Towards Domain-specific Computing for Stencil Codes in HPC

Richard Membarth¹, Frank Hannig¹, Jürgen Teich¹, and Harald Köstler²
¹Hardware/Software Co-Design, University of Erlangen-Nuremberg
²System Simulation, University of Erlangen-Nuremberg

WOLFHPC’12, November 16, 2012, Salt Lake City
Motivation: Exascale Performance for Stencil Codes

Exascale hardware will be heterogeneous:

- standard multi-core processors

  ![Intel Xeon](Intel_Xeon.png)
  ![AMD Opteron](AMD_Opteron.png)

- and accelerators (e.g., GPU)

  ![NVIDIA Tesla](NVIDIA_Tesla.png)
  ![AMD Radeon](AMD_Radeon.png)
  ![Intel MIC](Intel_MIC.png)
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: \( u_h^{(k+1)} = V_h \left( u_h^{(k)}, A^h, f^h, v_1, v_2 \right) \)

1 if coarsest level then
   2 solve \( A^h u^h = f^h \) exactly or by many smoothing iterations;
3 else
   4 \( \bar{u}_h^{(k)} = S_h^{v_1} \left( u_h^{(k)}, A^h, f^h \right) \); \{pre-smoothing\}
   5 \( r^h = f^h - A^h \bar{u}_h^{(k)} \); \{compute residual\}
   6 \( r^H = R r^h \); \{restrict residual\}
   7 \( e^H = V_H \left( 0, A^H, r^H, v_1, v_2 \right) \); \{recursion\}
   8 \( e^h = Pe^H \); \{interpolate error\}
   9 \( \tilde{u}_h^{(k)} = \bar{u}_h^{(k)} + e^h \); \{coarse grid correction\}
10 \( u_h^{(k+1)} = S_h^{v_2} \left( \tilde{u}_h^{(k)}, A^h, f^h \right) \); \{post-smoothing\}
11 end
Domain-Specific Language (DSL)
Images in the DSL

Define images of size $width \times height$

```
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

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

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

---

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

```cpp
class GradientKernel : public Kernel<float> {
private:
    Accessor<float> &In;
    Mask<float> &cMask;

public:
    GradientKernel(IterationSpace<float> &IS, Accessor<float> &In, Mask<float> &cMask) :
        Kernel(IS), In(In), cMask(cMask) {
        addAccessor(&In); }

    void kernel() {
        output() = convolve(cMask, SUM, [&] () -> float {
            return cMask() * In(cMask);
        });
    }
};

void kernel() {
    output() = - In(0, 1) - In(-1, 0) + 4*In() - In(1, 0) - In(0, -1);
}
```
The Heterogeneous Image Processing Acceleration (HIPA\textsuperscript{cc}) Framework

C++ Embedded DSL

Source-to-Source Compiler
Clang/LLVM

CUDA (GPU)
OpenCL (GPU)
OpenCL (x86)
C/C++ (x86)

CUDA/OpenCL Runtime Library
Compiler Work Flow

C++
embedded DSL

Clang AST

Match

C/C++

DSL - Host Code

DSL - Device Code

Rewrite

C/C++ & HIPA^cc Runtime

Analysis

Clone/Translate

PrettyPrint

CUDA/OpenCL
HDR Compression: Implementations

• using HIPA\textsuperscript{cc}
  • high-level implementation
  • \(\omega\)-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
  • $\omega$-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

<table>
<thead>
<tr>
<th></th>
<th>Tesla C2050</th>
<th></th>
<th>Quadro FX 5800</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Manual</td>
<td>OpenCL</td>
<td>CUDA</td>
<td>Manual</td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.53</td>
<td>0.58</td>
<td>0.79</td>
<td>1.35</td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.57</td>
<td>0.79</td>
<td>1.48</td>
<td>0.99</td>
</tr>
<tr>
<td>L1: residual</td>
<td>0.67</td>
<td>0.57</td>
<td>0.79</td>
<td>1.65</td>
</tr>
<tr>
<td>L1: restrict</td>
<td>0.28</td>
<td>0.28</td>
<td>0.59</td>
<td>0.53</td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.12</td>
<td>0.16</td>
<td>0.26</td>
<td>0.35</td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.16</td>
<td>0.26</td>
<td>0.44</td>
<td>0.27</td>
</tr>
<tr>
<td>L2: residual</td>
<td>0.19</td>
<td>0.16</td>
<td>0.25</td>
<td>0.44</td>
</tr>
<tr>
<td>L2: restrict</td>
<td>0.08</td>
<td>0.12</td>
<td>0.18</td>
<td>0.16</td>
</tr>
<tr>
<td>L3–L6</td>
<td>0.70</td>
<td>0.63</td>
<td>1.85</td>
<td>1.33</td>
</tr>
<tr>
<td>L2: interpolate</td>
<td></td>
<td>0.21</td>
<td>0.17</td>
<td>0.29</td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.15</td>
<td>0.16</td>
<td>0.27</td>
<td>0.34</td>
</tr>
<tr>
<td>L2: smooth</td>
<td>0.16</td>
<td>0.27</td>
<td>0.44</td>
<td>0.27</td>
</tr>
<tr>
<td>L1: interpolate</td>
<td>0.34</td>
<td>0.16</td>
<td>0.27</td>
<td>0.86</td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.53</td>
<td>0.57</td>
<td>0.89</td>
<td>1.35</td>
</tr>
<tr>
<td>L1: smooth</td>
<td>0.53</td>
<td>0.57</td>
<td>0.88</td>
<td>1.35</td>
</tr>
<tr>
<td>(\sum V)-cycle</td>
<td>3.90</td>
<td>5.75</td>
<td>8.31</td>
<td>9.02</td>
</tr>
</tbody>
</table>

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\textsuperscript{cc} framework sources released under \textit{Simplified BSD License}.

https://sourceforge.net/projects/hipacc