Westfälische
Wilhelms-Universität
Münster

> Performance Portable Stencil Code Generation in LIFT

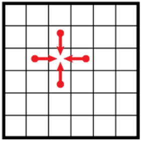
Bastian Hagedorn | Larisa Stoltzfus |
Michel Steuwer | Sergei Gorlatch | Christophe Dubach

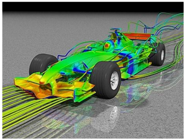
Introduction

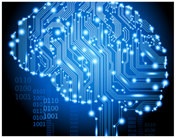
Bastian Hagedorn:

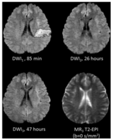
- PhD Student: University of Münster
- Group: Parallel and Distributed Systems
- Work: Stencil Computations - LIFT Project

Stencil Computations are all around...

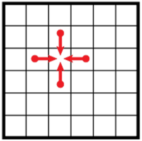
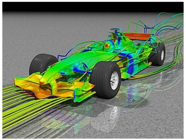
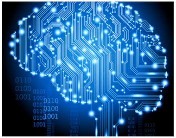
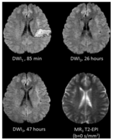




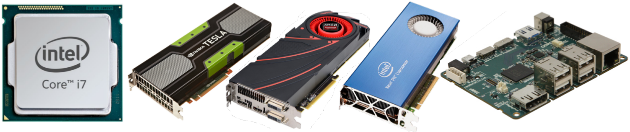




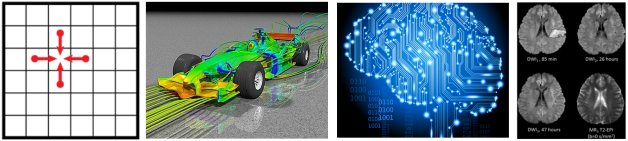
Stencil Computations are all around...

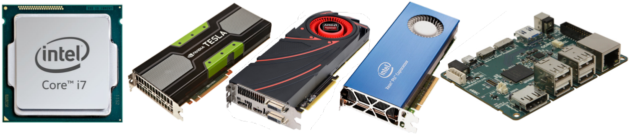
...and are executed on a wide range of hardware...



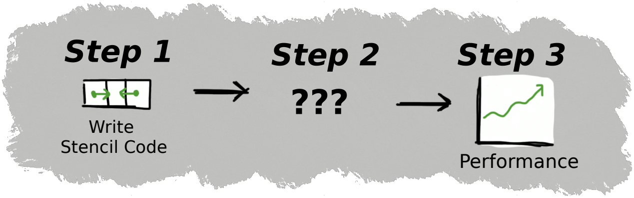
Stencil Computations are all around...



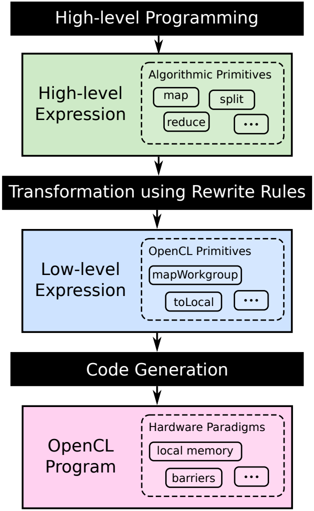
...and are executed on a wide range of hardware...



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



The LIFT Project



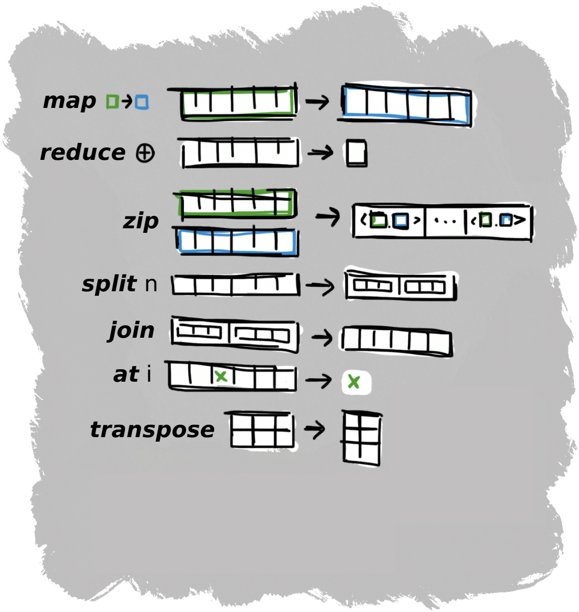
High-Level LIFT provides high-level interface of composable functional *primitives*

Optimization Automatic rewriting to optimize high-level expressions

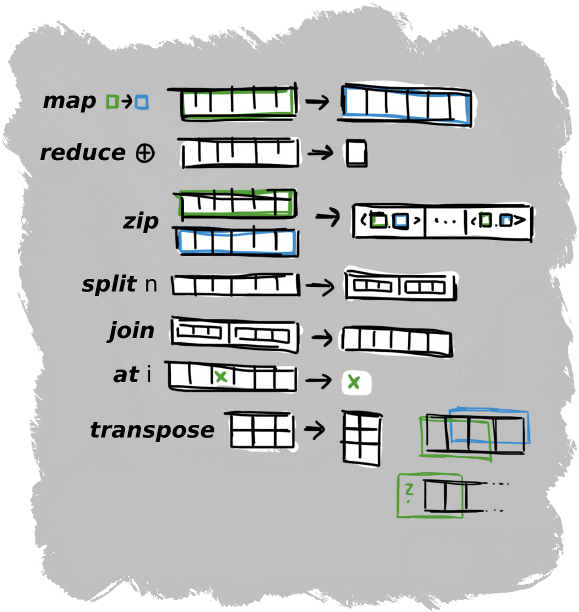
Performance High-performance OpenCL Code Generation

So far mainly used for linear algebra applications

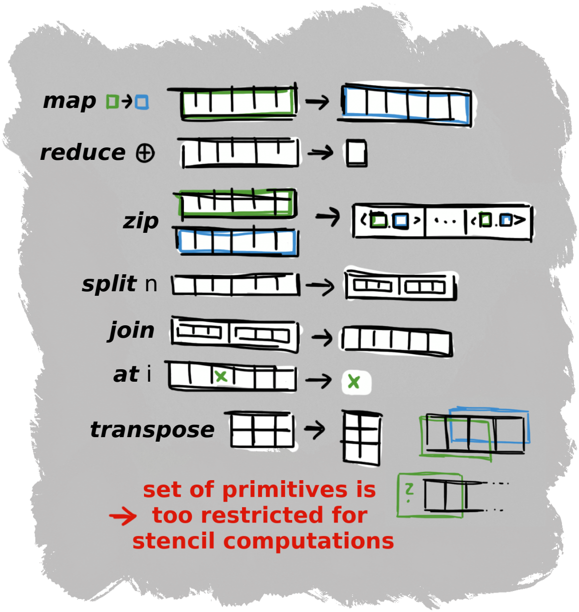
Existing Primitives in LIFT



Existing Primitives in LIFT



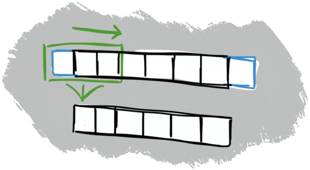
Existing Primitives in LIFT



Decomposing Stencil Computations

stencil.c

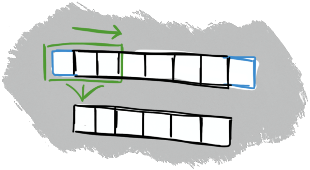
```
for (int i = 0; i < N; i++) {
  int sum = 0;
  for (int j = -1; j <= 1; j++) {       // (a)
    int pos = i + j;
    pos = pos < 0 ? 0 : pos;            // (b)
    pos = pos > N - 1 ? N - 1 : pos;
    sum += A[pos]; }                    // (c)
  B[i] = sum;
}
```



Decomposing Stencil Computations

stencil.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;
}
```

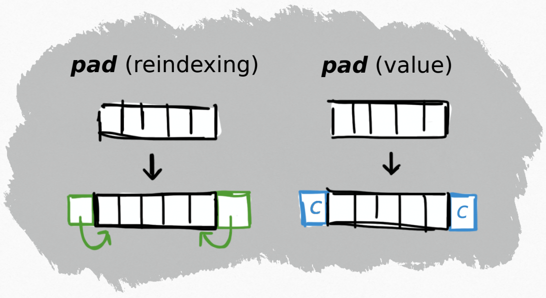


(a) Neighborhood: accessing neighboring elements according to stencil shape

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

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

(b) Boundary Handling using Pad



reindexing.example

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

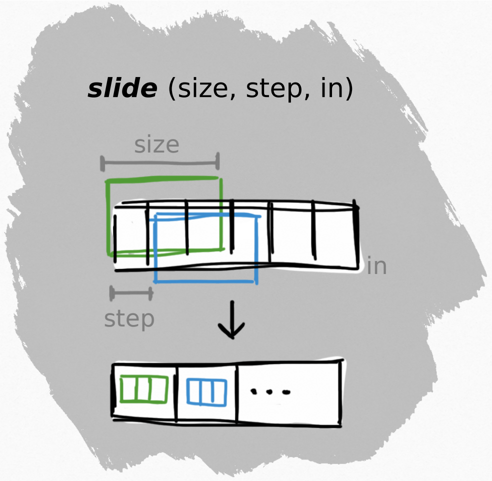
$$\textbf{\textit{pad}}\ (1, 1, clamp, [a, b, c, d]) = [a, a, b, c, d, d]$$

value.example

```
constant(i, n) = C
```

$$\textbf{\textit{pad}}\ (1, 1, constant, [a, b, c, d]) = [C, a, b, c, d, C]$$

(a) Create Neighborhoods using Slide

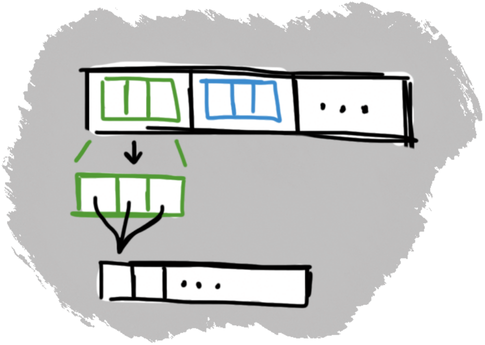


slide.example

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

$$\textit{slide} : (\textit{size} : \texttt{Int},\ \textit{step} : \texttt{Int},\ \textit{in} : [T]_n) \rightarrow [[T]_{\textit{size}}]_{\frac{n - \textit{size} + \textit{step}}{\textit{step}}}$$

(c) Apply Stencil Function using Map



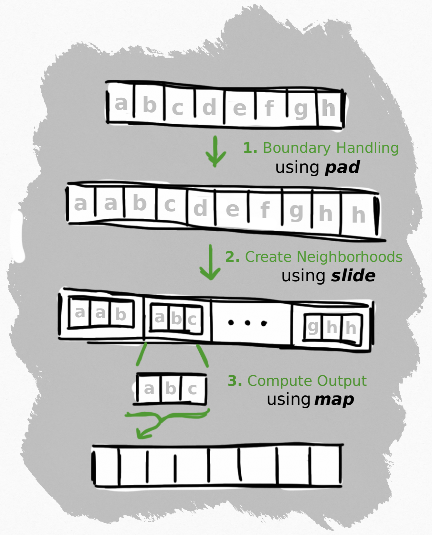
stencil-fun.example

$$\textbf{\textit{map}}(nbh \Rightarrow \textbf{\textit{reduce}}(\texttt{add, 0.0f}, nbh), [[0,1,2],[1,2,3]]) = [[3],[6]]$$

Expressing Stencil Computations in LIFT

stencil.lift

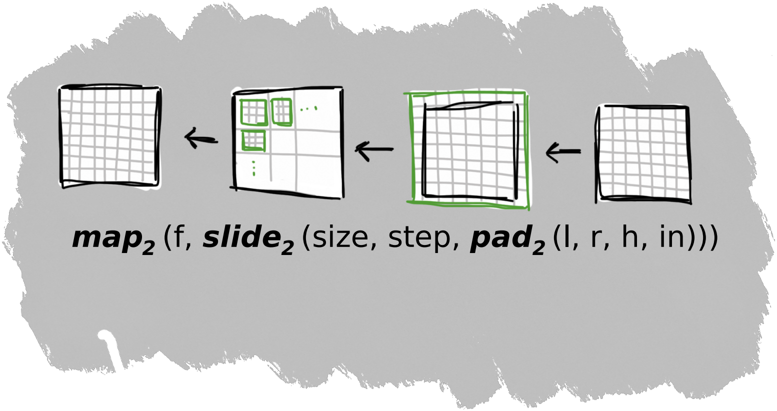
```
val sumNbh = fun(nbh =>
  reduce(add, 0.0f, nbh))

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

$$\textbf{\textit{map}}_{\textbf{\textit{n}}}(f,\ \textbf{\textit{slide}}_{\textbf{\textit{n}}}(size,\ step,\ \textbf{\textit{pad}}_{\textbf{\textit{n}}}(l,\ r,\ h,\ input)))$$



Multidimensional Boundary Handling

$$\boldsymbol{pad}_1(l,\ r,\ h,\ input) = \boldsymbol{pad}(l,\ r,\ h,\ input)$$

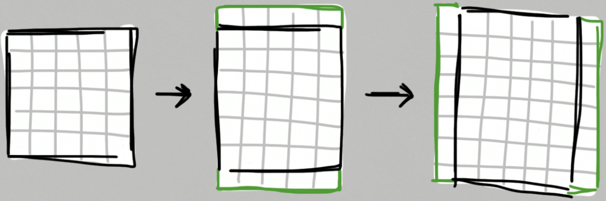
$$\boldsymbol{pad}_n(l,\ r,\ h,\ input) = \boldsymbol{map}_{n-1}(\boldsymbol{pad}(l,\ r,\ h),\ \boldsymbol{pad}_{n-1}(l,\ r,\ h,\ input))$$

where $\boldsymbol{map}_n$ are n nested *maps*:

$$\boldsymbol{map}_1(f,\ input) = \boldsymbol{map}(f,\ input)$$

$$\boldsymbol{map}_n(f,\ input) = \boldsymbol{map}_{n-1}(\boldsymbol{map}(f),\ input)$$

$\boldsymbol{pad_2} =$ $\boldsymbol{map}(\boldsymbol{pad}_{(1,1h)},\ \boldsymbol{pad}_{(1,1,h,in)})$



Multidimensional Neighborhood Creation

$$\textbf{\textit{slide}}_2(2, 1, \begin{bmatrix} [a,\ b,\ c], \\ [d,\ e,\ f], \\ [g,\ h,\ i] \end{bmatrix}) =$$

$$\textbf{\textit{map}}(\textbf{\textit{transpose}},$$

$$\textbf{\textit{slide}}(2, 1,$$

$$\textbf{\textit{map}}(\textbf{\textit{slide}}(2, 1),\ \big[\ [a,\ b,\ c],\ [d,\ e,\ f],\ [g,\ h,\ i]\ \big]))) =$$

Multidimensional Neighborhood Creation

$$\textit{\textbf{slide}}_2(2,1,\begin{bmatrix}[a,\ b,\ c],\\ [d,\ e,\ f],\\ [g,\ h,\ i]\end{bmatrix}) =$$

$$\textit{\textbf{map}}(\textit{\textbf{transpose}},$$
$$\textit{\textbf{slide}}(2,1,$$
$$\textit{\textbf{map}}(\textit{\textbf{slide}}(2,1),\ \big[\ [a,\ b,\ c],\ [d,\ e,\ f],\ [g,\ h,\ i]\ \big]))) =$$

$$\textit{\textbf{map}}(\textit{\textbf{transpose}},\ \textit{\textbf{slide}}(2,1,$$
$$\big[\ [\ [a,\ b],\ [b,\ c]\ ],\ [\ [d,\ e],\ [e,\ f]\ ],\ [\ [g,\ h],\ [h,\ i]\ ]\ \big])) =$$

Multidimensional Neighborhood Creation

$$\textbf{\textit{slide}}_2(2, 1, \begin{bmatrix} [a,\ b,\ c], \\ [d,\ e,\ f], \\ [g,\ h,\ i] \end{bmatrix}) =$$

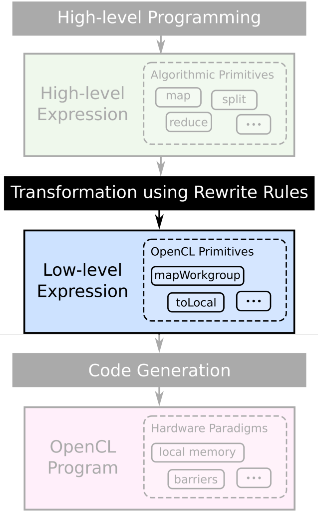
$$\textbf{\textit{map}}(\textbf{\textit{transpose}},$$
$$\textbf{\textit{slide}}(2, 1,$$
$$\textbf{\textit{map}}(\textbf{\textit{slide}}(2, 1),\ \big[\ [a,\ b,\ c],\ [d,\ e,\ f],\ [g,\ h,\ i]\ \big]))) =$$

$$\textbf{\textit{map}}(\textbf{\textit{transpose}},\ \textbf{\textit{slide}}(2, 1,$$
$$\big[\ [\ [a,\ b],\ [b,\ c]\ ],\ [\ [d,\ e],\ [e,\ f]\ ],\ [\ [g,\ h],\ [h,\ i]\ ]\ \big])) =$$

$$\textbf{\textit{map}}(\textbf{\textit{transpose}},$$
$$\begin{bmatrix} [\ [\ [a,\ b],\ [b,\ c]\ ],\ [\ [d,\ e],\ [e,\ f]\ ]\ ], \\ [\ [\ [d,\ e],\ [e,\ f]\ ],\ [\ [g,\ h],\ [h,\ i]\ ]\ ] \end{bmatrix})) =$$

$$\begin{bmatrix} \begin{bmatrix} [a,\ b], \\ [d,\ e] \end{bmatrix}, & \begin{bmatrix} [b,\ c], \\ [e,\ f] \end{bmatrix}, \\ \begin{bmatrix} [d,\ e], \\ [g,\ h] \end{bmatrix}, & \begin{bmatrix} [e,\ f], \\ [h,\ i] \end{bmatrix} \end{bmatrix}$$

Optimizing by Rewriting



Optimizations are encoded as Rewrite Rules:

map-fusion

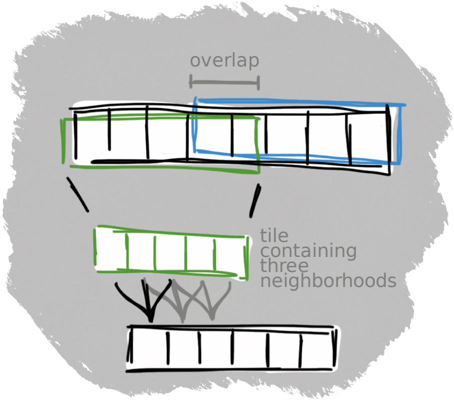
$$\textbf{\textit{map}}(f, \textbf{\textit{map}}(g, in)) \mapsto \textbf{\textit{map}}((f \circ g), in)$$

divide-and-conquer

$$\textbf{\textit{map}}(f, in) \mapsto \textbf{\textit{join}}(\textbf{\textit{map}}(\textbf{\textit{map}}(f), \textbf{\textit{split}}(n, in)))$$

Optimizing Stencil Computations

Exploiting Locality through Overlapped Tiling:



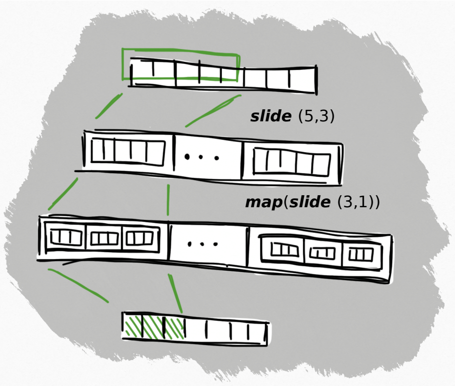
Locality Close neighborhoods share elements that can be grouped in tiles

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:



↑ $u = 5, v = 3, size = 3, step = 1$

Tiling as a rewrite rule:

overlapped-tiling

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

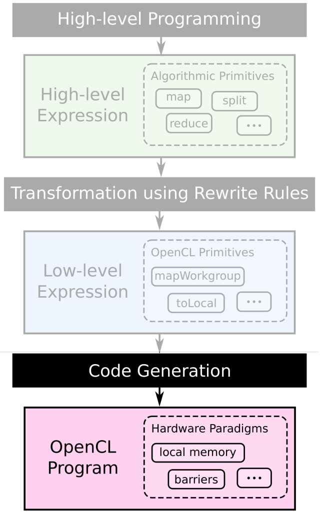
$\mapsto$

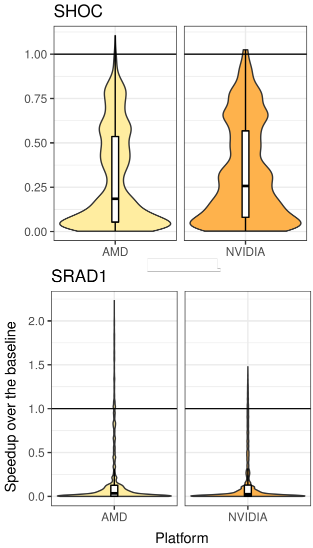
join$($***map***$(tile \Rightarrow$

map$(f,$ ***slide***$(size, step, tile)),$

slide$(u, v, input)))$

Code Generation and Exploration



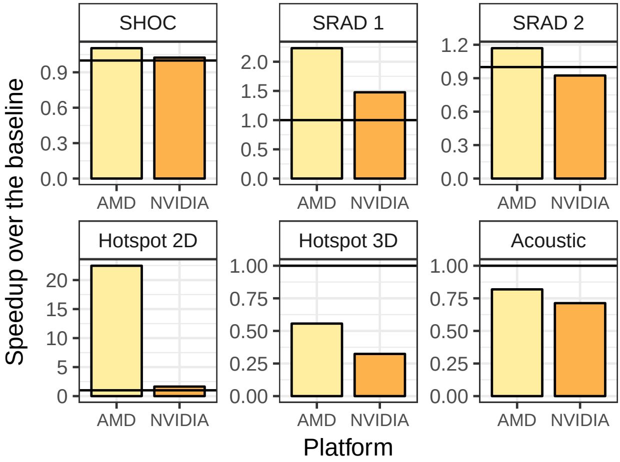


Benchmarks implemented in LIFT

Benchmark	Dim	Stencil	Input size	# Input grids
SHOC	2	9-point	8194^2	1
SRAD 1	2	5-point	504×458	1
SRAD 2	2	3-point	504×458	2
Hotspot2D	2	5-point	8192^2	2
Hotspot3D	3	7-point	$512^2 \times 8$	2
Acoustic	3	7-point	$512^2 \times 404$	2

Table: Benchmarks used in the evaluation.

Evaluation

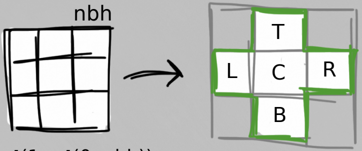

SHOC
SRAD 1
SRAD 2
Hotspot 2D
Hotspot 3D
Acoustic
Speedup over the baseline
Platform
AMD
NVIDIA
0.0
0.3
0.6
0.9
0.5
1.0
1.5
2.0
1.2
0
5
10
15
20
0.00
0.25
0.50
0.75
1.00

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)) → stencilFunction
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__15[v_gl_id_8] = id(v__11);
}}
```