Performance Portable Stencil Code Generation in LIFT

Bastian Hagedorn | Larisa Stoltzfus |
Michel Steuwer | Sergei Gorlatch | Christophe Dubach
Bastian Hagedorn:

- PhD Student: University of Münster
- Group: Parallel and Distributed Systems
- Work: Stencil Computations - LIFT Project
Stencil Computations are all around...
Stencil Computations are all around...

...and are executed on a wide range of hardware...
Stencil Computations are all around...

...and are executed on a wide range of hardware...

...which requires experts to achieve high-performance!

Step 1
Write Stencil Code

Step 2
???

Step 3
Performance
The LIFT Project

High-Level Programming

High-level Expression

Transformation using Rewrite Rules

Low-level Expression

Code Generation

OpenCL Program

High-Level LIFT provides high-level interface of composable functional *primitives*

Optimization Automatic rewriting to optimize high-level expressions

Performance High-performance OpenCL Code Generation

So far mainly used for linear algebra applications
Existing Primitives in LIFT

- **map**
- **reduce**
- **zip**
- **split n**
- **join**
- **at i**
- **transpose**
Existing Primitives in LIFT

- **map**: \(\square \rightarrow \square\)
- **reduce**: \(\oplus\)
- **zip**: \(\square \rightarrow \langle \square, \square, \ldots, \square \rangle\)
- **split n**: \(\square \rightarrow \square\)
- **join**: \(\square \rightarrow \square\)
- **at i**: \(\square \rightarrow \times\)
- **transpose**: \(\square \rightarrow \square\)
Existing Primitives in LIFT

- **map**
- **reduce**
- **zip**
- **split n**
- **join**
- **at i**
- **transpose**

Set of primitives is too restricted for stencil computations.
Decomposing Stencil Computations

```c
for (int i = 0; i < N; i++) {
    int sum = 0;
    for (int j = -1; j <= 1; j++) { // (a)
        int pos = i + j;
        pos = pos < 0 ? 0 : pos; // (b)
        pos = pos > N - 1 ? N - 1 : pos;
        sum += A[pos]; } // (c)
    B[i] = sum;
}
```

(a) Neighborhood: accessing neighboring elements according to stencil shape

(b) Boundary Handling: what happens at the border of the input array?

(c) Stencil Function: compute single output element for a given neighborhood
Decomposing Stencil Computations

(a) **Neighborhood:** accessing neighboring elements according to stencil shape
(b) **Boundary Handling:** what happens at the border of the input array?
(c) **Stencil Function:** compute single output element for a given neighborhood
(b) Boundary Handling using Pad

**reindexing.example**

clamp(i, n) = (i < 0) ? 0 : ((i >= n) ? n-1 : i)

\[
\text{pad} (1, 1, \text{clamp}, [a, b, c, d]) = [a, a, b, c, d, d]
\]

**value.example**

constant(i, n) = C

\[
\text{pad} (1, 1, \text{constant}, [a, b, c, d]) = [C, a, b, c, d, C]
\]
(a) Create Neighborhoods using Slide

\[ \text{slide}(3, 1, [a, b, c, d, e]) = [[[a, b, c], [b, c, d], [c, d, e]]} \]

\[ \text{slide} : (\text{size} : \text{Int}, \text{step} : \text{Int}, \text{in} : [T]_n) \rightarrow [[T]_{\text{size}}^{n - \text{size} + \text{step}}/\text{step}] \]
(c) Apply Stencil Function using Map

\[
\text{map}(\text{nbh} \Rightarrow \\
\text{reduce}(\text{add}, 0.0f, \text{nbh}), \\
[[0, 1, 2], [1, 2, 3]]) = \\
[[3], [6]]
\]
Expressing Stencil Computations in LIFT

```
val sumNbh = fun(nbh =>
    reduce(add, 0.0f, nbh))

val stencil = fun (A: Array (Float, N) =>
    map(sumNbh, // 3.
        slide(3, 1, // 2.
            pad(1, 1, clamp, A)))) // 1.
```
Multidimensional Stencil Computations

Idea: Express complex computations as compositions of simple primitives

$$map_n(f, slide_n(size, step, pad_n(l, r, h, input)))$$

$$map_2(f, slide_2(size, step, pad_2(l, r, h, in)))$$
Multidimensional Boundary Handling

\[
\begin{align*}
\text{pad}_1(l, r, h, \text{input}) &= \text{pad}(l, r, h, \text{input}) \\
\text{pad}_n(l, r, h, \text{input}) &= \text{map}_{n-1}(\text{pad}(l, r, h), \\
&\quad \text{pad}_{n-1}(l, r, h, \text{input})) \\
\text{where } \text{map}_n \text{ are } n \text{ nested maps:} \\
\text{map}_1(f, \text{input}) &= \text{map}(f, \text{input}) \\
\text{map}_n(f, \text{input}) &= \text{map}_{n-1}(\text{map}(f), \text{input})
\end{align*}
\]

\[
\text{pad}_2 = \text{map}(\text{pad}(1,1h), \\
\text{pad}(1,1,h,in))
\]
Multidimensional Neighborhood Creation

\[
\text{slide}_2(2, 1, \begin{bmatrix} [a, b, c], \\ [d, e, f], \\ [g, h, i] \end{bmatrix}) = \\
\text{map}(\text{transpose}, \\
\text{slide}(2, 1, \\
\text{map}(\text{slide}(2, 1), \begin{bmatrix} [a, b, c], [d, e, f], [g, h, i] \end{bmatrix}))) = \\
\begin{bmatrix} [a, b, c], \\ [d, e, f], \\ [g, h, i] \end{bmatrix}
\]
Multidimensional Neighborhood Creation

\[
\text{slide}_2(2, 1, \begin{bmatrix}
[a, b, c], \\
[d, e, f], \\
g, h, i
\end{bmatrix}) = \\
\text{map}(\text{transpose, slide}(2, 1, \\
\text{map}(\text{slide}(2, 1), \begin{bmatrix}
[a, b, c], [d, e, f], [g, h, i]
\end{bmatrix}))) = \\
\text{map}(\text{transpose, slide}(2, 1, \\
\begin{bmatrix}
\begin{bmatrix} a, b \end{bmatrix}, [b, c], \\
\begin{bmatrix} d, e \end{bmatrix}, [e, f], \\
\begin{bmatrix} g, h \end{bmatrix}, [h, i]
\end{bmatrix})) = 
\]
Multidimensional Neighborhood Creation

\[
\text{slide}_2(2, 1, \begin{bmatrix} [a, b, c], \\
[d, e, f], \\
[g, h, i] \end{bmatrix}) = \\
\text{map}(\text{transpose}, \\
\text{slide}(2, 1, \\
\text{map}(\text{slide}(2, 1, [ [a, b, c], [d, e, f], [g, h, i] ])))) = \\
\text{map}(\text{transpose}, \text{slide}(2, 1, \\
[ [ [a, b], [b, c]], [ [d, e], [e, f]], [ [g, h], [h, i]] ]))) = \\
\text{map}(\text{transpose}, \\
\begin{bmatrix} \\
[ [ [a, b], [b, c]], [ [d, e], [e, f]]], \\
[ [ [d, e], [e, f]], [ [g, h], [h, i]]] \end{bmatrix}) = \\
\begin{bmatrix} \\
[ [a, b], [b, c]], [ [d, e], [e, f]] \\
[ [d, e], [e, f]], [ [g, h], [h, i]] \end{bmatrix}
\]
Optimizing by Rewriting

Optimizations are encoded as Rewrite Rules:

**map-fusion**

\[
\text{map}(f, \text{map}(g, \text{in})) \\
\rightarrow \\
\text{map}((f \circ g), \text{in})
\]

**divide-and-conquer**

\[
\text{map}(f, \text{in}) \\
\rightarrow \\
\text{join}(\text{map}(\text{map}(f), \text{split}(n, \text{in})))
\]
Optimizing Stencil Computations

Exploiting Locality through Overlapped Tiling:

- **Locality**: Close neighborhoods share elements that can be grouped in tiles.
- **Local Memory**: On GPUs, local memory can be used to cache tiles.
- **Overlap**: The shape of the stencil determines the overlap at the edges of tiles.
Overlapped Tiling in LIFT

We reuse the *slide* primitive to represent overlapped tiles:

\[
\begin{align*}
\text{slide} & \ (5,3) \\
\text{map} & \ (\text{slide} \ (3,1)) \\
\uparrow \quad u & = 5, \quad v = 3, \quad \text{size} = 3, \quad \text{step} = 1
\end{align*}
\]

Tiling as a rewrite rule:

\[
\text{overlapped-tiling} \\
\map(f, \ \text{slide}(\text{size}, \text{step}, \text{input})) \\
\map(\text{slide} \ (u, v, \text{input}))
\]

map(f,  slide(size, step, input))

\[\map(\text{slide}(u, v, \text{input}))\]
Code Generation and Exploration

High-level Programming

Algorithmic Primitives:
- map
- split
- reduce
- ...

Transformation using Rewrite Rules

Low-level Expression

OpenCL Primitives:
- mapWorkgroup
- toLocal
- ...

Code Generation

OpenCL Program

Hardware Paradigms:
- local memory
- barriers
- ...

SHOC

Speedup over the baseline

SRAD1

Speedup over the baseline

Platform

AMD

NVIDIA
## Benchmarks implemented in LIFT

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dim</th>
<th>Stencil</th>
<th>Input size</th>
<th># Input grids</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHOC</td>
<td>2</td>
<td>9-point</td>
<td>$8194^2$</td>
<td>1</td>
</tr>
<tr>
<td>SRAD 1</td>
<td>2</td>
<td>5-point</td>
<td>$504 \times 458$</td>
<td>1</td>
</tr>
<tr>
<td>SRAD 2</td>
<td>2</td>
<td>3-point</td>
<td>$504 \times 458$</td>
<td>2</td>
</tr>
<tr>
<td>Hotspot2D</td>
<td>2</td>
<td>5-point</td>
<td>$8192^2$</td>
<td>2</td>
</tr>
<tr>
<td>Hotspot3D</td>
<td>3</td>
<td>7-point</td>
<td>$512^2 \times 8$</td>
<td>2</td>
</tr>
<tr>
<td>Acoustic</td>
<td>3</td>
<td>7-point</td>
<td>$512^2 \times 404$</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table:** Benchmarks used in the evaluation.
Evaluation

Platform

Speedup over the baseline

SHOC

SRAD 1

SRAD 2

Hotspot 2D

Hotspot 3D

Acoustic

AMD NVIDIA

AMD NVIDIA

AMD NVIDIA

AMD NVIDIA

AMD NVIDIA

AMD NVIDIA

Platform
Questions?
b.hagedorn@wwu.de
http://www.lift-project.org/
Non-Rectengular Stencils

val T = at(1. at(0, nbh))
val B = at(1. at(2, nbh))
val C = at(1. at(1, nbh))
val L = at(0. at(1, nbh))
val R = at(2. at(1, nbh))
return T+B+C+L+R

map(stencilFunction, nbh)
// toGlobal(mapGlobal(id)) o
// mapGlobal(reduceSeq (+) 0) o slide 3 1 o pad 1 1 clamp input

float add(float x, float y){
    return x+y;
}

float id(float x){
    return x;
}

kernel void KERNEL(const global float* restrict v__9,
                   global float* v__15,
                   int v_N_0){
    /* Static local memory */
    /* Typed Value memory */
    float v__11;
    /* Private Memory */
    for (int v_gl_id_6 = get_global_id(0); v_gl_id_6<v_N_0;
    v_gl_id_6 = (v_gl_id_6 + get_global_size(0))){
        float v_tmp_20 = 0.0f;
        v__11 = v_tmp_20;
        /* reduce_seq */
        for (int v_i_7 = 0; v_i_7<3; v_i_7 = (1 + v_i_7)){
            v__11 = add(v__11,
                         v__9[((-1 + v_gl_id_6 + v_i_7) >= 0) ?
                            ((-1 + v_gl_id_6 + v_i_7) < v_N_0) ?
                             (-1 + v_gl_id_6 + v_i_7) : (-1 + v_N_0) : 0 )]);
        }
        /* end reduce_seq */
    }
    for (int v_gl_id_8 = get_global_id(0); v_gl_id_8<v_N_0;
    v_gl_id_8 = (v_gl_id_8 + get_global_size(0))){
        v__11 = id(v__11);
    }
}