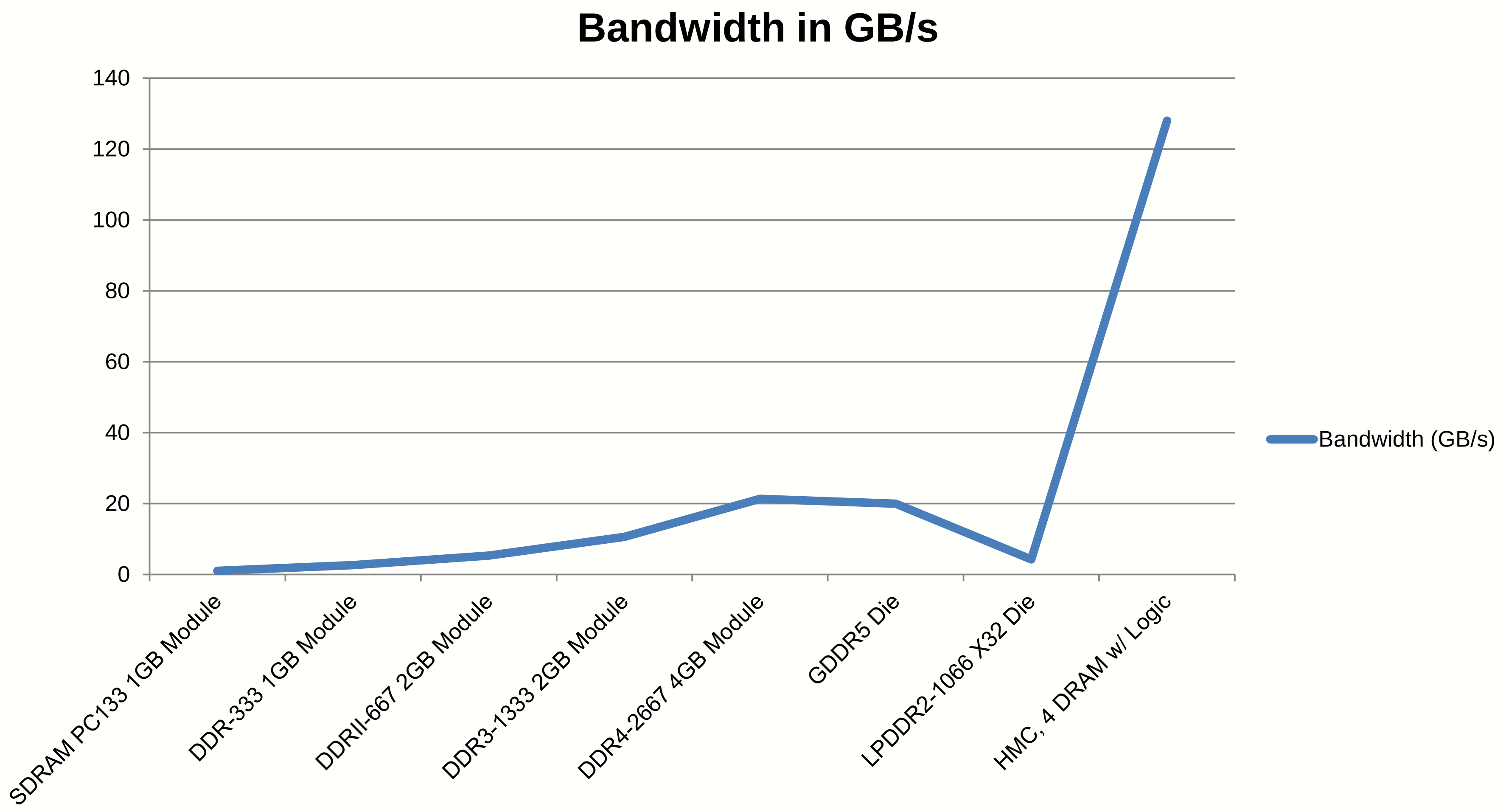
– Aragorn in the eve of the Battle of Pelennor Minas Tirith

THANK YOU!

ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

SOME IMPORTANT TRENDS

DRAM trends in bandwidth

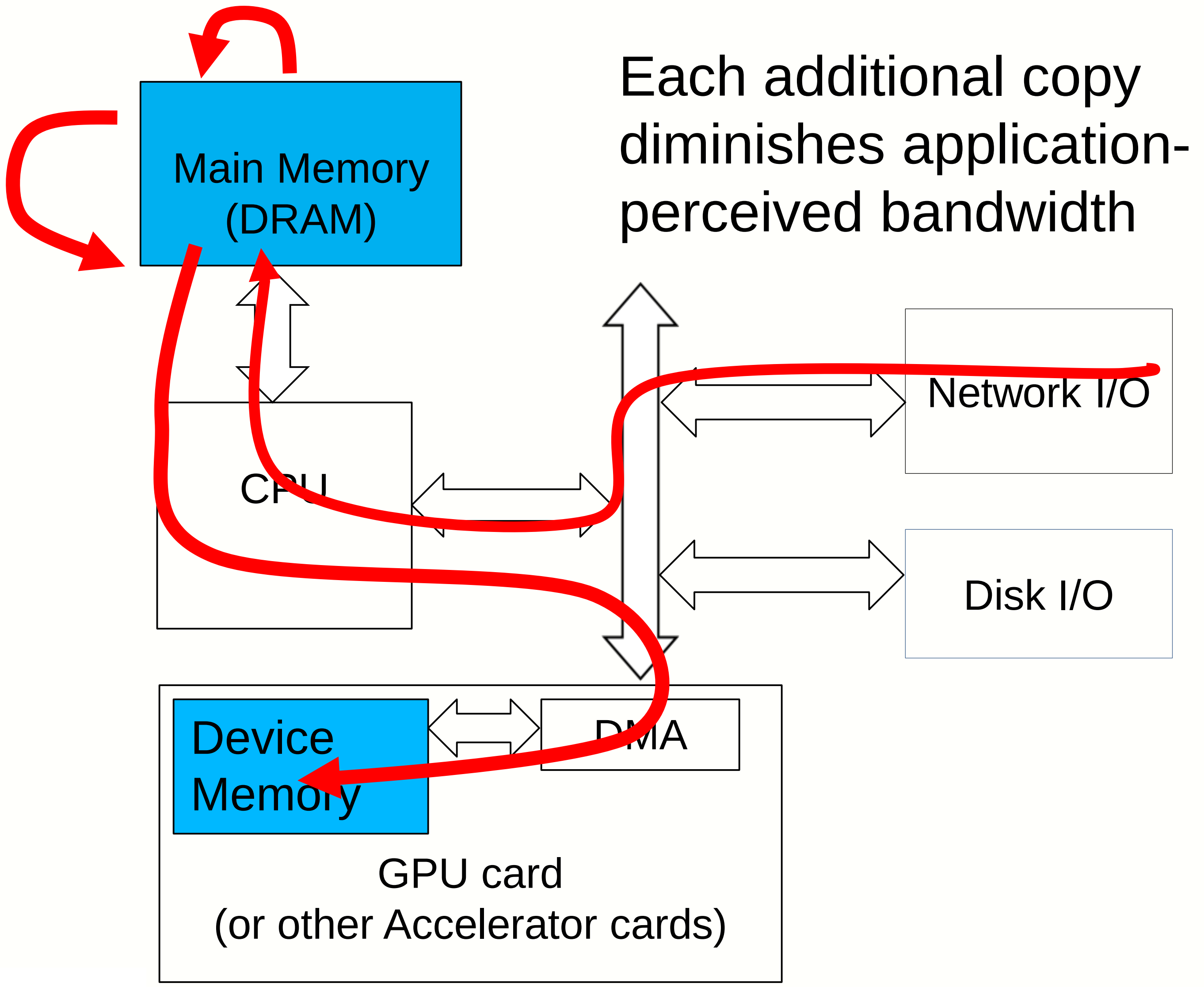


Source: J. Thomas Pawlowski, “Hybrid Memory Cube (HMC)”, Hot Chips 23

Exascale Energy Pressure

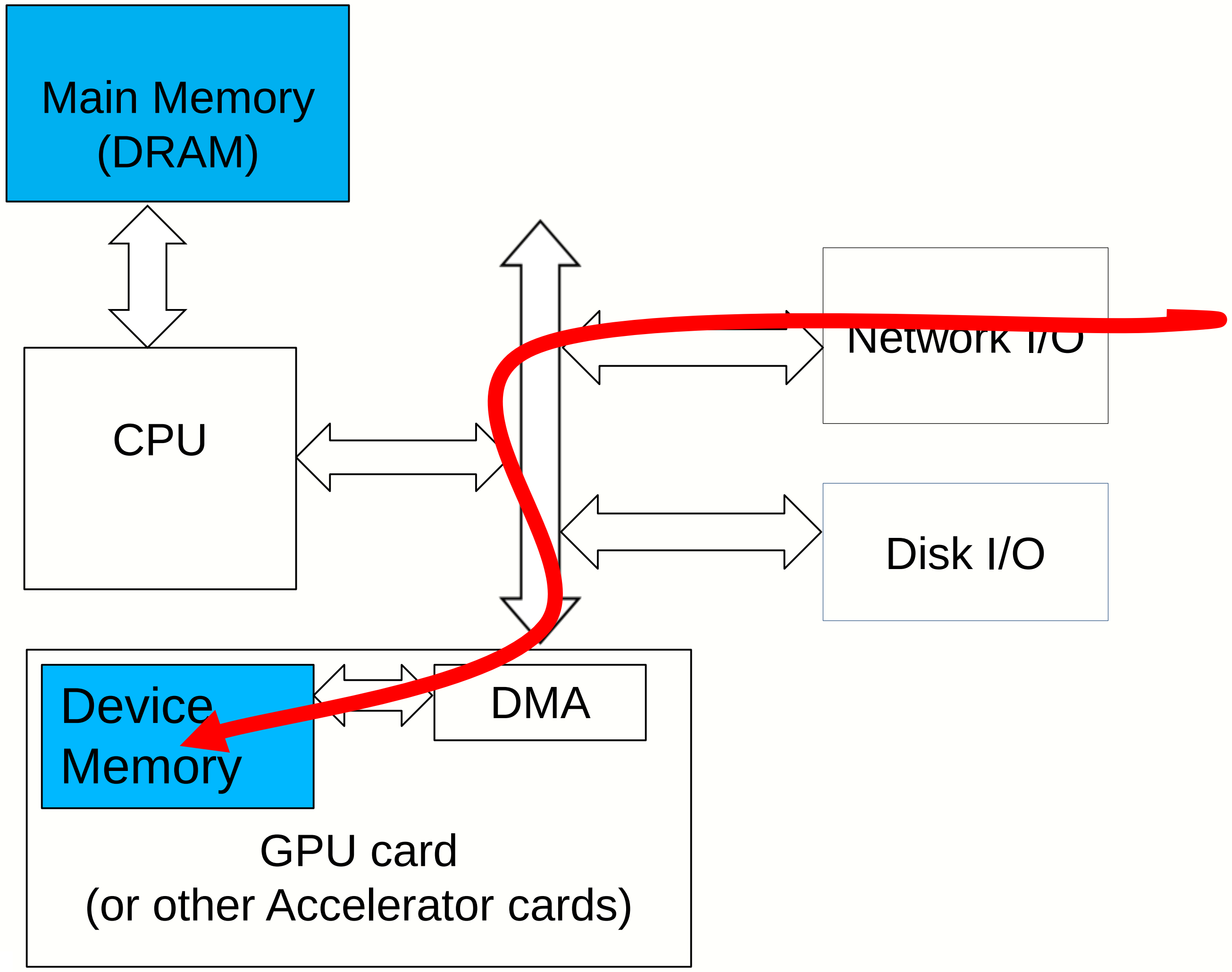
- Pressure for higher energy efficiency will likely make processors more difficult to program
  - More specialized processor data path (width, connectivity, etc.)
  - Wider SIMD
  - More system-level data movement control
  - Smaller on-chip storage per thread
  - …

Today's Data Transfer Behavior



Each additional copy diminishes application-perceived bandwidth

Desired Data Transfer Behavior with UVAS/UPAS/P2P DMA



Algorithm Design Challenges

Parallelism

- Parallelism to fill growing HW parallelism

Data Scalability

- Operations should grow linearly with data size

Locality

- DRAM burst and cache space utilization

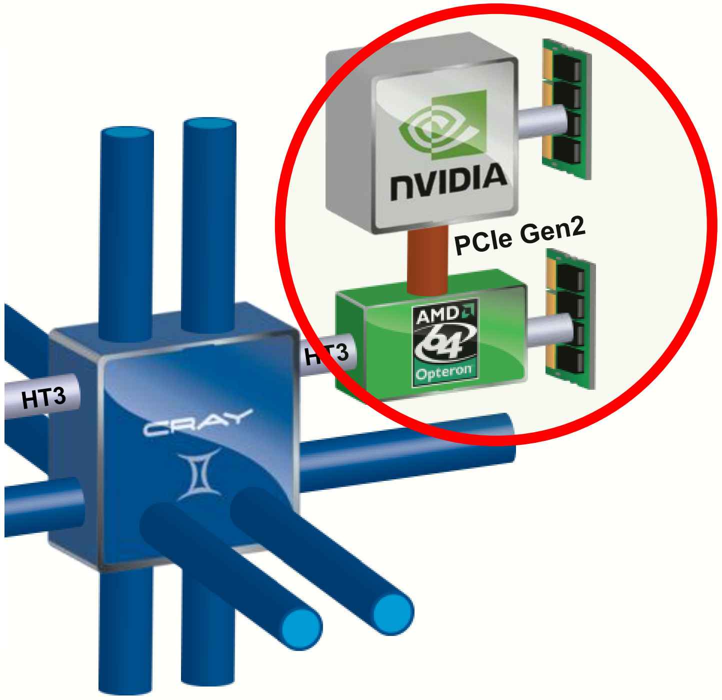
Regularity

- SIMD utilization and load balance

Numerical Stability

- Pivoting for linear system solvers

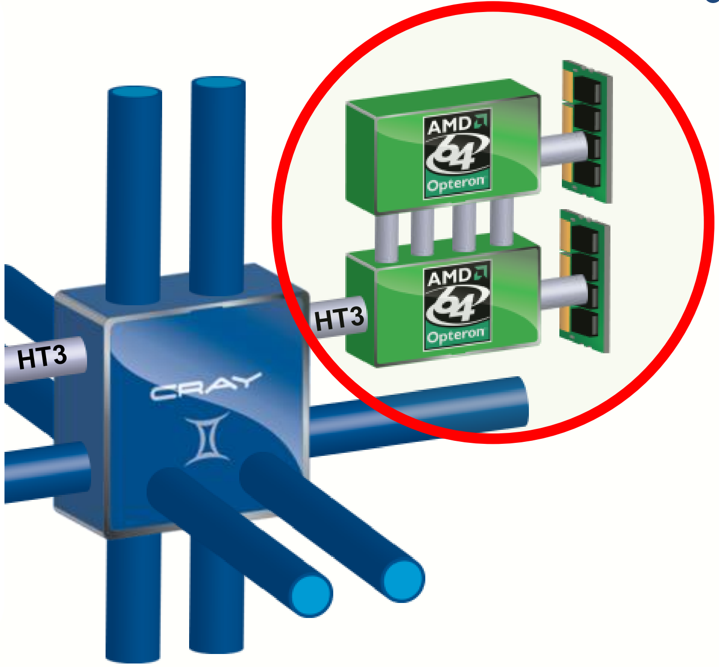
Cray XK7 Nodes



- Dual-socket Node
  - One AMD Interlagos chip
    - 8 core modules, 32 threads
    - 156.5 GFs peak performance
    - 32 GBs memory
      - 51 GB/s bandwidth
  - One NVIDIA Kepler chip
    - 1.3 TFs peak performance
    - 6 GBs GDDR5 memory
      - 250 GB/sec bandwidth
  - Gemini Interconnect
    - Same as XE6 nodes

Blue Waters contains 4,224 Cray XK7 compute nodes.

Cray XE6 Nodes



- Dual-socket Node
  - Two AMD Interlagos chips
    - 16 core modules, 64 threads
    - 313 GFs peak performance
    - 64 GBs memory
      - 102 GB/sec memory bandwidth
  - Gemini Interconnect
    - Router chip & network interface
    - Injection Bandwidth (peak)
      - 9.6 GB/sec per direction

Blue Waters contains 22,640 Cray XE6 compute nodes.

Scalable GPU Libraries

- Dense Linear algebra—BLAS, LU, Cholesky, Eigen solvers (CUBLAS, CULA, MAGMA)
- Sparse Matrix Vector Multiplication, Tridiagonal solvers (CUSPARSE, QUDA, ViennaCL, Parboil)
- FFTs, Convolutions (CUFFT, ViennaCL, Parboil)
- N-Body (NAMD/VMD, FMM BU, Parboil)
- Histograms (CUB, Parboil)
- Some PDE solvers (CURRENT, Parboil)
- Graphs – Breadth-First Search (Parboil)
- Image Processing (OpenCV)

Example of Library Needs

- Sparse linear algebra
  - Sparse LU, Cholesky factorization(?)
  - Sparse Eigen solvers
- Graph algorithm
  - Graph partitioning
  - Depth first search
  - …
- …

ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN