Autotuning Stencil-Based Computations on GPUs

Azamat Mametjanov, Daniel Lowell, Ching-Chen Ma,
Boyana Norris

LANS Performance Group
Mathematics and Computer Science Division
Argonne National Laboratory

*Thanks to **CACHE**: Algorithms and Software for Communication Avoidance and Communication Hiding at the Extreme Scale*

*and **SUPER**: Sustained Performance, Energy and Resilience*

September 26, 2012

Motivation

- Finite-difference stencils are very common in numerical modeling. They exhibit high degree of data parallelism and regular structure. However, their memory requirements hinder the performance.
- Our solution consists of
  - Exploitation of a stencil's data access pattern
  - Automatic conversion of C loops to CUDA C host+kernel code
  - Automatic tuning of CUDA C performance parameters

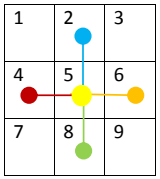
Outline

- Introduction
- **Stencil data structures**
- Transformation and tuning framework of Orio
- Our approach
- Results

Stencils

- Sets of neighboring discrete points in a structured grid
- Stencil pattern determines the interaction among points
  - Domain dimension: 1D, 2D, 3D
  - Stencil shape: star, box
  - Stencil width: distance from stencil center
  - Boundary condition: Dirichlet, periodic

Grid, adjacency matrix and its compression



(a)

Grid Element	1		2		3		4		5		6		7		8		9	
1	57	58	93	94			125	126										
	59	60	95	96			127	128										
2	25	26	61	62	97	98			129	130								
	27	28	63	64	99	100			131	132								
3			29	30	65	66	101	102			133	134						
			31	32	67	68	103	104			135	136						
4	1	2			33	34	69	70	105	106			137	138				
	3	4			35	36	71	72	107	108			139	140				
5			5	6			37	38	73	74	109	110			141	142		
			7	8			39	40	75	76	111	112			143	144		
6					9	10			41	42	77	78	113	114			145	146
					11	12			43	44	79	80	115	116			147	148
7							13	14			45	46	81	82	117	118		
							15	16			47	48	83	84	119	120		
8									17	18			49	50	85	86	121	122
									19	20			51	52	87	88	123	124
9											21	22			53	54	89	90
											23	24			55	56	91	92

(b)

-3		-1		0		1		3	
00	00	00	00	57	58	93	94	125	126
00	00	00	00	59	60	95	96	127	128
00	00	25	26	61	62	97	98	129	130
00	00	27	28	63	64	99	100	131	132
00	00	29	30	65	66	101	102	133	134
00	00	31	32	67	68	103	104	135	136
1	2	33	34	69	70	105	106	137	138
3	4	35	36	71	72	107	108	139	140
5	6	37	38	73	74	109	110	141	142
7	8	39	40	75	76	111	112	143	144
9	10	41	42	77	78	113	114	145	146
11	12	43	44	79	80	115	116	147	148
13	14	45	46	81	82	117	118	00	00
15	16	47	48	83	84	119	120	00	00
17	18	49	50	85	86	121	122	00	00
19	20	51	52	87	88	123	124	00	00
21	22	53	54	89	90	00	00	00	00
23	24	55	56	91	92	00	00	00	00

(c)

Outline

- Introduction
- Stencil data structures
- **Transformation and tuning framework of Orio**
- Overview of the approach
- Results

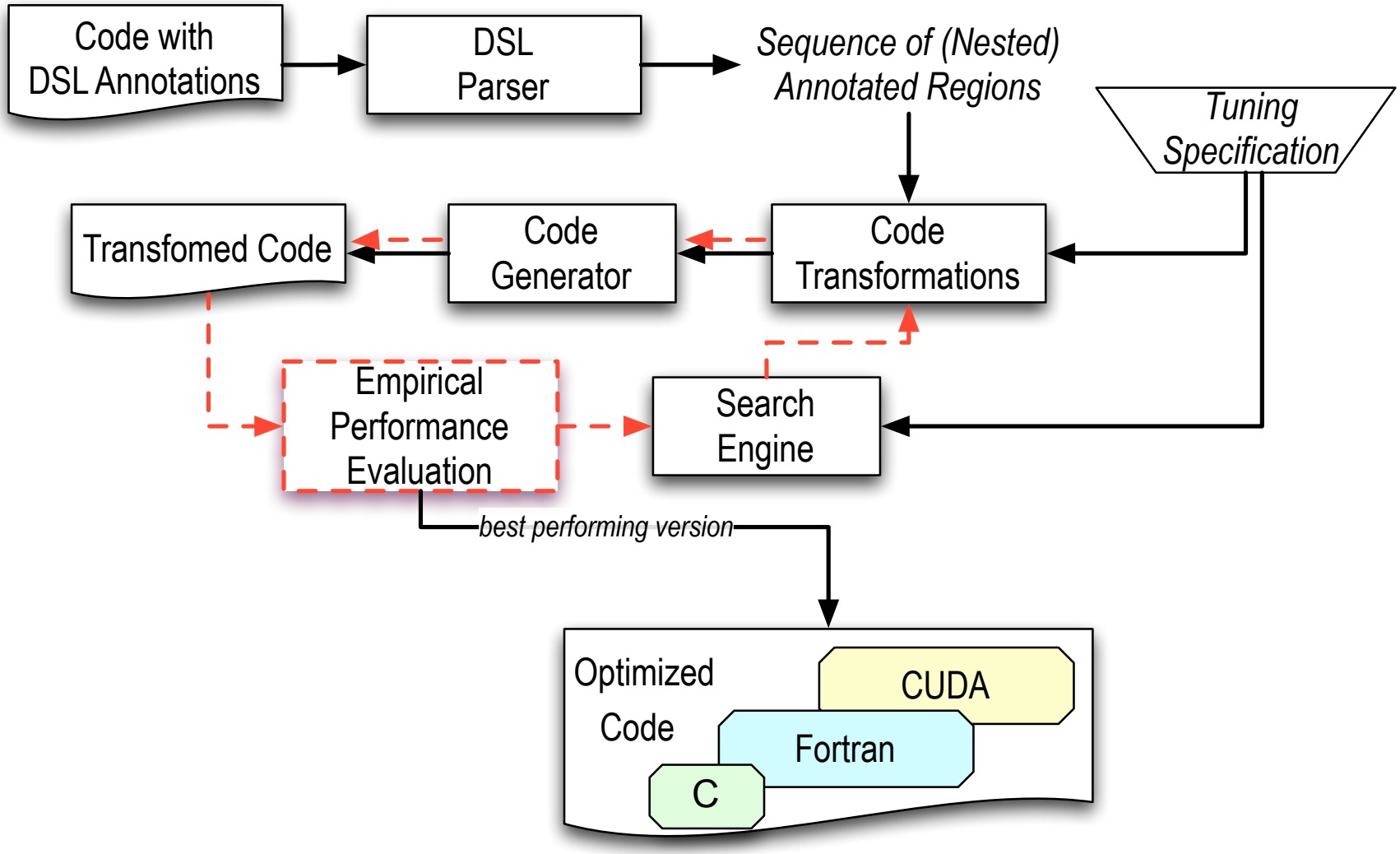
Method: Code Transformation

- Motivation
  - Compilation: HL source code into LL portable executable code
  - Optimization: performance, energy
  - Refactoring: resiliency, maintainability, readability
- Workflow
  - Parse: any structured source text into abstract syntax tree
  - Analyze: common intermediate representation
  - Transform: compositions of reusable transforms
  - Generate: any structured target text
- Challenges
  - Create source and target domains
  - Create analysis and transformation rules

Method: Code Tuning

- Motivation
  - Deep component stacks
  - Each component is adjustable
- Workflow
  - System model: pre-specified, learned
  - Application profile: memory-/compute-bound
  - Configure: create a valid configuration of parameters
  - Select: the best performing parameter configuration
- Challenges
  - Auto-profile
  - Auto-modify
  - Search
  - Whole-app autotuning

Orio autotuning framework



Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- **Overview of the approach**
- Results

Begin with reference C code

```
for(i=0; i<=nrows-1; i++) {
  for(j=0; j<=ndiags-1; j++){
    col = i+offsets[j];
    if(col>=0&&col<nrows)
      y[i] += A[i+j*nrows] * x[col];
  }
}
```

Add a DSL annotation

```
/*@ begin Loop(…

 for(i=0; i<=nrows-1; i++) {
   for(j=0; j<=ndiags-1; j++){
     col = i+offsets[j];
     if(col>=0&&col<nrows)
       y[i] += A[i+j*nrows] * x[col];
   }
  }
) @*/
for …
/*@ end @*/
```

Specify performance parameters (optional)

```
/*@ begin Loop(transform CUDA(
        threadCount=TC,
        blockCount=BC,
        streamCount=SC,
        preferL1Size=PL,
        unrollInner=UIF, …
    )
for …
) @*/
for …
/*@ end @*/
```

Specify parameter search ranges

```
/*@ begin PerfTuning(
   def performance_params{
      param TC[] = range(32,1025,32);
      param BC[] = range(14,113,14);
      param SC[] = range(1,17);
      param PL[]  = [16,48];
      param UIF[] = range(1,8); ...
   }
) @*/
/*@ begin Loop(transform CUDA(
…
/*@ end @*/
```

Define empirical experiment inputs

```
/*@ begin PerfTuning(
   def input_params {
      param M[] = [16,32,64,128,256]; ...
   }
   def input_vars {
      decl static double A[M*N*P*NOS*DOF] = random;
      decl static double x[M*N*P*DOF]     = random;
      decl static double y[M*N*P*DOF]     = 0; …
}
…
) @*/
```

Define build and search parameters

```
/*@ begin PerfTuning(
   def build {
      arg build_command = 'nvcc -arch=sm_20 @CFLAGS';
   }
   def performance_counter {
      arg repetitions = 10;
   }
   ...
) @*/
```

Launch

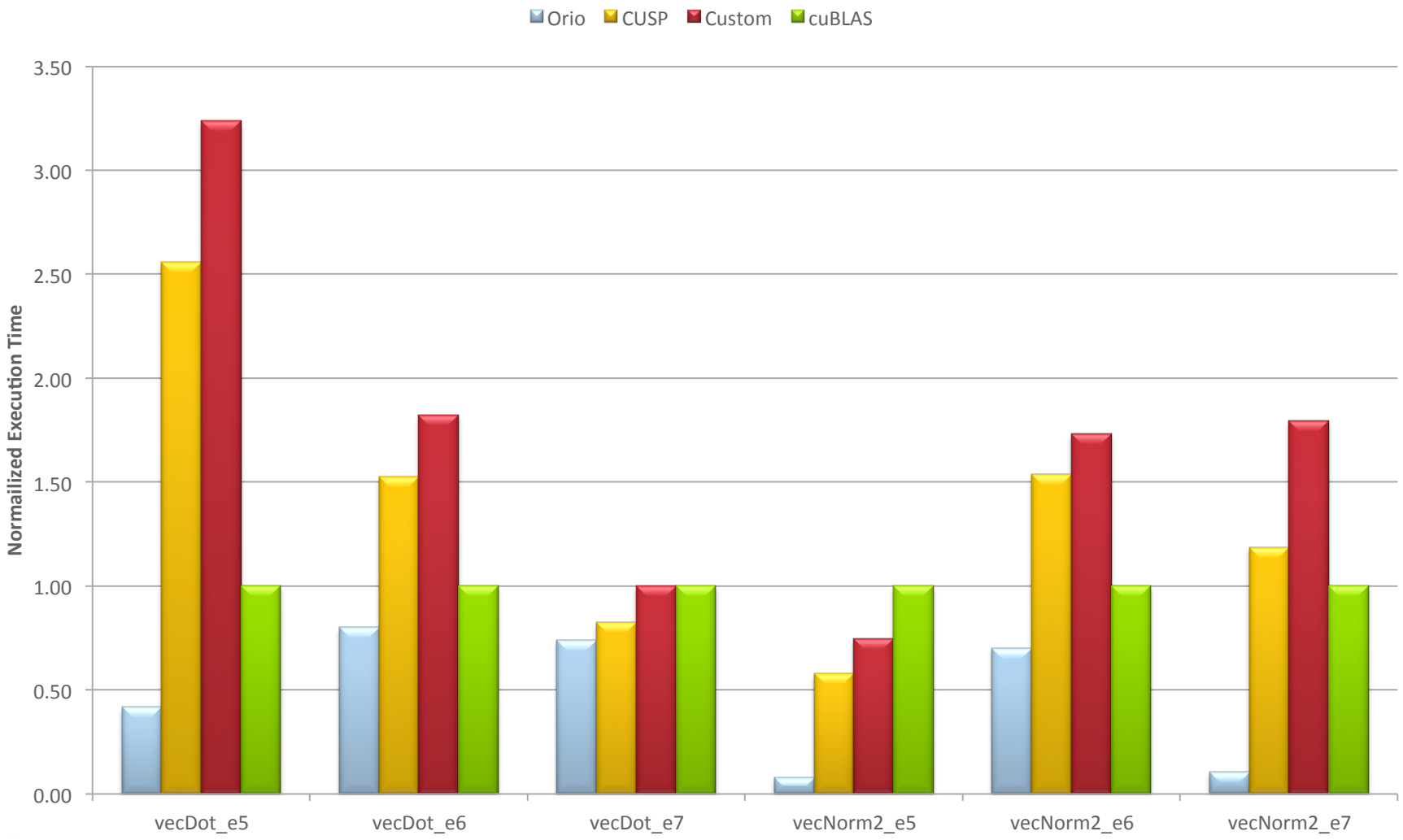
```
./orcuda matVec3D.c
…
Search_Space        = 1.024e+04
Number_of_Parameters = 05
Numeric_Parameters   = 05
Binary_Parameters    = 00
['TC', 'BC', 'UIF', 'PL', 'CFLAGS']
[[32, 64, 96, 128, 160, 192, 224, 256, 288, 320, 352, 384, 416, 448, 480, 512, 544, 576, 608, 640, 672, 704, 736, 768, 800, 832, 864, 896, 928, 960, 992, 1024], [14, 28, 42, 56, 70, 84, 98, 112], [1, 2, 3, 4, 5], [16, 48], ['', '-O1', '-O2', '-O3']]
```

Outline

- Introduction
- Stencil data structures
- Transformation and tuning framework of Orio
- Overview of the approach
- **Results**

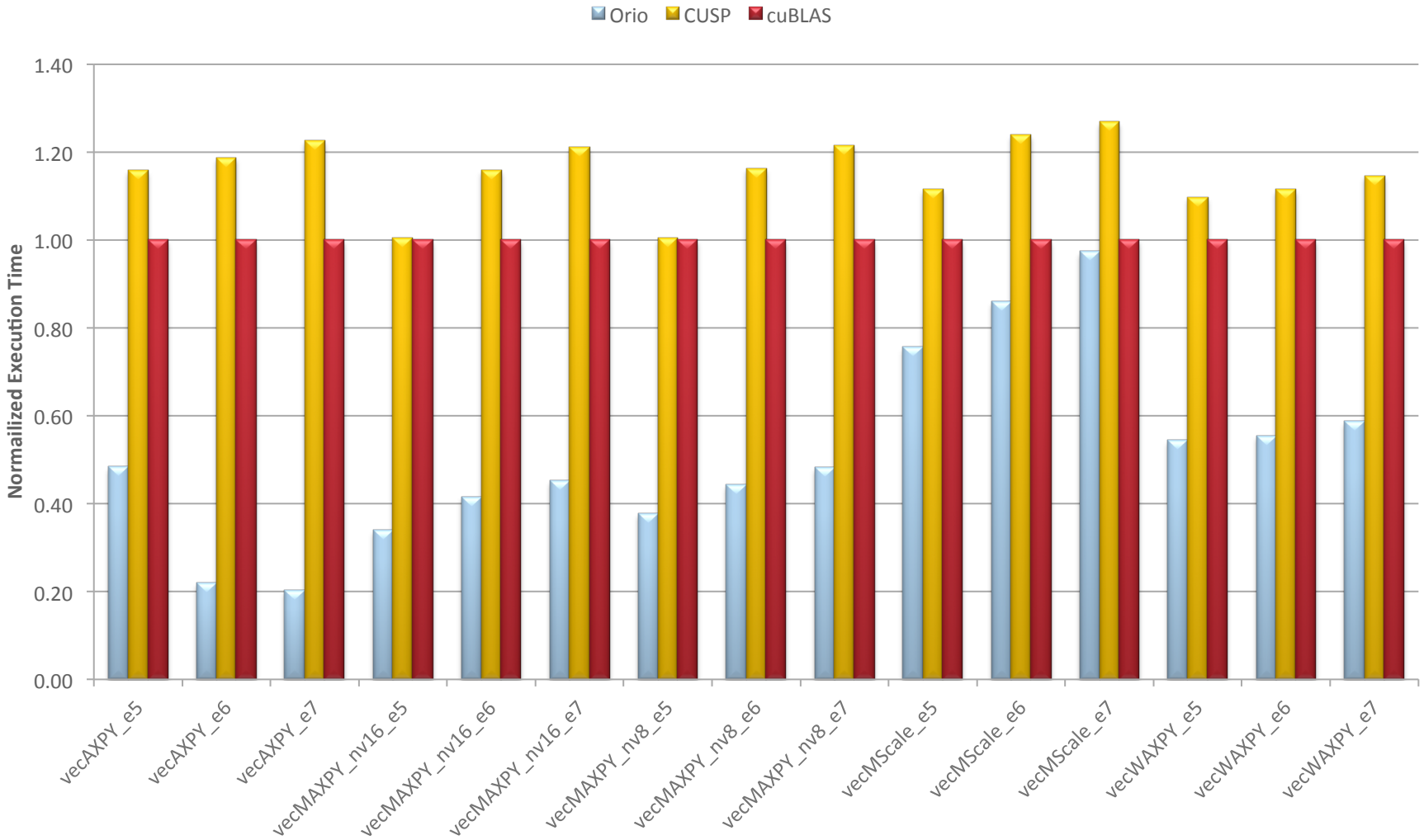
Reduction kernels

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070



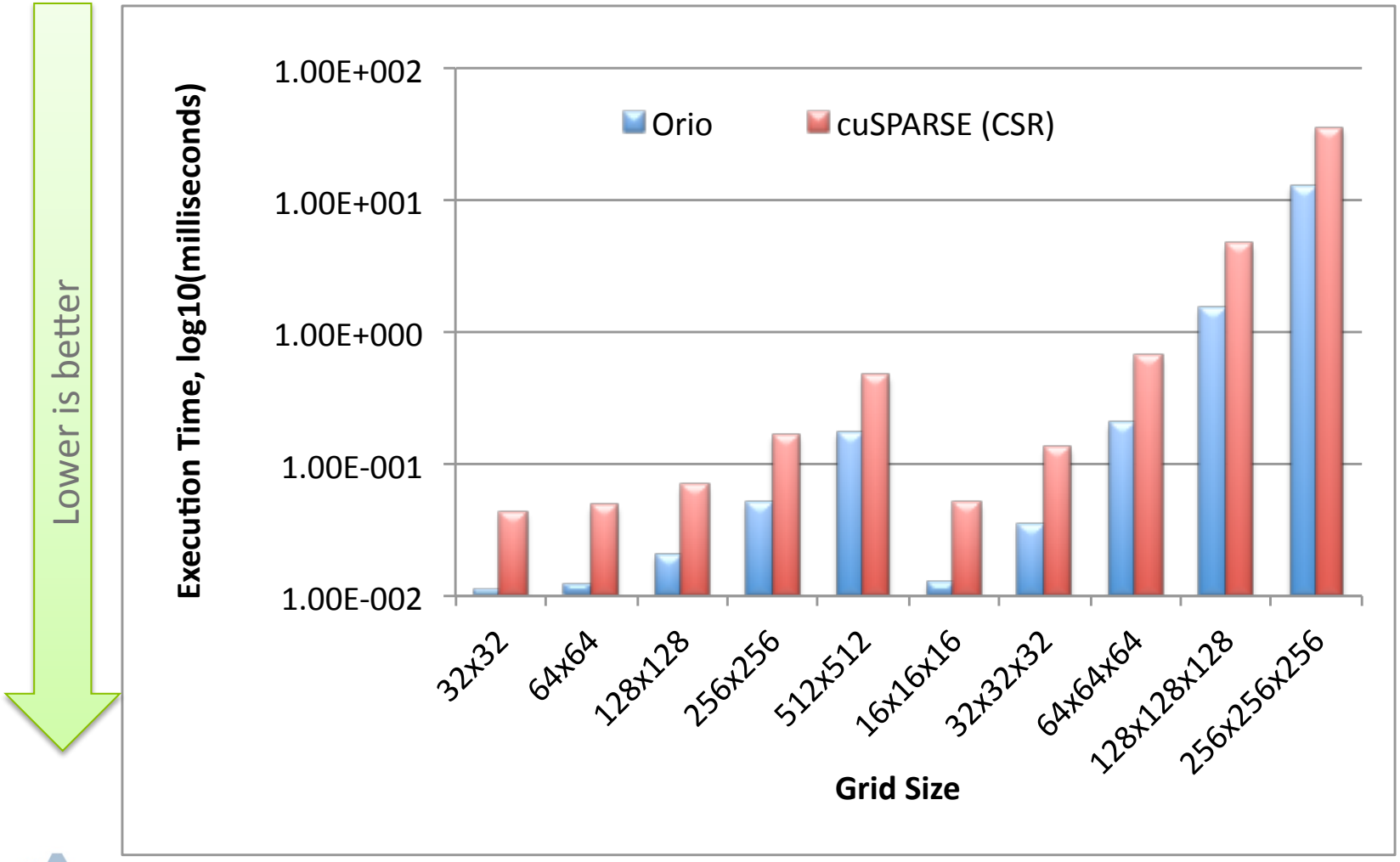
Pointwise kernels

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070

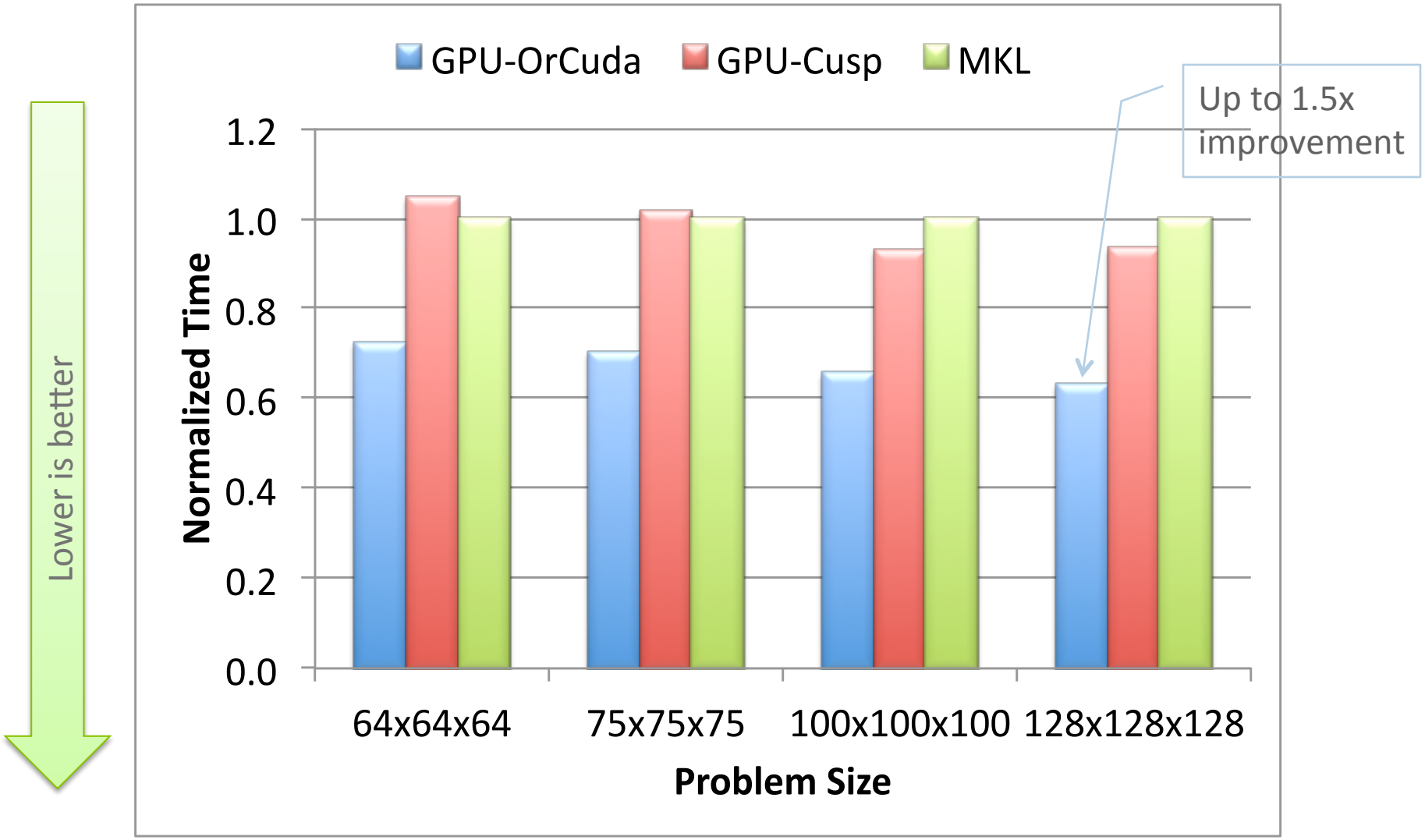


Example: Sparse matrix-vector product (5- and 7-point stencil) on a GPU

Intel Xeon (dual quad-core E5462 processors), 2.8GHz; GPU: NVIDIA Fermi C2070



Application: Bratu solid fuel ignition problem



Conclusion

- Workflow:
  - Functionality
  - Performance
  - Stability
- Application stack is complex
  - Dependency depth
  - Heterogeneity at each level
- Autotuning provides end-to-end integration
  - Hardware and components will continue to change
  - Application only needs to be written once
  - Programmability through portability

Thank you

http://tinyurl.com/OrioTool