

The LIFT Project

Performance Portable Parallel
Code Generation via Rewrite Rules

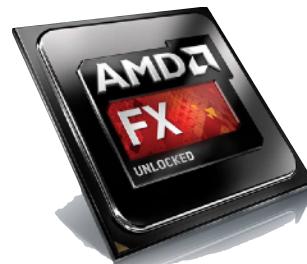
Michel Steuwer — *michel.steuwer@glasgow.ac.uk*



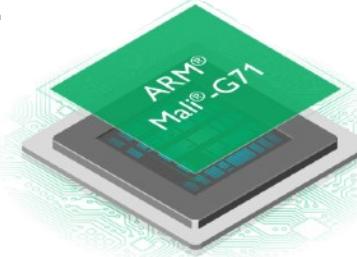
INSPIRING
PEOPLE

What are the problems LIFT tries to tackle?

- Parallel processors everywhere
- Many different types: CPUs, GPUs, ...
- Parallel programming is hard
- Optimising is even harder
- **Problem:**
No portability of performance!



CPU



GPU



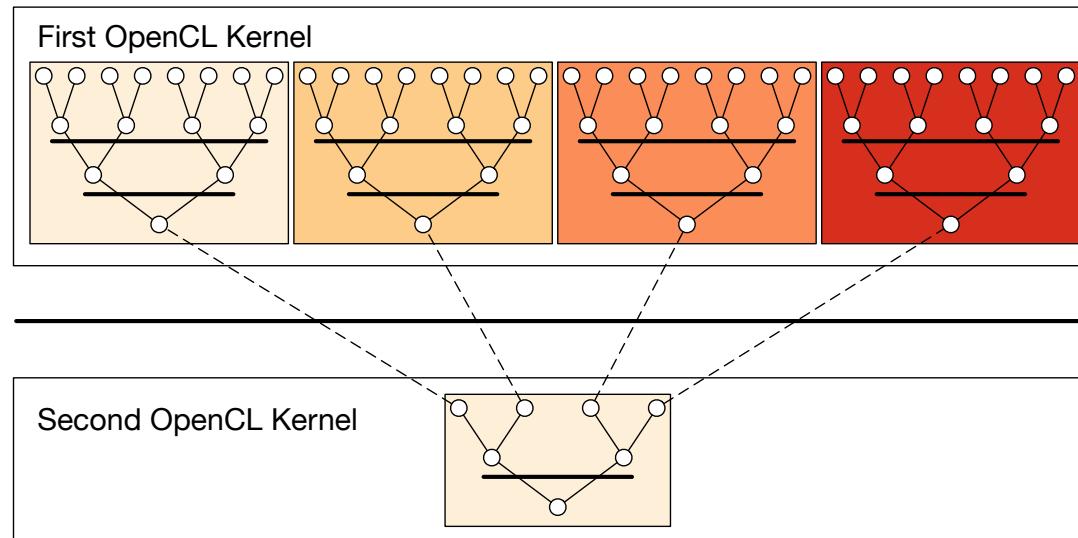
Accelerator



FPGA

Case Study: Parallel Reduction in OpenCL

- Summing up all values of an array
- Comparison of 7 implementations by Nvidia
- Investigating complexity and efficiency of optimisations



Unoptimised Implementation Parallel Reduction

```
kernel void reduce0(global float* g_idata, global float* g_odata,
                    unsigned int n, local float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i   = get_global_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    barrier(CLK_LOCAL_MEM_FENCE);
    // do reduction in local memory
    for (unsigned int s=1; s < get_local_size(0); s*= 2) {
        if ((tid % (2*s)) == 0) {
            l_data[tid] += l_data[tid + s];
            barrier(CLK_LOCAL_MEM_FENCE);
        }
    }
    // write result for this work-group to global memory
    if (tid == 0) g_odata[get_group_id(0)] = l_data[0];
}
```

Avoid Divergent Branching

```
kernel void reduce1(global float* g_idata, global float* g_odata,
                    unsigned int n, local float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i   = get_global_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    barrier(CLK_LOCAL_MEM_FENCE);

    for (unsigned int s=1; s < get_local_size(0); s*= 2) {
        // continuous work-items remain active
        int index = 2 * s * tid;
        if (index < get_local_size(0)) {
            l_data[index] += l_data[index + s];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (tid == 0) g_odata[get_group_id(0)] = l_data[0];
}
```

Avoid Interleaved Addressing

```
kernel void reduce2(global float* g_idata, global float* g_odata,
                    unsigned int n, local float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i   = get_global_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    barrier(CLK_LOCAL_MEM_FENCE);

    // process elements in different order
    // requires commutativity
    for (unsigned int s=get_local_size(0)/2; s>0; s>>=1) {
        if (tid < s) {
            l_data[tid] += l_data[tid + s];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (tid == 0) g_odata[get_group_id(0)] = l_data[0];
}
```

Increase Computational Intensity per Work-Item

```
kernel void reduce3(global float* g_idata, global float* g_odata,
                    unsigned int n, local float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i = get_group_id(0) * (get_local_size(0)*2)
                    + get_local_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    // performs first addition during loading
    if (i + get_local_size(0) < n)
        l_data[tid] += g_idata[i+get_local_size(0)];
    barrier(CLK_LOCAL_MEM_FENCE);

    for (unsigned int s=get_local_size(0)/2; s>0; s>>=1) {
        if (tid < s) {
            l_data[tid] += l_data[tid + s];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (tid == 0) g_odata[get_group_id(0)] = l_data[0];
}
```

Avoid Synchronisation inside a Warp

```
kernel void reduce4(global float* g_idata, global float* g_odata,
                    unsigned int n, local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i = get_group_id(0) * (get_local_size(0)*2)
                    + get_local_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    if (i + get_local_size(0) < n)
        l_data[tid] += g_idata[i+get_local_size(0)];
    barrier(CLK_LOCAL_MEM_FENCE);

    # pragma unroll 1
    for (unsigned int s=get_local_size(0)/2; s>32; s>>=1) {
        if (tid < s) { l_data[tid] += l_data[tid + s]; }
        barrier(CLK_LOCAL_MEM_FENCE); }

    // this is not portable OpenCL code!
    if (tid < 32) {
        if (WG_SIZE >= 64) { l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) { l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) { l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >=  8) { l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >=  4) { l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >=  2) { l_data[tid] += l_data[tid+ 1]; } }

    if (tid == 0) g_odata[get_group_id(0)] = l_data[0]; }
```

Complete Loop Unrolling

```
kernel void reduce5(global float* g_idata, global float* g_odata,
                    unsigned int n, local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i = get_group_id(0) * (get_local_size(0)*2)
                    + get_local_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    if (i + get_local_size(0) < n)
        l_data[tid] += g_idata[i+get_local_size(0)];
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) { l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }

    if (WG_SIZE >= 128) {
        if (tid < 64) { l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }

    if (tid < 32) {
        if (WG_SIZE >= 64) { l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) { l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) { l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) { l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) { l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) { l_data[tid] += l_data[tid+ 1]; } }

    if (tid == 0) g_odata[get_group_id(0)] = l_data[0]; }
```

Fully Optimised Implementation

```
kernel void reduce6(global float* g_idata, global float* g_odata,
                    unsigned int n, local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i = get_group_id(0) * (get_local_size(0)*2)
                    + get_local_id(0);
    unsigned int gridSize = WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) { l_data[tid] += g_idata[i];
                      if (i + WG_SIZE < n)
                          l_data[tid] += g_idata[i+WG_SIZE];
                      i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) { l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }

    if (WG_SIZE >= 128) {
        if (tid < 64) { l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }

    if (tid < 32) {
        if (WG_SIZE >= 64) { l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) { l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) { l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >=  8) { l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >=  4) { l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >=  2) { l_data[tid] += l_data[tid+ 1]; } }

    if (tid == 0) g_odata[get_group_id(0)] = l_data[0]; }
```

Reduction Case Study

- Optimising OpenCL is complex
 - Understanding of target hardware required
- Program changes not obvious
- Is it worth it? ...

```
kernel
void reduce0(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i = get_global_id(0);
    l_data[tid] = (i < n) ? g_idata[i] : 0;
    barrier(CLK_LOCAL_MEM_FENCE);

    for (unsigned int s=1;
         s < get_local_size(0); s*= 2) {
        if ((tid % (2*s)) == 0) {
            l_data[tid] += l_data[tid + s];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

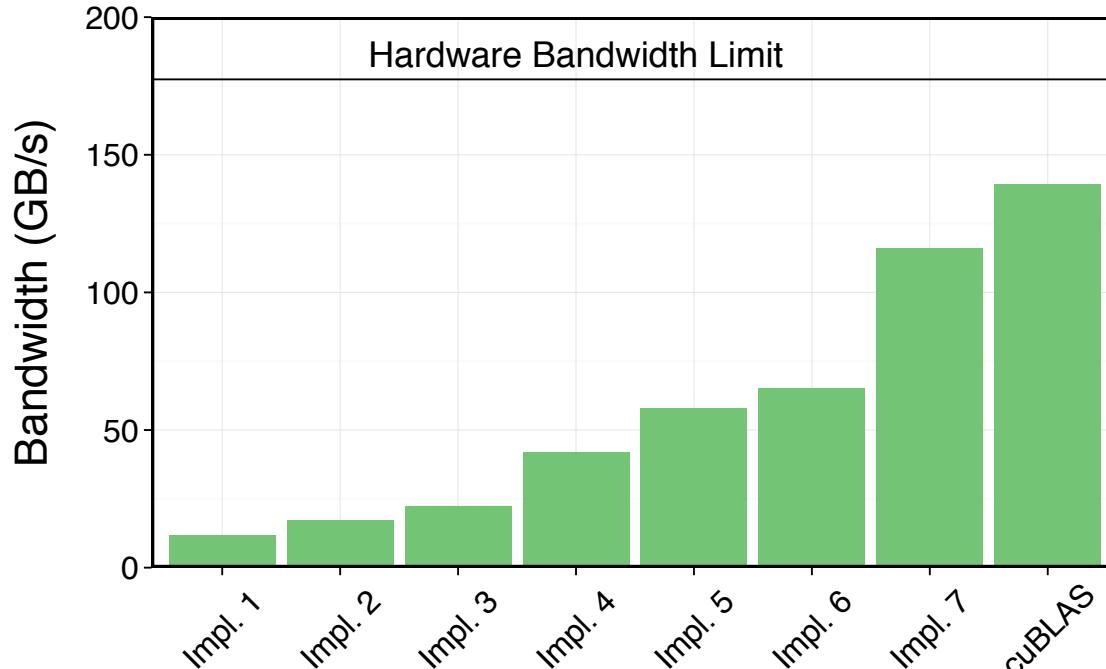
Unoptimized Implementation

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize;
    }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32];
        }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16];
        }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8];
        }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4];
        }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2];
        }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1];
        }
    }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

Fully Optimized Implementation

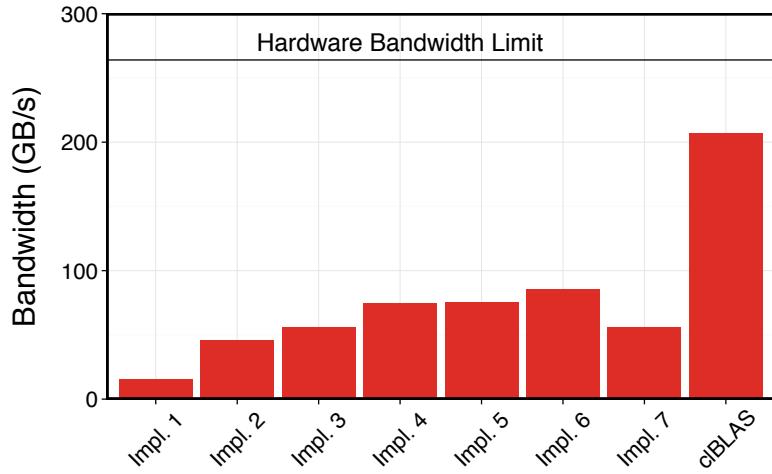
Performance Results Nvidia



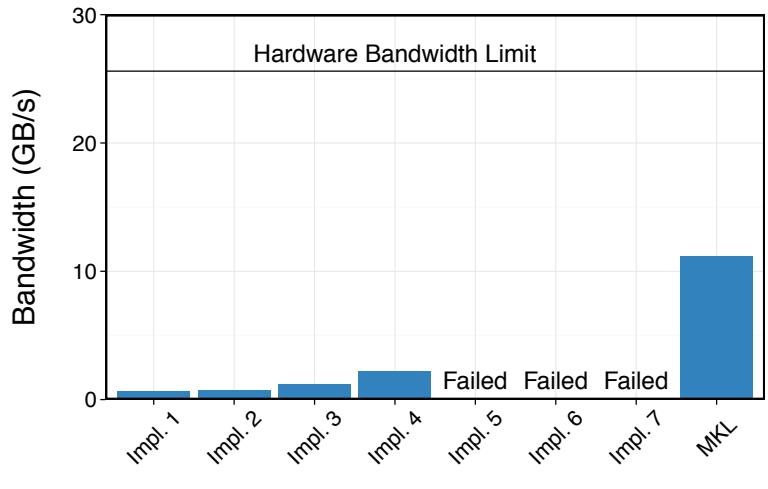
(a) Nvidia's GTX 480 GPU.

- ... Yes! Optimising improves performance by a factor of 10!
- Optimising is important, but ...

Performance Results AMD and Intel



(b) AMD's HD 7970 GPU.



(c) Intel's E5530 dual-socket CPU.

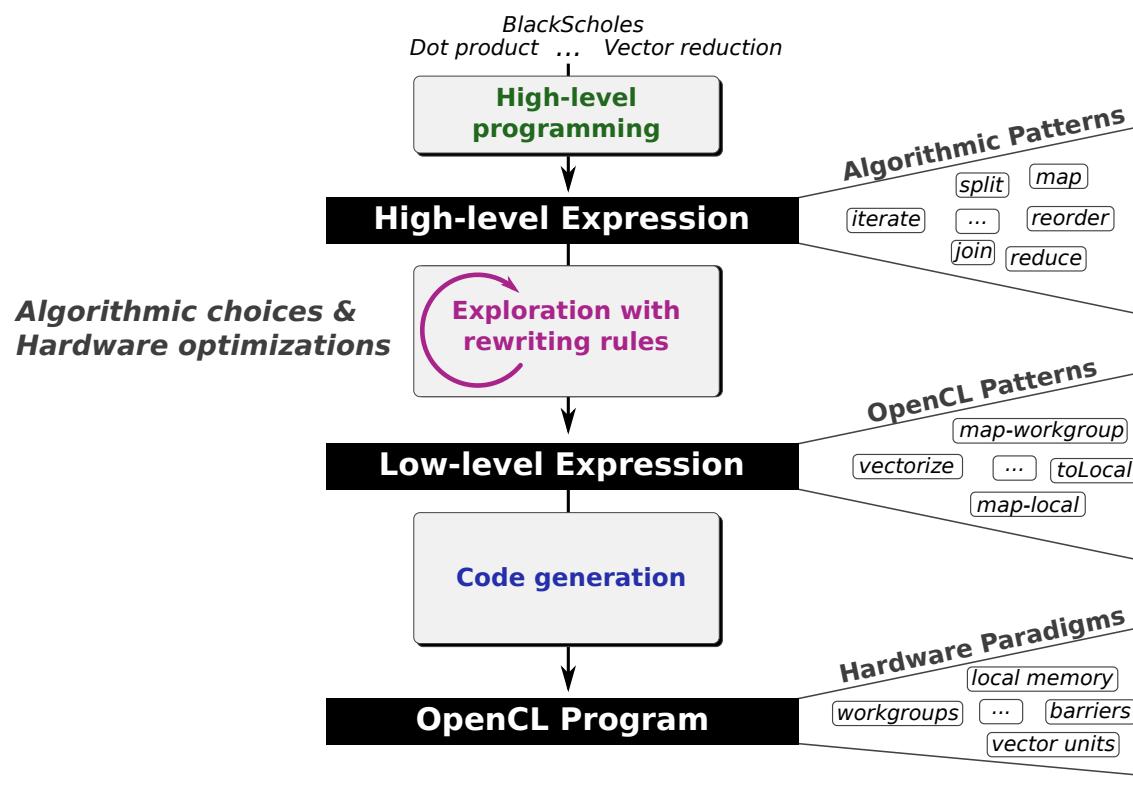
- ... unfortunately, optimisations in OpenCL are not portable!
- **Challenge:** how to achieving portable performance?

LIFT: Performance Portable GPU Code Generation via Rewrite Rules

[ICFP 2015]

[GPGPU 2016]
[CASES 2016]

[CGO 2017]



Ambition: automatic generation of *Performance Portable* code

Walkthrough

① $\text{sum}(\text{vec}) = \text{reduce}(+, 0, \text{vec})$

|
rewrite rules

code generation

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128 ∘
) ∘ split blockSize
```

③

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1]; } }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

Walkthrough

① $\text{sum}(\text{vec}) = \text{reduce}(+, 0, \text{vec})$

rewrite rules

code generation

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128 ∘
) ∘ split blockSize
```

③

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

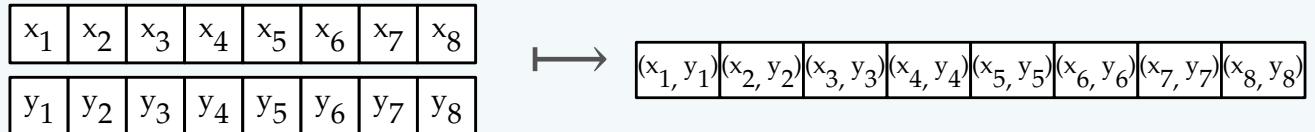
    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1]; } }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

① Algorithmic Primitives (a.k.a. algorithmic skeletons)

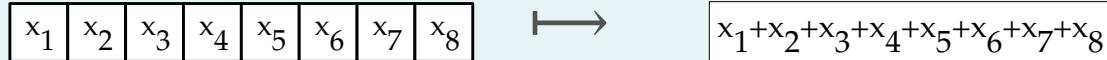
$\text{map}(f, x)$:



$\text{zip}(x, y)$:



$\text{reduce}(+, 0, x)$:



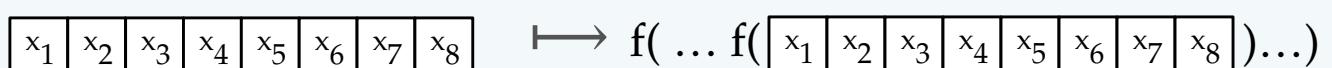
$\text{split}(n, x)$:



$\text{join}(x)$:



$\text{iterate}(f, n, x)$:



$\text{reorder}(\sigma, x)$:



① High-Level Programs

```
scal(a, vec) = map(λ x ↦ x*a, vec)
```

```
asum(vec) = reduce(+, 0, map(abs, vec))
```

```
dotProduct(x, y) = reduce(+, 0, map(*, zip(x, y)))
```

```
gemv(mat, x, y, α, β) =
  map(+, zip(
    map(λ row ↦ scal(α, dotProduct(row, x)), mat),
    scal(β, y) ))
```

Walkthrough

① $\text{sum}(\text{vec}) = \text{reduce}(+, 0, \text{vec})$

|
rewrite rules

code generation

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128 ∘
) ∘ split blockSize
```

③

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1]; } }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

Walkthrough

① $\text{sum}(\text{vec}) = \text{reduce}(+, 0, \text{vec})$

I
rewrite rules

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128 ∘
) ∘ split blockSize
```

③

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize;
    }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64];
        }
        barrier(CLK_LOCAL_MEM_FENCE);
    }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32];
        }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16];
        }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8];
        }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4];
        }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2];
        }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1];
        }
    }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

② Algorithmic Rewrite Rules

- **Provably correct** rewrite rules
- Express algorithmic implementation choices

Split-join rule:

$$\text{map } f \rightarrow \text{join} \circ \text{map} (\text{map } f) \circ \text{split } n$$

Map fusion rule:

$$\text{map } f \circ \text{map } g \rightarrow \text{map} (f \circ g)$$

Reduce rules:

$$\text{reduce } f z \rightarrow \text{reduce } f z \circ \text{reducePart } f z$$

$$\text{reducePart } f z \rightarrow \text{reducePart } f z \circ \text{reorder}$$

$$\text{reducePart } f z \rightarrow \text{join} \circ \text{map} (\text{reducePart } f z) \circ \text{split } n$$

$$\text{reducePart } f z \rightarrow \text{iterate } n (\text{reducePart } f z)$$

② OpenCL Primitives

Primitive

mapGlobal

mapWorkgroup

mapLocal

mapSeq

reduceSeq

toLocal , toGlobal

mapVec,
splitVec, joinVec

OpenCL concept

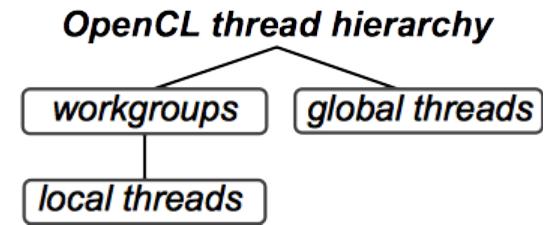
Work-items

Work-groups

Sequential implementations

Memory areas

Vectorisation



② OpenCL Rewrite Rules

- Express low-level implementation and optimisation choices

Map rules:

$$\text{map } f \rightarrow \text{mapWorkgroup } f \mid \text{mapLocal } f \mid \text{mapGlobal } f \mid \text{mapSeq } f$$

Local/ global memory rules:

$$\text{mapLocal } f \rightarrow \text{toLocal} (\text{mapLocal } f) \quad \text{mapLocal } f \rightarrow \text{toGlobal} (\text{mapLocal } f)$$

Vectorisation rule:

$$\text{map } f \rightarrow \text{joinVec} \circ \text{map} (\text{mapVec } f) \circ \text{splitVec } n$$

Fusion rule:

$$\text{reduceSeq } f \ z \circ \text{mapSeq } g \rightarrow \text{reduceSeq} (\lambda (acc, x). \ f (acc, g \ x)) \ z$$

Walkthrough

① $\text{vecSum} = \text{reduce } (+) 0$

|
rewrite rules

code generation

③

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1]; } }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128 ∘
) ∘ split blockSize
```

Walkthrough

① $\text{vecSum} = \text{reduce } (+) \ 0$

rewrite rules

code generation

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128
) ∘ split blockSize
```

③

```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1]; } }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

③ Pattern based OpenCL Code Generation

- Generate OpenCL code for each OpenCL primitive

mapGlobal f xs →

```
for (int g_id = get_global_id(0); g_id < n;  
     g_id += get_global_size(0)) {  
    output[g_id] = f(xs[g_id]);  
}
```

reduceSeq f z xs →

```
T acc = z;  
for (int i = 0; i < n; ++i) {  
    acc = f(acc, xs[i]);  
}
```

⋮

⋮

- A lot more details about the code generation implementation can be found in our [CGO 2017 paper](#)

Walkthrough

① $\text{vecSum} = \text{reduce } (+) 0$

|
rewrite rules

code generation

③

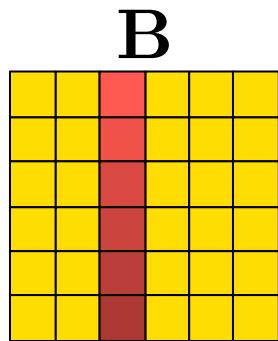
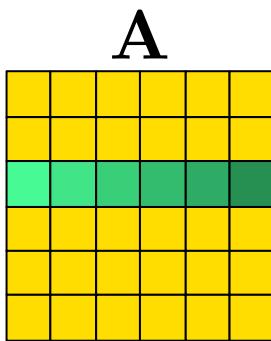
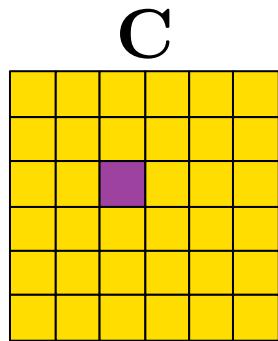
```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);
    unsigned int gridSize =
        WG_SIZE * get_num_groups(0);
    l_data[tid] = 0;
    while (i < n) {
        l_data[tid] += g_idata[i];
        if (i + WG_SIZE < n)
            l_data[tid] += g_idata[i+WG_SIZE];
        i += gridSize; }
    barrier(CLK_LOCAL_MEM_FENCE);

    if (WG_SIZE >= 256) {
        if (tid < 128) {
            l_data[tid] += l_data[tid+128]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (WG_SIZE >= 128) {
        if (tid < 64) {
            l_data[tid] += l_data[tid+ 64]; }
        barrier(CLK_LOCAL_MEM_FENCE); }
    if (tid < 32) {
        if (WG_SIZE >= 64) {
            l_data[tid] += l_data[tid+32]; }
        if (WG_SIZE >= 32) {
            l_data[tid] += l_data[tid+16]; }
        if (WG_SIZE >= 16) {
            l_data[tid] += l_data[tid+ 8]; }
        if (WG_SIZE >= 8) {
            l_data[tid] += l_data[tid+ 4]; }
        if (WG_SIZE >= 4) {
            l_data[tid] += l_data[tid+ 2]; }
        if (WG_SIZE >= 2) {
            l_data[tid] += l_data[tid+ 1]; } }
    if (tid == 0)
        g_odata[get_group_id(0)] = l_data[0];
}
```

②

```
vecSum = reduce ∘ join ∘ map-workgroup (
    join ∘ toGlobal (map-local (map-seq id)) ∘ split 1 ∘
    join ∘ map-warp (
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 1 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 2 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 4 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 8 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 16 ∘
        join ∘ map-lane (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 32 ∘
    ) ∘ split 64 ∘
    join ∘ map-local (reduce-seq (+) 0) ∘ split 2 ∘ reorder-stride 64 ∘
    join ∘ toLocal (map-local (reduce-seq (+) 0)) ∘
    split (blockSize/128) ∘ reorder-stride 128 ∘
) ∘ split blockSize
```

Case Study: Matrix Multiplication



$A \times B =$

```
map(λ rowA ↦  
    map(λ colB ↦  
        dotProduct(rowA, colB)  
        , transpose(B))  
    , A)
```

Tiling as a Rewrite Rules

Naïve matrix multiplication

```

1 map(λ arow .
2   map(λ bcol .
3     reduce(+, 0) ∘ map(×) ∘ zip(arow, bcol)
4     , transpose(B))
5   , A)

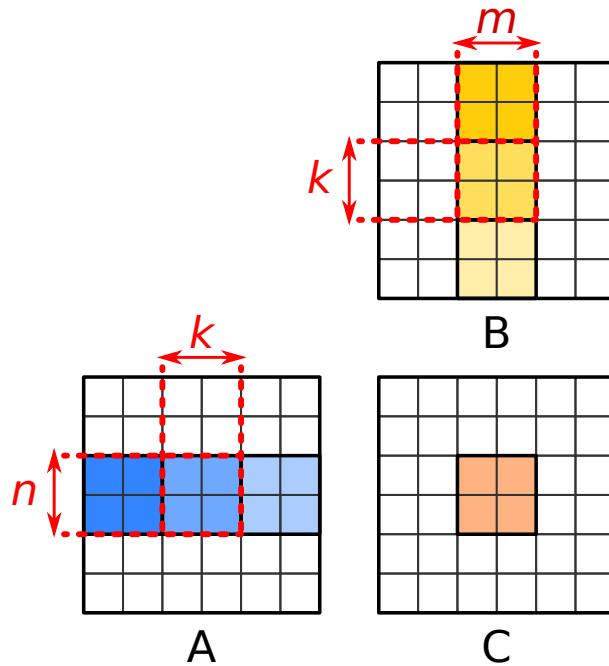
```

↓
Apply tiling rules

```

1 untile ∘ map(λ rowOfTilesA .
2   map(λ colOfTilesB .
3     toGlobal(copy2D) ∘
4     reduce(λ (tileAcc, (tileA, tileB)) .
5       map(map(+)) ∘ zip(tileAcc) ∘
6       map(λ as .
7         map(λ bs .
8           reduce(+, 0) ∘ map(×) ∘ zip(as, bs)
9           , toLocal(copy2D(tileB)))
10          , toLocal(copy2D(tileA)))
11        ,0, zip(rowOfTilesA, colOfTilesB))
12      ) ∘ tile(m, k, transpose(B))
13    ) ∘ tile(n, k, A)

```



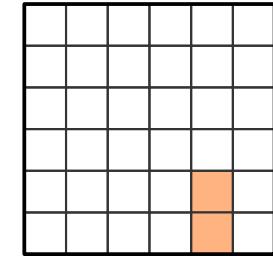
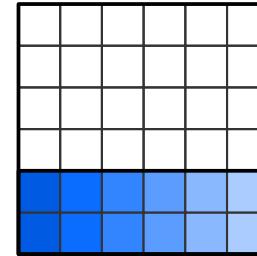
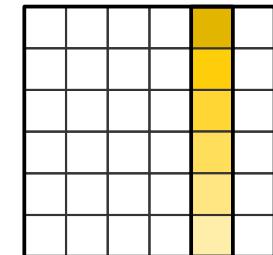
Register Blocking as a Rewrite Rules

```
1  untile o map(λ rowOfTilesA .  
2    map(λ colOfTilesB .  
3      toGlobal(copy2D) o  
4      reduce(λ (tileAcc, (tileA, tileB)) .  
5        map(map(+)) o zip(tileAcc) o  
6        map(λ as .  
7          map(λ bs .  
8            reduce(+, 0) o map(×) o zip(as, bs)  
9            , toLocal(copy2D(tileB)))  
10           , toLocal(copy2D(tileA)))  
11           ,0, zip(rowOfTilesA, colOfTilesB))  
12     ) o tile(m, k, transpose(B))  
13   ) o tile(n, k, A)
```



Apply blocking rules

```
1  untile o map(λ rowOfTilesA .  
2    map(λ colOfTilesB .  
3      toGlobal(copy2D) o  
4      reduce(λ (tileAcc, (tileA, tileB)) .  
5        map(map(+)) o zip(tileAcc) o  
6        map(λ aBlocks .  
7          map(λ bs .  
8            reduce(+, 0) o  
9            map(λ (aBlock, b) .  
10           map(λ (a,bp) . a × bp  
11             , zip(aBlock, toPrivate(id(b))))  
12           ) o zip(transpose(aBlocks), bs)  
13             , toLocal(copy2D(tileB)))  
14             , split(l, toLocal(copy2D(tileA))))  
15             ,0, zip(rowOfTilesA, colOfTilesB))  
16           ) o tile(m, k, transpose(B))  
17         ) o tile(n, k, A)
```



Register Blocking as a Rewrite Rules

registerBlocking =

$\text{Map}(f) \Rightarrow \text{Join}() \circ \text{Map}(\text{Map}(f)) \circ \text{Split}(k)$

$\text{Map}(a \mapsto \text{Map}(b \mapsto f(a, b))) \Rightarrow \text{Transpose}() \circ \text{Map}(b \mapsto \text{Map}(a \mapsto f(a, b)))$

$\text{Map}(f \circ g) \Rightarrow \text{Map}(f) \circ \text{Map}(g)$

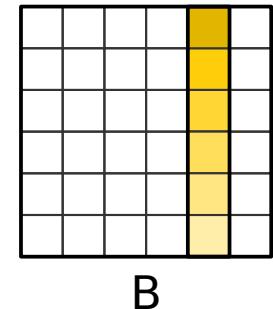
$\text{Map}(\text{Reduce}(f)) \Rightarrow \text{Transpose}() \circ \text{Reduce}((\text{acc}, x) \mapsto \text{Map}(f) \circ \text{Zip}(\text{acc}, x))$

$\text{Map}(\text{Map}(f)) \Rightarrow \text{Transpose}() \circ \text{Map}(\text{Map}(f)) \circ \text{Transpose}()$

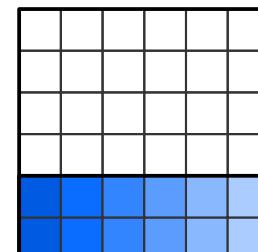
$\text{Transpose}() \circ \text{Transpose}() \Rightarrow id$

$\text{Reduce}(f) \circ \text{Map}(g) \Rightarrow \text{Reduce}((\text{acc}, x) \mapsto f(\text{acc}, g(x)))$

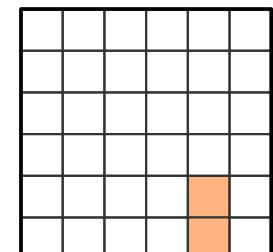
$\text{Map}(f) \circ \text{Map}(g) \Rightarrow \text{Map}(f \circ g)$



B

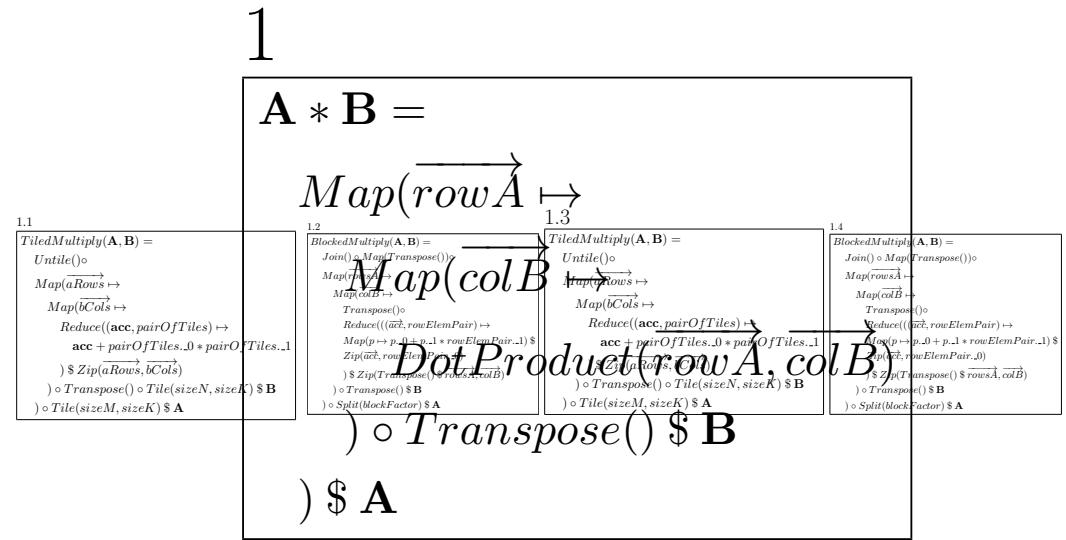
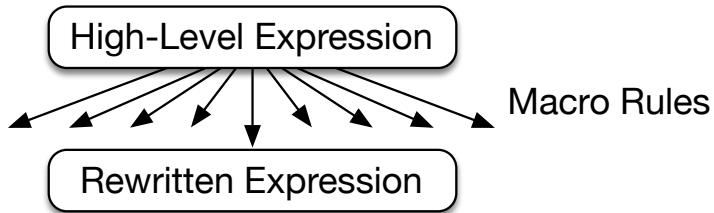


A

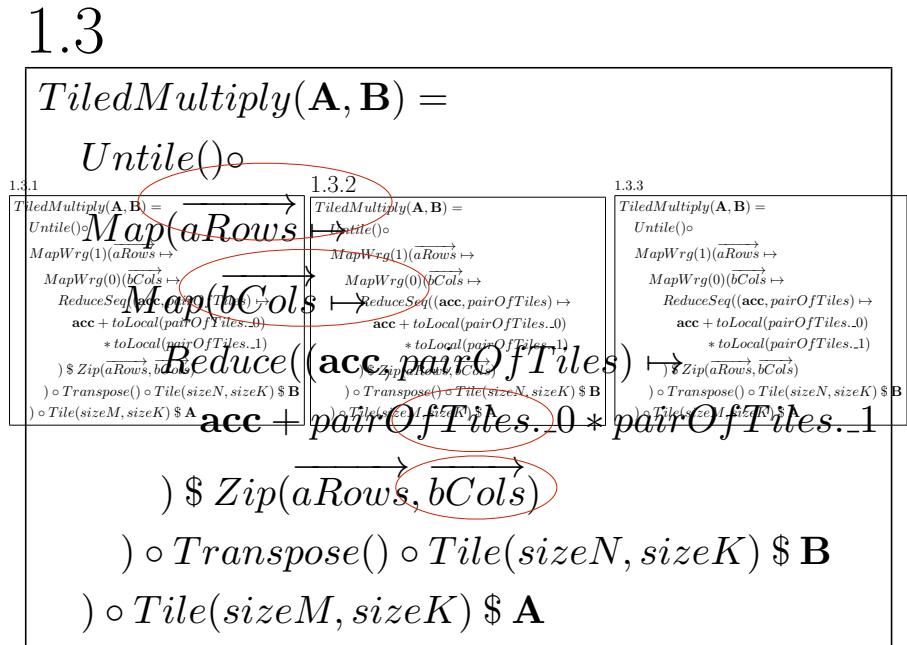
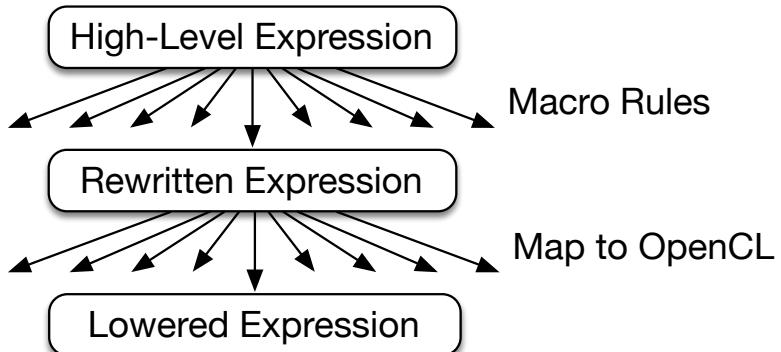


C

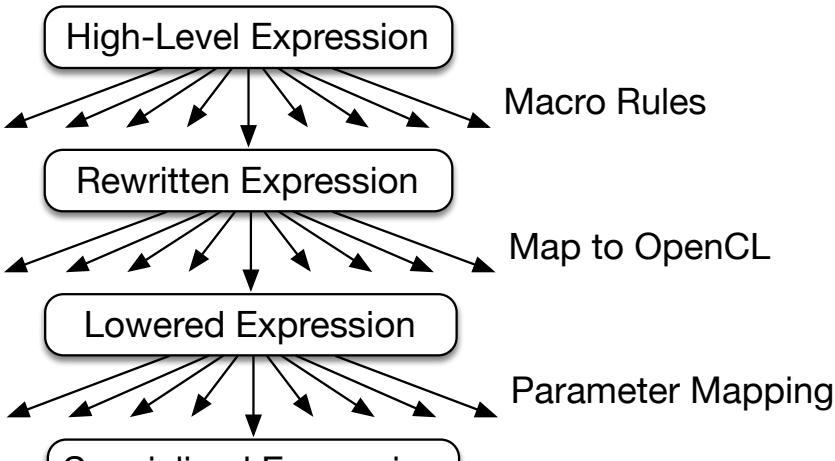
Exploration Strategy



Exploration Strategy



Exploration Strategy



1.3.2

$$\begin{aligned} TiledMultiply(\mathbf{A}, \mathbf{B}) &= \\ &\quad Untile() \circ \end{aligned}$$

1.3.2.1 $\overrightarrow{MapWrg(1)(\overrightarrow{aRows} \mapsto$

<i>TiledMultiply(A, B) =</i>	<i>TiledMultiply(A, B) =</i>
<i>Untile()</i> \circ	<i>Untile()</i> \circ
<i>MapWrg(1)(\overrightarrow{aRows}) \mapsto</i>	<i>MapWrg(1)(\overrightarrow{aRows}) \mapsto</i>
<i>MapWrg(0)(\overrightarrow{bCols}) \mapsto</i>	<i>MapWrg(0)(\overrightarrow{bCols}) \mapsto</i>
<i>ReduceSeq((acc, pairOfTiles..))</i>	<i>ReduceSeq((acc, pairOfTiles..))</i>
<i>acc + toLocal(pairOfTiles..0)</i>	<i>acc + toLocal(pairOfTiles..0)</i>
<i>* toLocal(pairOfTiles..1)</i>	<i>* toLocal(pairOfTiles..1)</i>
<i>) \\$ Zip(\overrightarrow{aRows}, \overrightarrow{bCols})</i>	<i>acc +</i>
<i>) o Transpose() o Tile(128, 16) \\$ B</i>	<i>toLocal(pairOfTiles..0)</i>
<i>) o Tile(128, 16) \\$ A</i>	<i>\\$ Transpose() o Tile(128, 16)</i>
3 2 4	<i>* toLocal(pairOfTiles..1)</i>

$\begin{aligned} & \text{TiledMultiply}(\mathbf{A}, \mathbf{B}) = \\ & \quad \text{Untile}(\mathbf{A}) \circ \\ & \quad \text{MapWrg}(1) \left(\overrightarrow{\text{aRows}} \right) \circ \text{Zip}(\text{aRows}, \mathbf{B}) \\ & \quad \text{MapWrg}(0) \left(\overrightarrow{\text{bCols}} \right) \mapsto \\ & \quad \text{ReduceSeq}(\text{acc}, \text{pairOfTiles}_0) \\ & \quad \text{acc} \circ \text{toLocal}(\text{pairOfTiles}_0) \\ & \quad \left(\begin{array}{c} * \text{toLocal}(\text{pairOfTiles}_{-1}) \\ \text{Zip}(\text{aRows}, \text{bCols}) \end{array} \right) \circ \\ & \quad \text{Transpose}(\text{sizeM}, \text{sizeK}) \circ \text{Tile}(128, 16) \circ \mathbf{B} \\ & \circ \text{Tile}(128, 16) \circ \mathbf{A} \end{aligned}$	$\begin{aligned} & \text{TiledMultiply}(\mathbf{A}, \mathbf{B}) = \\ & \quad \text{Untile}(\mathbf{A}) \circ \\ & \quad \text{MapWrg}(1) \left(\overrightarrow{\text{aRows}} \right) \\ & \quad \text{MapWrg}(0) \left(\overrightarrow{\text{bCols}} \right) \mapsto \\ & \quad \text{Transpose}(\text{oTile}(128, 16)) \\ & \quad \text{ReduceSeq}(\text{acc}, \text{pairOfTiles}_0) \\ & \quad \text{acc} \circ \text{toLocal}(\text{pairOfTiles}_0) \\ & \quad \left(\begin{array}{c} * \text{toLocal}(\text{pairOfTiles}_{-1}) \\ \text{Transpose}(\text{oTile}(128, 16)) \end{array} \right) \circ \\ & \quad \text{Transpose}(\text{oTile}(128, 16)) \circ \mathbf{B} \\ & \circ \text{Tile}(128, 16) \circ \mathbf{A} \end{aligned}$
---	--

```

13.2.3
TiledMultiply(A, B) =
  Untile() ◦
    MapWrg(1)( $\overrightarrow{aRows} \mapsto$ 
      MapWrg(0)( $\overrightarrow{bCols} \mapsto$ 
        ReduceSeq((acc, pairOfTiles)  $\mapsto$ 
          acc  $\dagger$  tbLocal(pairOfTiles..0)
          *  $\mapsto$  Local(pairOfTiles..1))
        fTiles..1)
         $\overrightarrow{aRows} \mapsto$ 
        )  $\circ$  Transpose()  $\circ$  Tile(128, 16) $ B
fTiles..1
 $\dagger$  TbLocal(128, 16) $ A
fTiles..1

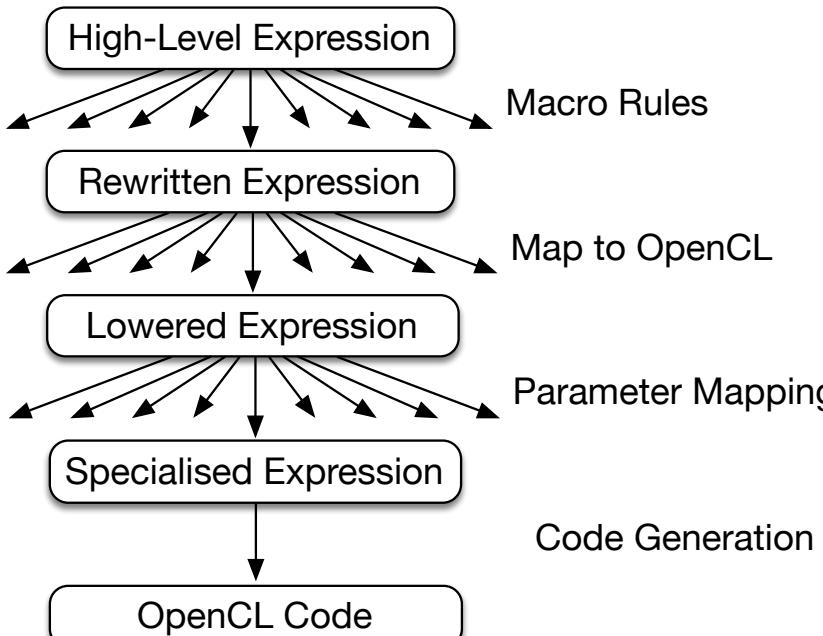
```

```

TiledMultiply(A, B) =
  Until(0)
    MapWrg(1)( $\overrightarrow{aRows}$ )  $\mapsto$ 
      MapWrg(0)( $\overrightarrow{bCols}$ )  $\mapsto$ 
        Reduce( $\overrightarrow{pairOfTiles}$ , pairOfTiles)
          acc + toLocal(pairOfTiles..0)
        * toLocal(pairOfTiles..1)
      ) $ Zip( $\overrightarrow{aRows}$ ,  $\overrightarrow{bCols}$ )
    )  $\circ$  Transpose()  $\circ$  Tile(128, 16) $ B
  )  $\circ$  Tile(128, 16) $ A

```

Exploration Strategy



1.3.2.5

```

1 kernel mm_and_opt(global float *A, B,C,
2   int aRows, int aCols, int bRows, int bCols,
3   local float tileA[512]; tileB[512];
4
5 private float acc_0; ...; acc_31;
6 private float blockOfA_0; ...; blockOfA_7;
7 private float blockOfB_0; ...; blockOfB_7;
8
9 int lid0 = local_id(0); lid1 = local_id(1);
10 int wid0 = group_id(0)*wid + group_id(1);
11
12 for (int w1=wid1; w1<M/64; w1+=num_grps(1)) {
13   for (int w0=wid0; w0<N/64; w0+=num_grps(0)) {
14     acc_0 = 0.0f; acc_1 = 0.0f;
15     for (int i=0; i<K/8; i++) {
16       vstore4(vload4(lid1*M/4+2*i*M+16*w1+lid0,A), 16*lid1+lid0, tileA);
17       vstore4(vload4(lid1*M/4+2*i*M+N/16*w0+lid0,B), 16*lid1+lid0, tileB);
18       barrier (...);
19     }
20   }
21   for (int j = 0; j < 8; j++) {
22     blockOfA_0 = tileA[0+lid1*N+lid0*8]; blockOfA_1 = tileA[1+lid1*N+lid0*8];
23     blockOfB_0 = tileB[0+lid1*N+lid0]; ...; blockOfB_3 = tileB[3+lid1*N+lid0];
24
25     acc_0 += blockOfA_0 * blockOfB_0; ...; acc_28 += blockOfA_7 * blockOfB_0;
26     acc_1 += blockOfA_0 * blockOfB_1; ...; acc_29 += blockOfA_7 * blockOfB_1;
27     acc_2 += blockOfA_0 * blockOfB_2; ...; acc_30 += blockOfA_7 * blockOfB_2;
28     acc_3 += blockOfA_0 * blockOfB_3; ...; acc_31 += blockOfA_7 * blockOfB_3;
29   }
30   barrier (...);
31 } $ Zip(aRows, bCols)
32
33 C[ 0+8*lid1*N+64*w0+64*w1*N+0*N+lid0]=acc_0; ...; C[ 0+8*lid1*N+64*w0+64*w1*N+7*N+lid0]=acc_28;
34 C[16+8*lid1*N+64*w0+64*w1*N+8*N+lid0]=acc_1; ...; C[16+8*lid1*N+64*w0+64*w1*N+7*N+lid0]=acc_29;
35 C[32+8*lid1*N+64*w0+64*w1*N+9*N+lid0]=acc_2; ...; C[32+8*lid1*N+64*w0+64*w1*N+7*N+lid0]=acc_30;
36 C[48+8*lid1*N+64*w0+64*w1*N+0*N+lid0]=acc_3; ...; C[48+8*lid1*N+64*w0+64*w1*N+7*N+lid0]=acc_31;
37 } } } ) $ Tile(128, 16) $ A
  
```

The code snippet shows the generated OpenCL kernel for tiled matrix multiplication. It uses local memory tiles (tileA[512], tileB[512]) and global memory arrays A and B. The kernel iterates over 8x8 blocks of elements. Within each 8x8 block, it performs 8 dot products (acc_0 to acc_7) between corresponding rows from A and columns from B. The result is accumulated into acc_0 to acc_31. The final result is stored in C. The code is annotated with arrows indicating the mapping from the high-level expression to the generated code. For example, *MapWrg(1)(aRows)* points to the first iteration of the outer loop, and *MapWrg(0)(bCols)* points to the second iteration of the inner loop.

Heuristics for Matrix Multiplication

For Macro Rules:

- Nesting depth
- Distance of addition and multiplication
- Number of times rules are applied

For Map to OpenCL:

- Fixed parallelism mapping
- Limited choices for mapping to local and global memory
- Follows best practice

For Parameter Mapping:

- Amount of memory used
 - Global
 - Local
 - Registers
- Amount of parallelism
 - Work-items
 - Workgroup

Exploration in Numbers for Matrix Multiplication

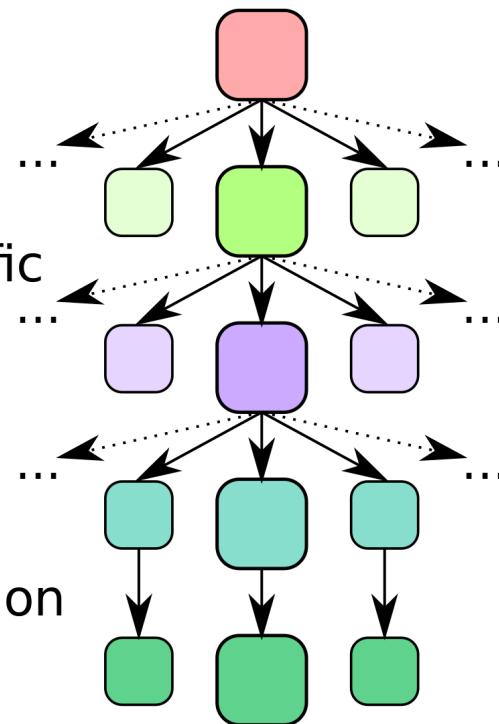
Phases:

Algorithmic Exploration

OpenCL specific Exploration

Parameter Exploration

Code Generation



Program Variants:

High-Level Program 1

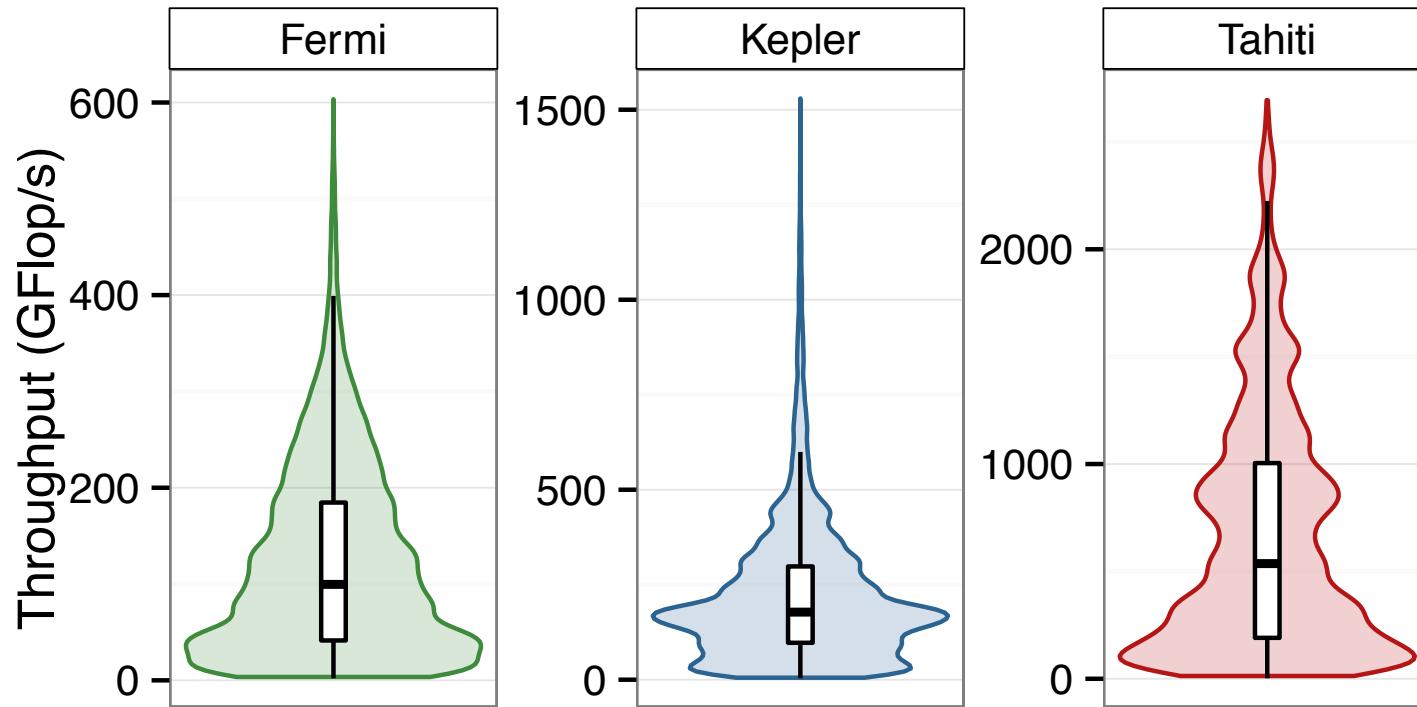
Algorithmic Rewritten Program 8

OpenCL Specific Program 760

Fully Specialized Program 46,000

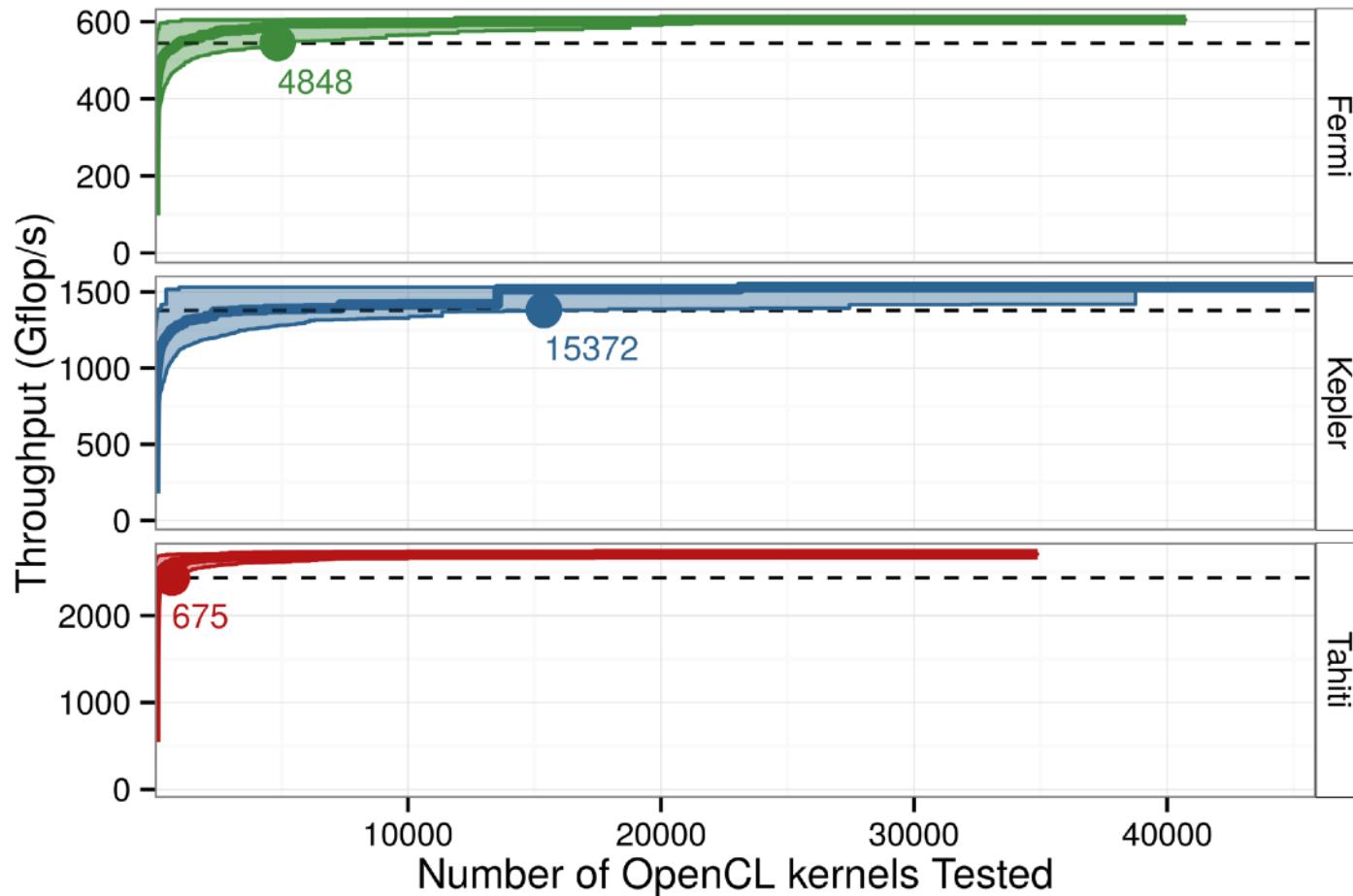
OpenCL Code 46,000

Exploration Space for Matrix Multiplication



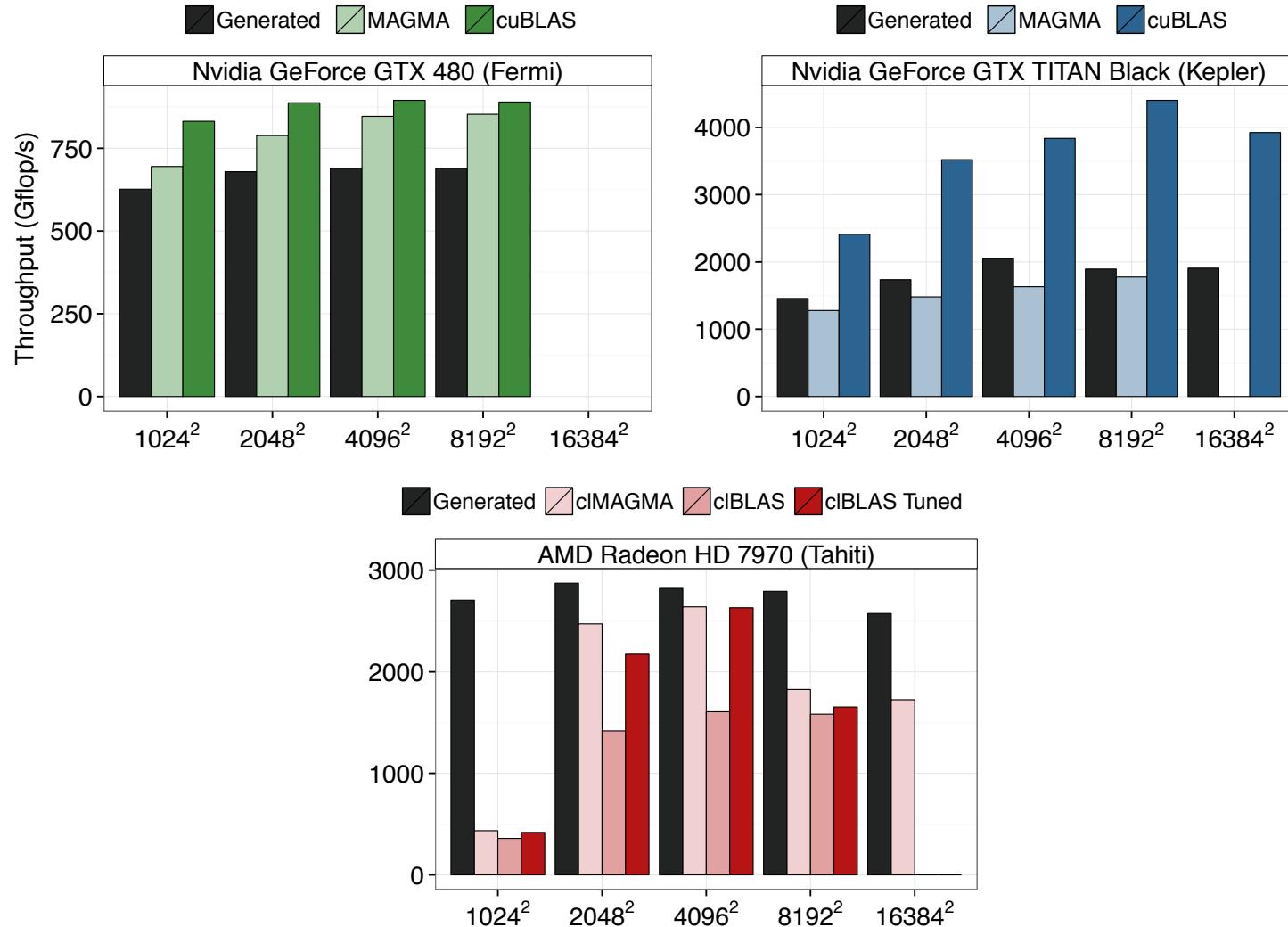
Only few OpenCL kernel with very good performance

Performance Evolution for Randomised Search



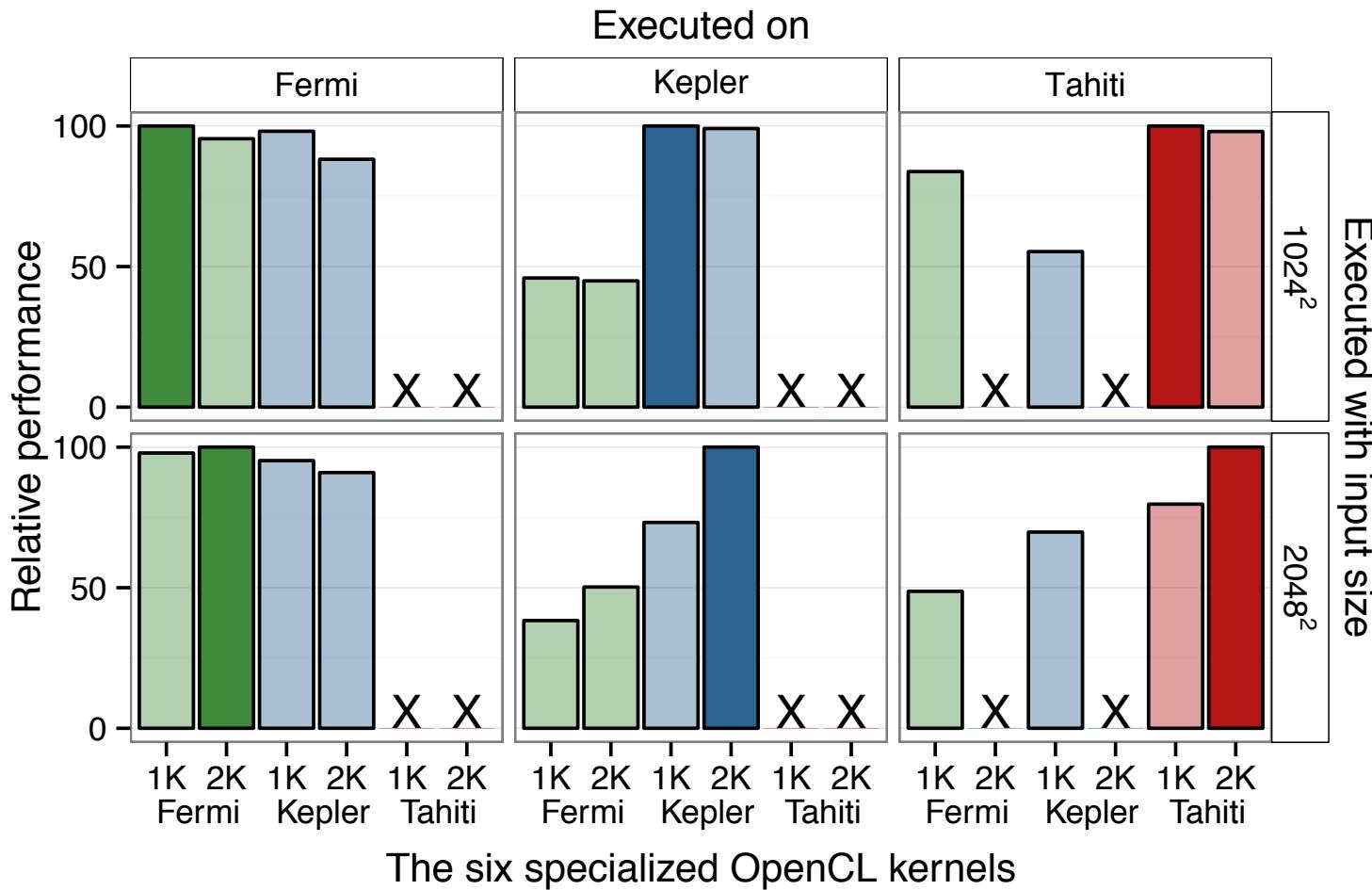
Even with a simple random search strategy one can expect to find a good performing kernel quickly

Performance Results Matrix Multiplication



Performance close or better than hand-tuned MAGMA library

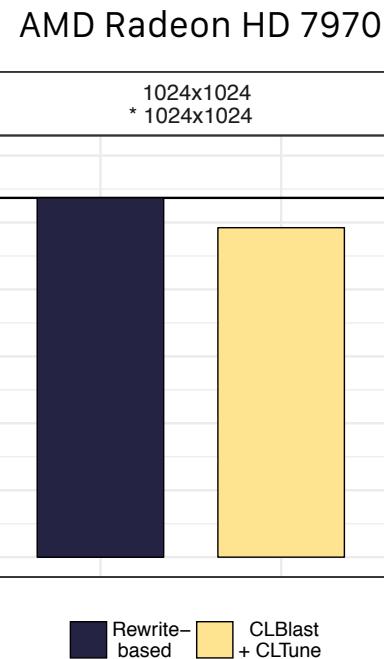
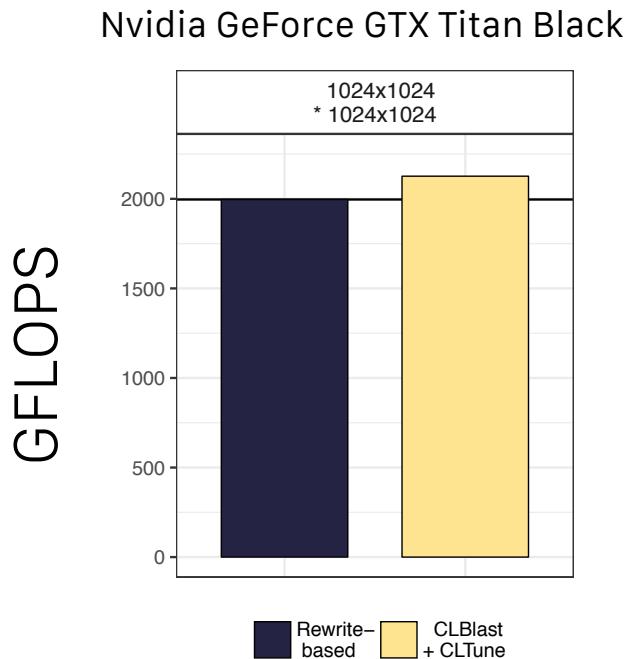
Performance Portability Matrix Multiplication



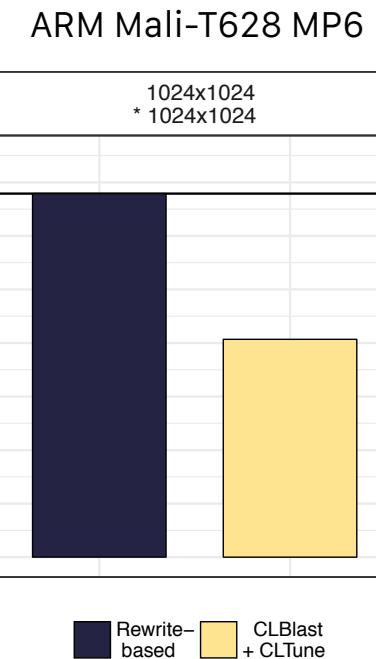
Generated kernels are specialised for device and input size

Desktop GPUs vs. Mobile GPU

Desktop GPUs



Mobile GPU



Performance portable even for mobile GPU device!

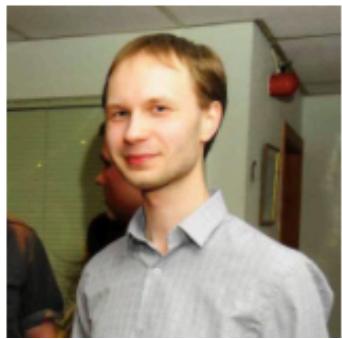


The LIFT Team



THE UNIVERSITY
of EDINBURGH

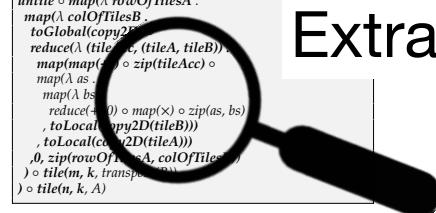




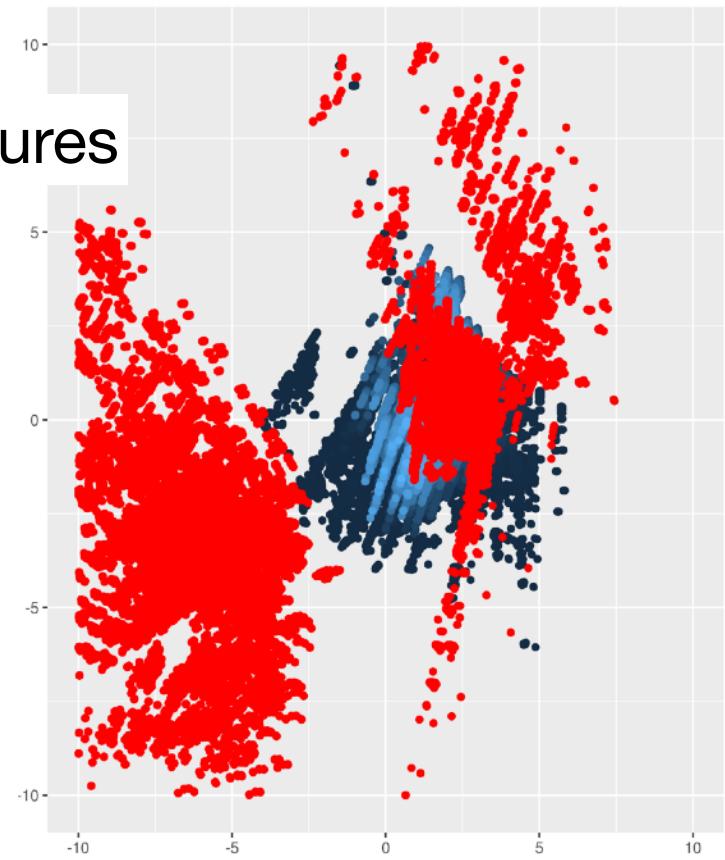
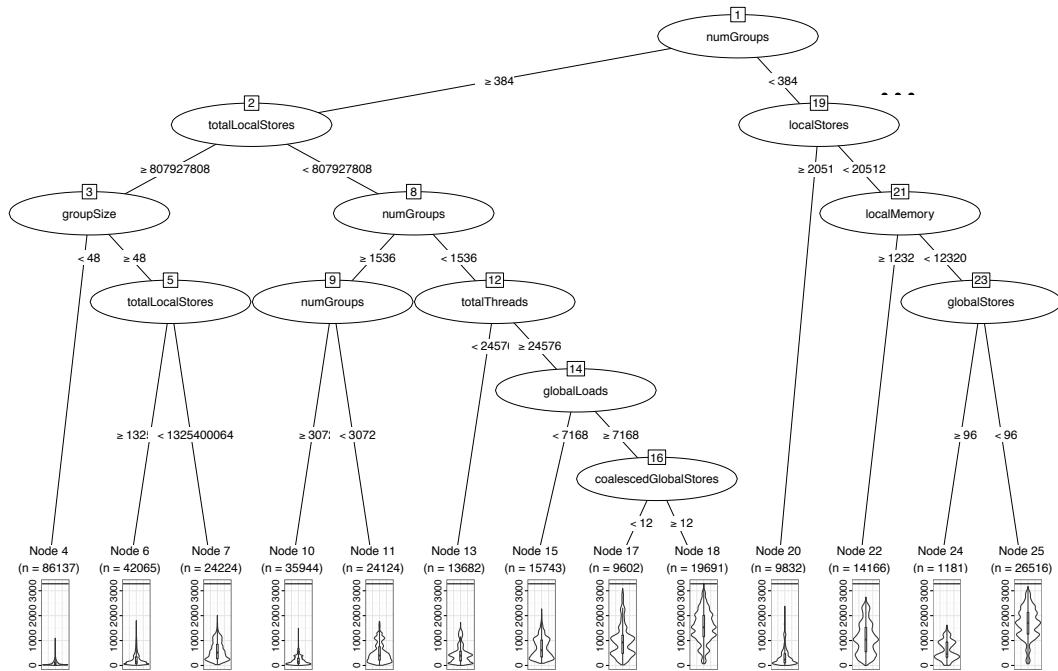
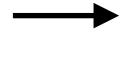
Toomas Remmelt
PhD Student
University of Edinburgh

Performance Modeling of LIFT Programs

```
untile o map(λ rowOfTilesA .  
map(λ colOfTilesB .  
toGlobal(copy2D(tileA, tileB))  
reduce(λ (tileAcc, (tileA, tileB)) .  
map(λ e .  
map(λ as .  
map(λ bs .  
reduce(λ (x) o map(x) o zip(as, bs)  
.toLocal(copy2D(tileB)))  
.toLocal(copy2D(tileA)))  
, λ, zip(rowOfTilesA, colOfTilesB))  
o tile(m, k, transpose(m))  
) o tile(n, k, transpose(m))
```



Extract Features



Predictions
used to drive the
rewrite process



Graph Algorithms via Sparse Linear Algebra in LIFT

Adam Harries
PhD Student
University of Edinburgh

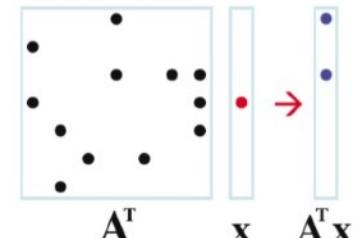
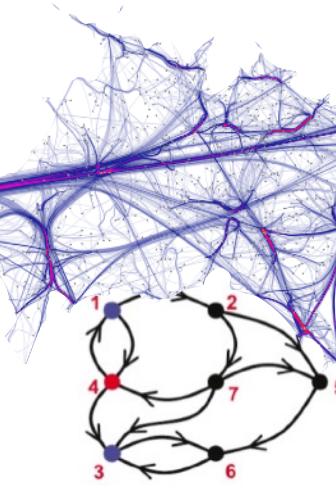
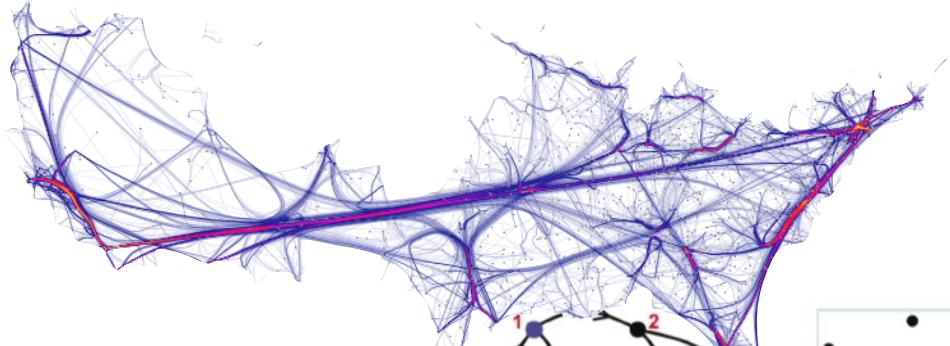
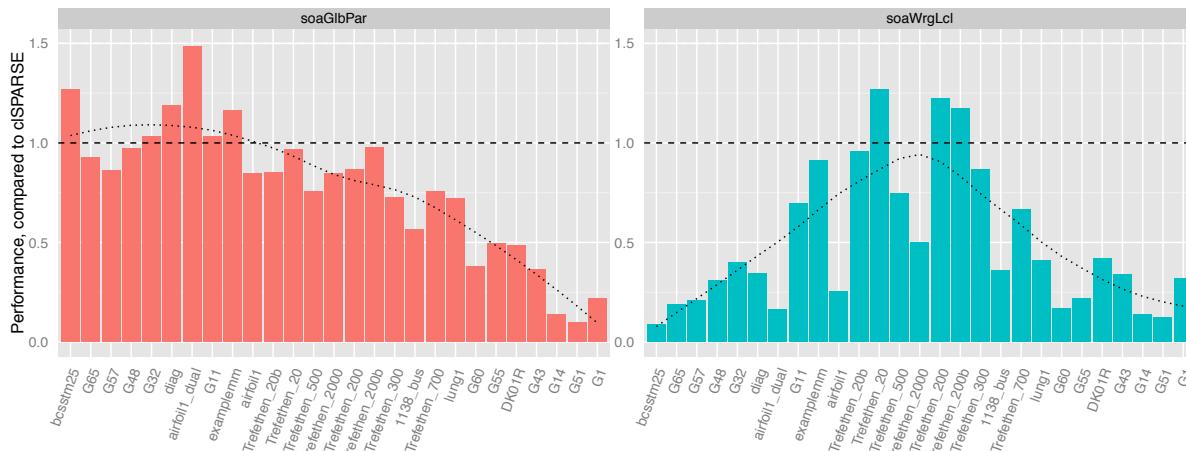


Image credit: [Kepner2011]

```
val sparseMatrixVector = fun(
  ArrayType(ArrayType(Int),N),
  ArrayType(ArrayType(ElemT),N),
  ArrayType(ElemT, M),
  (indices, values, vector) =>
  Map(fun(row =>
    sparseDotProduct(row, vector)),
  Map(Zip,Zip(indices, values))))
```



Differently optimised kernels for different inputs

Identify *hidden parallelism* in LIFT programs



Frederico Pizzuti
PhD Student
University of Edinburgh

Parallelising non-associative reductions

$x \leftarrow 0; \text{for } i = 0 \text{ to } n \text{ do } x \leftarrow c \cdot x + a[i] \text{ done.}$

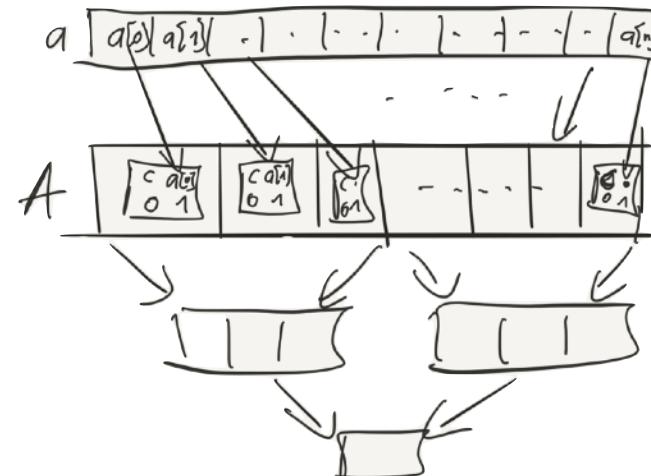
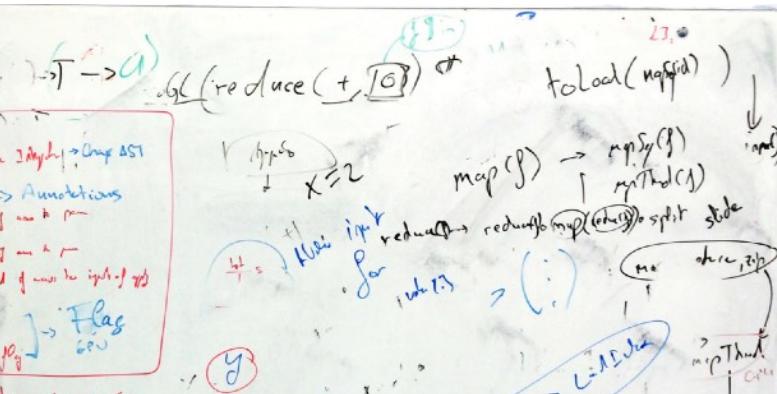


$x \leftarrow x_0; \text{for } i = 0 \text{ to } n \text{ do } x \leftarrow A_i \times x \text{ done,}$

where $x = \begin{pmatrix} x \\ 1 \end{pmatrix}$, $x_0 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$, $A_i = \begin{pmatrix} c & a[i] \\ 0 & 1 \end{pmatrix}$.



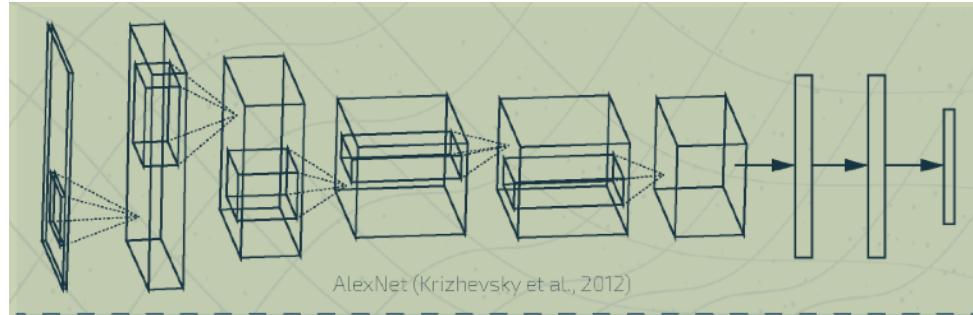
Key idea: Rearrange data as matrices to exploit associative matrix multiplication





Naums Mogers
PhD Student
University of Edinburgh

Optimising Deep Learning with LIFT



Express layers with LIFT primitives

```
fully_connected(f, weights, bias, inputs) :=  
    Map((neuron_weights, neuron_bias) → f() ∘ Reduce(add, neuron_bias) ∘  
        Map(mult) $ Zip(inputs, neuron_weights)) $ Zip(weights, bias)
```

Optimise individual layers and across layers via rewrites

FPGAs



Microsoft®
Research



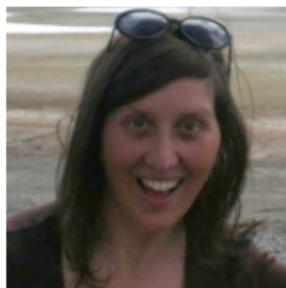
Google TPU

Low Power Devices





Bastian Hagedorn
PhD Student
University of Münster



Larisa Stoltzfus
PhD Student
University of Edinburgh

Stencil Computations in LIFT

Express Stencil with Skeletons

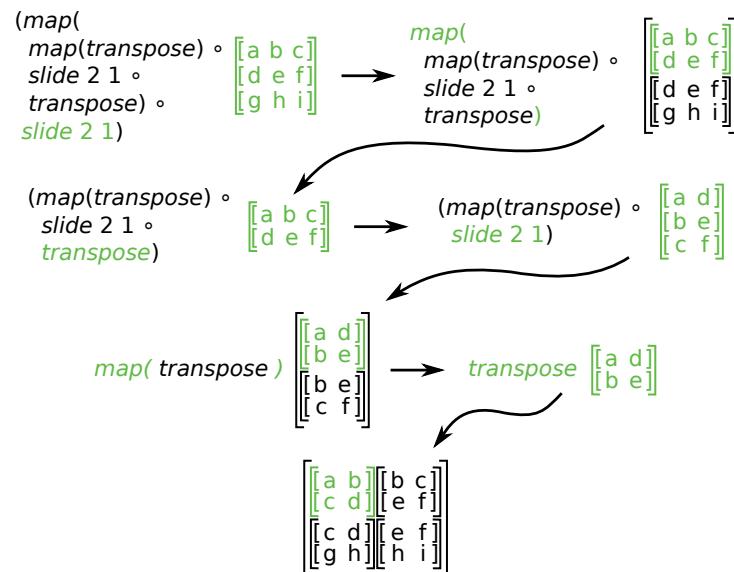
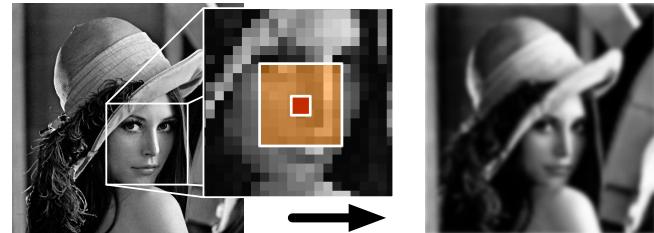
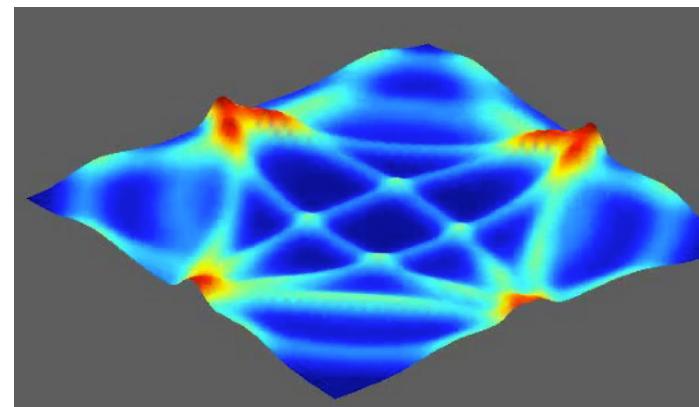


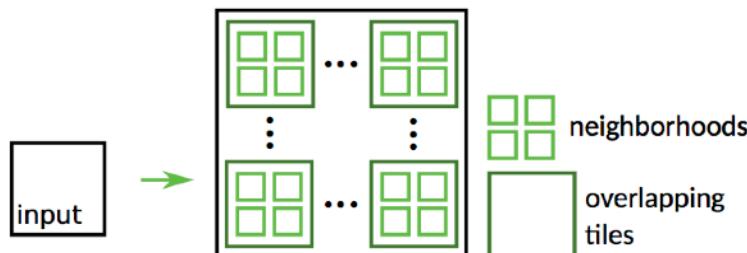
Image Processing



Acoustics Simulation



Explore optimisations as rewrites



Video

LIFT is Open-Source Software

<http://www.lift-project.org/>

<https://github.com/lift-project/lift>

The screenshot shows a web browser window with the GitHub repository for 'lift-project/lift'. The title bar reads 'lift-project/lift: The Lift program' and the address bar shows 'GitHub, Inc. [US] https://github.com/lift-project/lift'. The repository page displays basic statistics: 1,923 commits, 1 branch, 0 releases, 10 contributors, and an MIT license. A green 'Clone or download' button is prominent. Below the stats, a list of recent commits is shown, including one from 'michel-steuwer' and several from 'docker'. The commit from 'michel-steuwer' is dated '2 days ago'. The commit from 'docker' is dated '4 months ago'. The commit from 'highLevel' is dated '7 months ago'. The commit from 'lib' is dated '6 days ago'. The commit from 'native' is dated 'a year ago'. The commit from 'presentations' is dated 'a year ago'. The number '49' is visible at the bottom of the commit list.

The Lift programming language <http://www.lift-project.org/> — Edit

1,923 commits 1 branch 0 releases 10 contributors MIT

Branch: master ▾ New pull request Create new file Upload files Find file Clone or download ▾

michel-steuwer committed on GitHub Made LICENSE file parsable for github Latest commit 8b13aac 2 days ago

docker Cleaning up the top folder of the repo and restructuring the docker s... 4 months ago

highLevel refactoring 7 months ago

lib Bump ArithExpr 6 days ago

native Add support for querying if the device supports double 1 a year ago

presentations Added power point slides of ICFP, PL Interest and PENCIL meeting. 1 a year ago

49

The LIFT Project

Performance Portable Parallel
Code Generation via Rewrite Rules

Michel Steuwer — michel.steuwer@glasgow.ac.uk

www.lift-project.org

 @LIFTlang

INSPIRING
PEOPLE

#UofGWorldChangers



@UofGlasgow