Intentional Programming for HPC SoC Computing

Wen-mei Hwu
University of Illinois, Urbana-Champaign

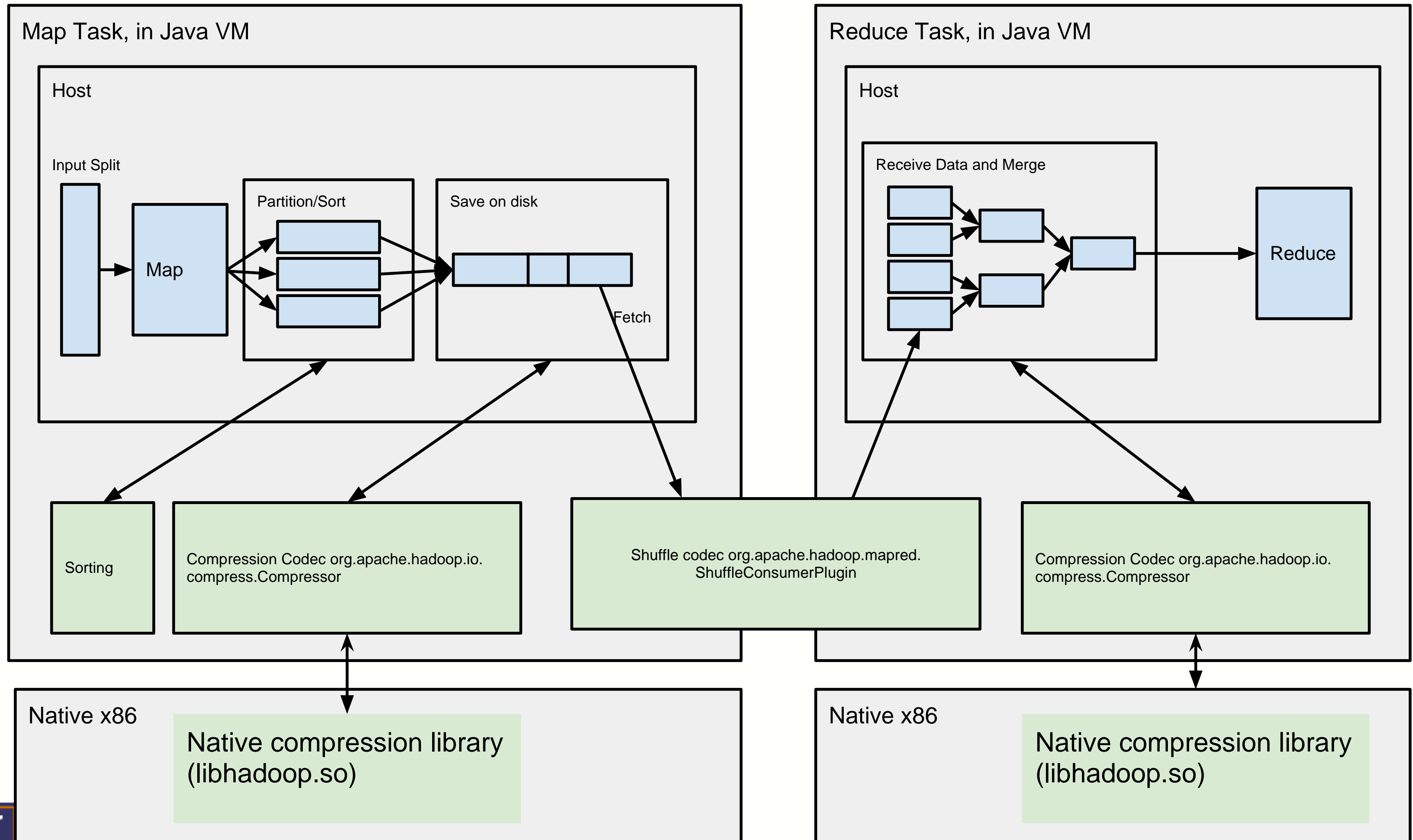
MulticoreWare, Inc.

SoC HPC Opportunities and Challenges

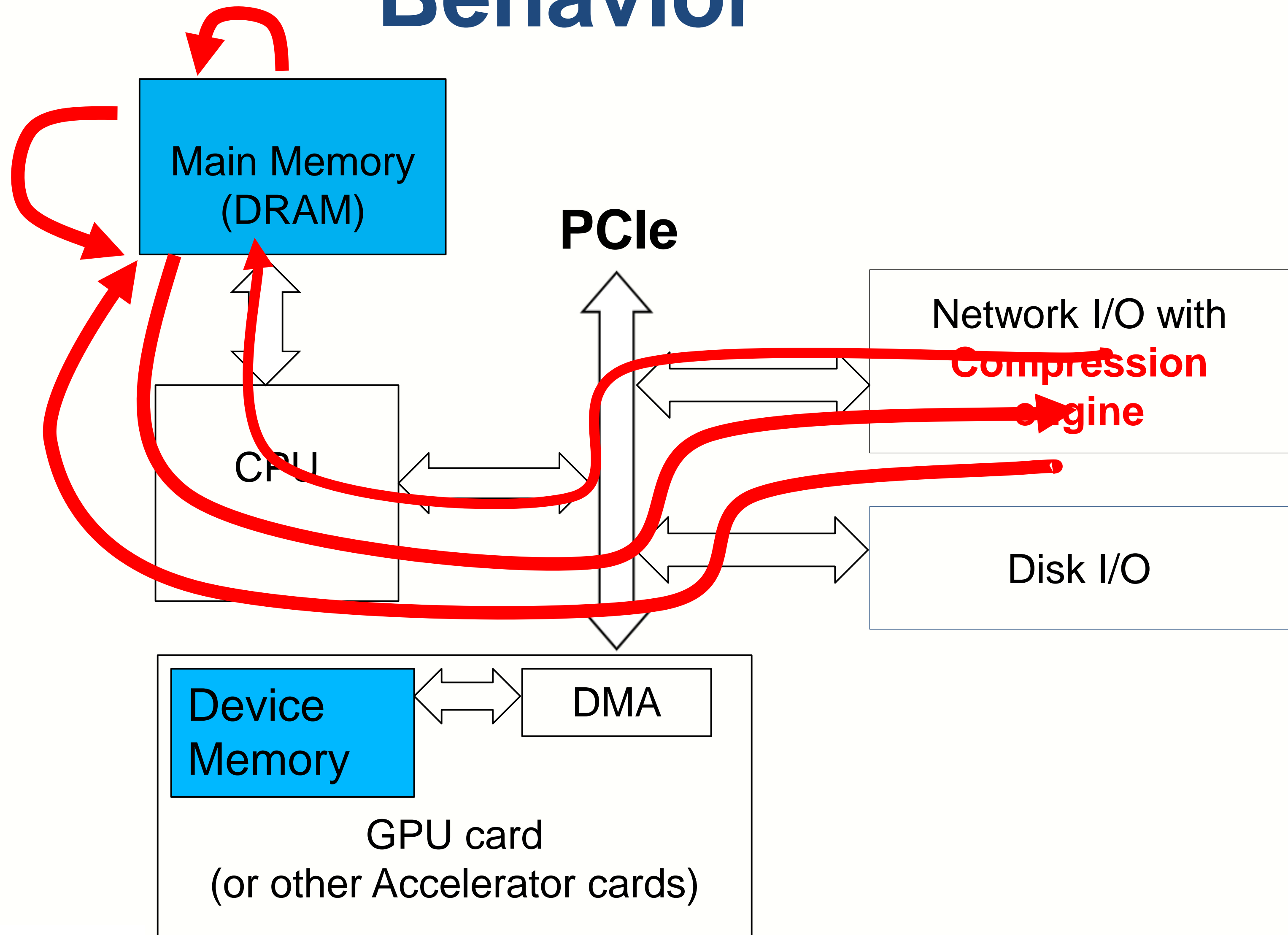
- Packaging and architecture innovations to
  - Reduce system-level data movement
    - Reduced latency and increased throughput
  - Enable distributed, heterogeneous computing

- Major impediment is software inertia.

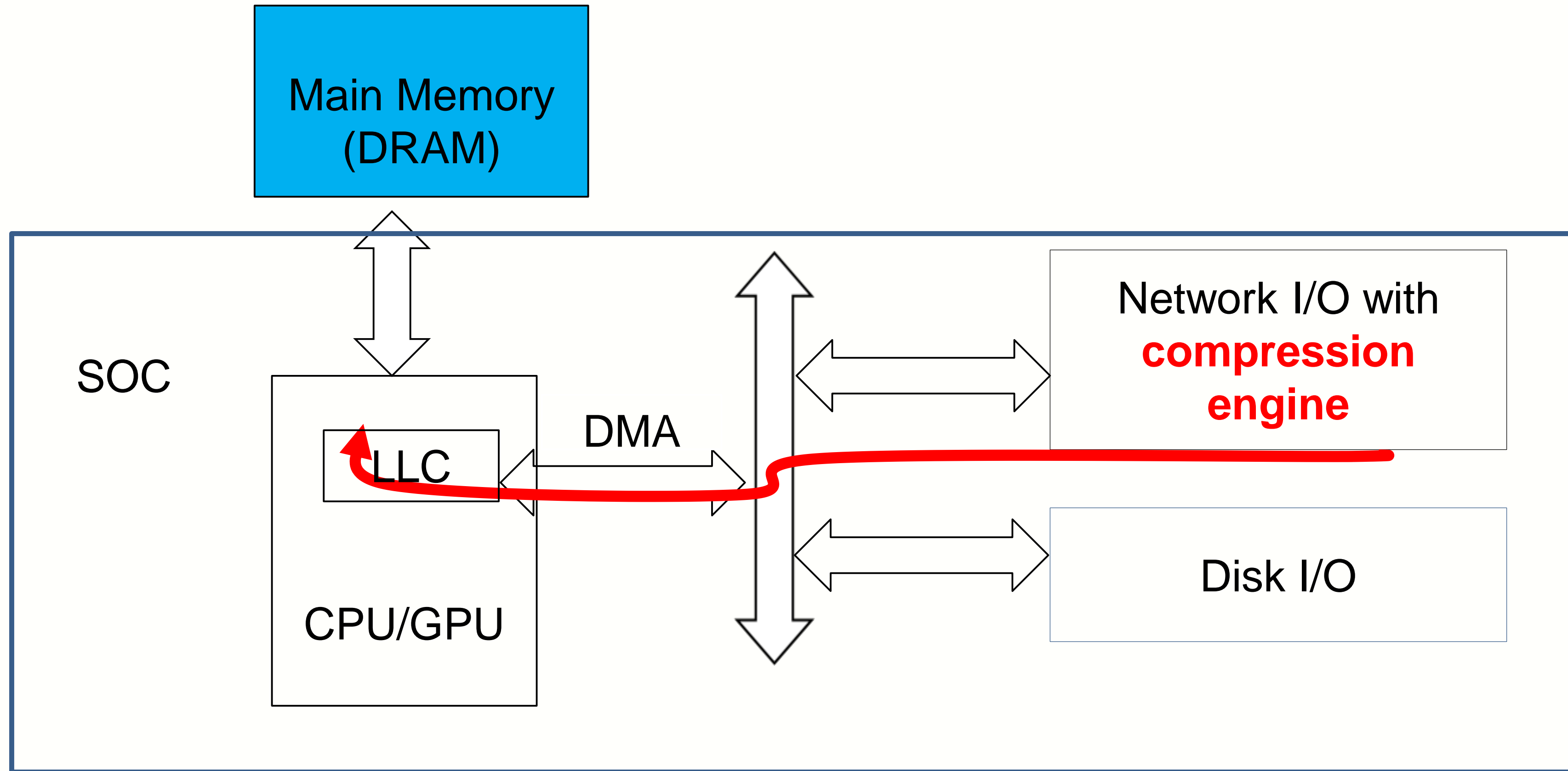
Acceleration Challenges in Hadoop



State of the Art Data Transfer Behavior



Desired SoC Data Transfer Behavior



ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Programming Heterogeneous Computing Systems

- Performance portability and locality with imperative parallel programming
  – OpenCL with MxPA (UIUC Research, now MulticoreWare Product)

- Algorithm selection, scalability, and efficiency with intentional parallel programming
  – Triolet [Rodrigues, et al, PPoPP 2014]

MxPA Results: Comparison to Intel/AMD

■ LC □ AMD

+ Locality improves = Locality stays the same - Locality gets worse

Speedup (normalized – higher is better)

1.00
0.80
0.60
0.40
0.20
0.00

sgm kmns sc mriq ctcp lud lkct mrig pf nw sad tpcf spmv rbfs pbfs hw fft hst cfd geo

+ Locality =Locality - Locality

■ LC ■ Intel

+ Locality improves = Locality stays the same - Locality gets worse

Speedup (normalized – higher is better)

1.00
0.80
0.60
0.40
0.20
0.00

sgm pf kmns sc mriq sad lmd spmv pbfs tpcf nw ctcp mrig hw lkct fft rbfs hst lud cfd geo

+ Locality =Locality - Locality

ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Programming in Triolet Python

Nonuniform FT (real part)

$$y_i = \sum_{j=0}^{n-1} x_j \cos(r_i k_j) \qquad \text{for all } 0 \le i < m$$

Programming in Triolet Python

Nonuniform FT (real part)

Inner loop

$$y_i = \sum_{j=0}^{n-1} x_j \cos(r_i k_j) \qquad \text{for all } 0 \le i < m$$

Inner loop

```
sum(x * cos(r*k) for (x, k) in zip(xs, ks))
```

Programming in Triolet Python

Nonuniform FT (real part)

Inner loop

Outer loop

$$y_i = \sum_{j=0}^{n-1} x_j \cos(r_i k_j) \qquad \text{for all } 0 \le i < m$$

Inner loop

Outer loop

```
ys = [ sum(x * cos(r*k) for (x, k) in zip(xs, ks)) for r in par(rs)]
```

ILLINOIS

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Programming in Triolet Python

Nonuniform FT (real part)

Inner loop

Outer loop

$$y_i = \sum_{j=0}^{n-1} x_j \cos(r_i k_j) \qquad \text{for all } 0 \le i < m$$

Inner loop

Outer loop

```
ys = [ sum(x * cos(r*k) for (x, k) in zip(xs, ks)) for r in par(rs)]
```

- "map and reduce" style programming—no new paradigm to learn
- Parallel details are implicit—easy to use
- Automated data partitioning, MPI rank generation, MPI messaging, OpenMP, etc.

Productivity

Triolet

```
ys = [sum(x * cos(r*k) for (x, k) in zip(xs, ks))
      for r in par(rs)]
```

- Library functions factor out data decomposition, parallelism, and communication

C with MPI+OpenMP

Setup

Cleanup

Productivity and Performance

Triolet

```
ys = [sum(x * cos(r*k) for (x, k) in zip(xs, ks))
      for r in par(rs)]
```

- Library functions factor out data decomposition, parallelism, and communication

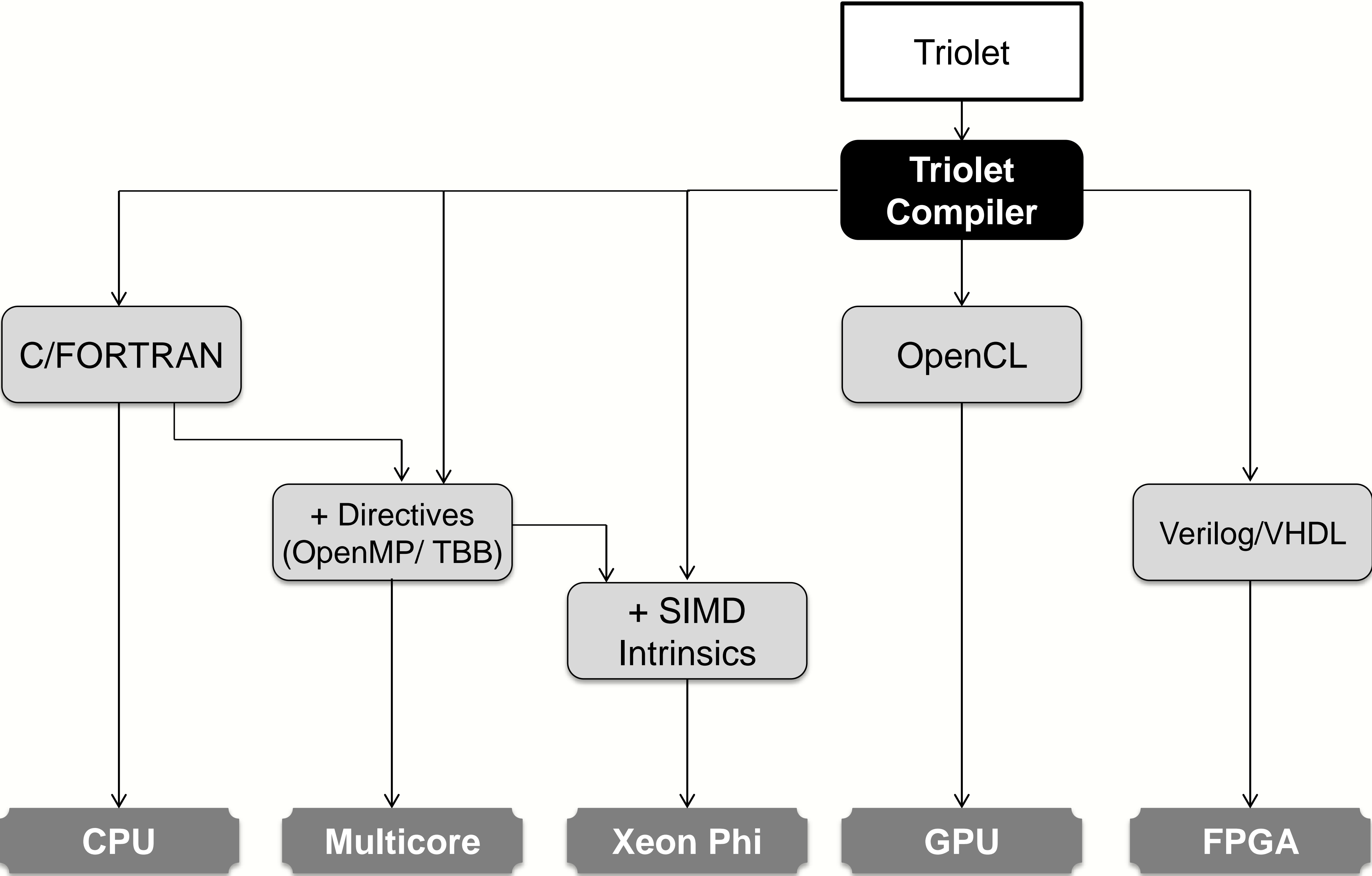
128-way Speedup *(16 cores × 8 nodes)*	Triolet	C with MPI+OpenMP
	99	115

C with MPI+OpenMP

Setup

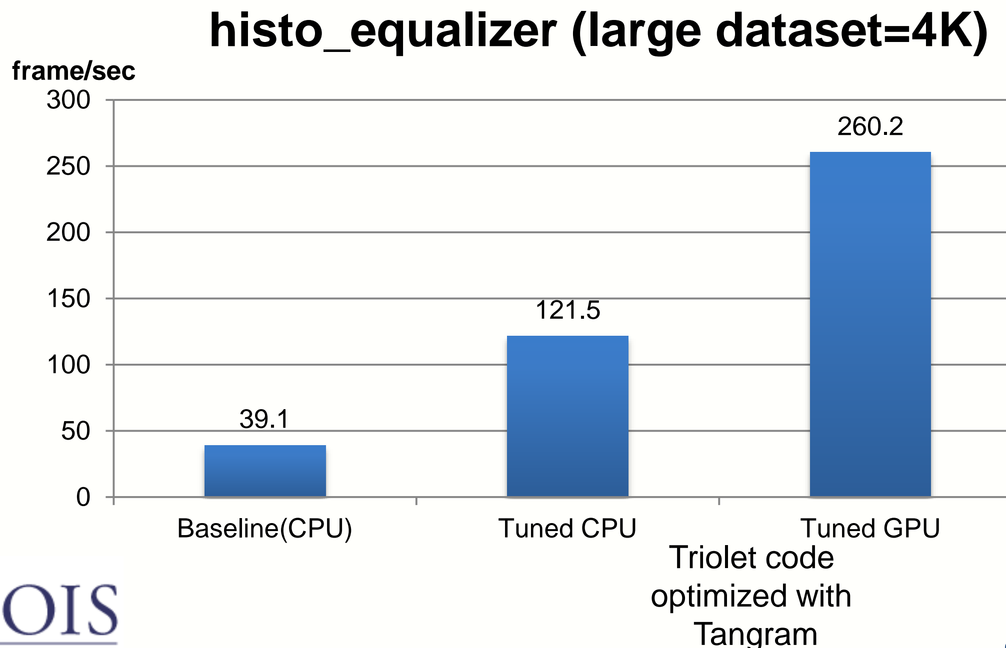
Cleanup

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN


Triolet
Triolet Compiler
C/FORTRAN
OpenCL
+ Directives (OpenMP/ TBB)
Verilog/VHDL
+ SIMD Intrinsics
CPU
Multicore
Xeon Phi
GPU
FPGA

Triolet equalize_frames Example

- Baseline is parallel code taken from DARPA PERFECT benchmark suite
- Tuned code outperforms baseline on both architectures



Triolet code optimized with Tangram

CPU time measured on Intel Xeon E5520 (4-core, hyperthreaded)

GPU time measured on Tesla C2050

Triolet Compiled Conv2D on BnB

6,000,000,000
5,000,000,000
4,000,000,000
3,000,000,000
2,000,000,000
1,000,000,000
0

insts
Cycles
Energy (nJ)

Base RISC
BnB

SoC for HPC

~10.5x improvement across the board
1024x1024 image, HMC memory system

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Collaboration with Andrew Chien (U Chicago) and Laura Carington (UCSD)

Conclusion

- Packing and architecture innovations enable continued improvement in performance and power
  - Hardware APIs for streamlined data movement
  - Hardware APIs for I/O-side and memory-side compute

- Intentional programming for system-level software mapping and transformation
  - Triolet Python brings intentional programming into mobile devices, HPC clusters, and distributed computing

Essential work in writing efficient parallel code.

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
- Migrate code across SW components

- Allocate memory
- Partition data
- Insert data movement code

- Reduction trees
- Array packing
- Boundary cell communication

Current State of Programming Heterogeneous Systems

CPU | Multicore | Xeon Phi | GPU | FPGA

Current State of Programming Heterogeneous Systems

C/FORTRAN

CPU

Multicore

Xeon Phi

GPU

FPGA

Current State of Programming Heterogeneous Systems

C/FORTRAN

+ Directives (OpenMP/ TBB)

CPU

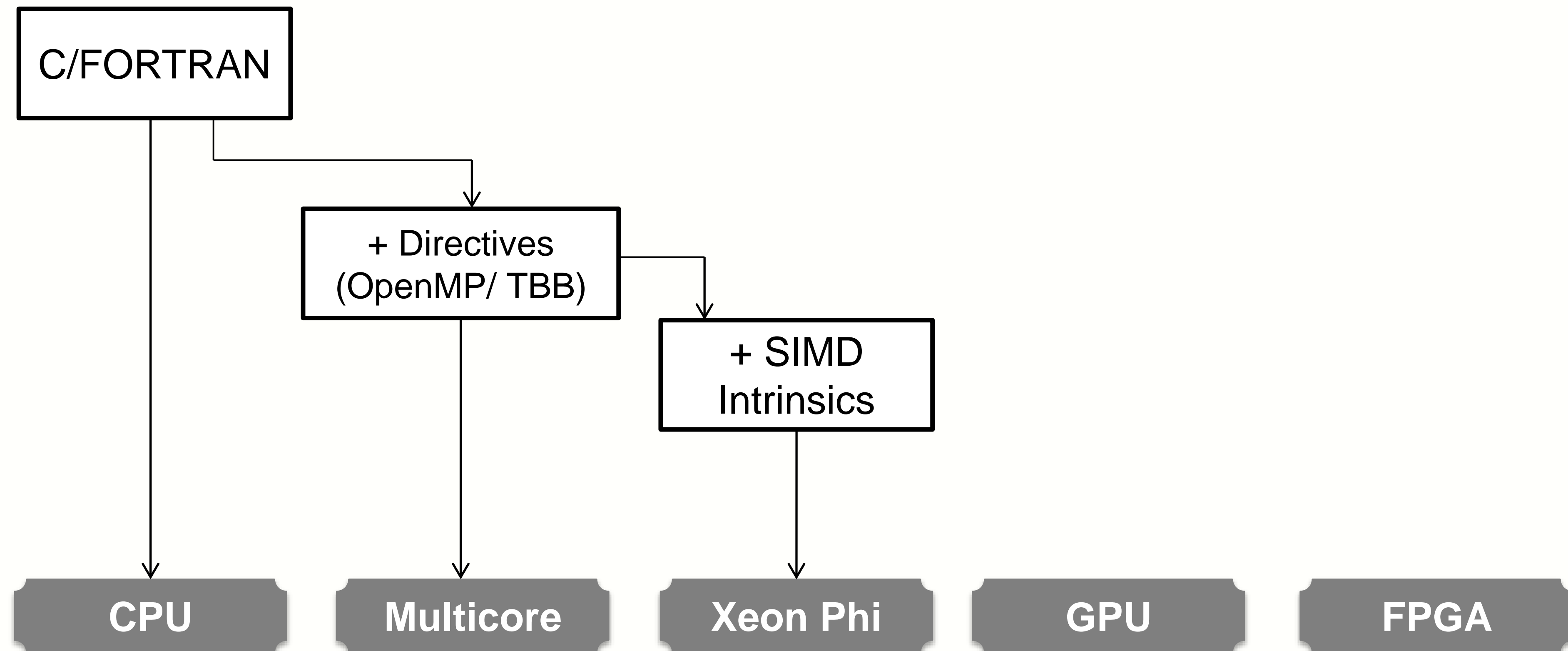
Multicore

Xeon Phi

GPU

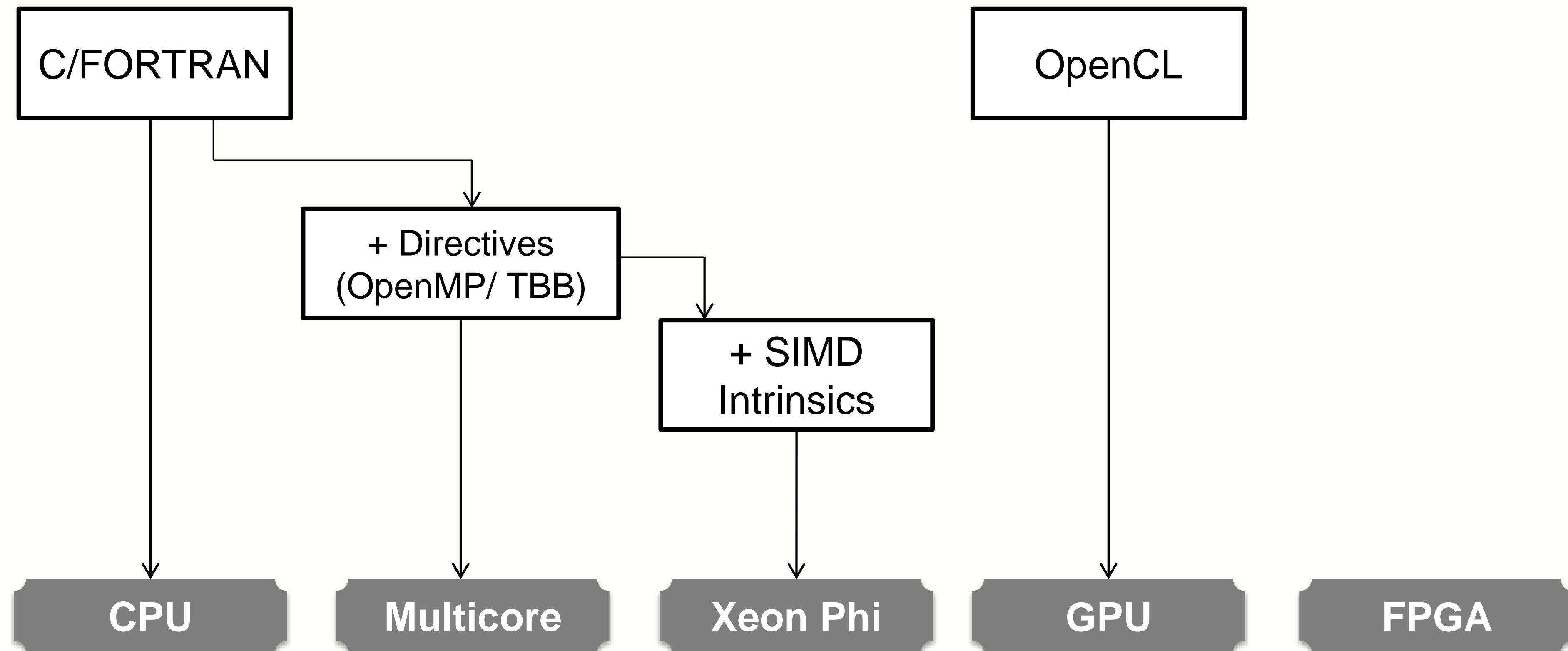
FPGA

Current State of Programming Heterogeneous Systems



ILLINOIS
1867
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Current State of Programming Heterogeneous Systems



Current State of Programming Heterogeneous Systems

C/FORTRAN

+ Directives (OpenMP/ TBB)

+ SIMD Intrinsics

OpenCL

Verilog/VHDL

CPU

Multicore

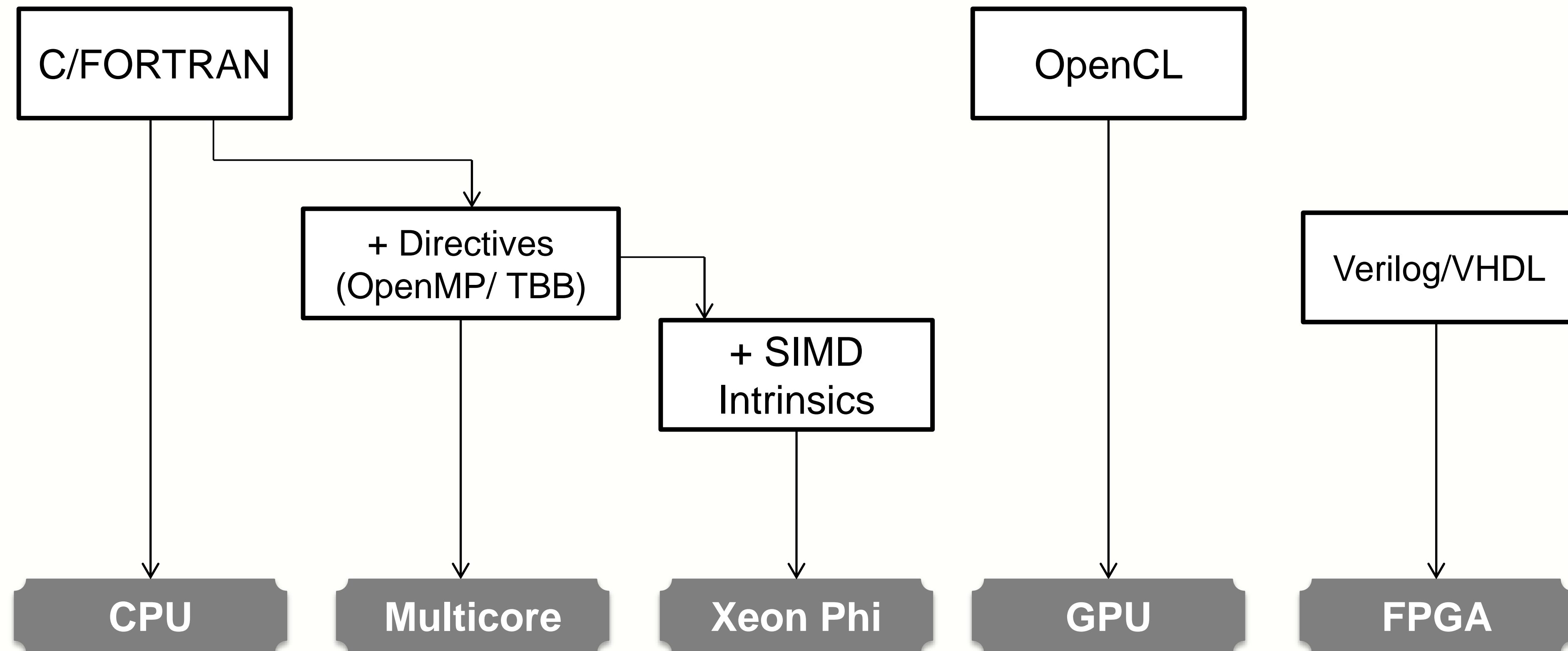
Xeon Phi

GPU

FPGA

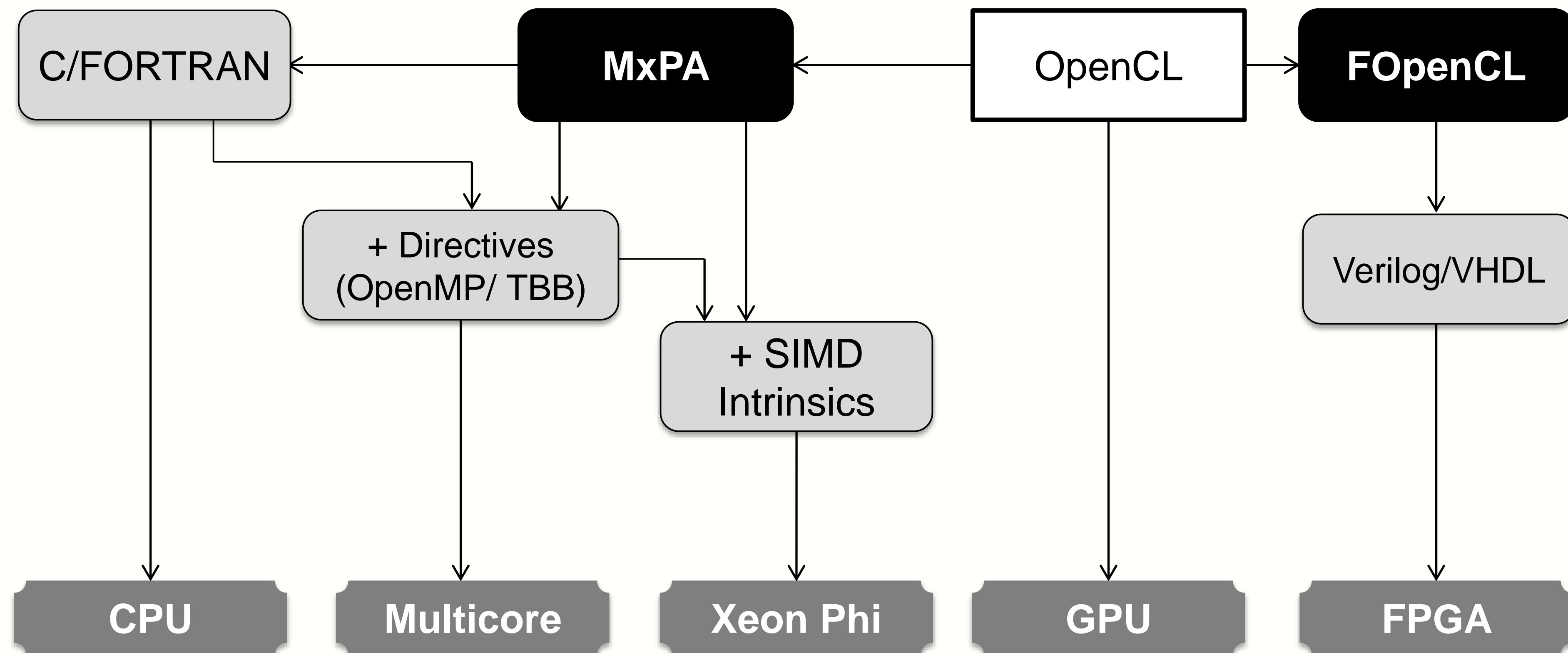
Current State of Programming Heterogeneous Systems

Programming heterogeneous systems requires too many versions of code!



Productivity in Programming Heterogeneous Systems

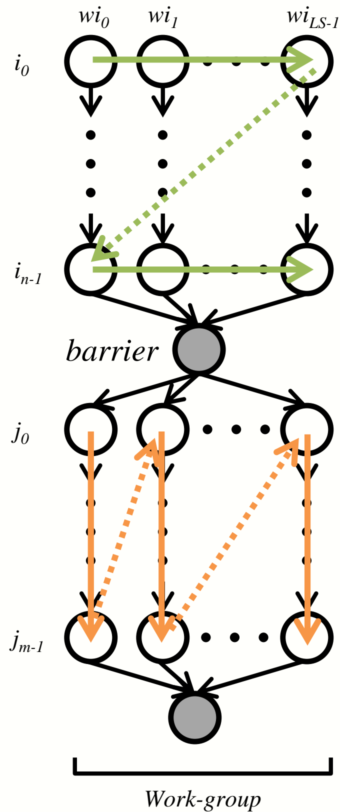
<u>Step #1:</u> Keep only one of the versions and use portability tools to generate the others.



MxPA: Overview

- Optimizes scheduling of work-item execution for **locality** and **vectorization** for the hardware
- Locality-centric compile-time scheduling selects between:
  - *BFO (Breadth-First Order):* issue each memory access for all work-items first
  - *DFO (Depth-First Order):* issue all memory accesses for a single work-item first
- Kernel fusion run-time scheduling reduces memory traffic for common data flow between kernels
- Dynamic vectorization maintains vector execution in spite of apparent control flow divergence

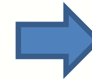
MxPA: Locality-centric Scheduling



Dependency in executing OpenCL kernels

```
CLKernel() {
  // e.g. SOA
  for (i = 0..N) {
    .. = A[c0*i + wid];
  }
  barrier();
  // e.g. AOS
  for (j = 0..M) {
    .. = B[c1*wid + j];
  }
}
```

An example OpenCL code

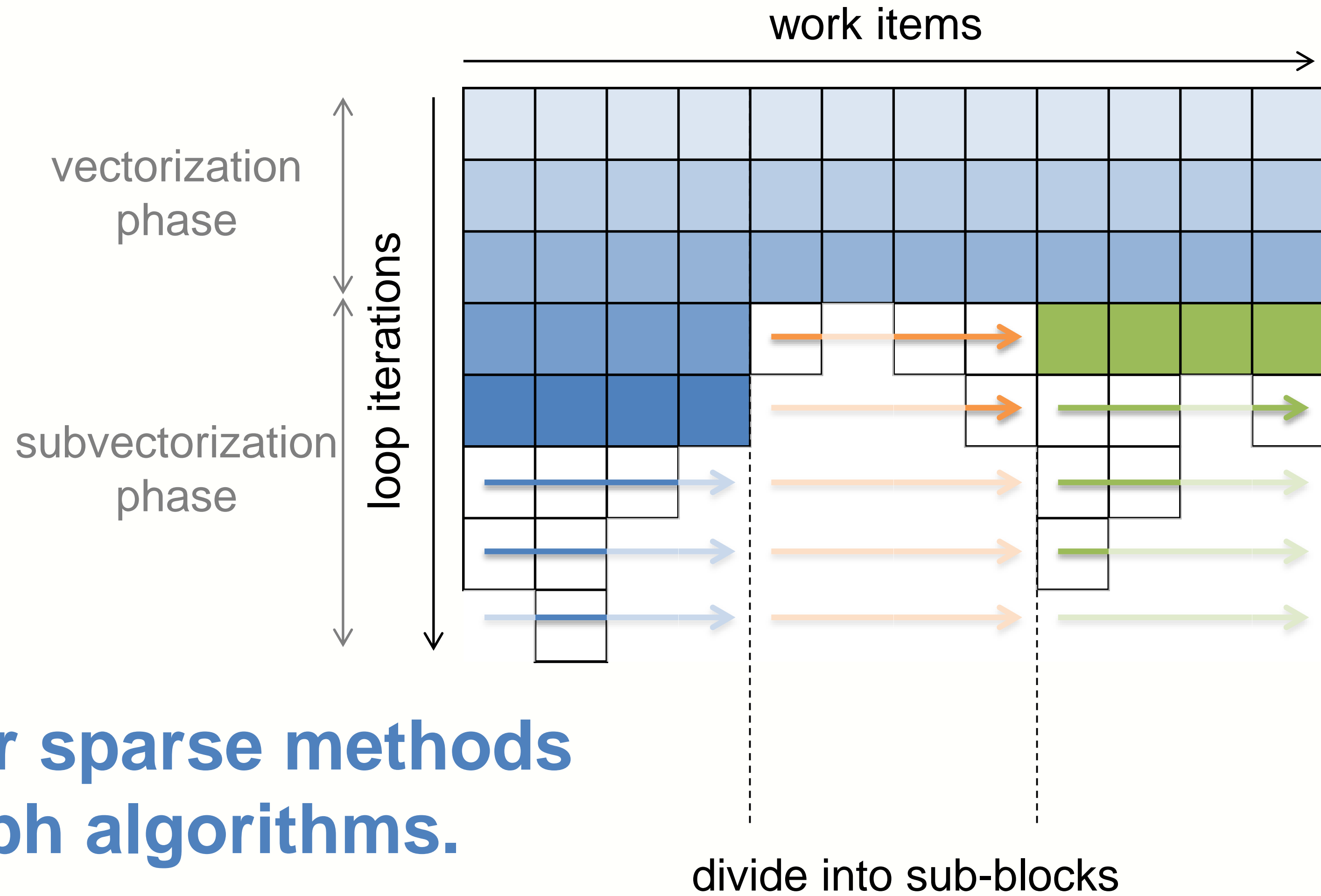


```
CLKernel_MXPA() {
  // BFO
  for (i = 0..N) {
    for (wid = 0..LS) {
      .. = A[c0*i + wid];
    }
  }

  // DFO
  for (wid = 0..LS) {
    for (j = 0..M) {
      .. = B[c1*wid + j];
    }
  }
}
```

MXPA-translated code

MxPA: Dynamic Vectorization of Control Divergent Loops



Effective for sparse methods and graph algorithms.

MxPA Results: Comparison to Industry

■ LC □ AMD

+ Locality improves = Locality stays the same - Locality gets worse

Speedup (normalized – higher is better)

1.00
0.80
0.60
0.40
0.20
0.00

sgm kmns sc mriq ctcp lud lkct mrig pf nw sad tpcf spmv rbfs pbfs hw fft hst cfd geo

+ Locality =Locality - Locality

■ LC ■ Intel

+ Locality improves = Locality stays the same - Locality gets worse

Speedup (normalized – higher is better)

1.00
0.80
0.60
0.40
0.20
0.00

sgm pf kmns sc mriq sad lmd spmv pbfs tpcf nw ctcp mrig hw lkct fft rbfs hst lud cfd geo

+ Locality =Locality - Locality

ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

MxPA Results: Comparison to Industry

Metric	LC/AMD	LC/Intel
Speedup	3.20x	1.62x
L1 Data Cache Misses	0.12x	0.36x
Data TLB Misses	0.23x	0.33x
LLC Misses	0.91x	0.97x

HIGH-LEVEL INTERFACE

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 + optimizing compiler
  - Library makes parallelization decisions
  - Uses a rich transformation, library aware compiler
  - Extensible—just add library functions