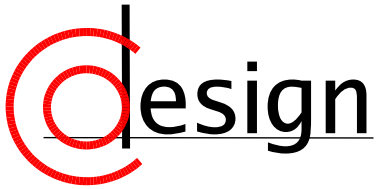
Towards Domain-specific Computing for Stencil Codes in HPC

Richard Membarth[1], Frank Hannig[1], Jürgen Teich[1], and Harald Köstler[2]

[1]Hardware/Software Co-Design, University of Erlangen-Nuremberg

[2]System Simulation, University of Erlangen-Nuremberg

WOLFHPC'12, November 16, 2012, Salt Lake City



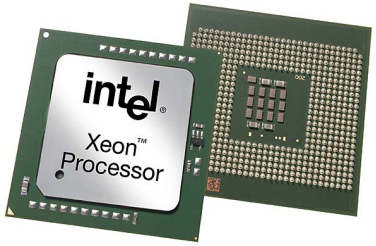
FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Motivation: Exascale Performance for Stencil Codes

Exascale hardware will be heterogeneous:

- standard multi-core processors



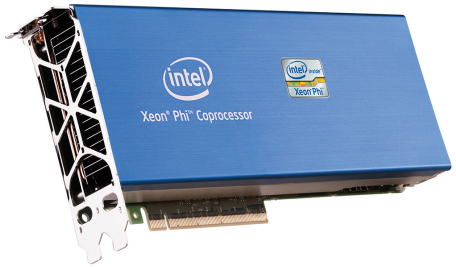
Intel Xeon

AMD Opteron

- and accelerators (e. g., GPU)

NVIDIA Tesla

AMD Radeon



Intel MIC

Challenge: 3P's

- **productivity**
  - algorithm description at a high-level
  - hide low-level details from programmer
- **portability**
  - support different target architectures from the same algorithm description
  - support different target languages from the same algorithm description
- **performance**
  - portable: high performance on different target hardware
  - competitive: comparable performance to hand-written code

Challenge: 3P's

- **productivity**
  - algorithm description at a high-level
  - hide low-level details from programmer
- **portability**
  - support different target architectures from the same algorithm description
  - support different target languages from the same algorithm description
- **performance**
  - portable: high performance on different target hardware
  - competitive: comparable performance to hand-written code

Challenge: 3P's

- **productivity**
  - algorithm description at a high-level
  - hide low-level details from programmer
- **portability**
  - support different target architectures from the same algorithm description
  - support different target languages from the same algorithm description
- **performance**
  - portable: high performance on different target hardware
  - competitive: comparable performance to hand-written code

Challenge: 3P's

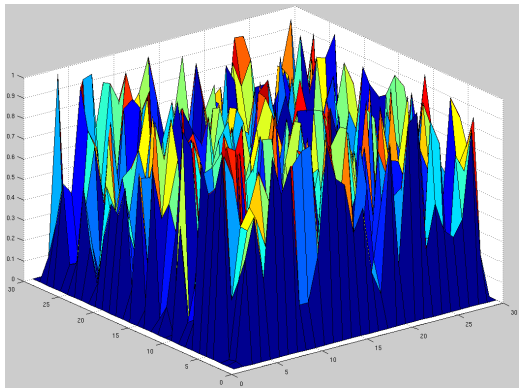
- **productivity**
  - algorithm description at a high-level
  - hide low-level details from programmer
- **portability**
  - support different target architectures from the same algorithm description
  - support different target languages from the same algorithm description
- **performance**
  - portable: high performance on different target hardware
  - competitive: comparable performance to hand-written code

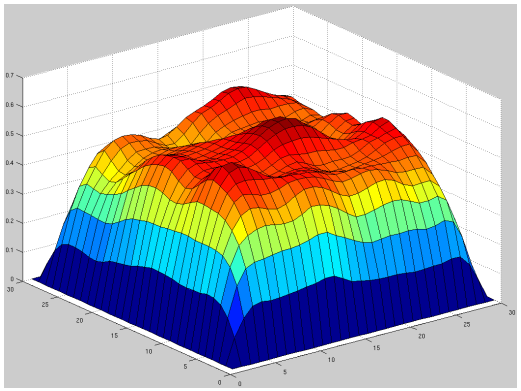
Remedy:

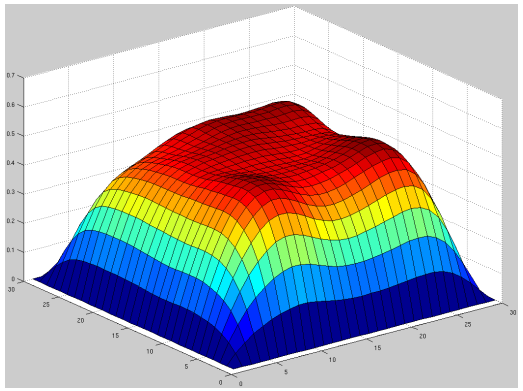
Domain-Specific Language (DSL) for stencil codes (multigrid)

Multigrid Idea

1. smoothing property
2. coarse grid principle



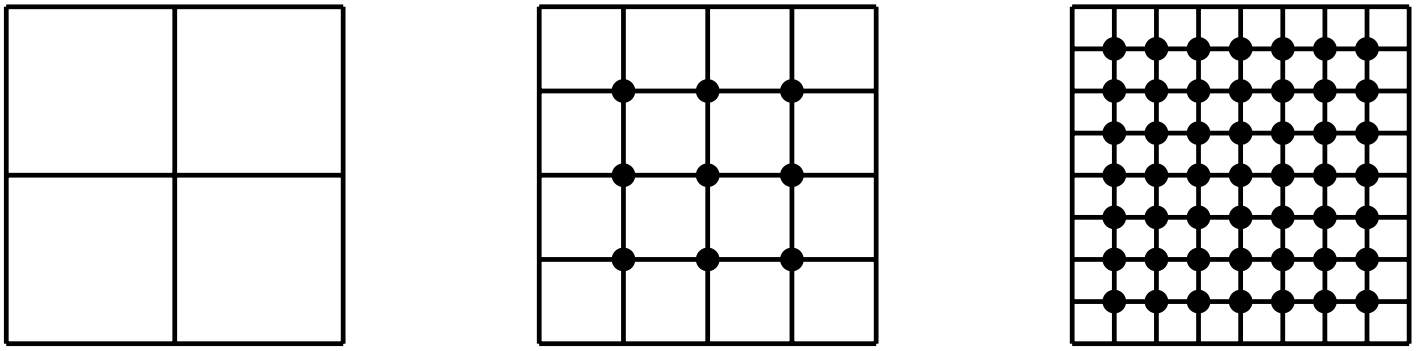




smooth error on fine grid

Multigrid Idea

1. smoothing property
2. coarse grid principle



approximate smooth error on coarser grids

Multigrid Correction Scheme

Recursive V-cycle: $u_h^{(k+1)} = V_h\left(u_h^{(k)}, A^h, f^h, \nu_1, \nu_2\right)$

if *coarsest level* **then**
  solve $A^h u^h = f^h$ exactly or by many smoothing iterations;
else
  $\bar{u}_h^{(k)} = \mathcal{S}_h^{\nu_1}\left(u_h^{(k)}, A^h, f^h\right)$; {pre-smoothing}
  $r^h = f^h - A^h \bar{u}_h^{(k)}$; {compute residual}
  $r^H = R r^h$; {restrict residual}
  $e^H = V_H(0, A^H, r^H, \nu_1, \nu_2)$; {recursion}
  $e^h = P e^H$; {interpolate error}
  $\tilde{u}_h^{(k)} = \bar{u}_h^{(k)} + e^h$; {coarse grid correction}
  $u_h^{(k+1)} = \mathcal{S}_h^{\nu_2}\left(\tilde{u}_h^{(k)}, A^h, f^h\right)$; {post-smoothing}
end

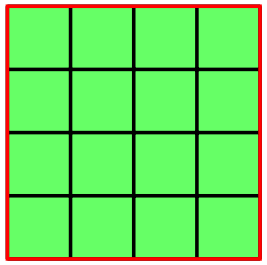
Domain-Specific Language (DSL)

Images in the DSL

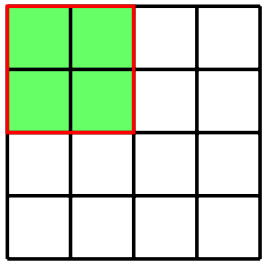
Define images of size *width*×*height*

```
1 Image<float> IN(width, height);
2 Image<float> OUT(width, height);
```

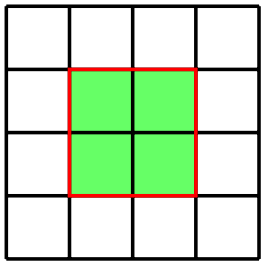
Writing to the *output* image: *Iteration Space*



Output image.



Crop of output image.



Crop of output image with offset.

```
1 IterationSpace<float> ISOut(OUT, width-10, height-10, 5, 5);
```

Images in the DSL

Reading from an *input* image: *Accessor*

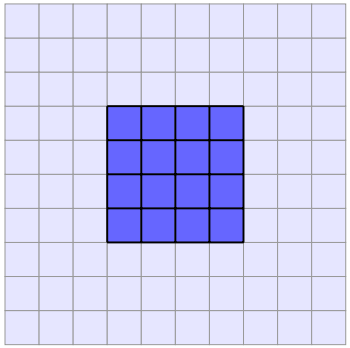


Image and boundary.

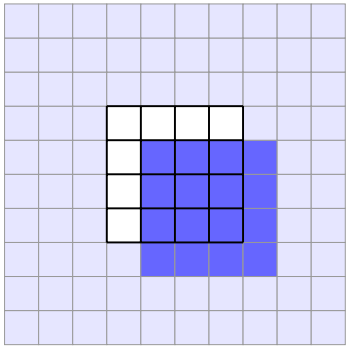


Image offset.

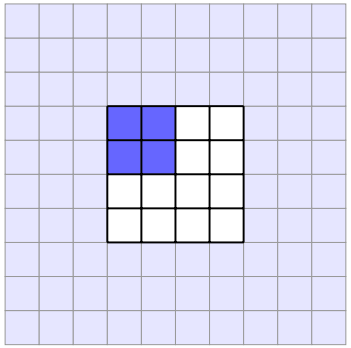


Image crop.

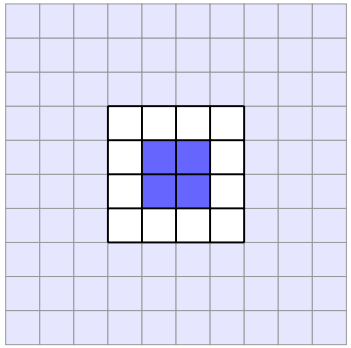


Image crop with offset.

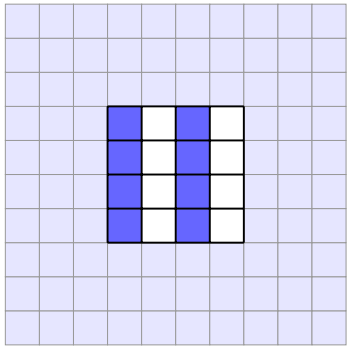
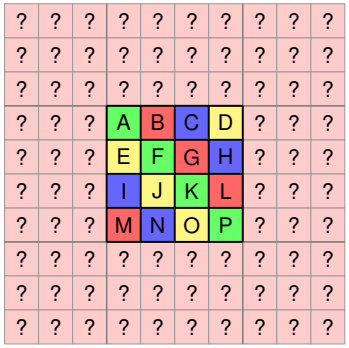


Image stride.

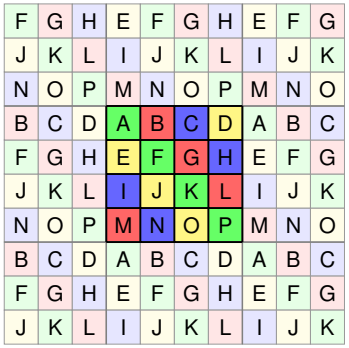
```
1 Accessor<float> AccIn(IN);
```

Different *Accessors* for interpolation: nearest neighbor, bilinear, bicubic, etc.

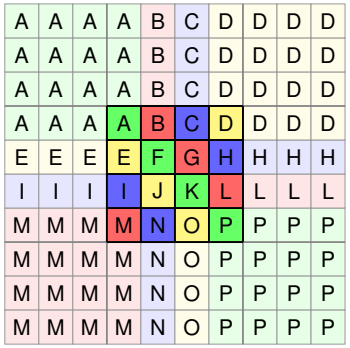
Accessing Pixels out of Bounds: Boundary Handling



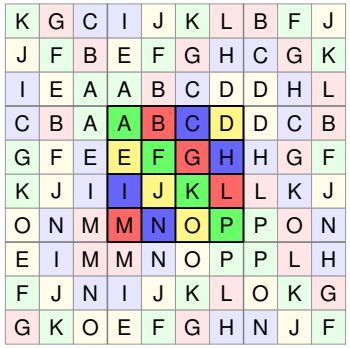
Undefined.



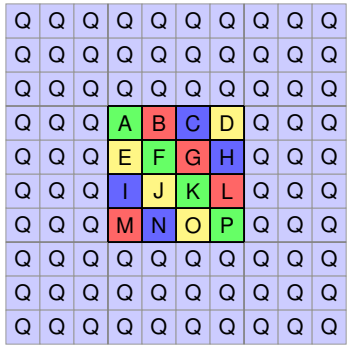
Repeat.



Clamp.



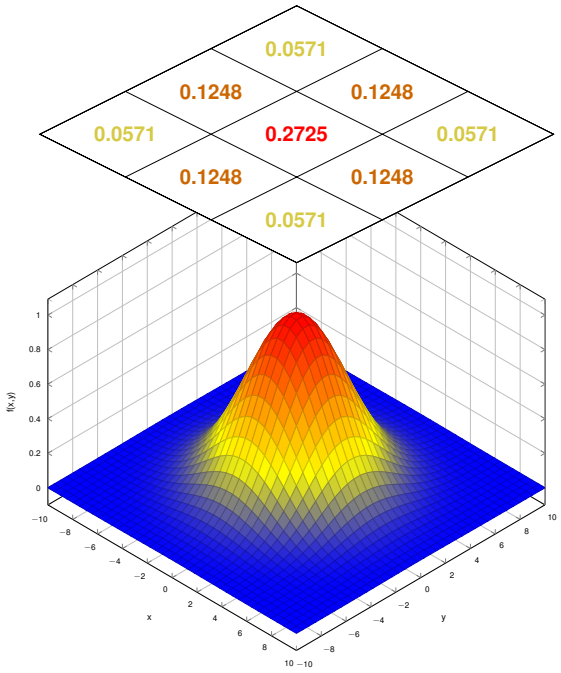
Mirror.



Constant.

```
1 Image<float> IN(width, height);
2 BoundaryCondition<float> BcIn(IN, size_x, size_y, BOUNDARY_CLAMP);
3 Accessor<float> AccIn(BcIn);
```

Filter Mask for Local Operators




```
1 float mask[] = { 0.0571, 0.1248, 0.0571, ... };
2 Mask<float> cMask(size_x, size_y);
3 cMask = mask;
4
5 // use Mask to define boundary handling
6 BoundaryCondition<float> BcIn(IN, cMask, BOUNDARY_CLAMP);
```

Application for Multigrid on GPU Accelerators

High Dynamic Range (HDR) Compression

- the dynamic range of an image refers to the ratio between the brightest and darkest portions of the image which is accurately captured or observed
- HDR compression is used to get more details out of the image [SIGGRAPH'02]

Input image.

Output image.

[SIGGRAPH'02] Raanan Fattal, Dani Lischinski, and Michael Werman. "Gradient Domain High Dynamic Range Compression". In: *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. ACM, July 2002, pp. 249–256.

Describing Stencils in the DSL

```
1 // filter mask for gradient calculation
2 const float filter_gradient[] = {
3    0, -1,  0,
4   -1,  4, -1,
5    0, -1,  0
6 };
7 Mask<float> Mgradient(size_x, size_y);
8 Mgradient = filter_gradient;
9
10 // image for RHS
11 Image<float> RHS(width, height);
12 IterationSpace<float> IsRHS(RHS);
13
14 // input image
15 int width, height;
16 image = read_image(&width, &height, "input.pgm");
17 Image<float> IN(width, height);
18 IN = image;
19
20 // reading from IN with mirroring as boundary condition
21 BoundaryCondition<float> BcInMirror(IN, Mgradient, BOUNDARY_MIRROR);
22 Accessor<float> AccInConst(BcInMirror);
23
24 // kernel declaration
25 GradientKernel Gradient(IsRHS, AccInConst, MGradient);
26
27 // first step: compute the gradient of the image
28 Gradient.execute();
```

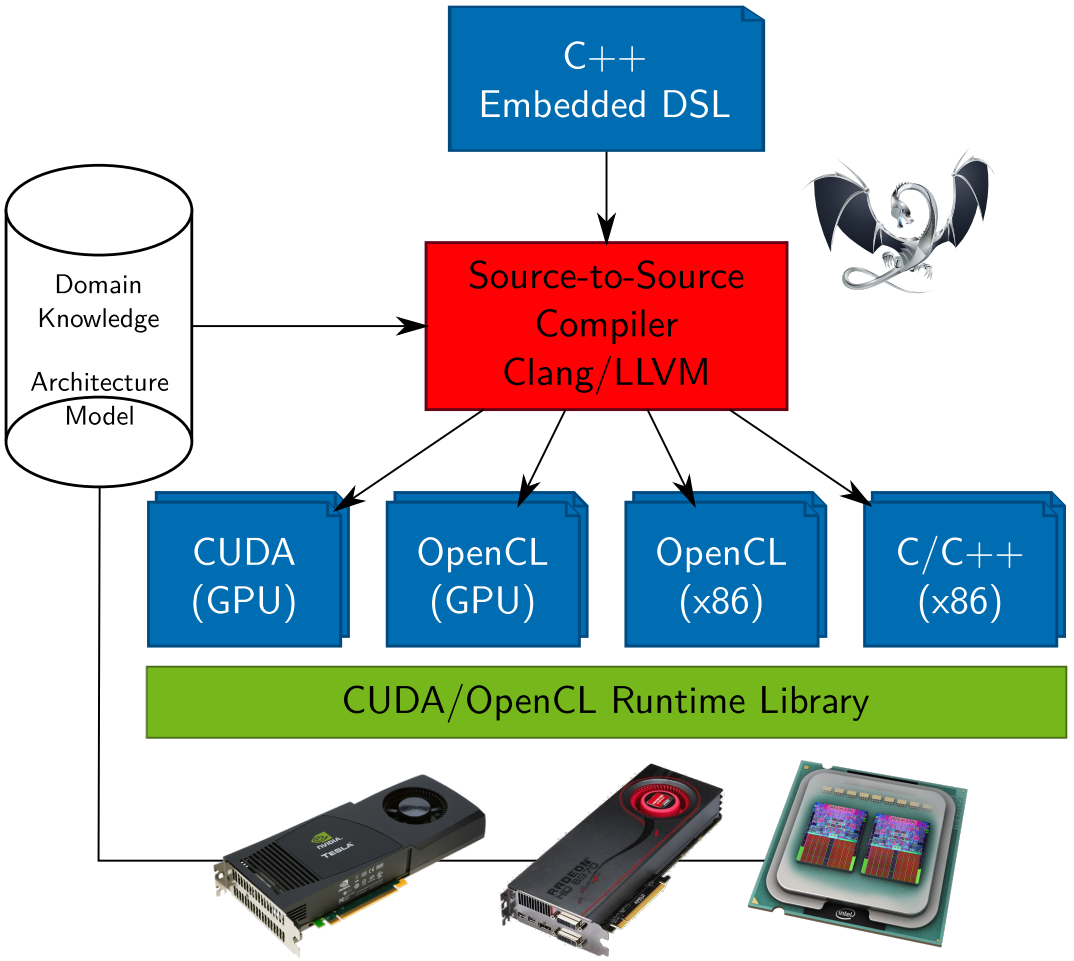
Describing Stencils in the DSL

```
1 class GradientKernel : public Kernel<float> {
2   private:
3     Accessor<float> &In;
4     Mask<float> &cMask;
5
6   public:
7     GradientKernel(IterationSpace<float> &IS, Accessor<float> &In, Mask<float> &cMask) :
8       Kernel(IS), In(In), cMask(cMask)
9     { addAccessor(&In); }
10
11    void kernel() {
12      output() = convolve(cMask, SUM, [&] () -> float {
13        return cMask() * In(cMask);
14      });
15    }
16 };
```

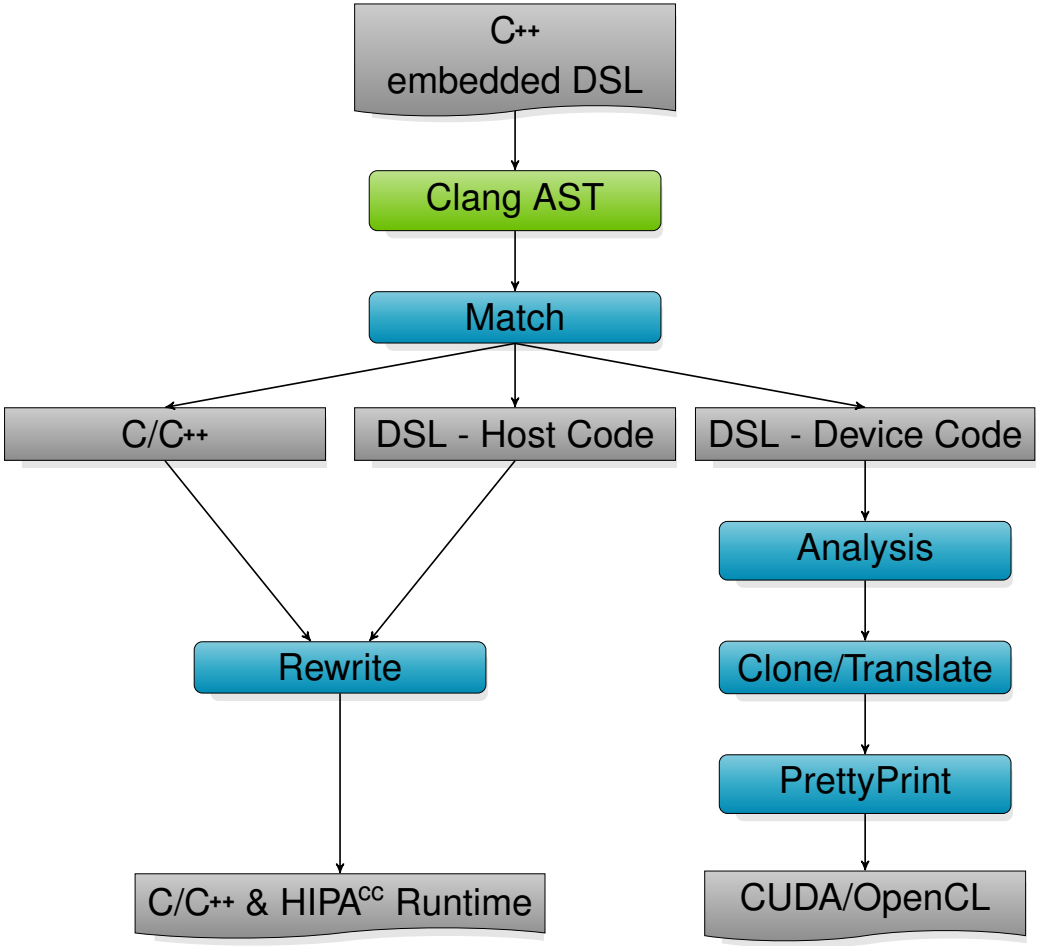
```
1     void kernel() {
2       output() = - In(0, 1) - In(-1, 0) + 4*In() - In(1, 0) - In(0, -1);
3     }
```

The Heterogeneous Image Processing Acceleration (HIPA^{cc}) Framework

HIPA^{cc}
Heterogeneous Image Processing Acceleration



Compiler Work Flow



HDR Compression: Implementations

- using HIPA^{cc}
  - high-level implementation
  - ω-Jacobi
  - one kernel per V-cycle component
- hand-tuned Graphics Processing Unit (GPU) implementation
  - OpenCL implementation
  - tuned for Fermi devices
  - red-black Gauss-Seidel
  - *kernel fusion & wavefront blocking*

HDR Compression: Implementations

- using HIPA^{cc}
  - high-level implementation
  - ω-Jacobi
  - one kernel per V-cycle component
- hand-tuned GPU implementation
  - OpenCL implementation
  - tuned for Fermi devices
  - red-black Gauss-Seidel
  - *kernel fusion* & *wavefront blocking*

Evaluation & Results

Evaluation

- **productivity**
  - DSL description: 3 lines per kernel computation
  - 1/2 day for whole implementation
  - reference implementation: 1200 lines of OpenCL code
  - 3 months optimization after basic implementation
- **portability**
  - we can generate different code variants for CUDA and OpenCL (device-specific)
  - reference implementation: implementation in OpenCL, optimized for Fermi hardware
- **performance**
  - portable & competitive performance on different target hardware
  - reference implementation: good performance on Fermi hardware

Evaluation

- **productivity**
  - DSL description: 3 lines per kernel computation
  - 1/2 day for whole implementation
  - reference implementation: 1200 lines of OpenCL code
  - 3 months optimization after basic implementation
- **portability**
  - we can generate different code variants for CUDA and OpenCL (device-specific)
  - reference implementation: implementation in OpenCL, optimized for Fermi hardware
- **performance**
  - portable & competitive performance on different target hardware
  - reference implementation: good performance on Fermi hardware

Evaluation

- **productivity**
  - DSL description: 3 lines per kernel computation
  - 1/2 day for whole implementation
  - reference implementation: 1200 lines of OpenCL code
  - 3 months optimization after basic implementation
- **portability**
  - we can generate different code variants for CUDA and OpenCL (device-specific)
  - reference implementation: implementation in OpenCL, optimized for Fermi hardware
- **performance**
  - portable & competitive performance on different target hardware
  - reference implementation: good performance on Fermi hardware

Results

	Tesla C2050			Quadro FX 5800		
	Manual	OpenCL	CUDA	Manual	OpenCL	CUDA
L1: smooth	0.53	0.58	0.79	1.35	1.50	1.01
L1: smooth		0.57	0.79		1.48	0.99
L1: residual	0.67	0.57	0.79	1.65	1.62	0.93
L1: restrict		0.28	0.28		0.59	0.53
L2: smooth	0.12	0.16	0.26	0.35	0.44	0.26
L2: smooth		0.16	0.26		0.44	0.27
L2: residual	0.19	0.16	0.25	0.44	0.46	0.26
L2: restrict		0.08	0.12		0.18	0.16
L3–L6	0.70	0.63	1.85	1.33	1.73	1.34
L2: interpolate		0.21	0.17		0.29	0.18
L2: smooth	0.15	0.16	0.27	0.34	0.45	0.27
L2: smooth	0.34	0.16	0.27	0.86	0.44	0.27
L1: interpolate		0.83	0.48		0.96	0.61
L1: smooth	0.53	0.57	0.89	1.35	1.48	1.01
L1: smooth	0.53	0.57	0.88	1.35	1.49	1.01
$\sum$V-cycle	3.90	5.75	8.31	9.02	13.54	9.07

Execution times in *ms* for the HDR compression on the **Quadro FX 5800** and **Tesla C2050** for an image of 2048×2048 pixels. Shown is the hand-tuned **OpenCL** as well as the generated **CUDA** and **OpenCL** implementations.

Conclusions

- DSLs provide a performance-portable solution across several architectures with respect to
  - productivity
  - portability (flexibility)
  - performance (competitive)
- extension of the DSL to match stencil codes
  - 2D domain $\rightarrow$ 3D domain
  - boundary handling
  - interpolation
  - concise syntax for different multigrid variants (V-cycle, W-cycle, etc.)

Future Directions

Combination of different disciplines:

- algorithmic engineering
- domain-specific representation and modeling
- domain-specific optimization and generation
- polyhedral optimization and code generation
- platform-specific code optimization and generation

ExaStencils: Advanced Stencil Code Engineering

`http://www.exastencils.org`

Questions?

HIPA^{cc} framework sources released under *Simplified BSD License*.

`https://sourceforge.net/projects/hipacc`