A Heterogeneous Parallel Framework for Domain-Specific Languages

Kevin J. Brown, Arvind K. Sujeeth, HyoukJoong Lee, Hassan Chafi, Kunle Olukotun
Stanford University

Tiark Rompf, Martin Odersky
EPFL

Programmability Chasm

Applications

Scientific Engineering

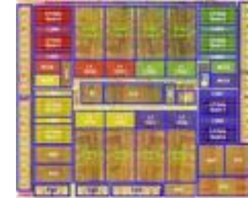
Virtual Worlds

Personal Robotics

Data Informatics

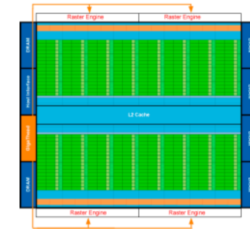
Parallel Programming Language

Pthreads
OpenMP



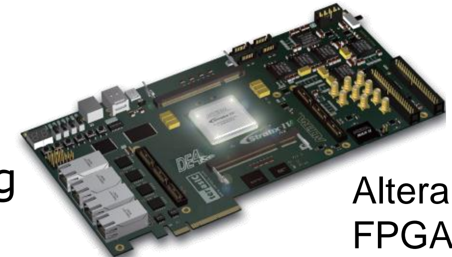
Sun
T2

CUDA
OpenCL



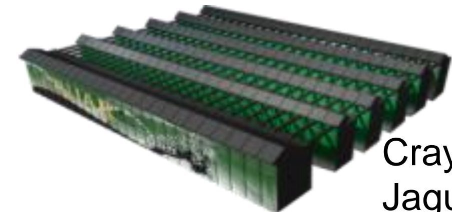
Nvidia
Fermi

Verilog
VHDL



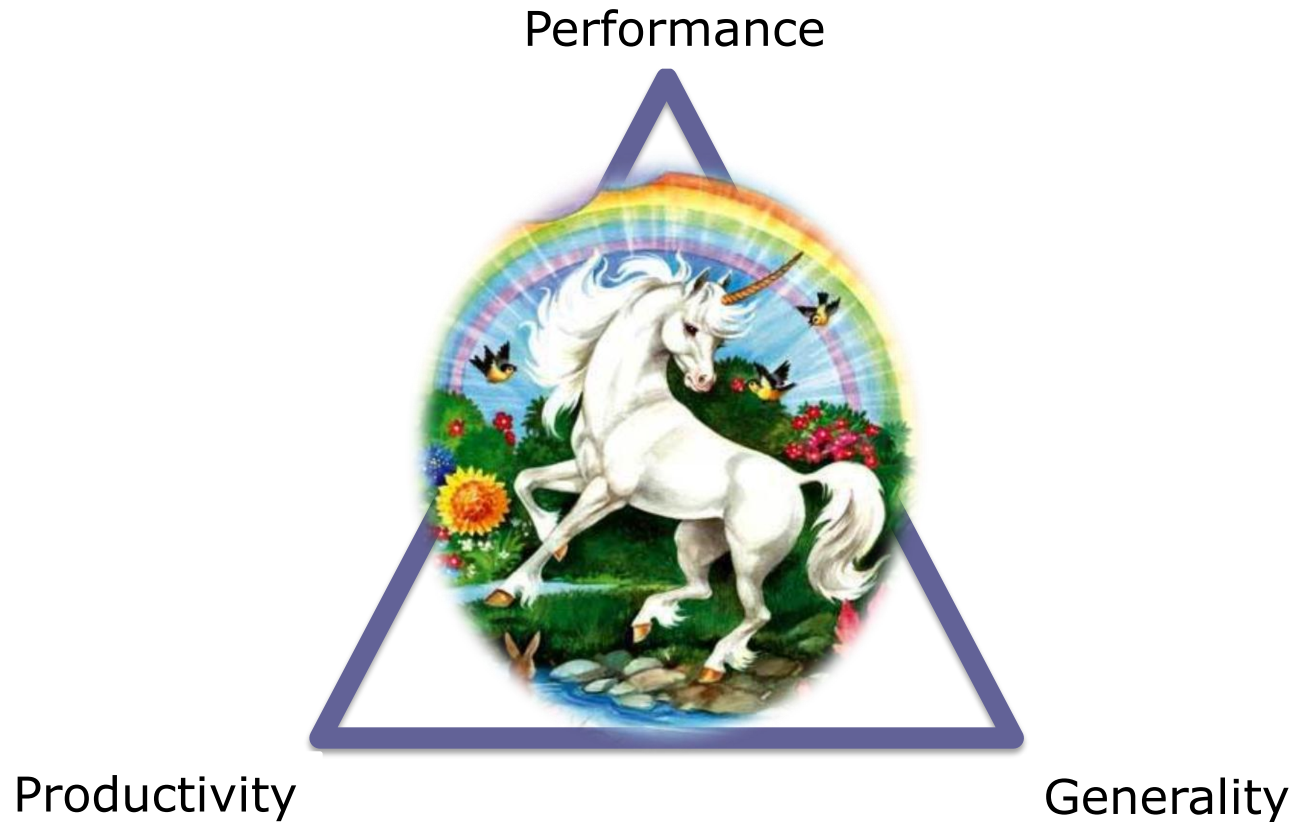
Altera
FPGA

MPI
PGAS

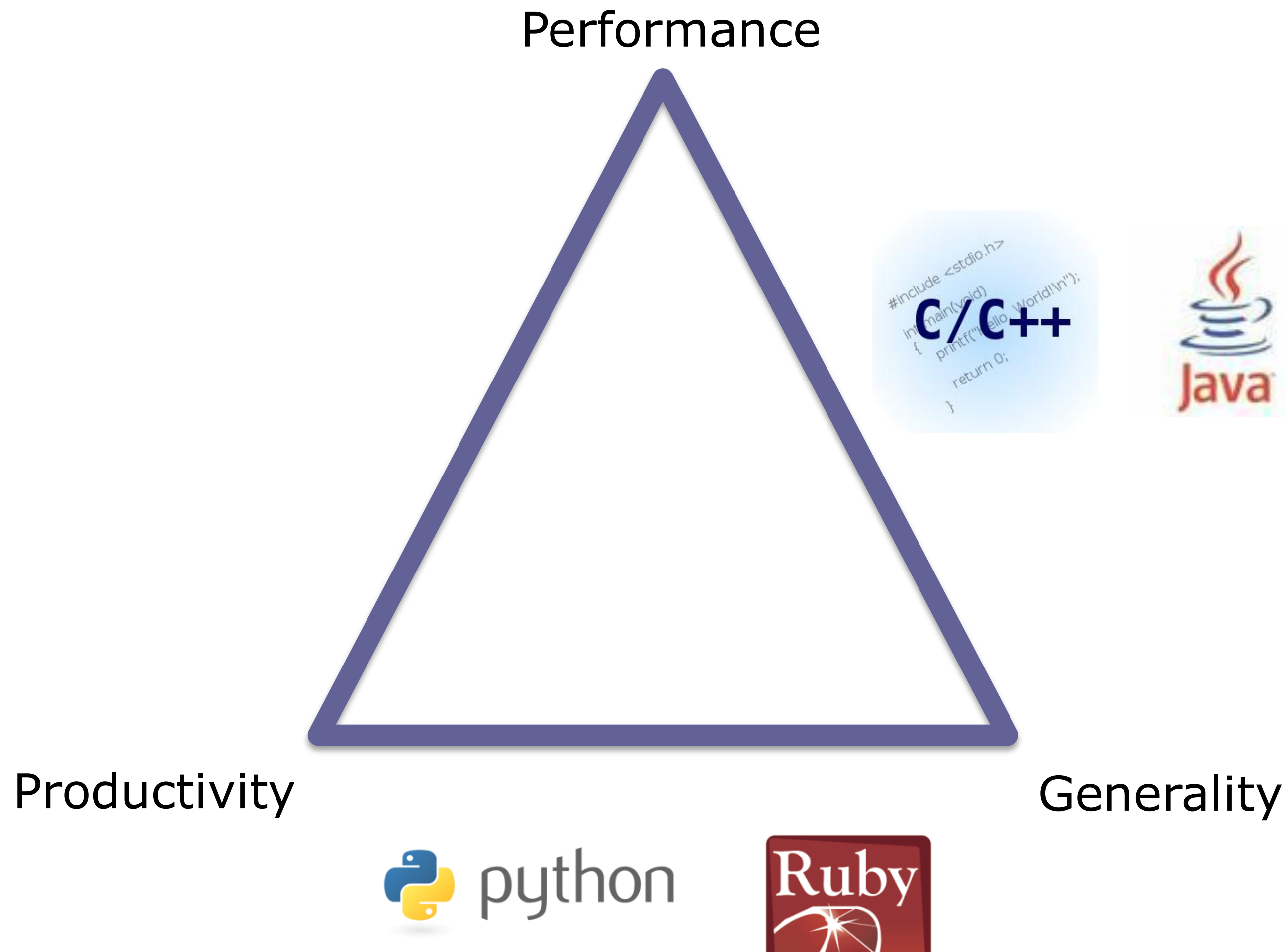


Cray
Jaguar

The Ideal Parallel Programming Language



Successful Languages



Domain Specific Languages

Performance
(Heterogeneous Parallelism)

Domain
Specific
Languages

Productivity

Generality

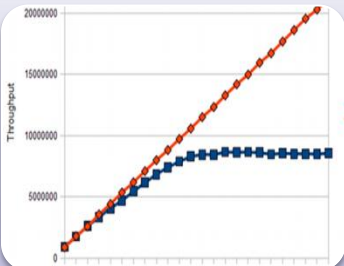
Benefits of Using DSLs for Parallelism

Productivity

- Shield most programmers from the difficulty of parallel programming
- Focus on developing algorithms and applications and not on low level implementation details

Performance

- Match high level domain abstraction to generic parallel execution patterns
- Restrict expressiveness to more easily and fully extract available parallelism
- Use domain knowledge for static/dynamic optimizations



Portability and forward scalability

- DSL & Runtime can be evolved to take advantage of latest hardware features
- Applications remain unchanged
- Allows innovative HW without worrying about application portability

DSLs: Compiler vs. Library

- *A Domain-Specific Approach to Heterogeneous Parallelism*, Chafi et al.
  - A framework for parallel DSL libraries
  - Used data-parallel patterns and deferred execution (transparent futures) to execute tasks in parallel
- Why write a compiler?
  - Static optimizations (both generic and domain-specific)
  - All DSL abstractions can be removed from the generated code
  - Generate code for hardware not supported by the host language
  - Full-program analysis

Common DSL Framework

- Building a new DSL
  - Design the language (syntax, operations, abstractions, etc.)
  - Implement compiler (parsing, type checking, optimizations, etc.)
  - Discover parallelism (understand parallel patterns)
  - Emit parallel code for different hardware (optimize for low-level architectural details)
  - Handle synchronization, multiple address spaces, etc.

- Need a DSL infrastructure
  - Embed DSLs in a common host language
  - Provide building blocks for common DSL compiler & runtime functionality

Delite Overview

Domain Specific Languages

Data Analytics (*OptiQL*)

Physics (*Liszt*)

Machine Learning (*OptiML*)

Delite: DSL Infrastructure

Domain Embedding Language (*Scala*)

Staged Execution

Delite Compiler

Parallel Patterns

Static Optimizations

Heterogeneous Code Generation

Delite Runtime

Walk-time Optimizations

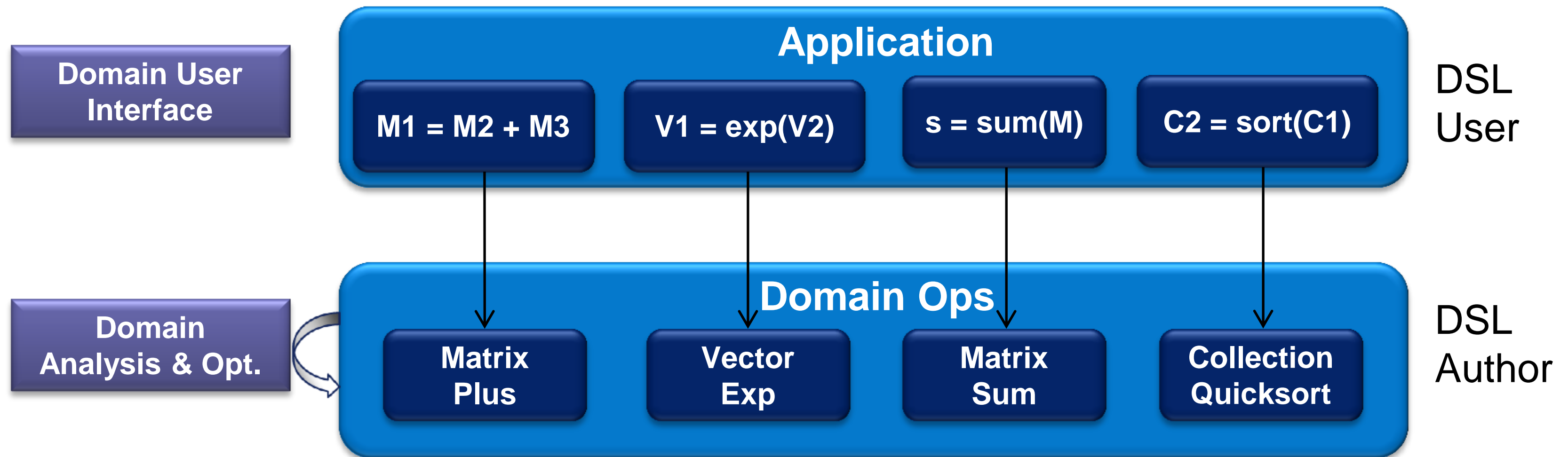
Locality-aware Scheduling

Heterogeneous Hardware

SMP

GPU

DSL Intermediate Representation (IR)



Building an IR

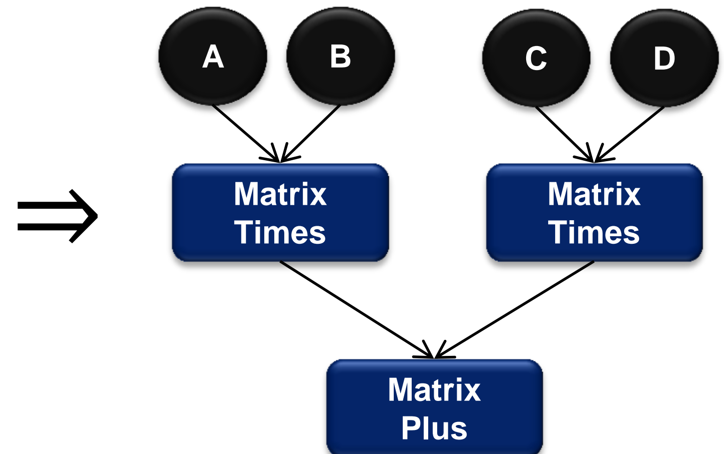
- OptiML: A DSL for machine learning
  - Built using Delite
  - Supports linear algebra (Matrix/Vector) operations

```
//a, b, c, d : Matrix
val x = a * b + c * d
```

```
def infix_+(a: Matrix, b: Matrix) =
  new MatrixPlus(a,b)

def infix_*(a: Matrix, b: Matrix) =
  new MatrixTimes(a,b)
```

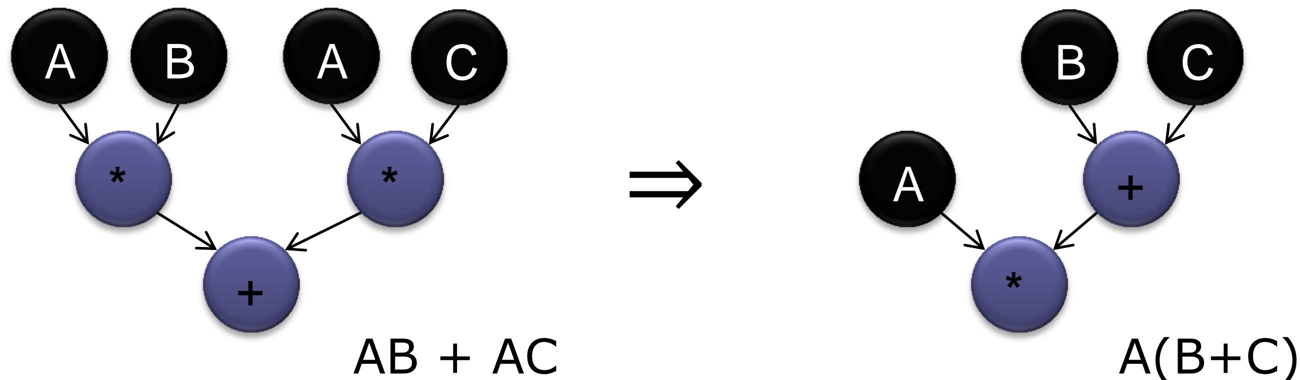
⇒



- DSL methods build IR as program runs

DSL Optimizations

- DSL developer defines how DSL operations create IR nodes
- Specialize implementation of operation for each *occurrence* by pattern matching on the IR
- This technique can be used to control merely what to add to IR or to perform IR rewrites
  - Use this to apply linear algebra simplification rules



OptiML Linear Algebra Rewrites

- A straightforward translation of the Gaussian Discriminant Analysis (GDA) algorithm from the mathematical description produces the following code:

```
val sigma = sum(0,m) { i =>
  val a = if (!x.labels(i)) x(i) - mu0
          else x(i) - mu1
  a.t ** a
}
```

- A much more efficient implementation recognizes that

$$\sum_{i=0}^{n} \overrightarrow{x_i} * \overrightarrow{y_i} \rightarrow \sum_{i=0}^{n} X(:,i) * Y(i,:) = X * Y$$

- Transformed code was 20.4x faster with 1 thread and 48.3x faster with 8 threads.

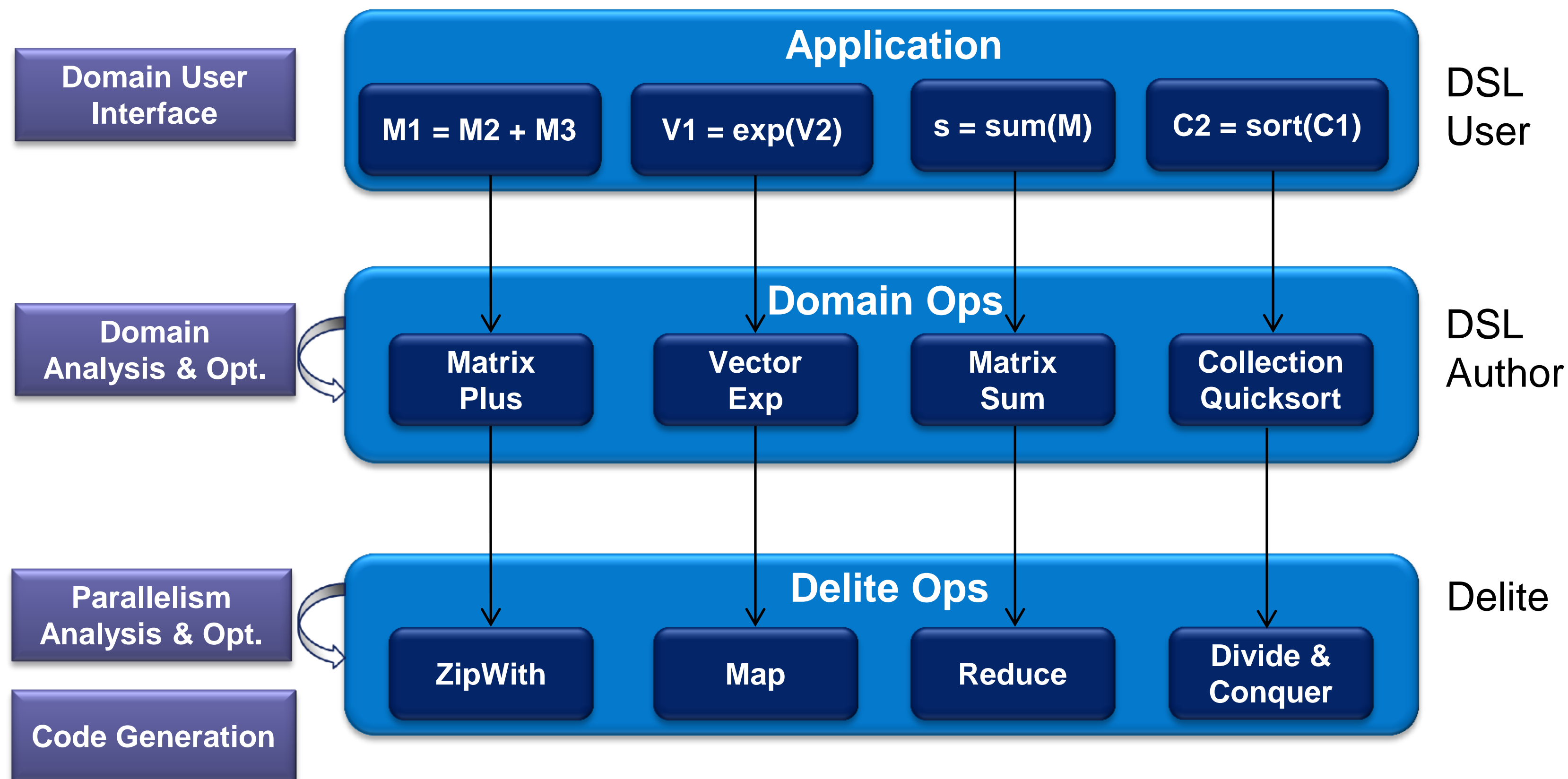
Delite DSL Framework

- Building a new DSL
  - Design the language (syntax, operations, abstractions, etc.)
  - Implement compiler
    - Domain-specific analysis and optimization
    - Lexing, parsing, type-checking, generic optimizations
  - Discover parallelism (understand parallel patterns)
  - Emit parallel code for different hardware (optimize for low-level architectural details)
  - Handle synchronization, multiple address spaces, etc.

Delite Ops

- Encode known parallel execution patterns
  - Map, filter, reduce, …
  - Bulk-synchronous foreach
  - Divide & conquer
- Delite provides implementations of these patterns for multiple hardware targets
  - e.g., multi-core, GPU
- DSL author maps each domain operation to the appropriate pattern
  - Delite handles parallel optimization, code generation, and execution for all DSLs

Multiview Delite IR



Delite Op Fusion

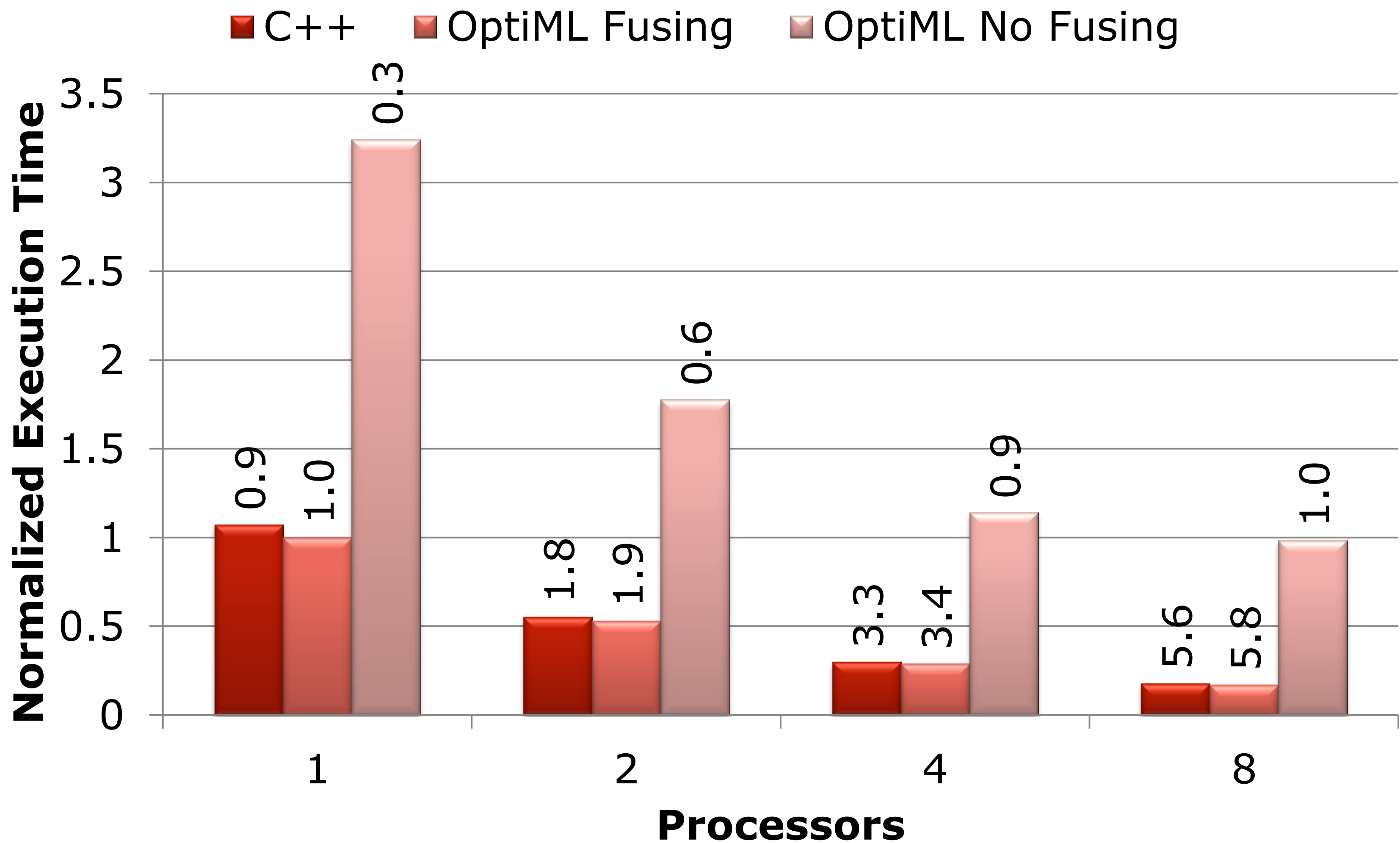
- Operates on all loop-based ops

- Reduces op overhead and improves locality
  - Elimination of temporary data structures
  - Merging loop bodies may enable further optimizations

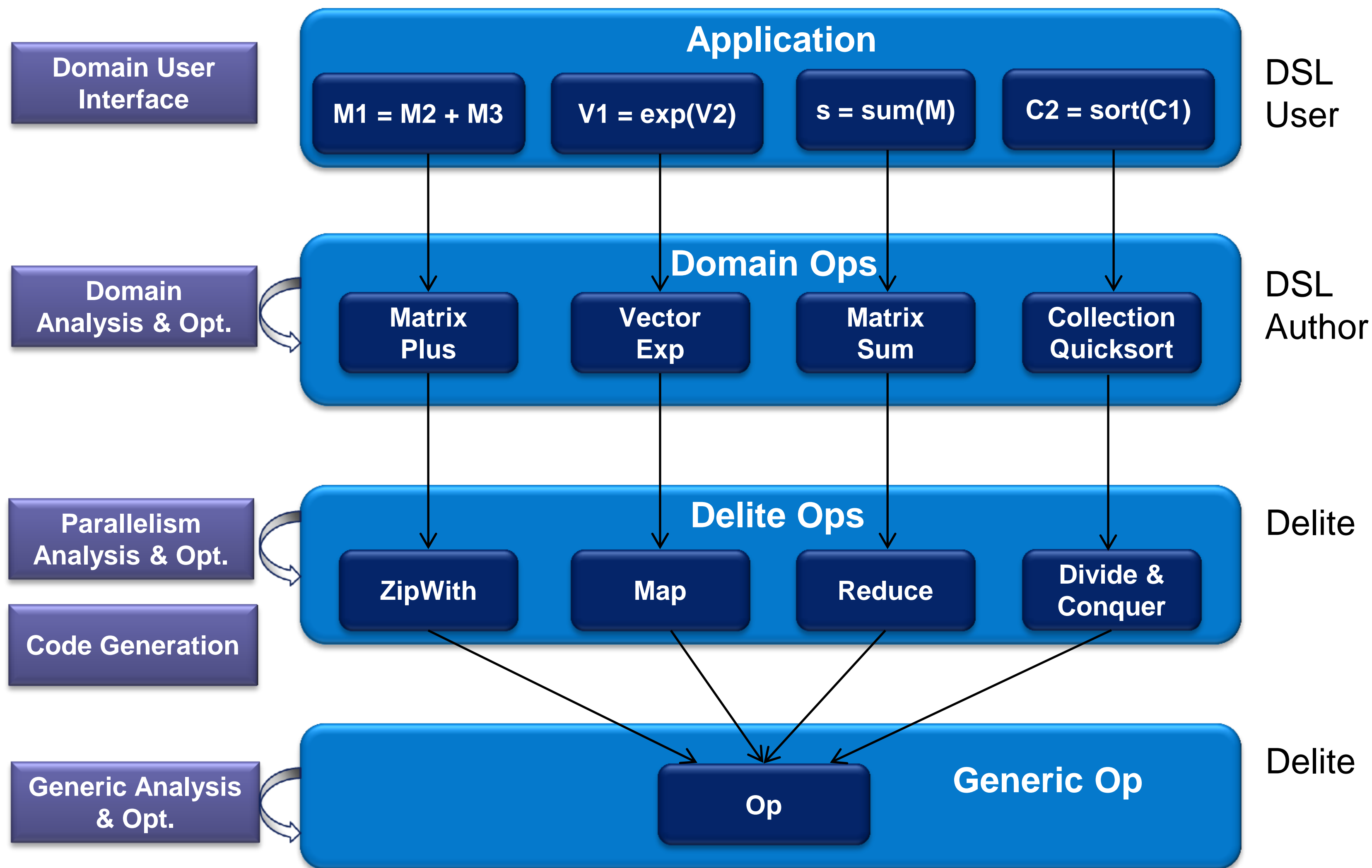
- Fuse both dependent and side-by-side operations
  - Fused ops can have multiple inputs & outputs

- Algorithm: fuse two loops if
  - size(loop1) == size(loop2)
  - No mutual dependencies (which aren't removed by fusing)

Downsampling in OptiML



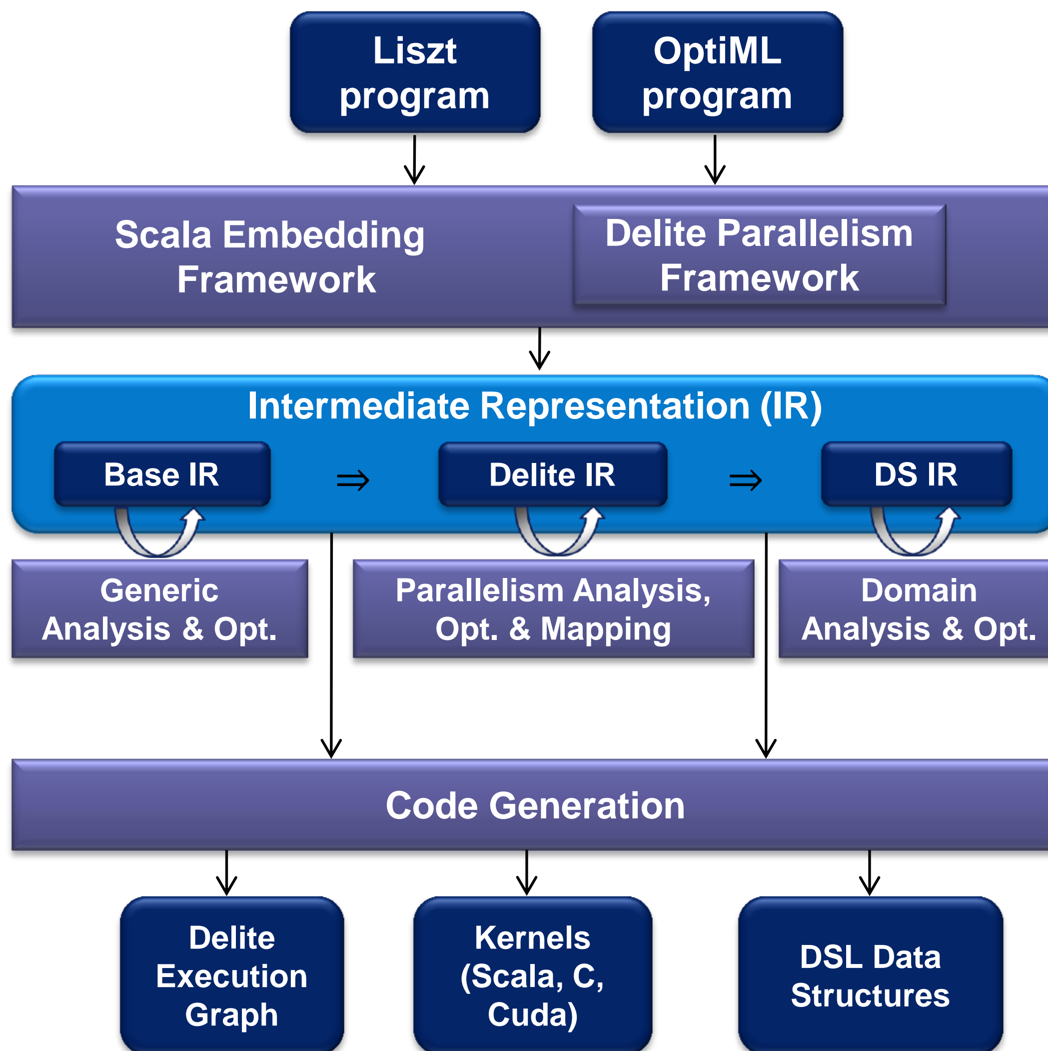
Multiview Delite IR



Generic IR

- Optimizations
  - Common subexpression elimination (CSE)
  - Dead code elimination (DCE)
  - Constant folding
  - Code motion (e.g., loop hoisting)
- Side effects and alias tracking
- All performed at the granularity of DSL operations
  - e.g., MatrixMultiply

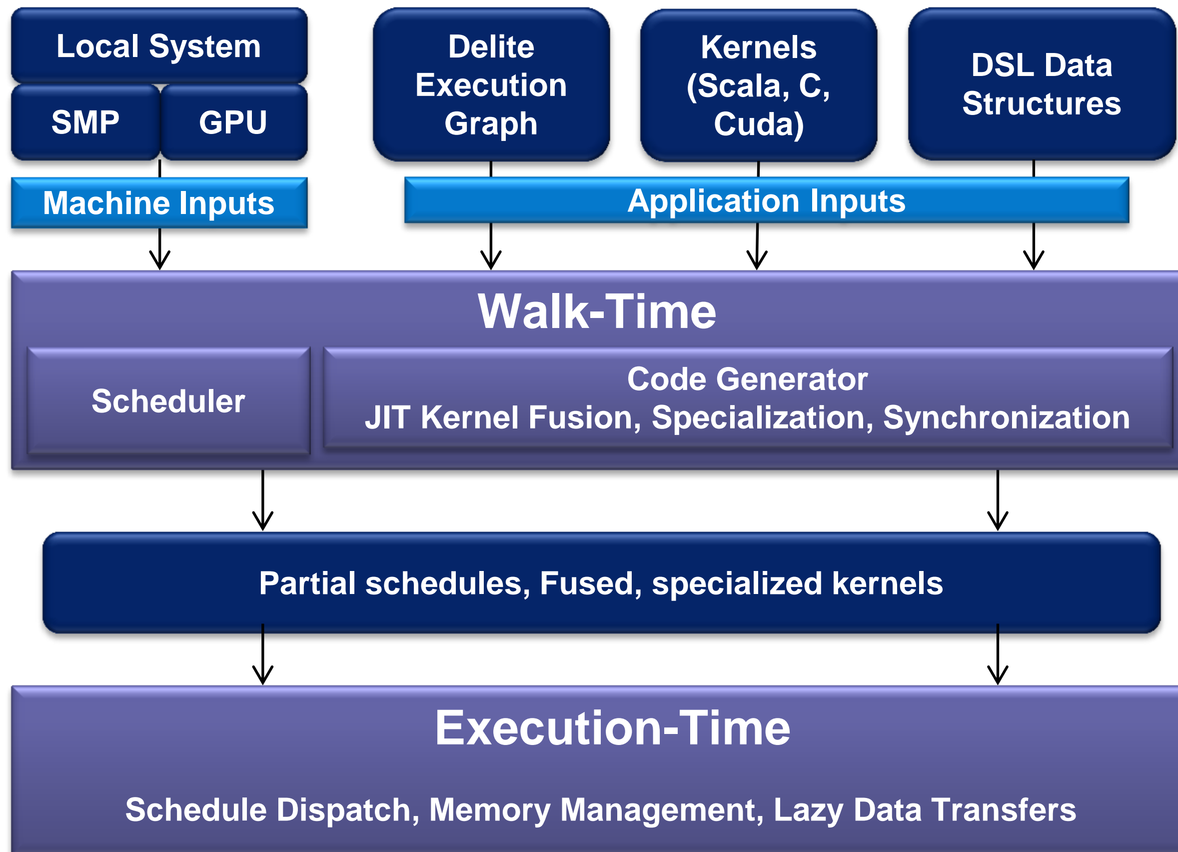
Delite DSL Compiler Infrastructure



Heterogeneous Code Generation

- Delite can have multiple registered target code generators (Scala, Cuda, …)
  - Calls all generators for each Op to create kernels
  - Only 1 generator has to succeed
- Generates an *execution graph* that enumerates all Delite Ops in the program
  - Encodes parallelism within the application
  - Contains all the information the Delite Runtime requires to execute the program
    - Op dependencies, supported targets, etc.

Delite Runtime

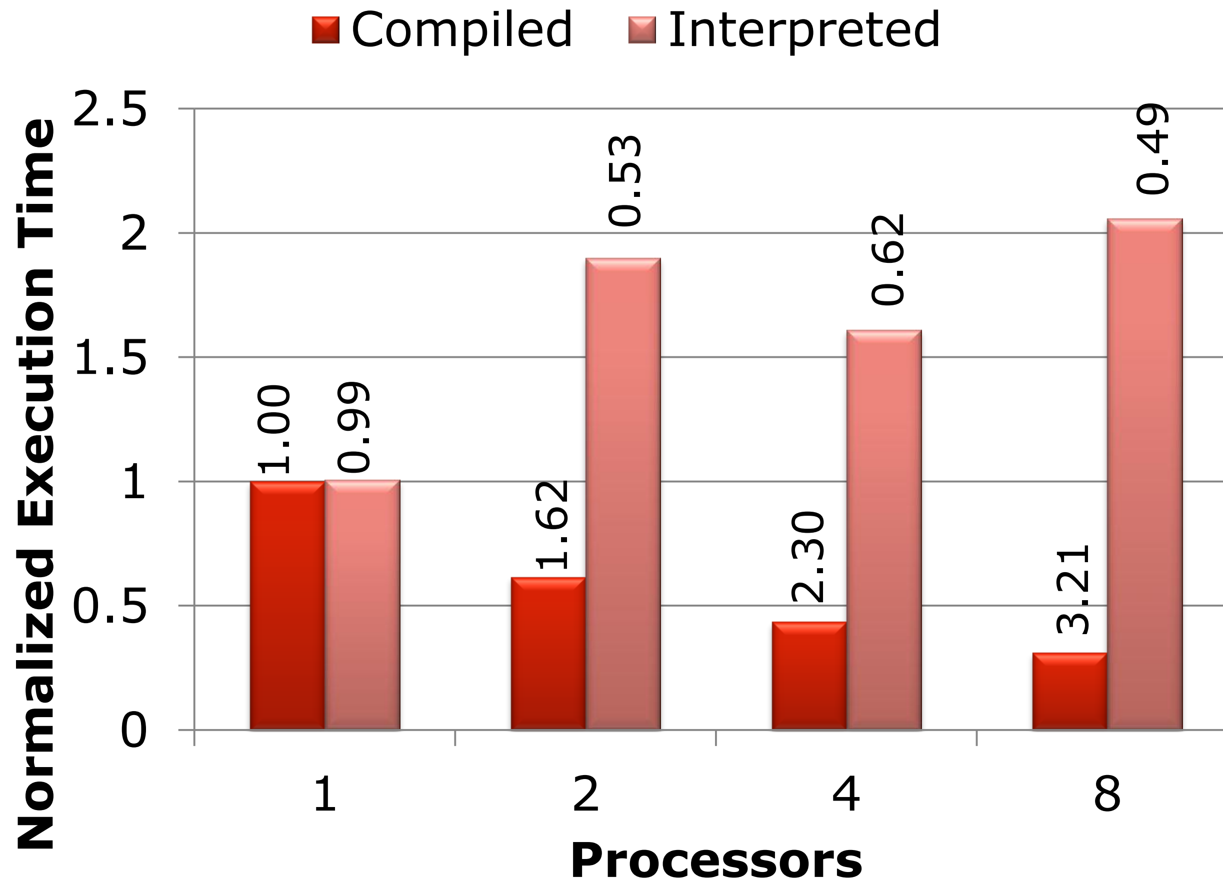


Schedule & Kernel Compilation

- Compile execution graph to executables for each resource after scheduling
  - Defer all synchronization to this point and optimize
- Kernels specialized based on number of processors allocated for it
  - e.g., specialize height of tree reduction
- Greatly reduces overhead compared to dynamic deferred execution model
  - Can have finer-grained Ops with less overhead

Benefits of Runtime Codegen

- GDA with 64 element input



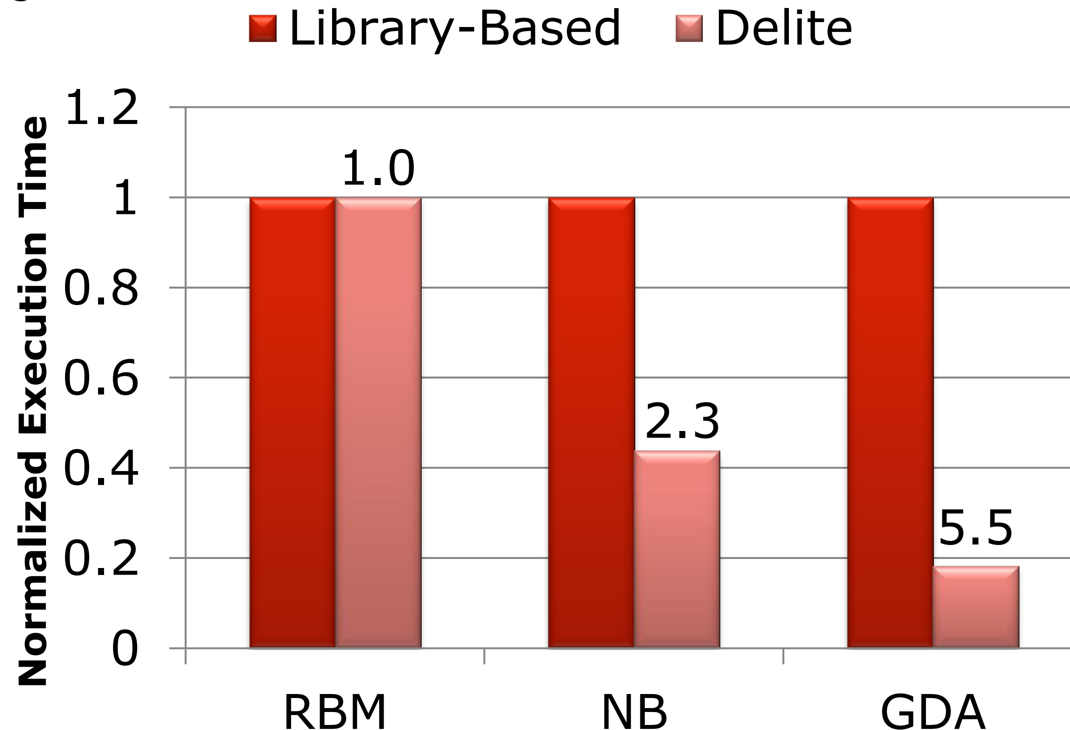
GPU Management

- Cuda host thread launches kernels and automatically performs data transfers as required by schedule
  - Compiler provides helper functions to
    - Copy data structures between address spaces
    - pre-allocate outputs and temporaries
    - select the number of threads & thread blocks

- Provides device memory management for kernels
  - Perform liveness analysis to determine when op inputs and outputs are dead on the GPU
  - Runtime frees dead data when it experiences memory pressure

Cuda Code Generation

- With a library approach we can only launch pre-written kernels
- Code generation enables kernels containing user-defined functions and optimization opportunities
  - e.g., fuse operations into one kernel and keep intermediate results in registers

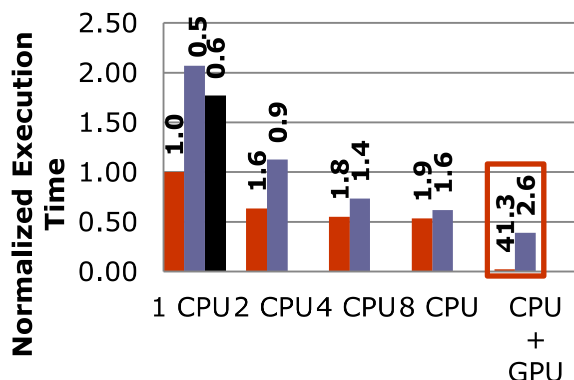


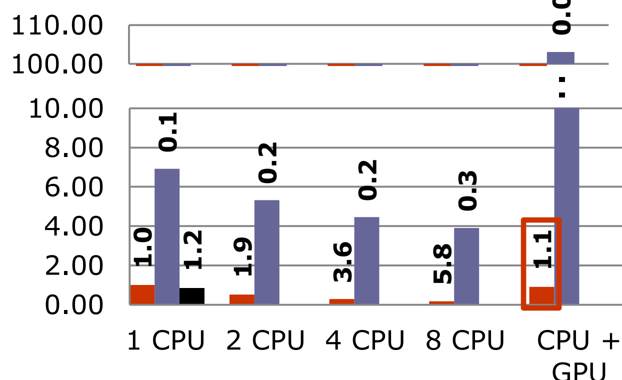
Performance Results

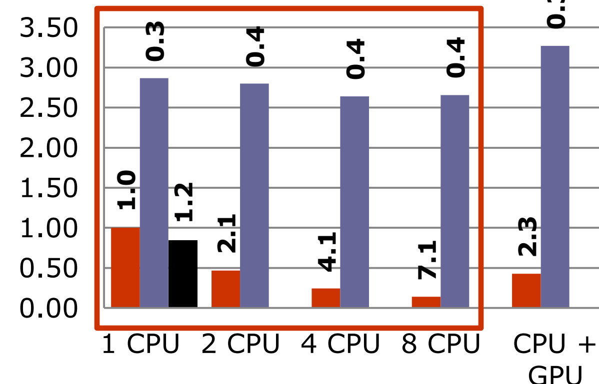
- Machine
  - Two quad-core Nehalem 2.67 GHz processors
  - NVidia Tesla C2050 GPU
- Application Versions
  - OptiML + Delite
  - MATLAB
    - version 1: multi-core (parallelization using “parfor” construct and BLAS)
    - version 2: GPU
  - C++
    - used Armadillo linear algebra library for a sequential baseline
    - Algorithmically identical to OptiML version

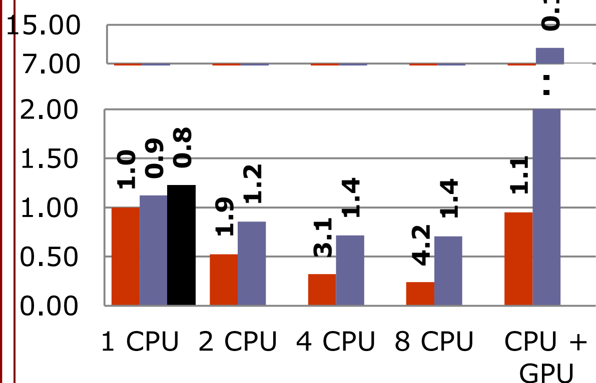
OptiML vs. MATLAB vs. Armadillo (C++)

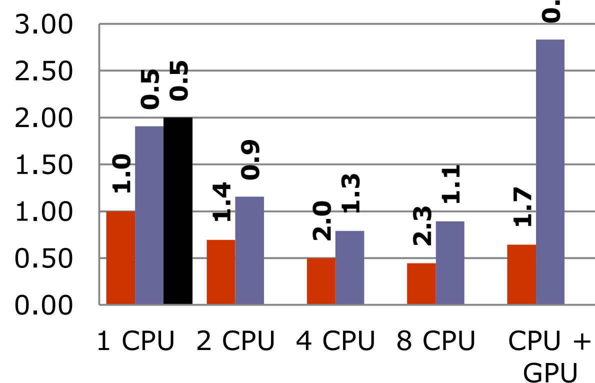
OptiML Parallelized MATLAB C++

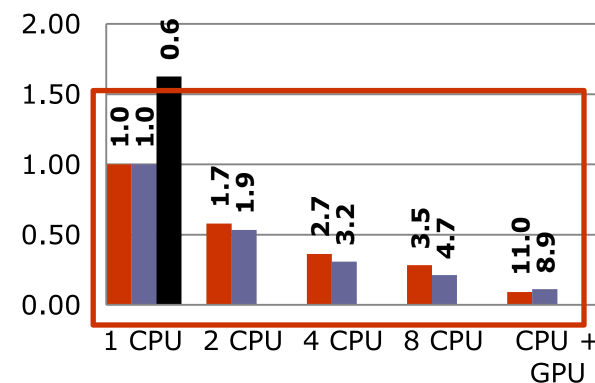












Conclusions

- DSLs can provide both productivity and performance on heterogeneous hardware
- Need to simplify the process of developing DSLs for parallelism
  - Delite provides a framework for creating heterogeneous parallel DSLs
  - Performs generic, parallel, and domain-specific optimizations in a single system
- Visit us at ppl.stanford.edu
  - Link to GitHub project
  - Related publications & projects