



University
of Glasgow

The LIFT Project

Performance Portable Parallel Code Generation via Rewrite Rules

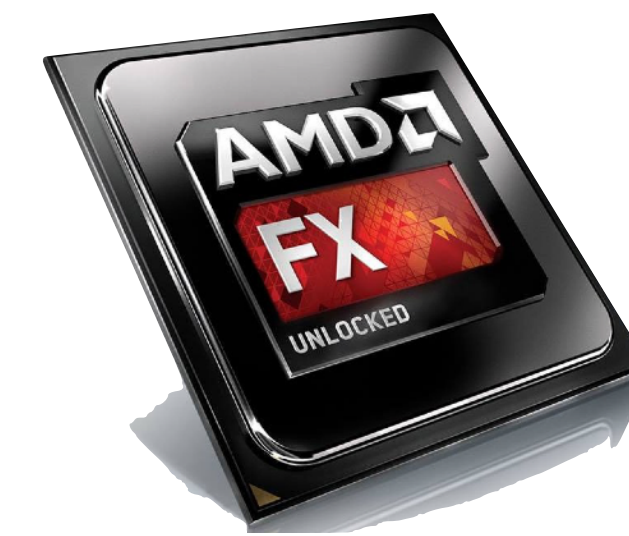
Michel Steuwer — michel.steuwer@glasgow.ac.uk

**INSPIRING
PEOPLE**

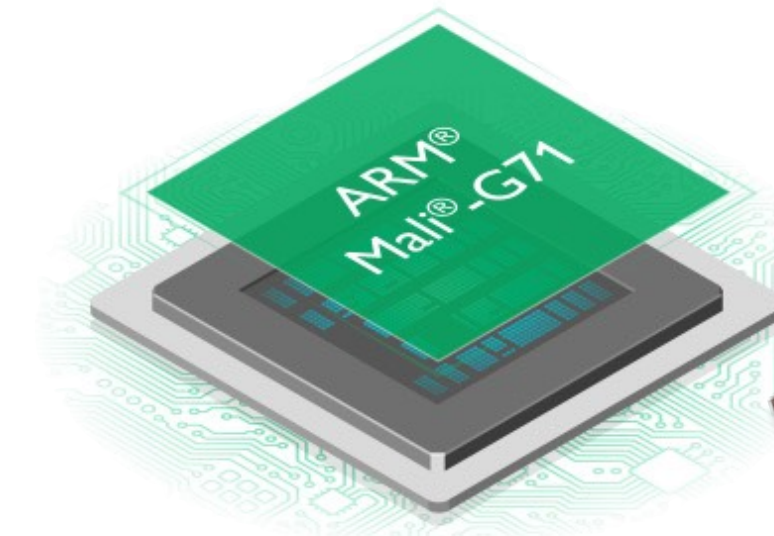


What are the problems LIFT tries to tackle?

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



CPU

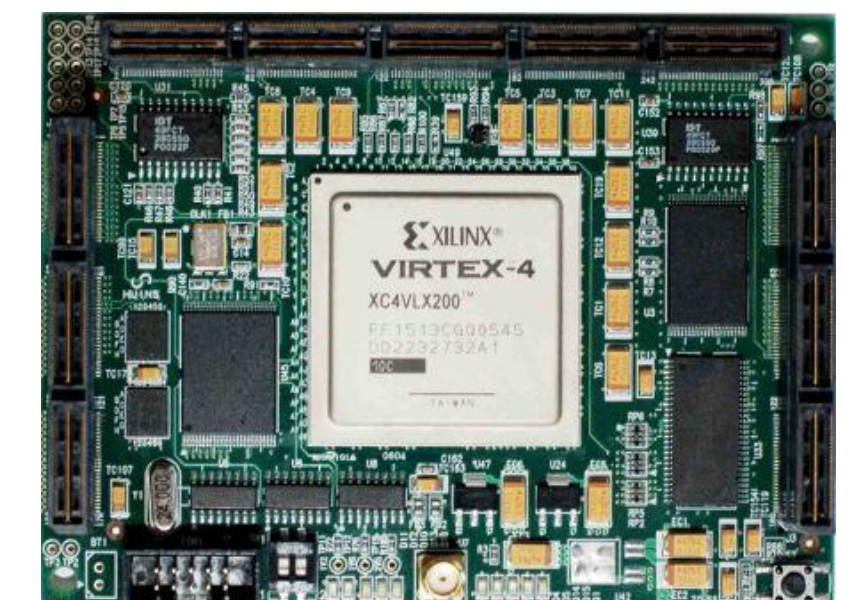


GPU



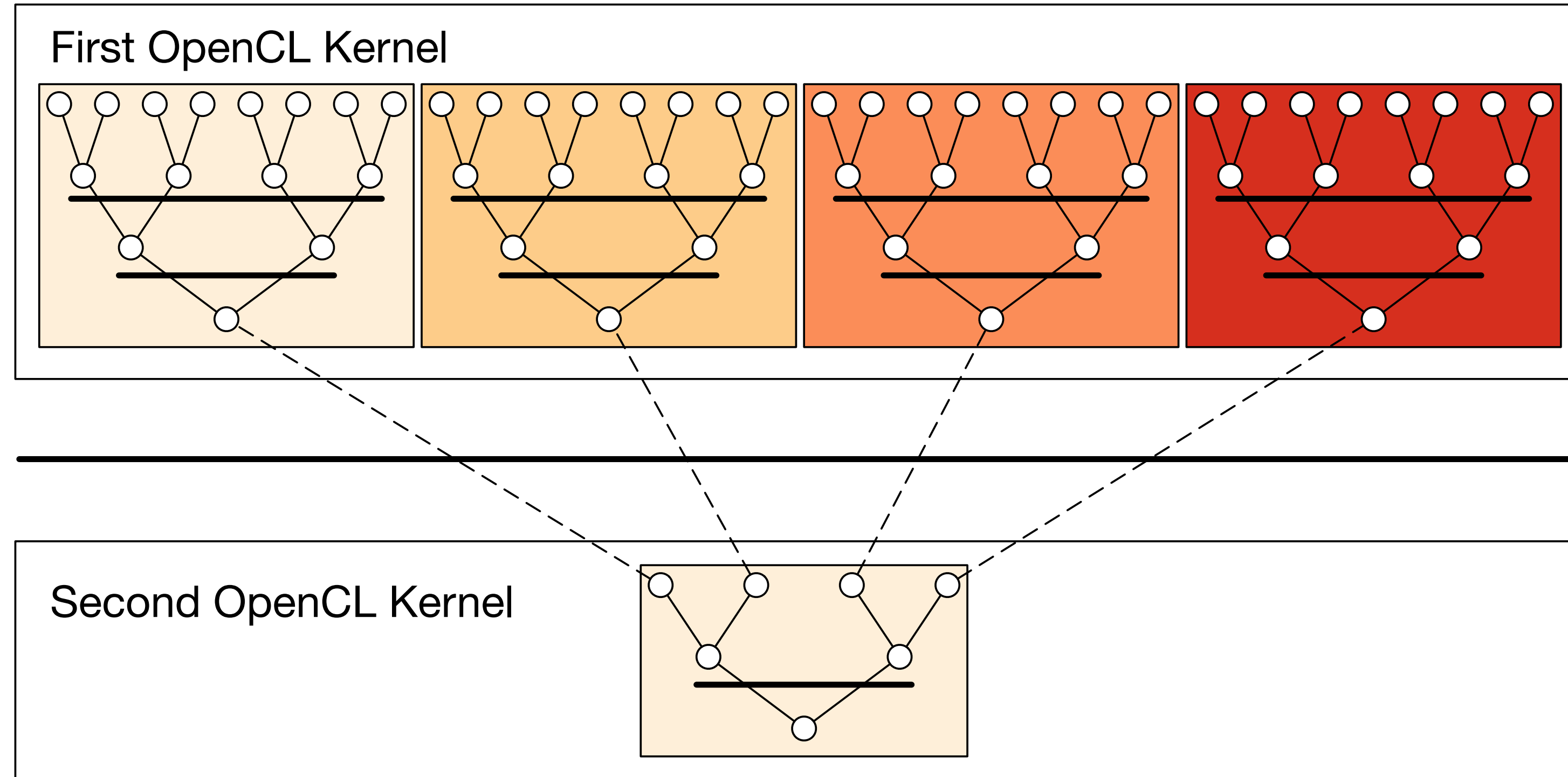
Accelerator

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



Parallel reduction with OpenCL

```
kernel void reduce(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];
}
```

Parallel reduction with OpenCL

Kernel function executed in parallel by multiple **work-items**

```
kernel void reduce(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];
}
```

Work-items are identified by a unique **global id**

Parallel reduction with OpenCL

Work-items are grouped into **work-groups**

Local id within work-group

```
kernel void reduce(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];
}
```

Parallel reduction with OpenCL

Big, but slow **global** memory

Small, but fast **local** memory

```
kernel void reduce(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];
}
```

Memory **barriers** for consistency

Parallel reduction with OpenCL

Potential **Deadlock!**

```
kernel void reduce(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];
}
```

Functionally correct implementations in OpenCL are hard!

1. Version: 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];
}
```

2. Version: 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];
}
```

3. Version: 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];
}
```

4. Version: 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];
}
```

5. Version: 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]; }
```

6. Version: 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]; }
```

7. Version: 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 Conclusions

- 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

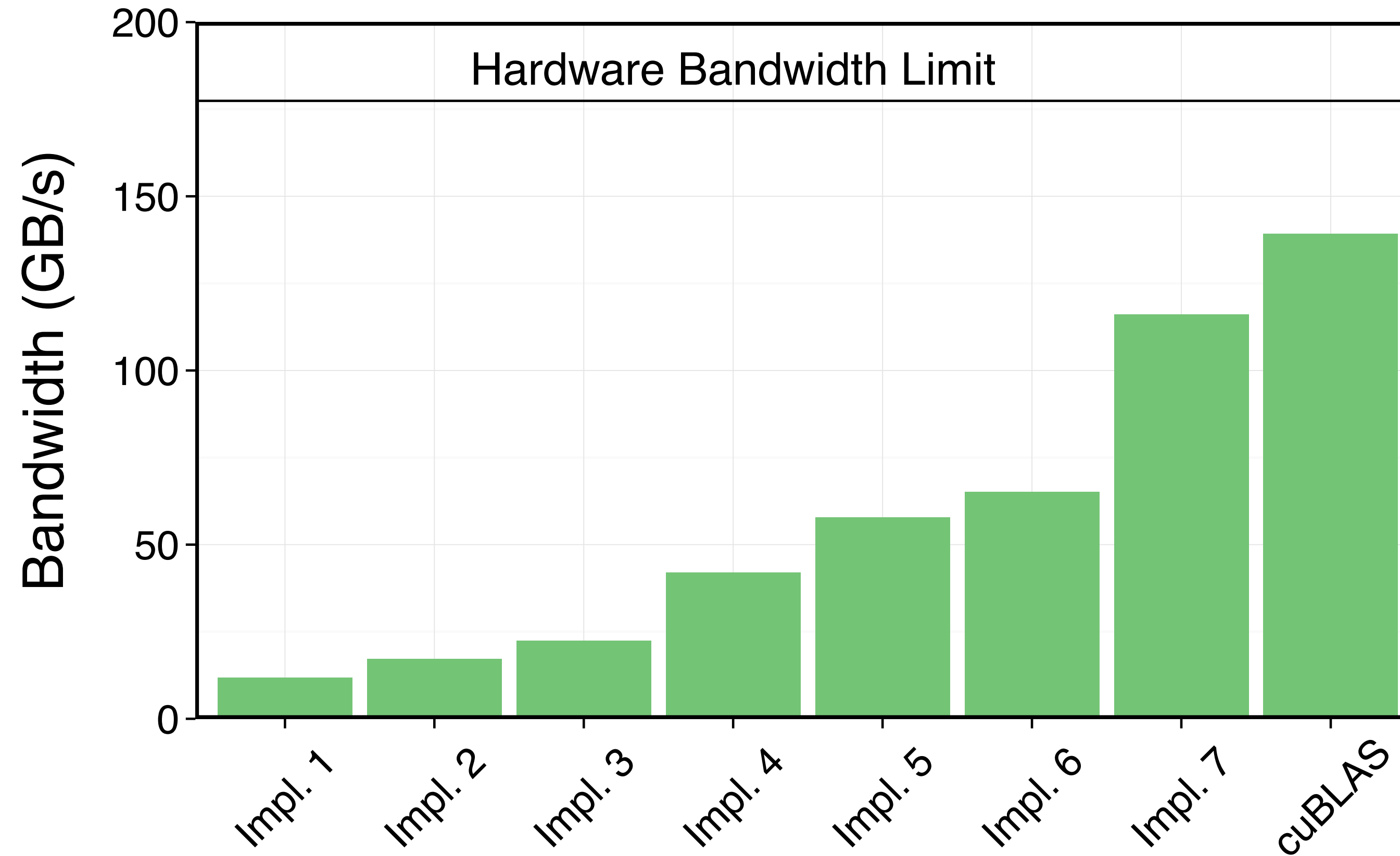
```
kernel
void reduce6(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local volatile float* l_data) {
    unsigned int tid = get_local_id(0);
    unsigned int i =
        get_group_id(0) * (get_local_size(0)*2)
        + get_local_id(0);

    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

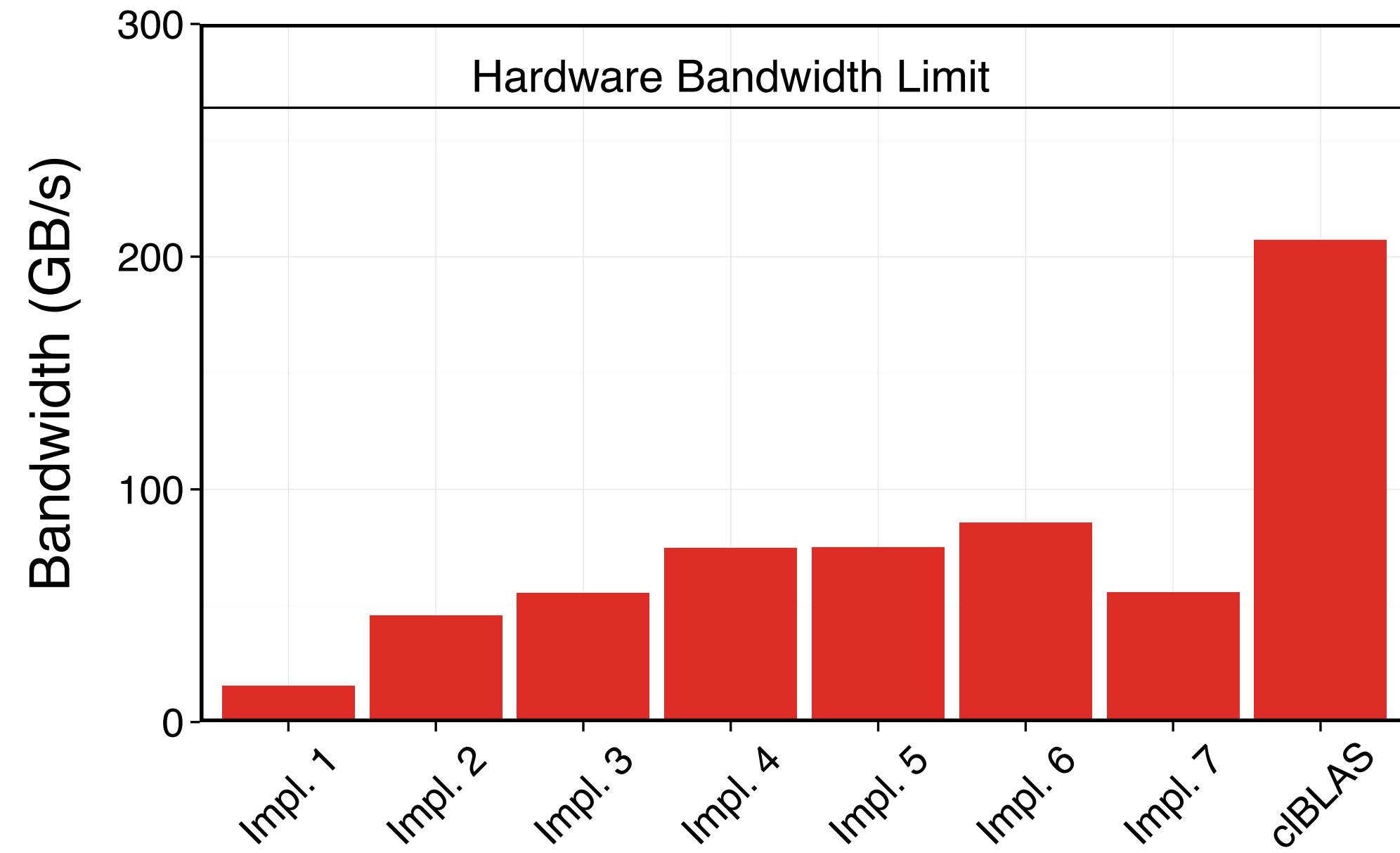
Performance Results Nvidia



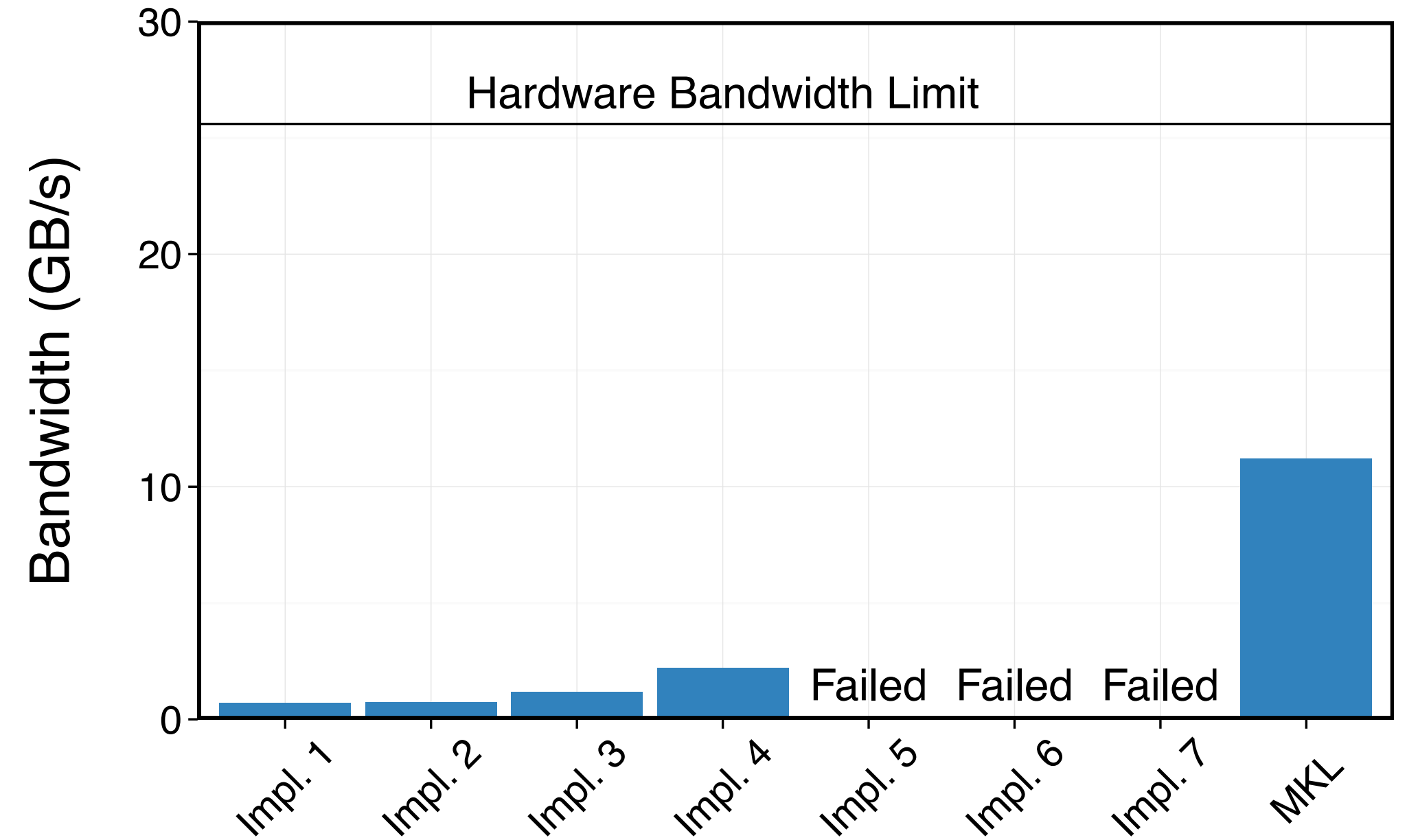
(a) Nvidia's GTX 480 GPU.

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

Performance Results AMD and Intel



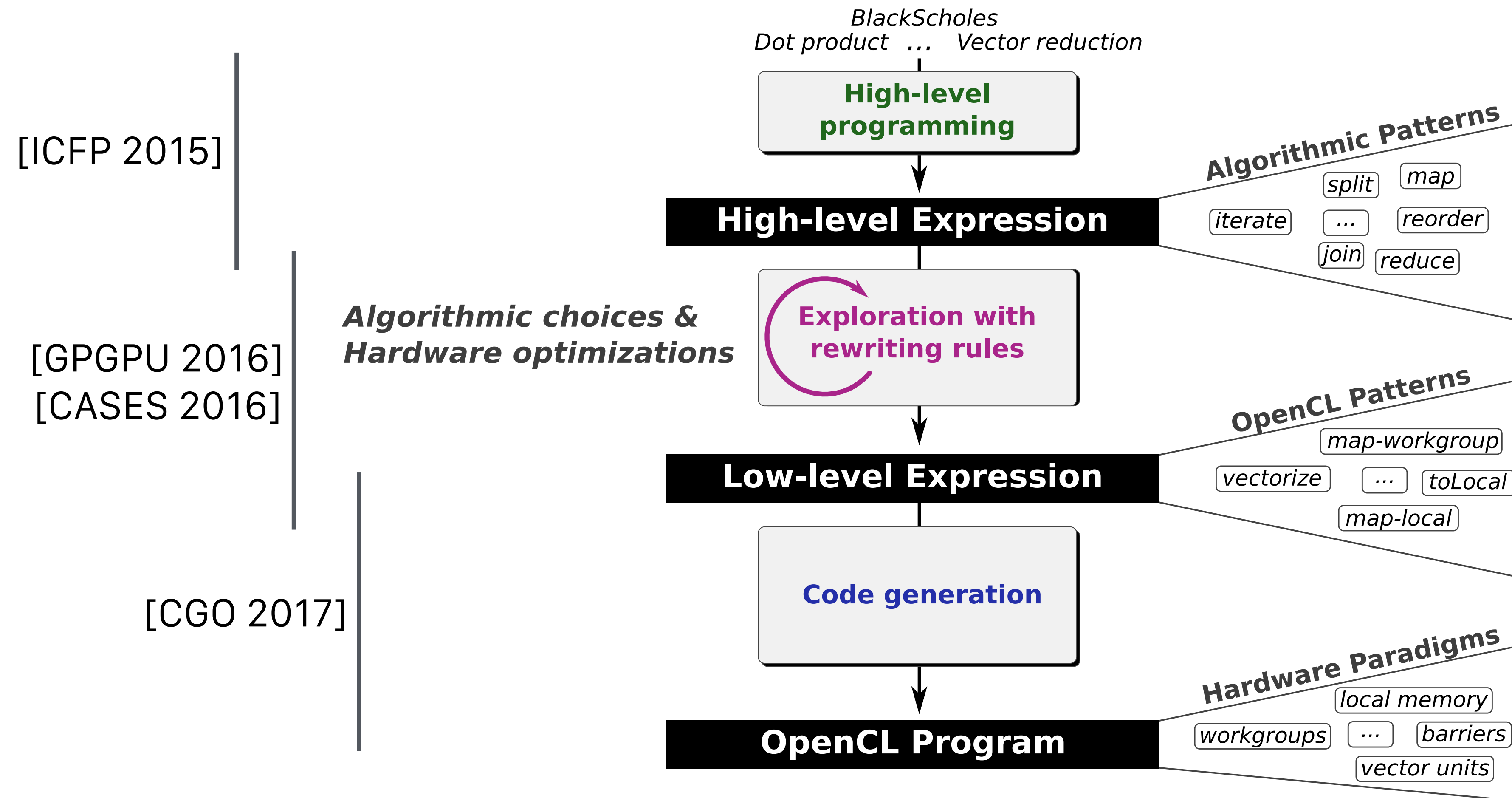
(b) AMD's HD 7970 GPU.



(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



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

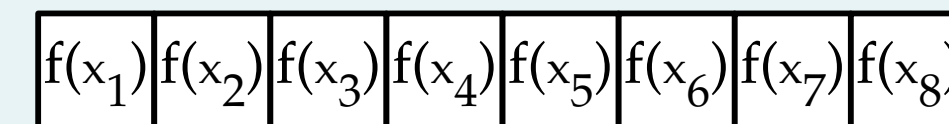
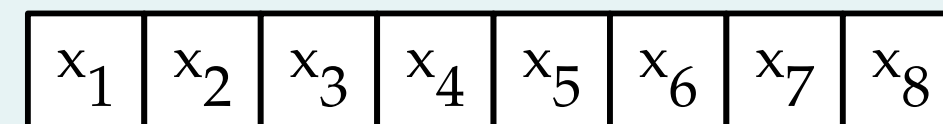
```
vecSum = reduce ◦ join ◦ map-workgroup (  
  join ◦ toGlobal (map-local (map-seq id)) ◦ split 1 ◦  
  join ◦ map-warp (  
    join ◦ map-lane (reduce-seq (+) 0) ◦ split 2 ◦ reorder-stride 1 ◦  
    join ◦ map-lane (reduce-seq (+) 0) ◦ split 2 ◦ reorder-stride 2 ◦  
    join ◦ map-lane (reduce-seq (+) 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

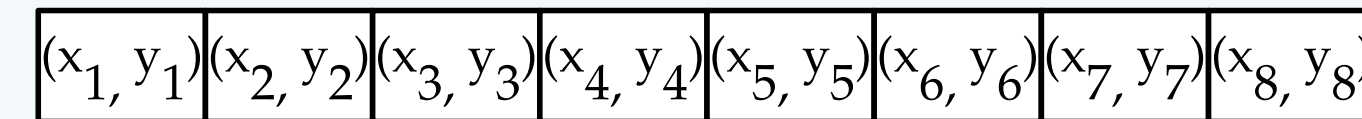
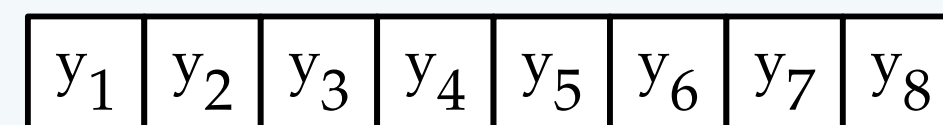
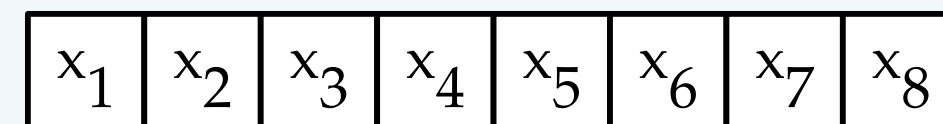
```
kernel  
void reduce6(global float* g_idata,  
             global float* g_odata,  
             unsigned int n,  
             local volatile float* l_data) {  
  unsigned int tid = get_local_id(0);  
  unsigned int i =  
    get_group_id(0) * (get_local_size(0)*2)  
    + get_local_id(0);  
  
  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)

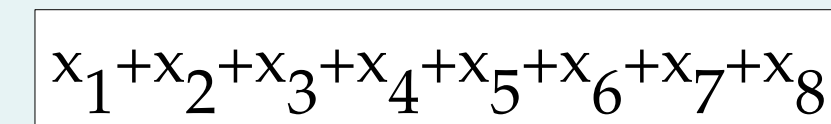
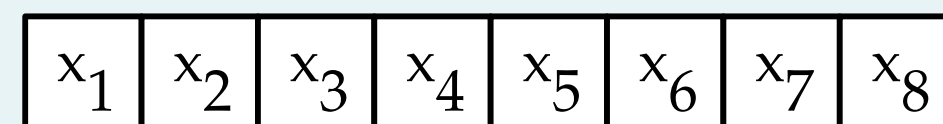
map(f, x):



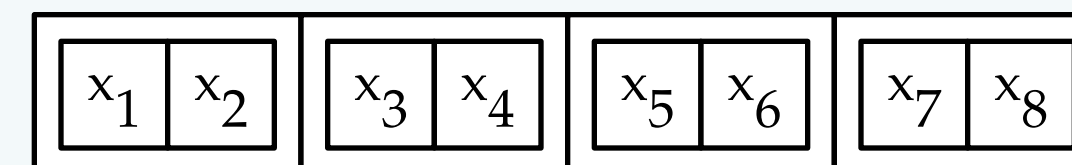
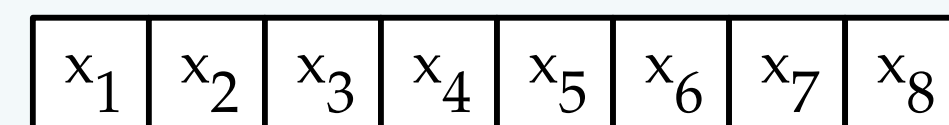
zip(x, y):



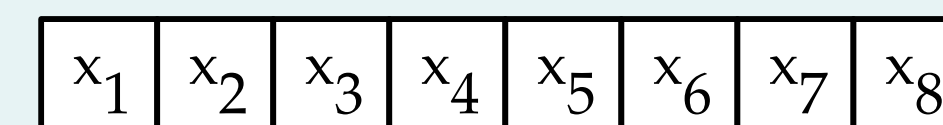
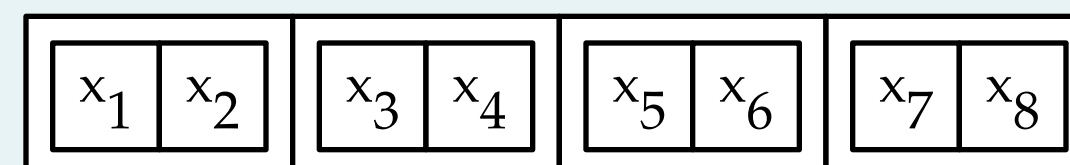
reduce($+, 0, x$):



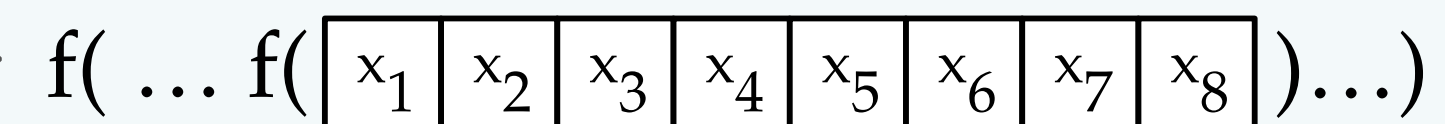
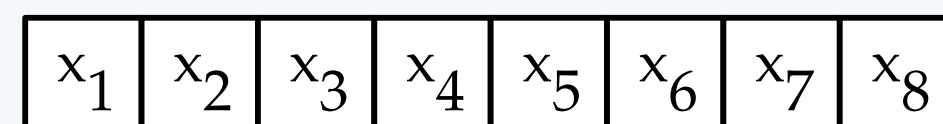
split(n, x):



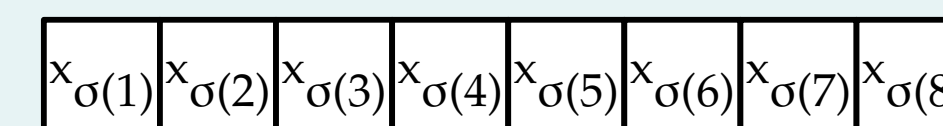
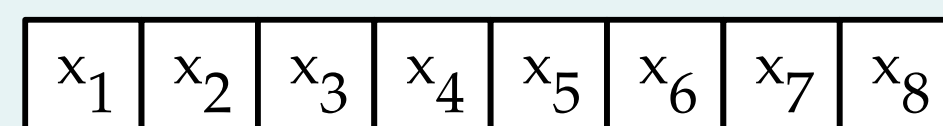
join(x):



iterate(f, n, x):



reorder(σ, x):



① High-Level Programs

`scal(a, vec) = map($\lambda x \mapsto x*a$, 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 \mapsto 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 $\text{sum}(\text{vec}) = \text{reduce}(+, 0, \text{vec})$

↓
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
```

↗
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 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

OpenCL concept

mapGlobal

Work-items

mapWorkgroup

Work-groups

mapLocal

mapSeq

Sequential implementations

reduceSeq

toLocal, toGlobal

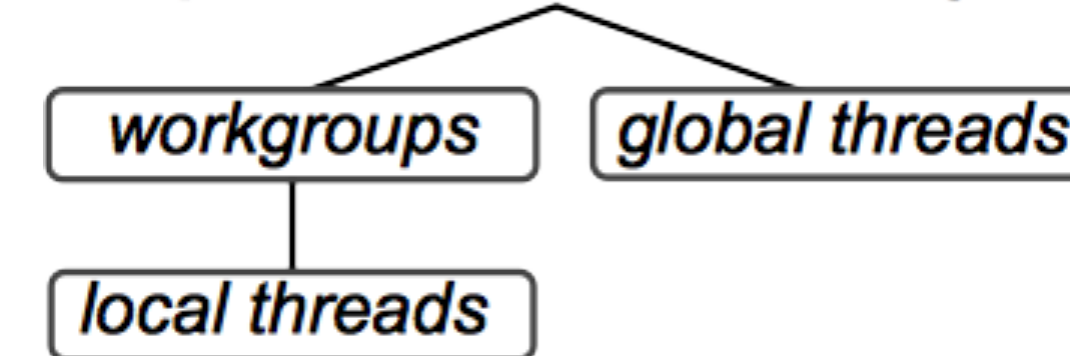
Memory areas

mapVec,

Vectorisation

splitVec, joinVec

OpenCL thread hierarchy



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 $vecSum = reduce (+) 0$

↓
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
```

↗ 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];
}
```

Walkthrough

① $vecSum = reduce (+) 0$

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
```

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];  
}
```

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 $vecSum = reduce (+) 0$

↓
rewrite rules

↗
code generation

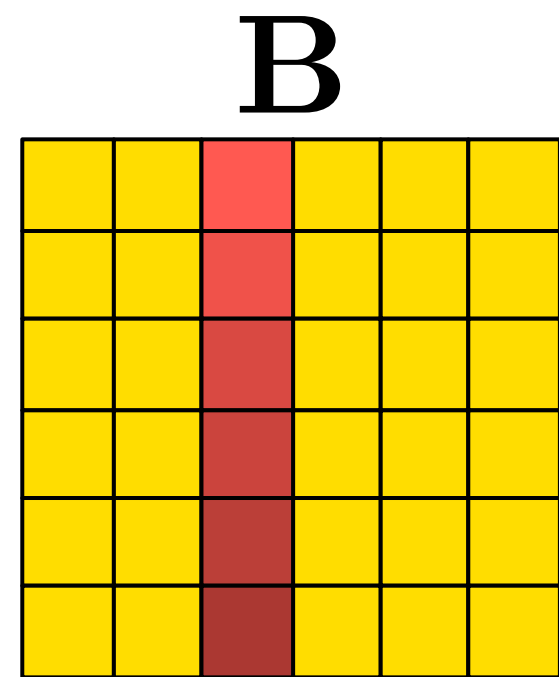
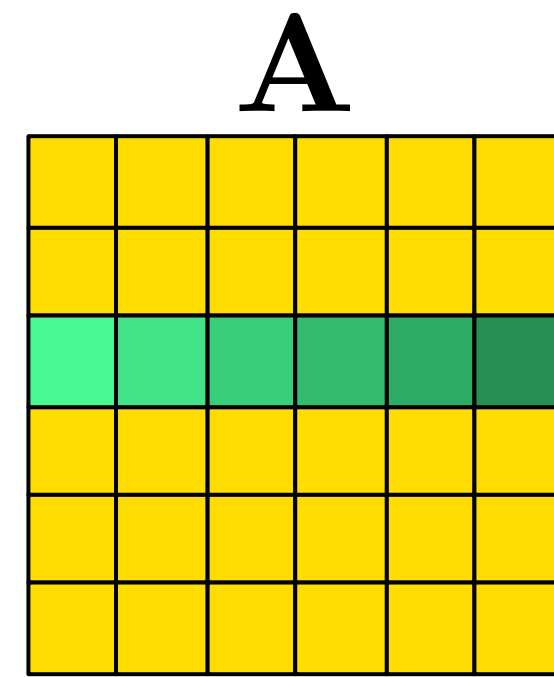
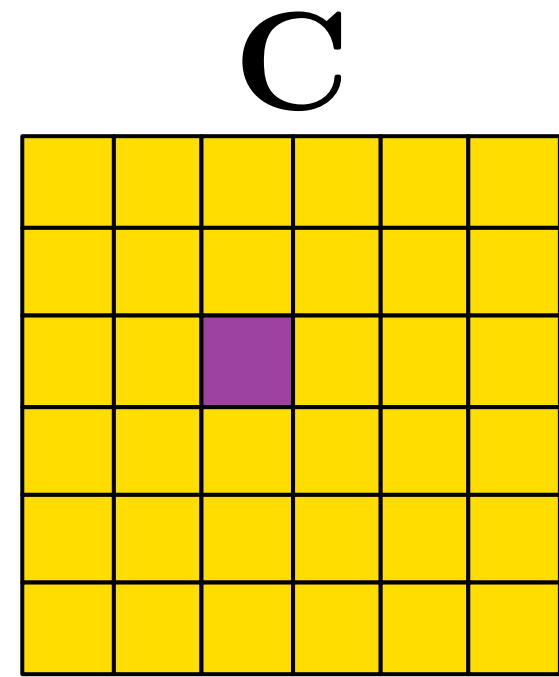
②

```
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];  
}
```


Case Study: Matrix Multiplication

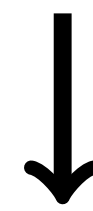


$A \times B =$
 $\text{map}(\lambda \text{ rowA} \mapsto$
 $\text{map}(\lambda \text{ colB} \mapsto$
 $\text{dotProduct}(\text{rowA}, \text{colB})$
 , $\text{transpose}(B)$)
 , A)

Tiling as a Rewrite Rules

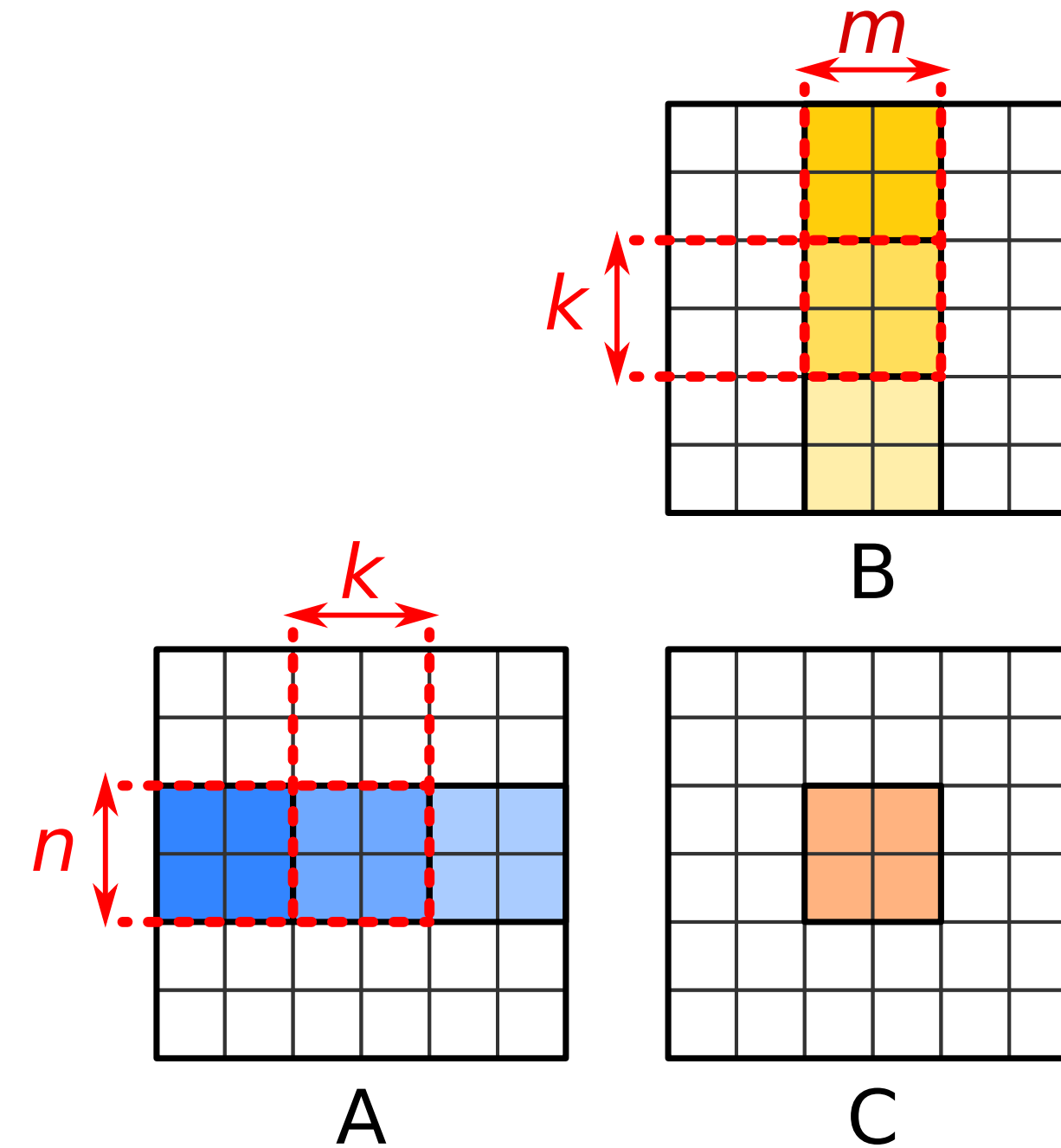
Naïve matrix multiplication

```
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  until ◦ 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)

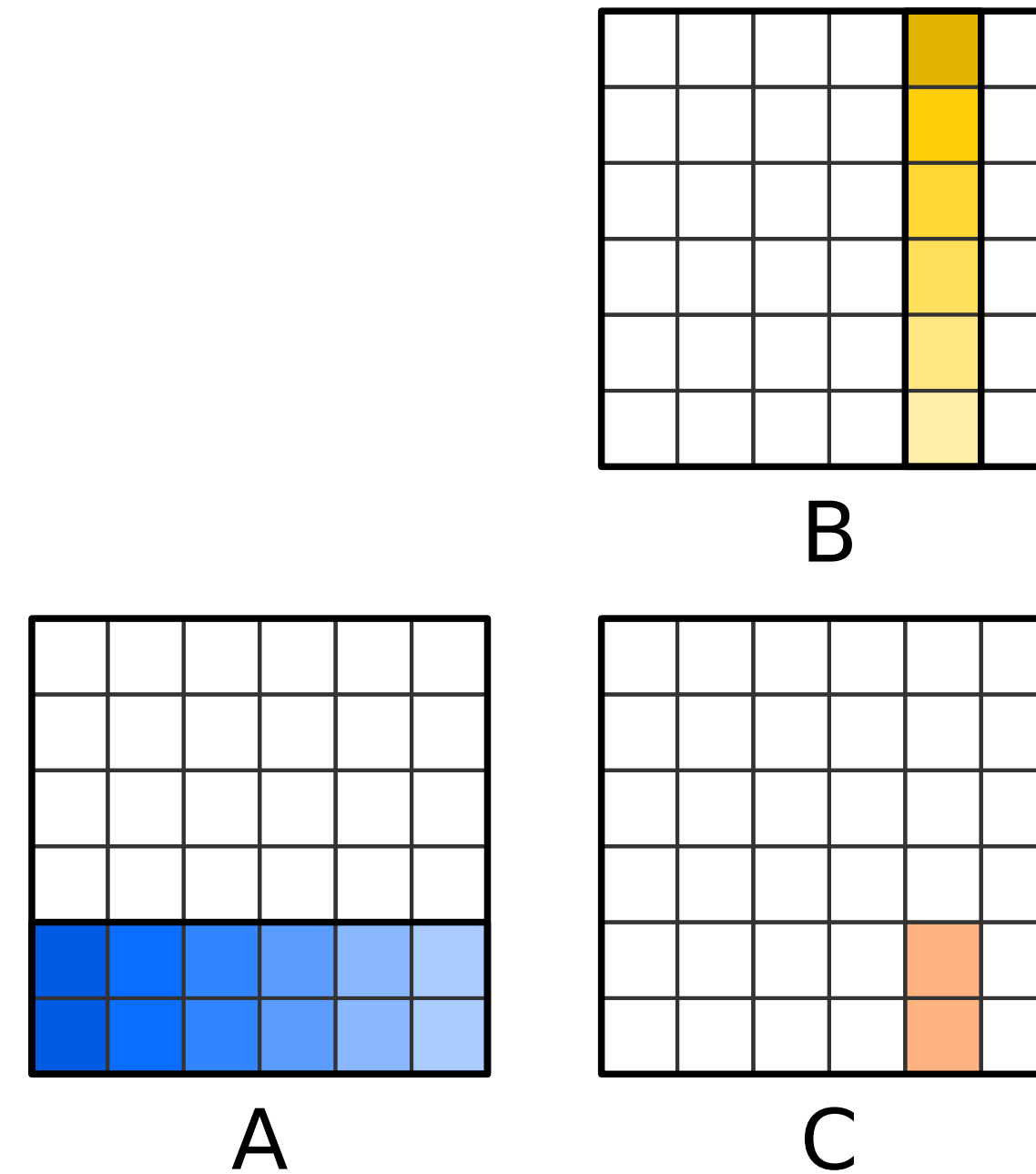
```

↓ Apply blocking rules

```

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

```



Register Blocking as a Rewrite Rules

registerBlocking =

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

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

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

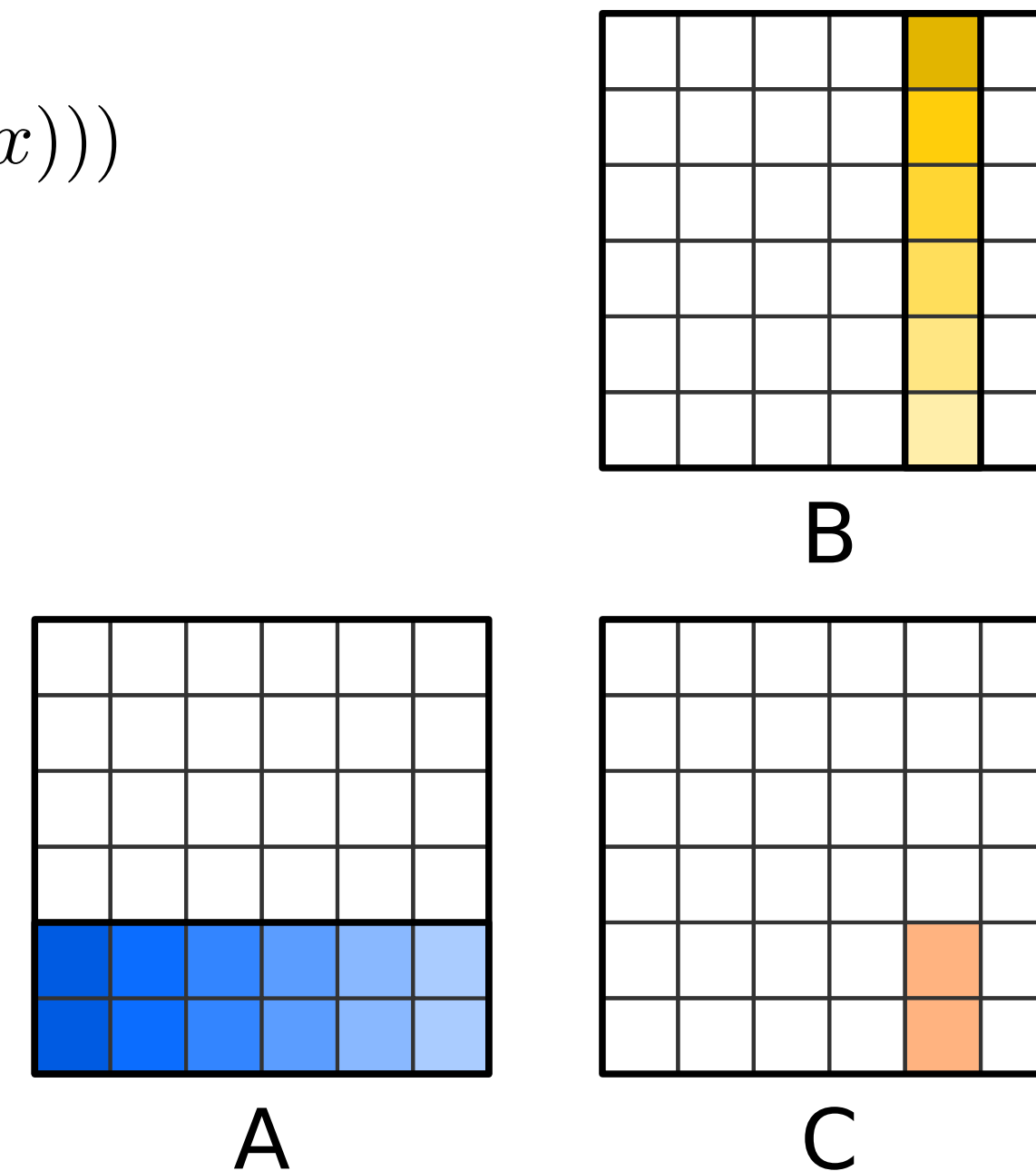
$$\text{Map}(\text{Reduce}(f)) \Rightarrow \text{Transpose}() \circ \text{Reduce}((acc, x) \mapsto \text{Map}(f) \circ \text{Zip}(acc, x))$$

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

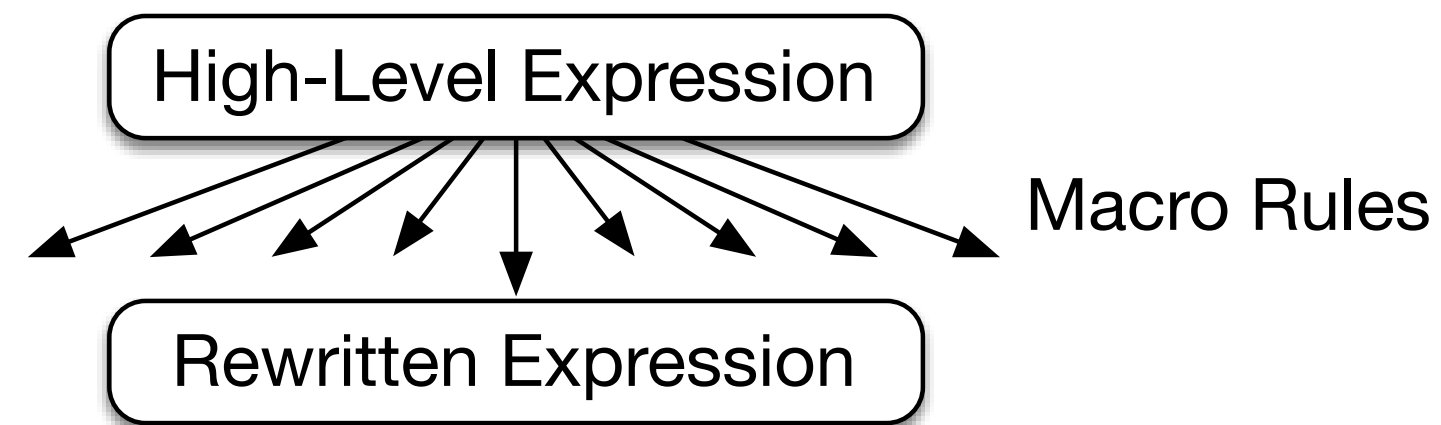
$$\text{Transpose}() \circ \text{Transpose}() \Rightarrow id$$

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

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



Exploration Strategy



1

$$\mathbf{A} * \mathbf{B} =$$

$$\text{Map}(\overrightarrow{\text{row } A} \mapsto$$

$$\text{Map}(\overrightarrow{\text{col } B} \mapsto$$

$$\text{DotProduct}(\overrightarrow{\text{row } A}, \overrightarrow{\text{col } B})$$

$$\text{Transpose}() \$ \mathbf{B}$$

$$) \$ \mathbf{A}$$

1.1

```
TiledMultiply(A, B) =
  Untile() o
  Map(aRows ->
  Map(bCols ->
  Reduce((acc, pairOfTiles) ->
    acc + pairOfTiles..0 * pairOfTiles..1
  ) $ Zip(aRows, bCols)
  ) o Transpose() o Tile(sizeN, sizeK) $ B
  ) o Tile(sizeM, sizeK) $ A
```

1.2

```
BlockedMultiply(A, B) =
  Join() o Map(Transpose()) o
  Map(aRows ->
  Map(bCols ->
  Transpose() o
  Reduce((acc, rowElemPair) ->
    acc + pairOfTiles..0 * pairOfTiles..1
  ) $ Zip(aRows, bCols)
  ) o Transpose() $ B
  ) o Split(blockFactor) $ A
```

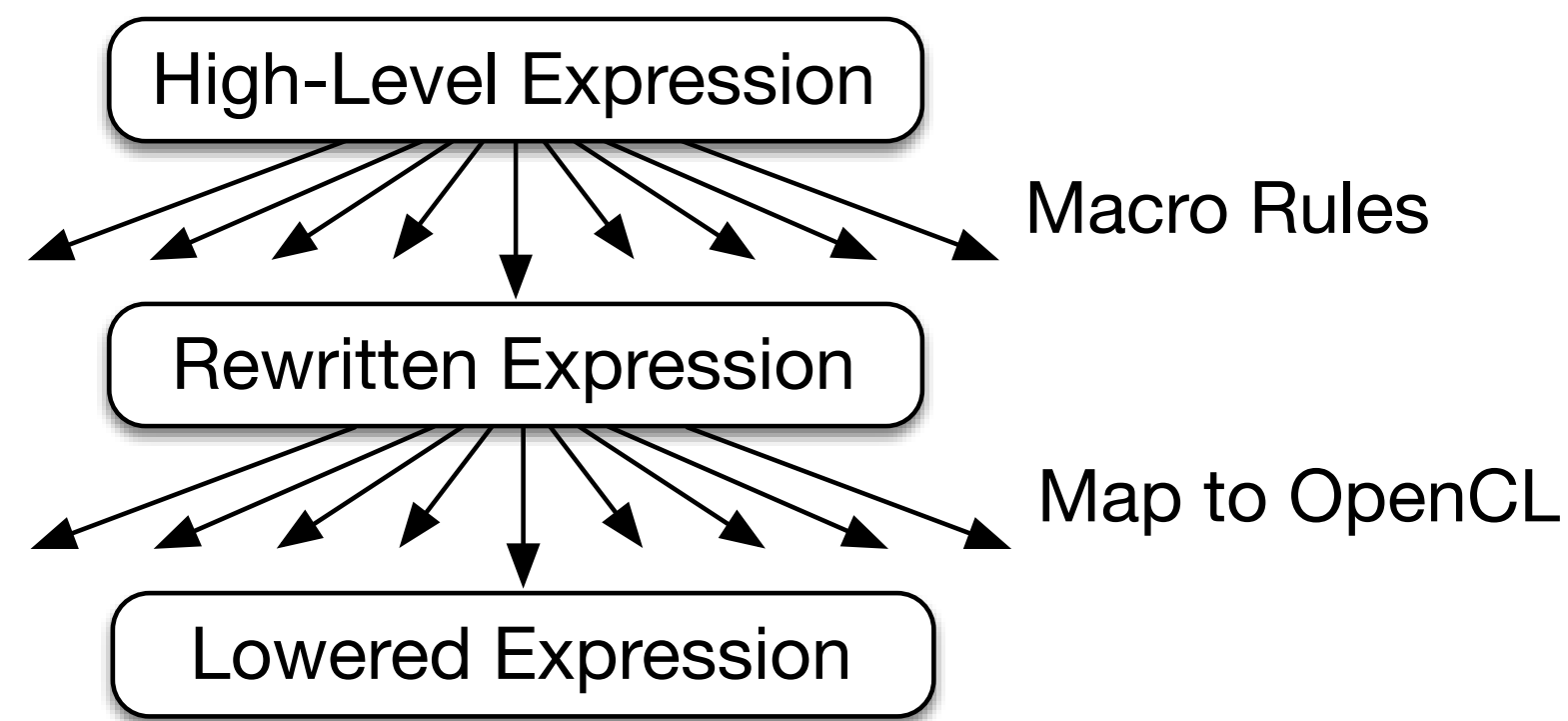
1.3

```
TiledMultiply(A, B) =
  Untile() o
  Map(aRows ->
  Map(bCols ->
  Reduce((acc, pairOfTiles) ->
    acc + pairOfTiles..0 * pairOfTiles..1
  ) $ Zip(aRows, bCols)
  ) o Transpose() o Tile(sizeN, sizeK) $ B
  ) o Tile(sizeM, sizeK) $ A
```

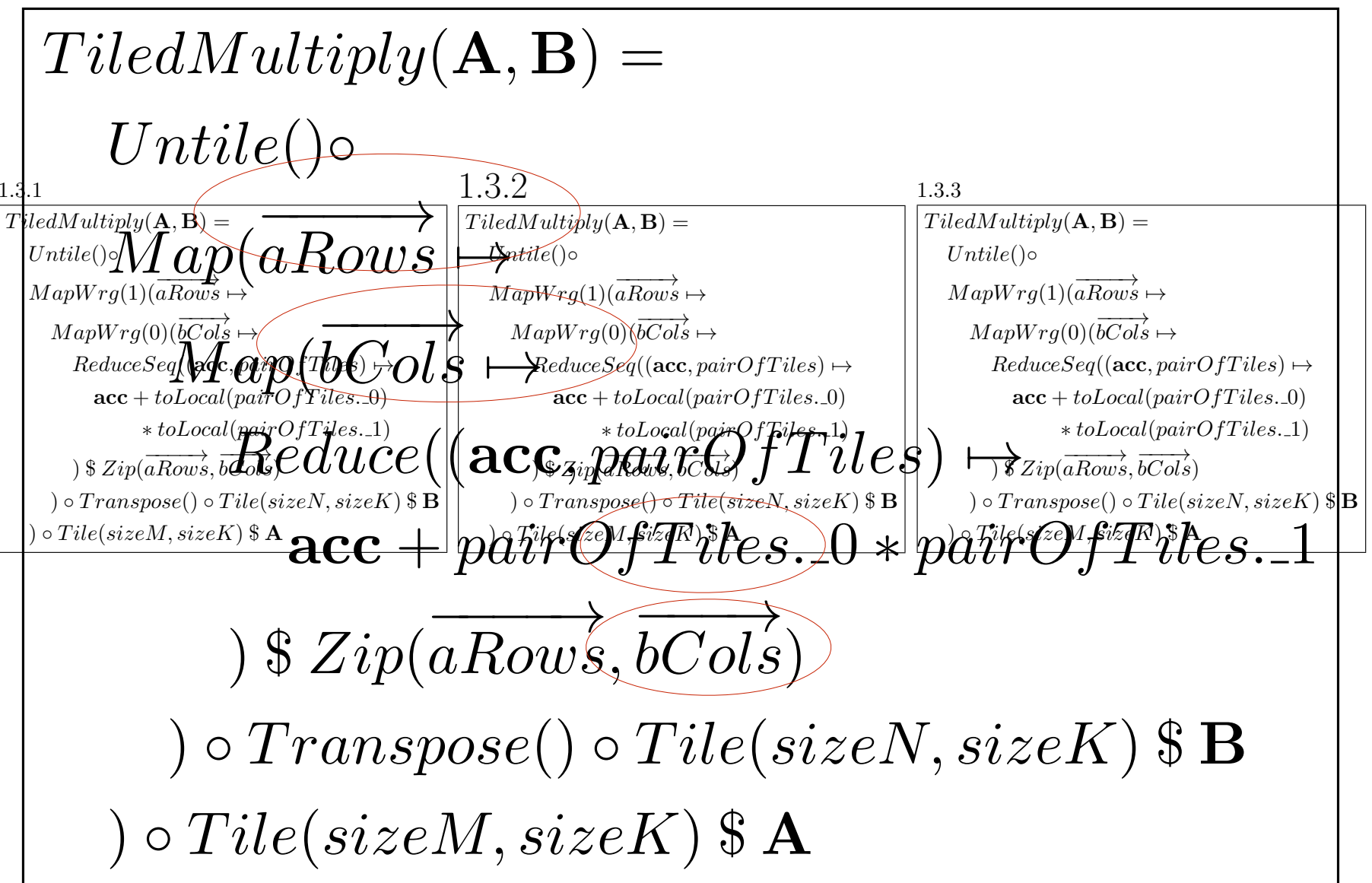
1.4

```
BlockedMultiply(A, B) =
  Join() o Map(Transpose()) o
  Map(aRows ->
  Map(bCols ->
  Transpose() o
  Reduce((acc, rowElemPair) ->
    acc + pairOfTiles..0 * pairOfTiles..1
  ) $ Zip(aRows, bCols)
  ) o Transpose() $ B
  ) o Split(blockFactor) $ A
```

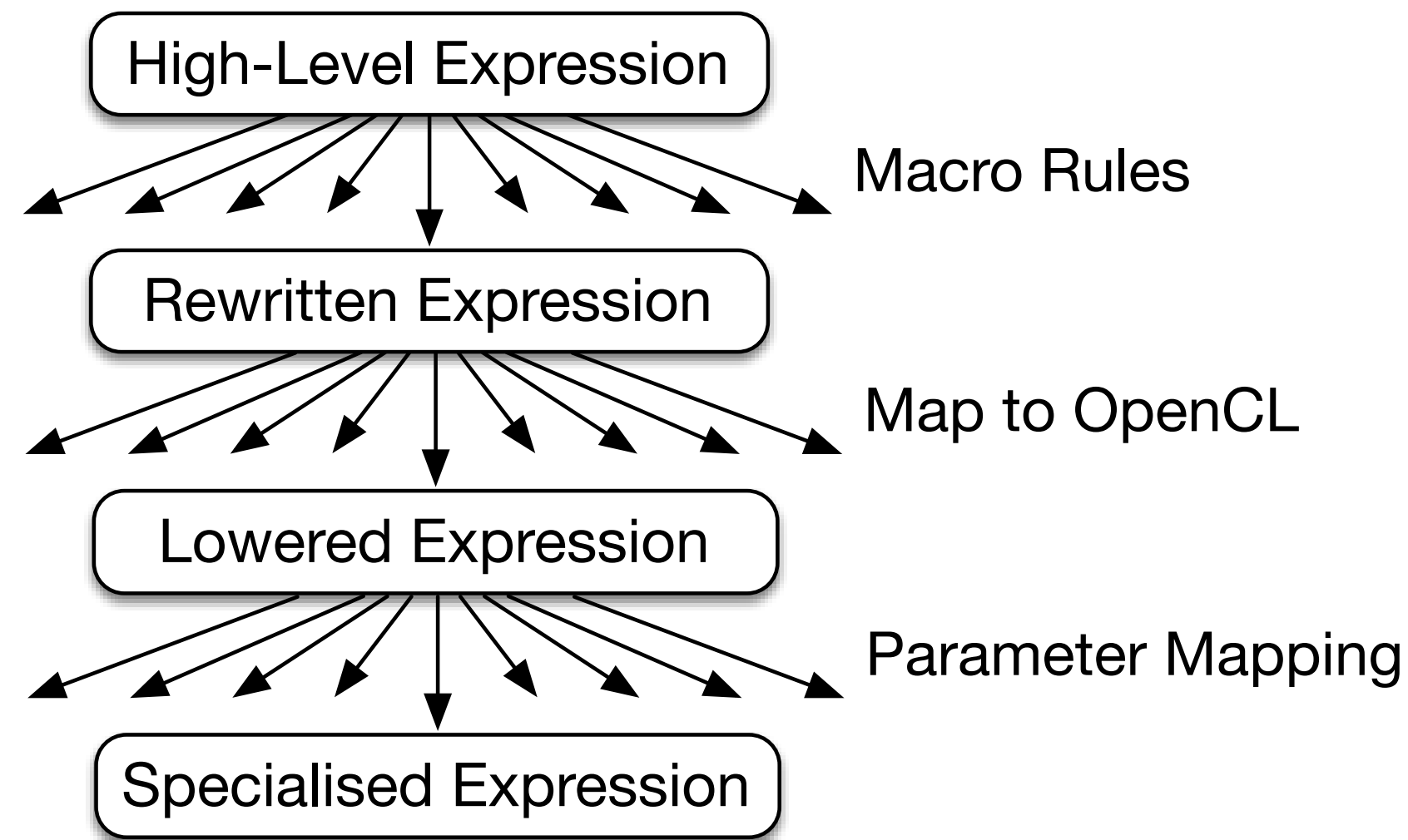
Exploration Strategy



1.3



Exploration Strategy



1.3.2

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(128, 16) \$ B

) ◦ Tile(128, 16) \$ A

1.3.2.1

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(128, 16) \$ B

) ◦ Tile(128, 16) \$ A

1.3.2.2

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(128, 16) \$ B

) ◦ Tile(128, 16) \$ A

1.3.2.3

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(128, 16) \$ B

) ◦ Tile(128, 16) \$ A

1.3.2.4

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(sizeM, sizeK) \$ B

) ◦ Tile(128, 16) \$ A

1.3.2.5

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(sizeN, sizeK) \$ B

) ◦ Tile(128, 16) \$ A

1.3.2.6

TiledMultiply(A, B) =

Untile() ◦

MapWrg(1)(\overrightarrow{aRows} ↦

MapWrg(0)(\overrightarrow{bCols} ↦

ReduceSeq((acc, pairOfTiles) ↦

acc + toLocal(pairOfTiles..0)

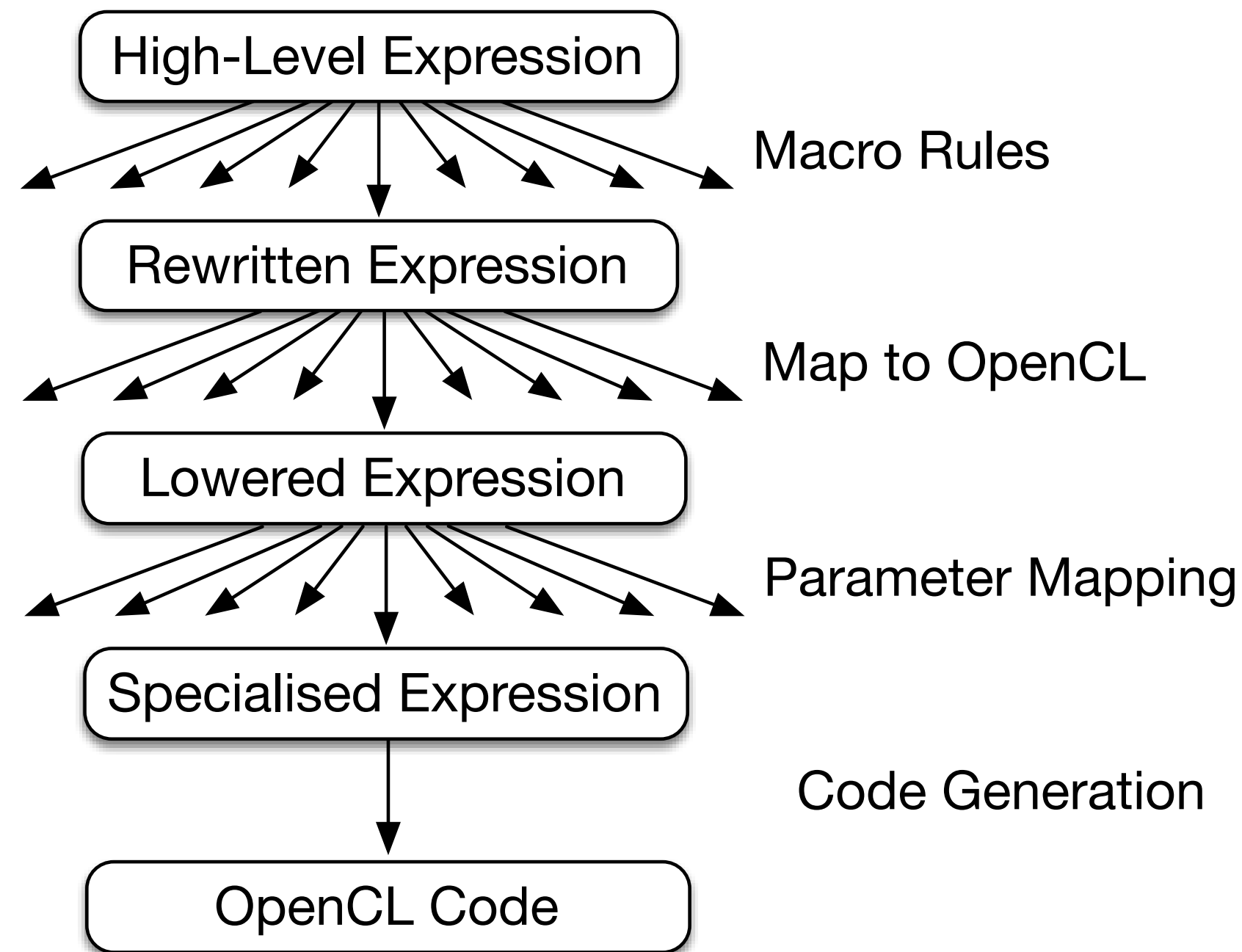
** toLocal(pairOfTiles..1)*

) \$ Zip(\overrightarrow{aRows} , \overrightarrow{bCols})

) ◦ Transpose() ◦ Tile(128, 16) \$ B

) ◦ Tile(128, 16) \$ A

Exploration Strategy



1.3.2.5

```

1 kernel mm_amd_opt(global float *A, B, C,
2   int K,M,N;
3   local float tileA [512], tileB [512];
4
5   private float acc_0; ...; acc_31;
6   private float blockOfA_0; ...; blockOfB_3;
7   private float blockOfA_0; ...; blockOfA_7;
8
9   int lid0 = local_id(0); lid1 = local_id(1);
10  int wid0 = group_id(0); wid1 = group_id(1);
11  MapWrg(1)(aRows) ↦
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      MapWrg(0)(bCols) ↦
15      acc_0 = 0.0f; ...; acc_31 = 0.0f;
16      for (int i=0; i<K/8; i++) {
17        vstore4(vload4(lid1*M/4+2*i*M+16*w1+lid0,A), 16*lid1+lid0, tileA);
18        vstore4(vload4(lid1*N/4+2*i*N+16*w0+lid0,B), 16*lid1+lid0, tileB);
19        barrier (...);
20        ReduceSeq(acc, pairOfTiles) ↦
21        for (int j = 0; j<8; j++) {
22          blockOfA_0 = tileA[0+8*j+lid0]; ...; blockOfA_7 = tileA[7+8*j+lid0];
23          blockOfB_0 = tileB[0+64*j+lid0]; ...; blockOfB_3 = tileB[3+64*j+lid0];
24
25          acc_0 += blockOfA_0 * blockOfB_0; ...; acc_28 += blockOfA_7 * blockOfB_8;
26          acc_1 += blockOfA_0 * blockOfB_1; ...; acc_29 += blockOfA_7 * blockOfB_9;
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      }
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+0*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+0*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  } } }
  
```

$\text{Tile}(128, 16) \$ B$
 $\text{Tile}(128, 16) \$ A$

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

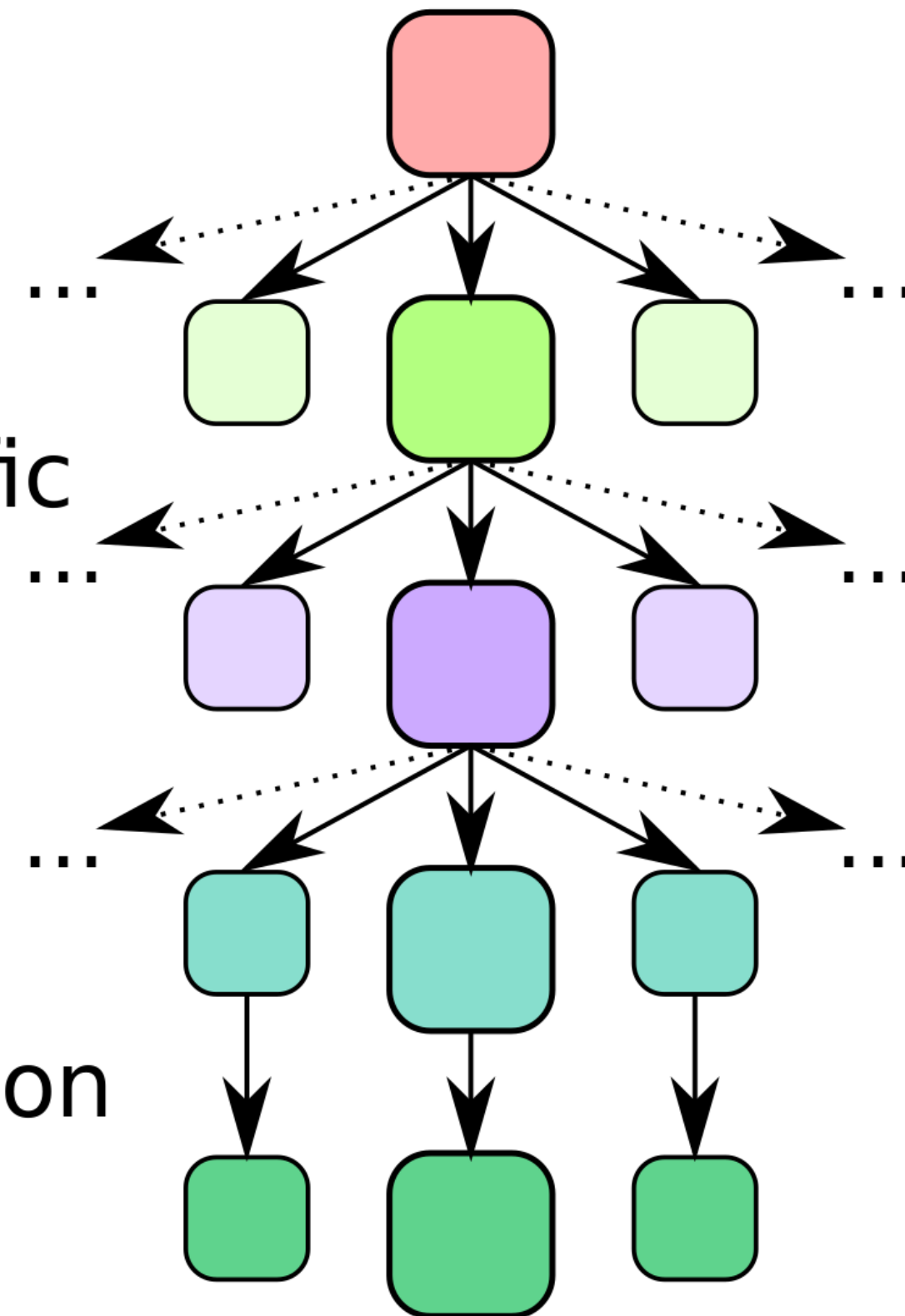
Phases:

Algorithmic Exploration

OpenCL specific Exploration

Parameter Exploration

Code Generation



Program Variants:

High-Level Program 1

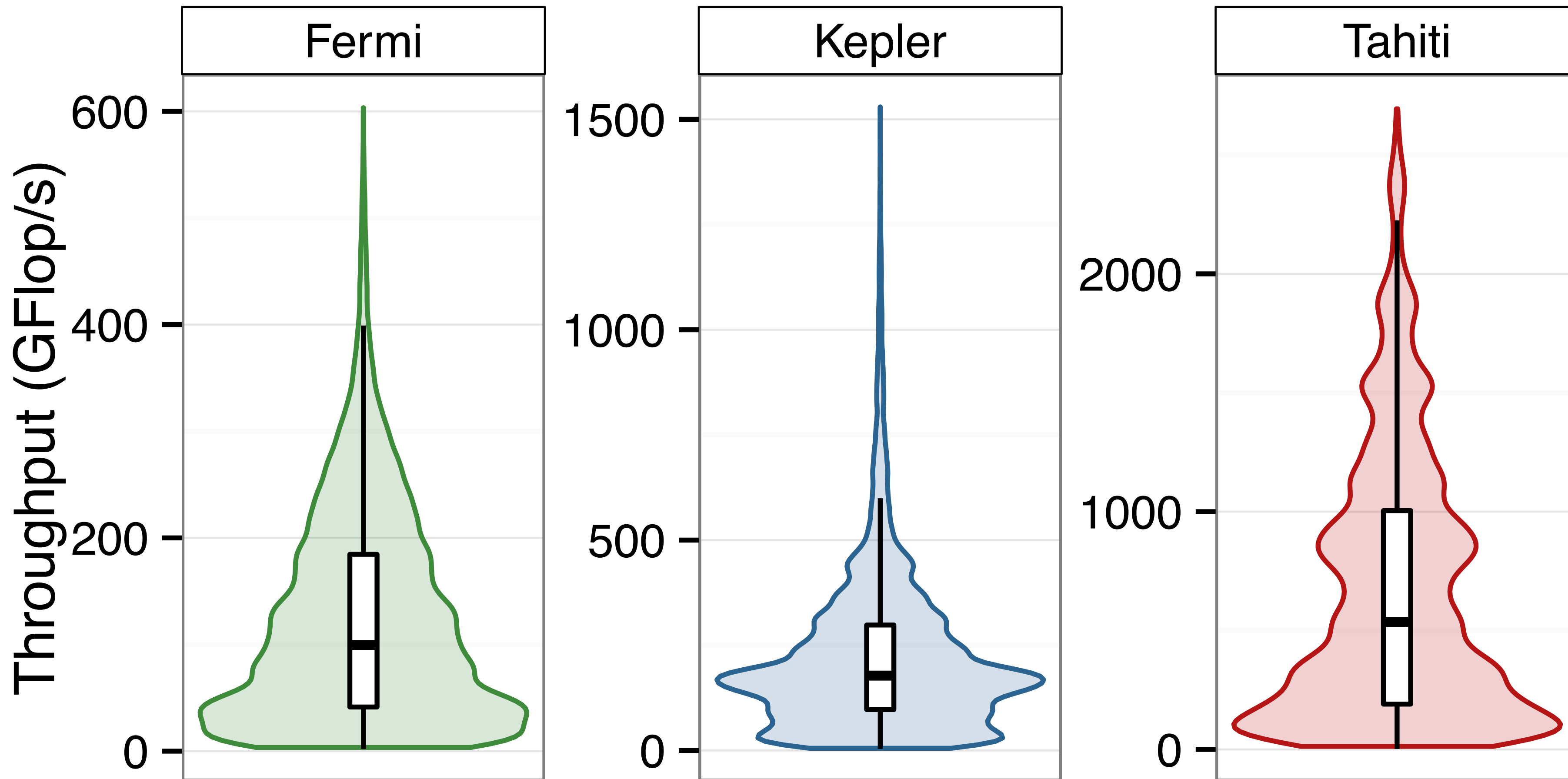
Algorithmic Rewritten Program 8

OpenCL Specific Program 760

Fully Specialized Program 46,000

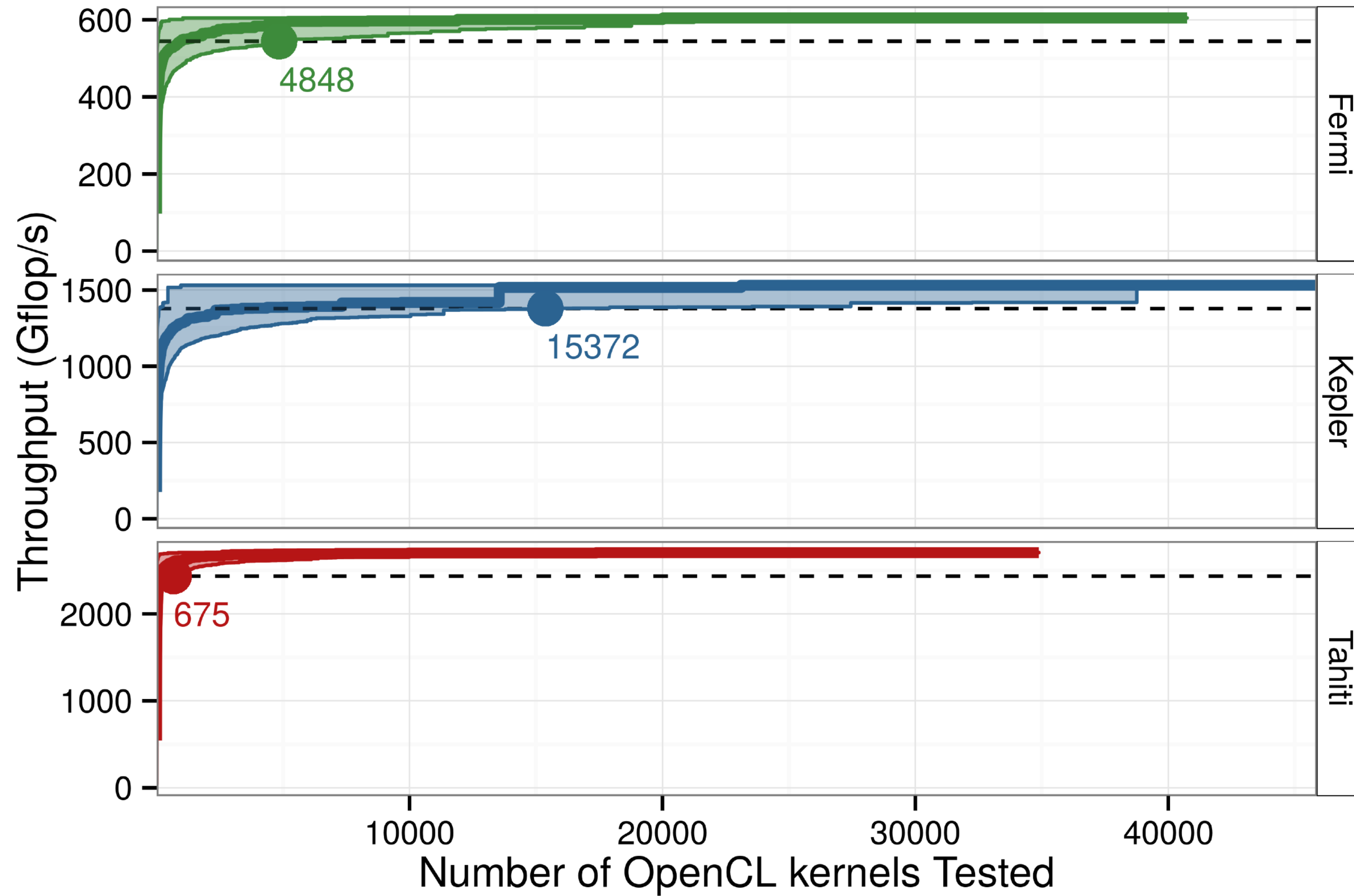
OpenCL Code 46,000

Exploration Space for Matrix Multiplication



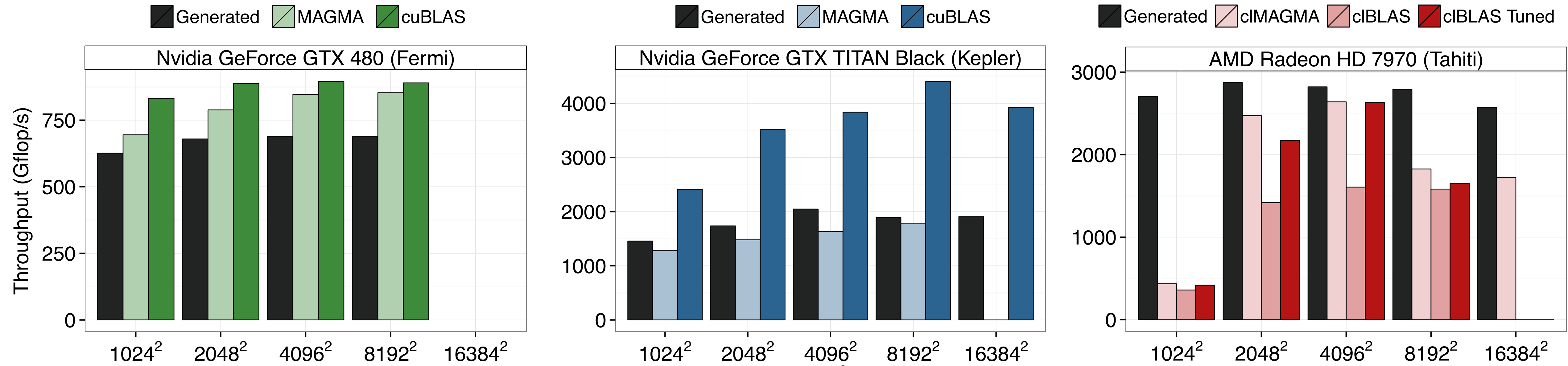
Only few OpenCL kernel with very good performance

Performance Evolution for Randomised Search



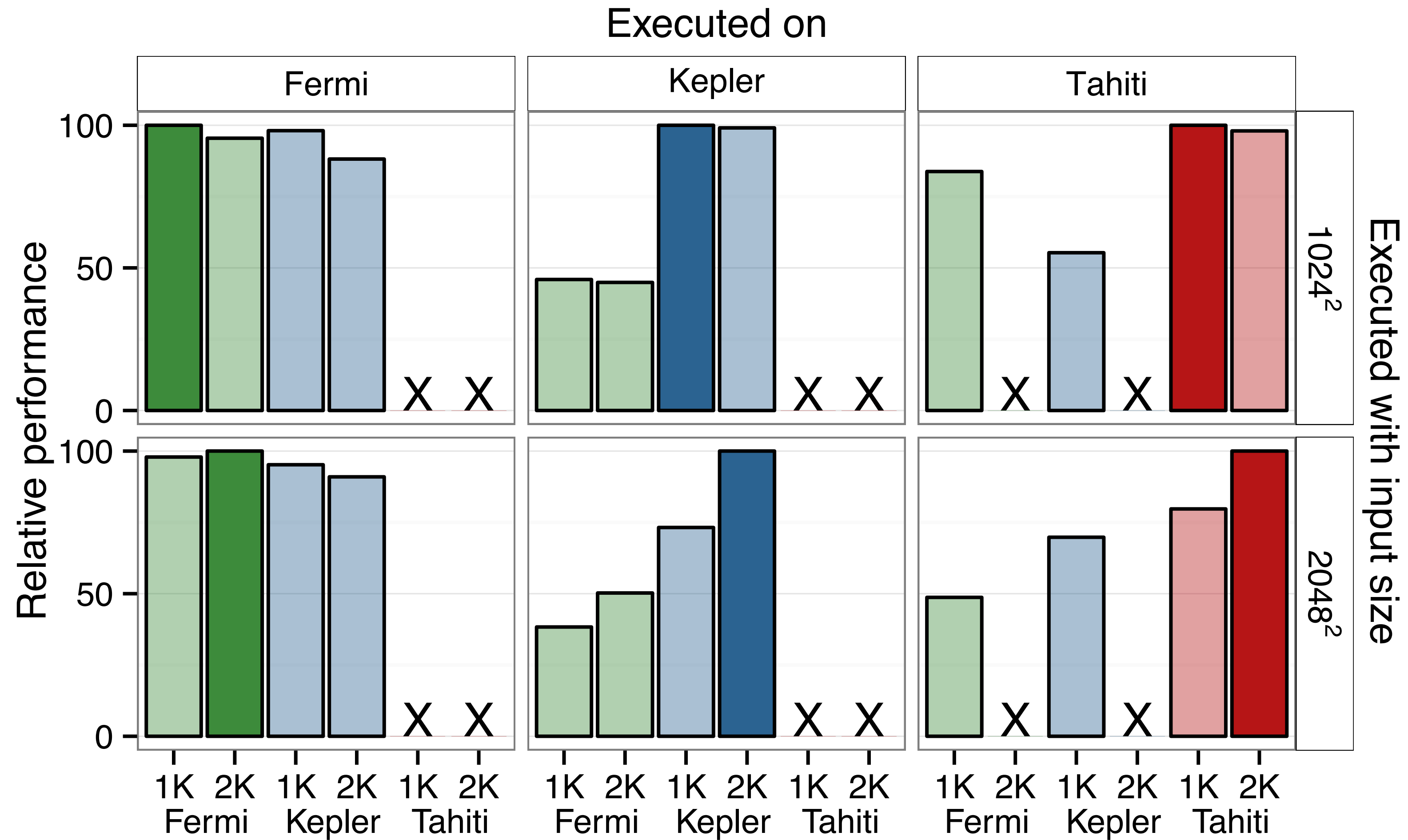
Even with a simple random search strategy one can expect to find a good performing kernel quickly

Performance Results Matrix Multiplication



Performance close or better than hand-tuned MAGMA library

Performance Portability Matrix Multiplication



The six specialized OpenCL kernels

Generated kernels are specialised for device and input size

Desktop GPUs vs. Mobile GPU

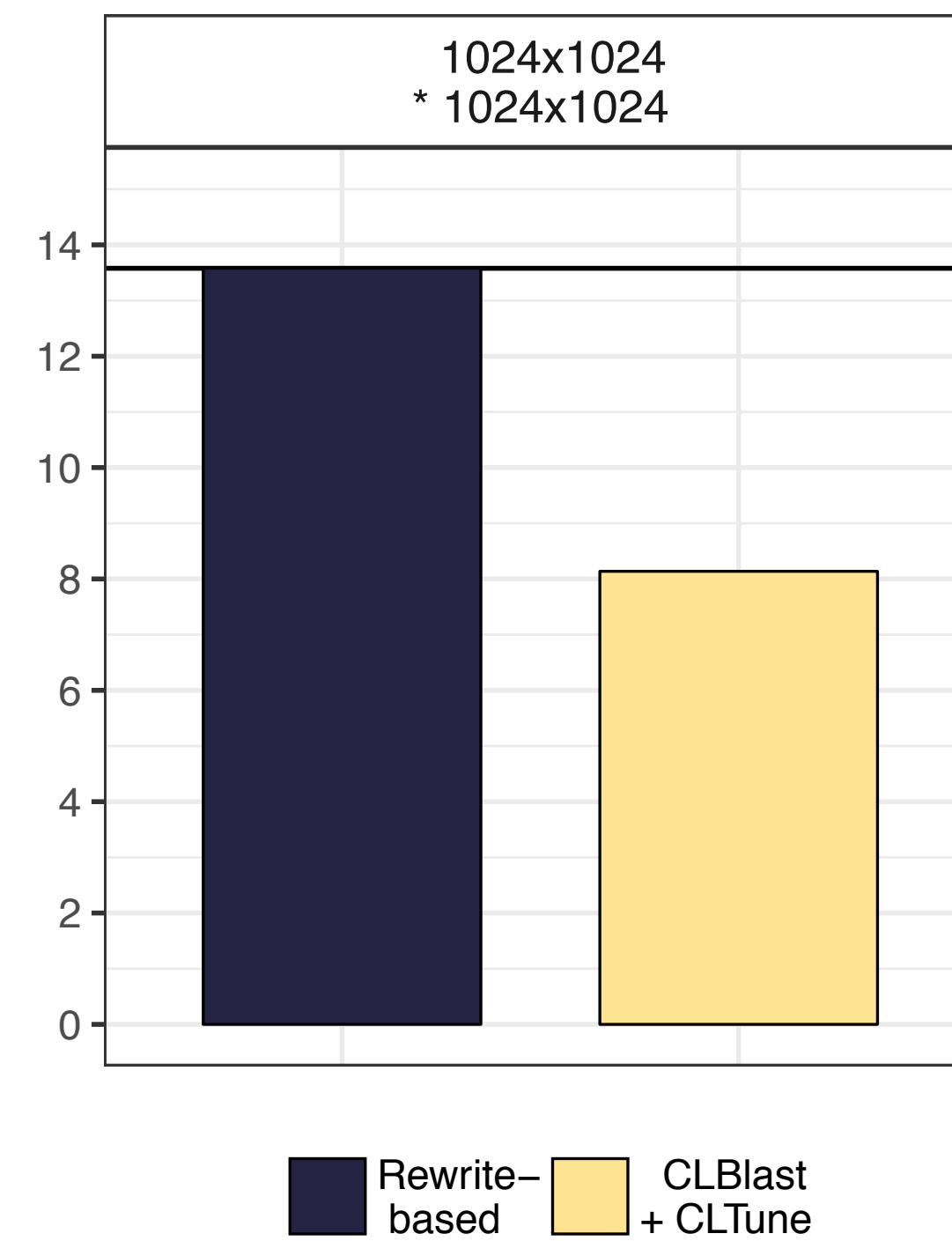
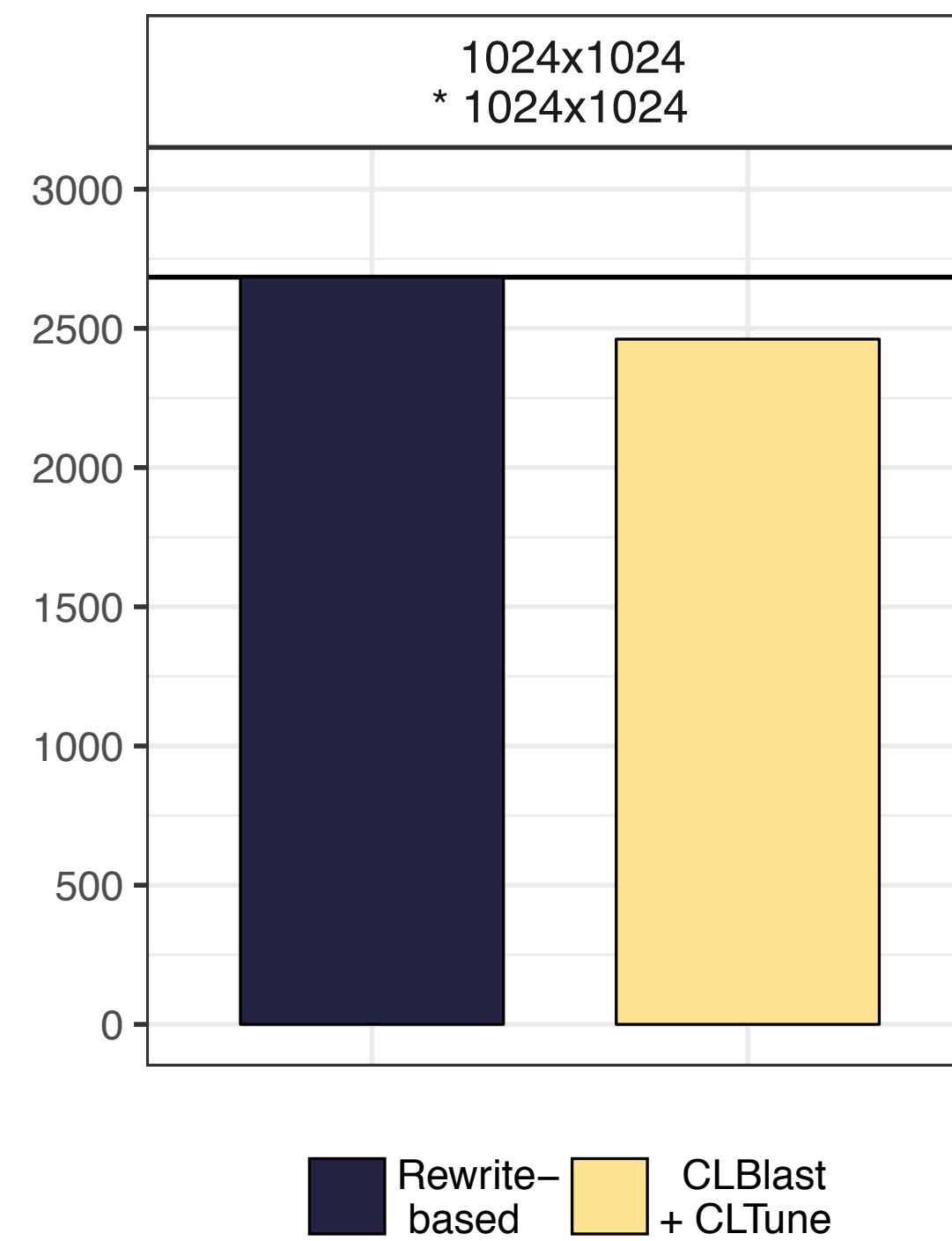
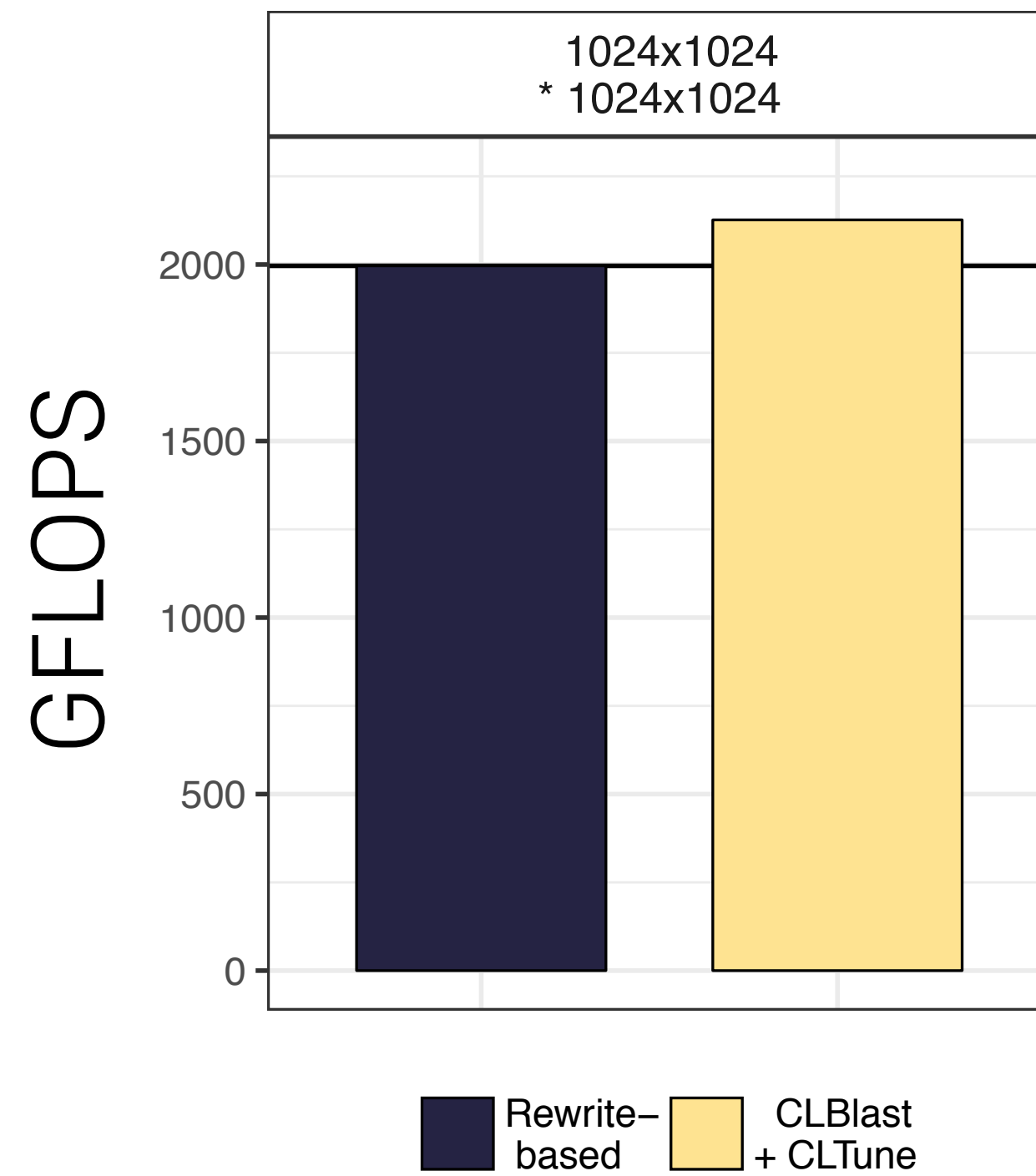
Desktop GPUs

Mobile GPU

Nvidia GeForce GTX Titan Black

AMD Radeon HD 7970

ARM Mali-T628 MP6



Performance portable even for mobile GPU device!



The LIFT Team



University
of Glasgow



THE UNIVERSITY
of EDINBURGH





Identify *hidden parallelism* in LIFT programs

Parallelising non-associative reductions

Federico Pizzuti
PhD Student
University of Edinburgh

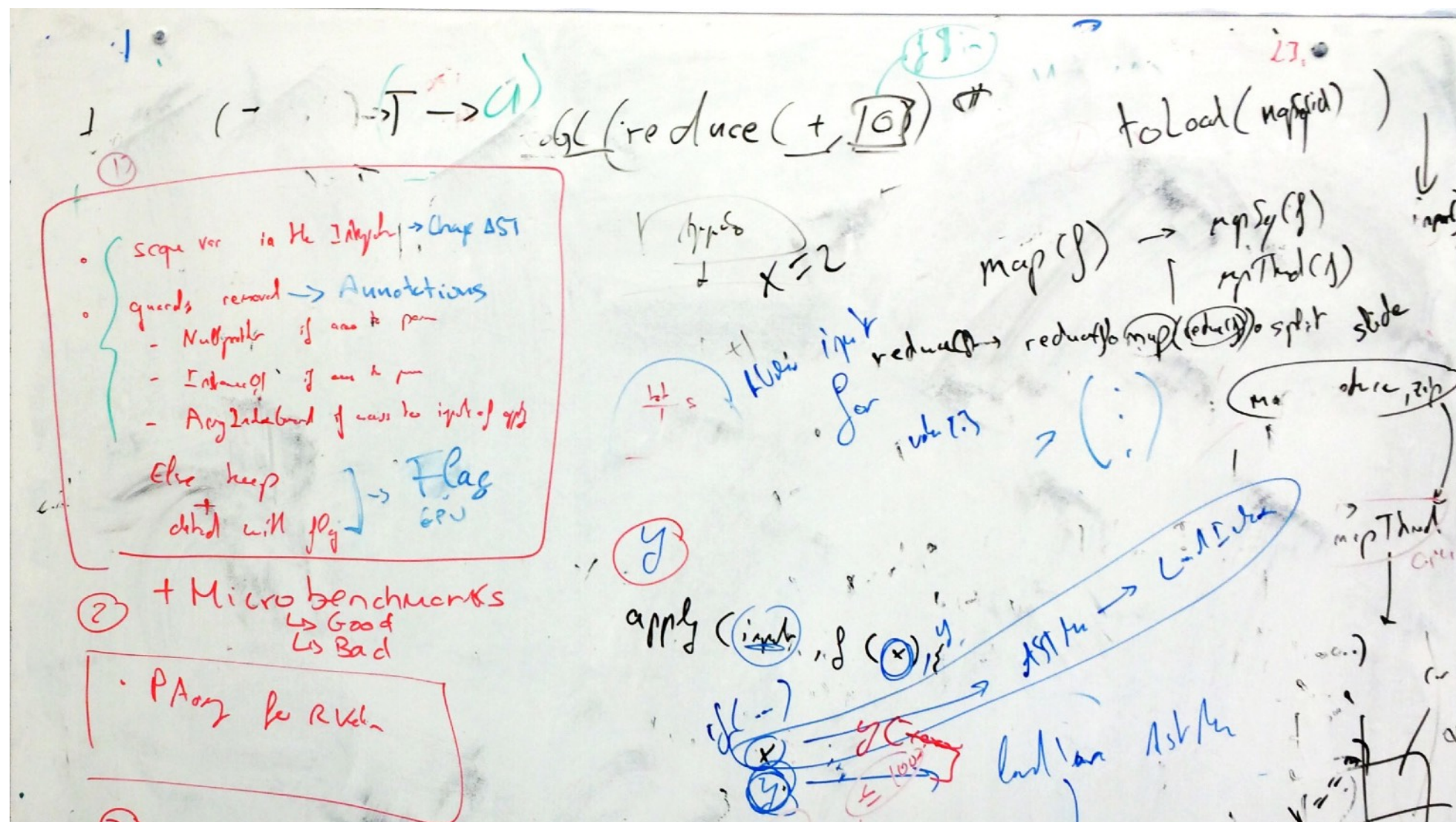
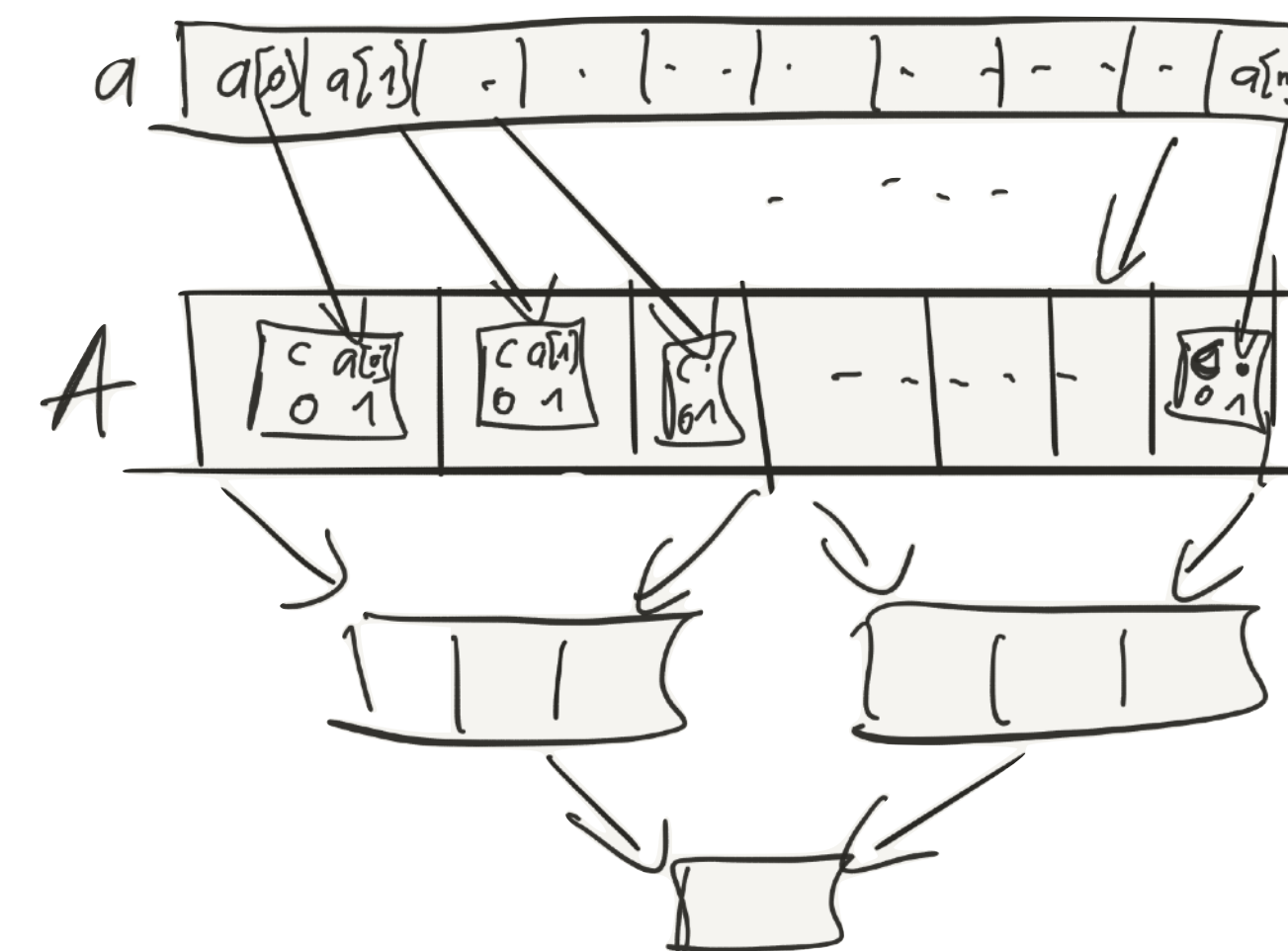
Key idea: Rearrange data as matrices to exploit associative matrix multiplication

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



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

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





Adam Harries
PhD Student
University of Edinburgh

Graph Algorithms via Sparse Linear Algebra in LIFT

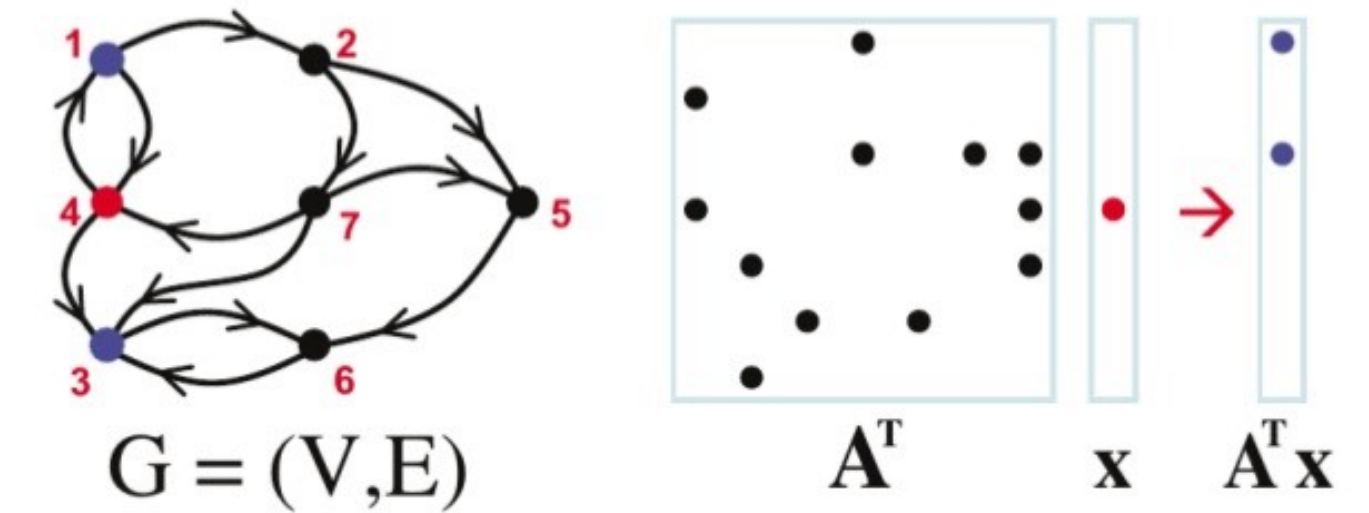
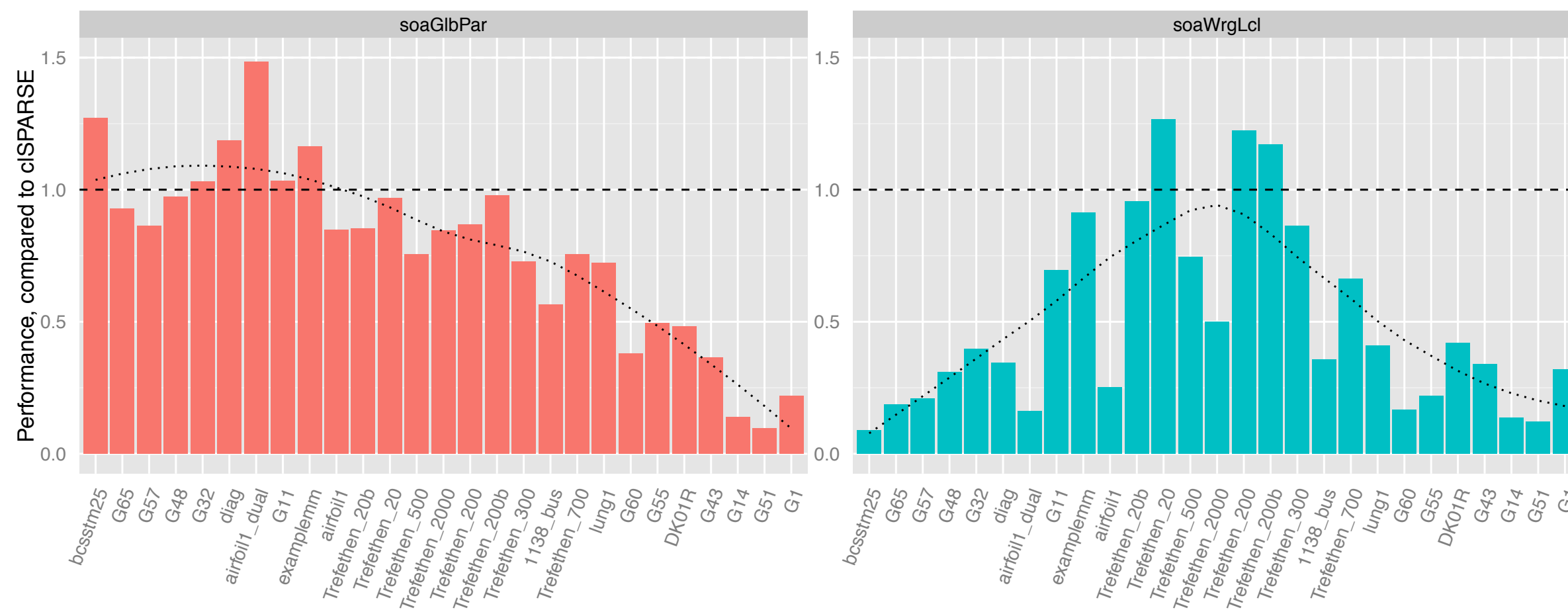


Image credit: [Kepner2011]

```
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))) )
```

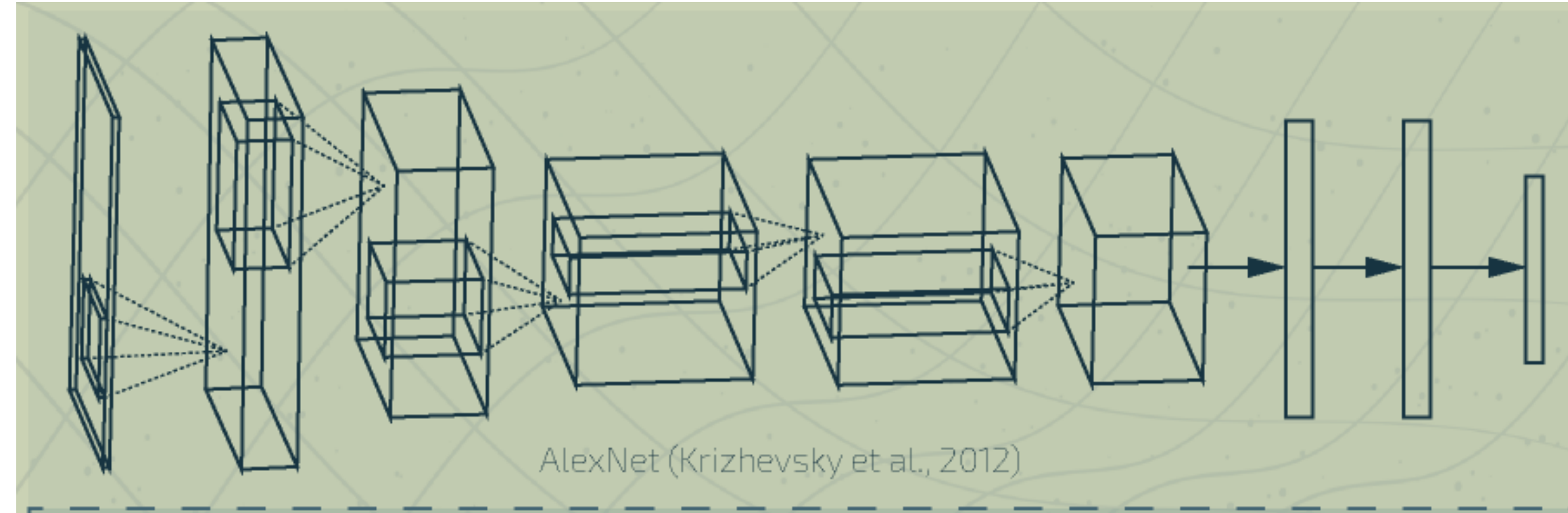


Differently
optimised kernels
for different inputs



Naums Mogers
PhD Student
University of Edinburgh

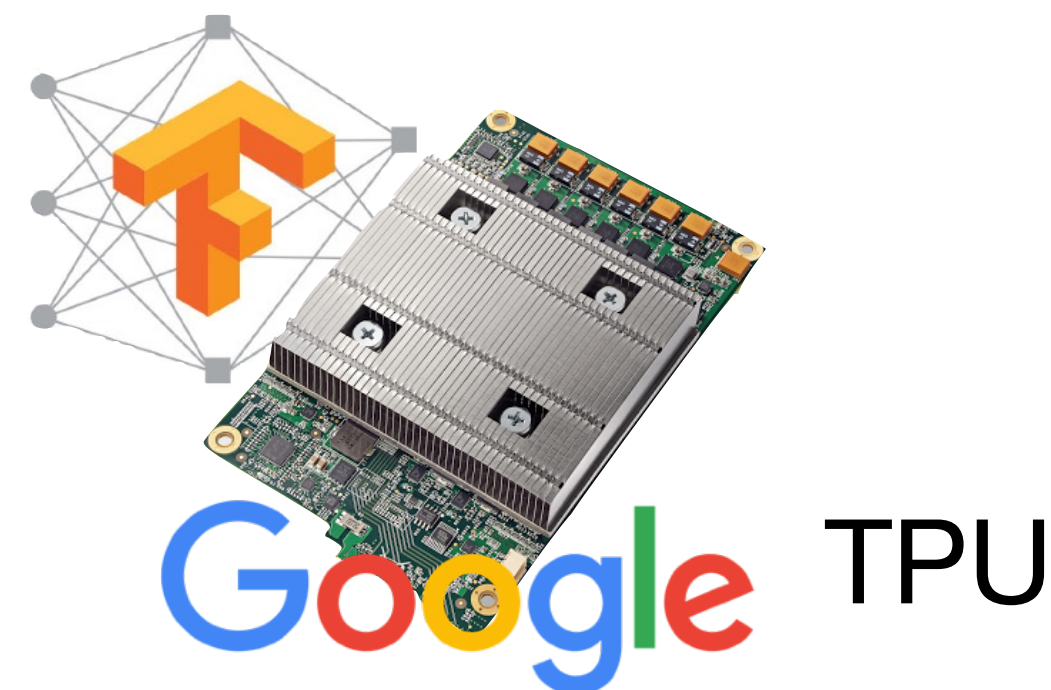
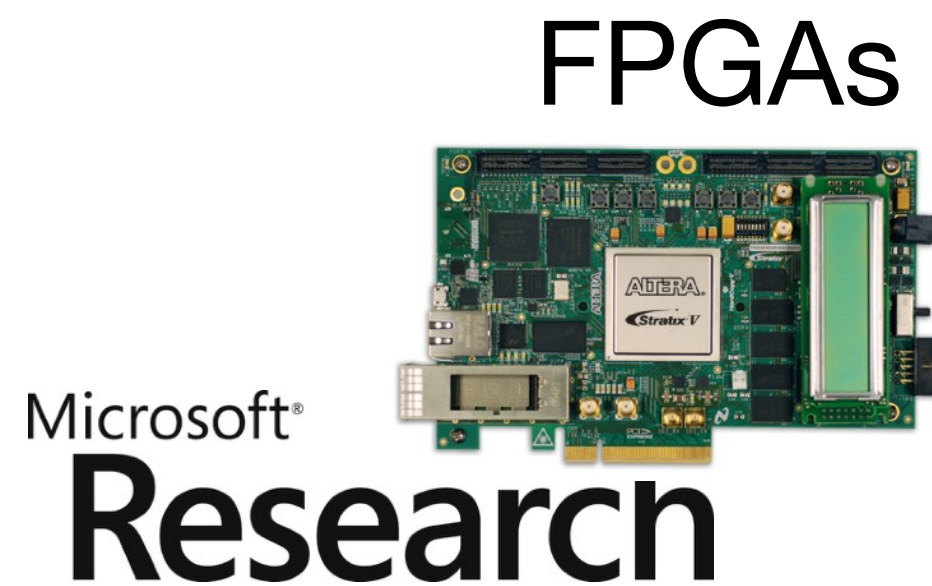
Optimising Deep Learning with LIFT



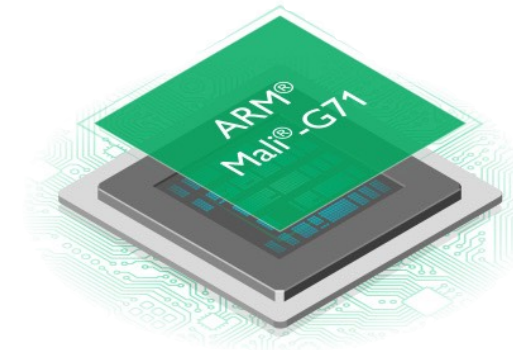
Express layers with LIFT primitives

```
fully_connected(f, weights, bias, inputs) :=  
  Map((neuron_weights, neuron_bias) -> f() o Reduce(add, neuron_bias) o  
  Map(mult) $ Zip(inputs, neuron_weights)) $ Zip(weights, bias)
```

Optimise individual layers and across layers via rewrites



Low Power Devices



Stencil Computations in LIFT



Bastian Hagedorn
PhD Student
University of Münster



Larisa Stoltzfus
PhD Student
University of Edinburgh

Express Stencil with Skeletons

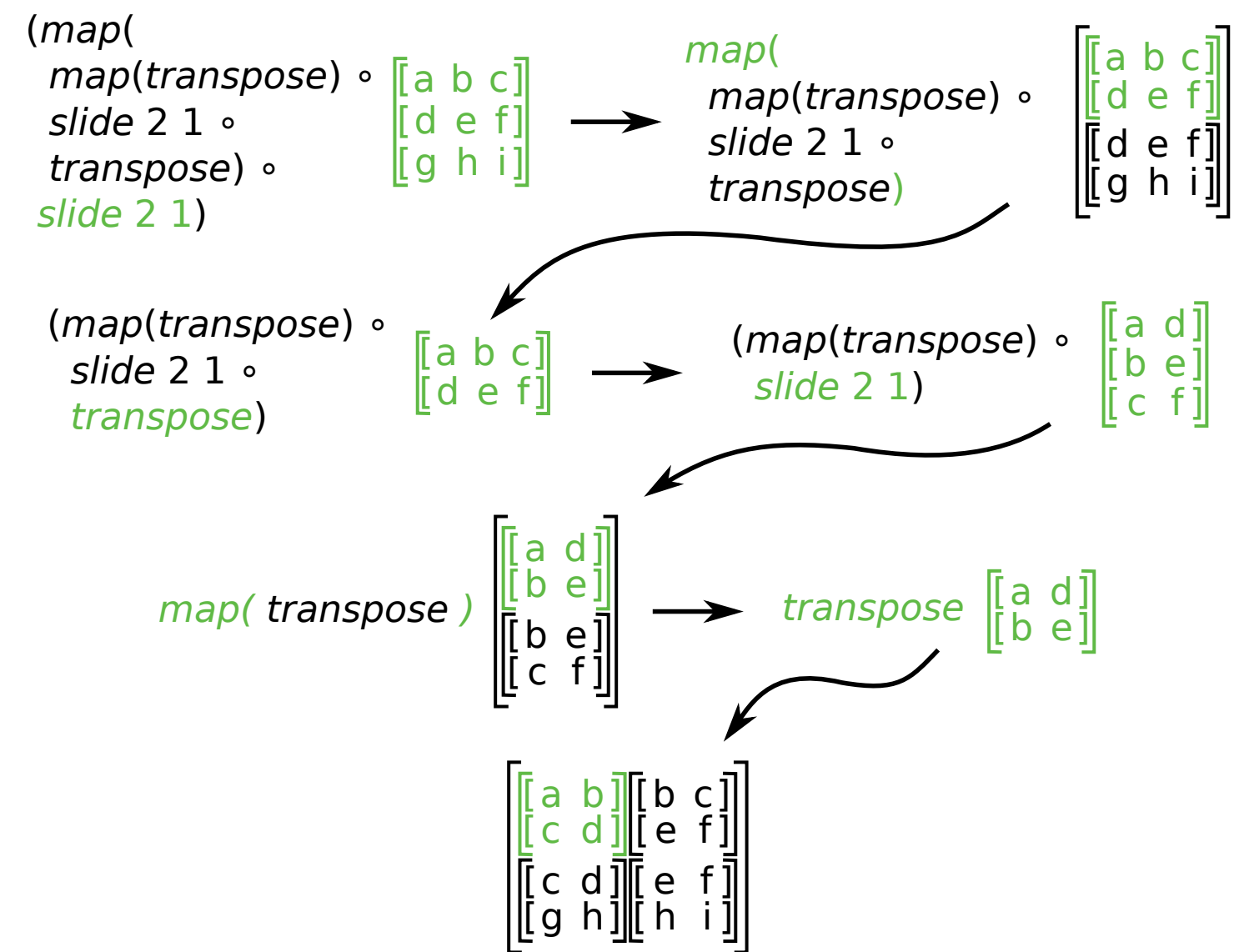
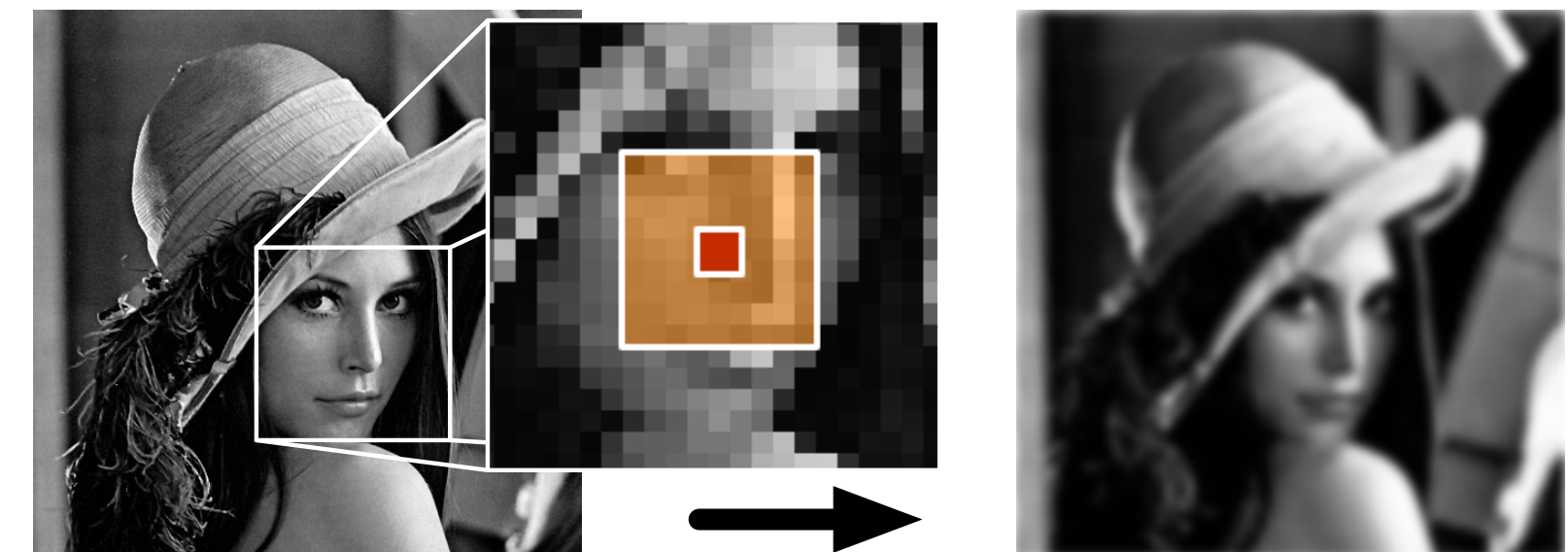
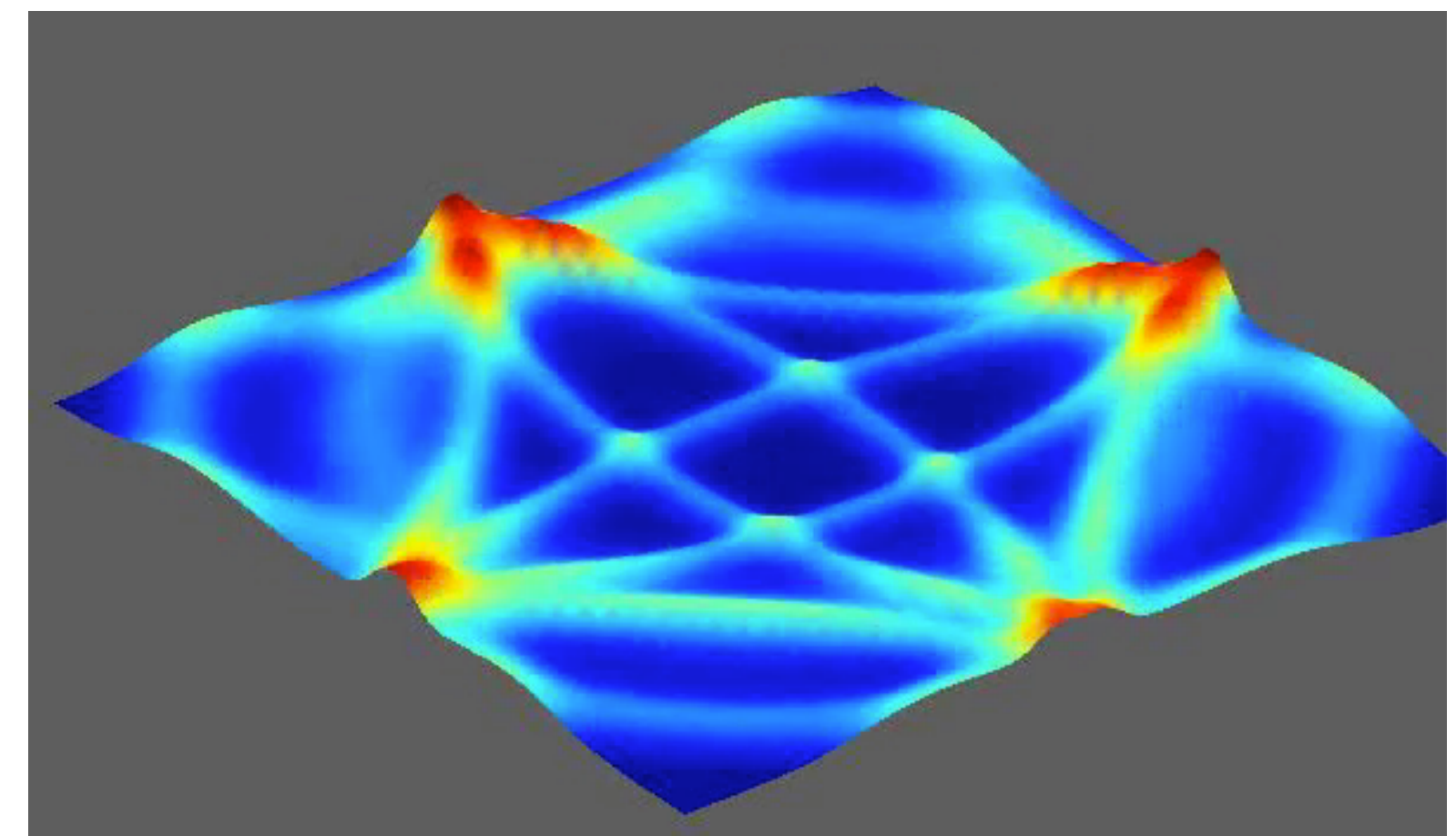


Image Processing

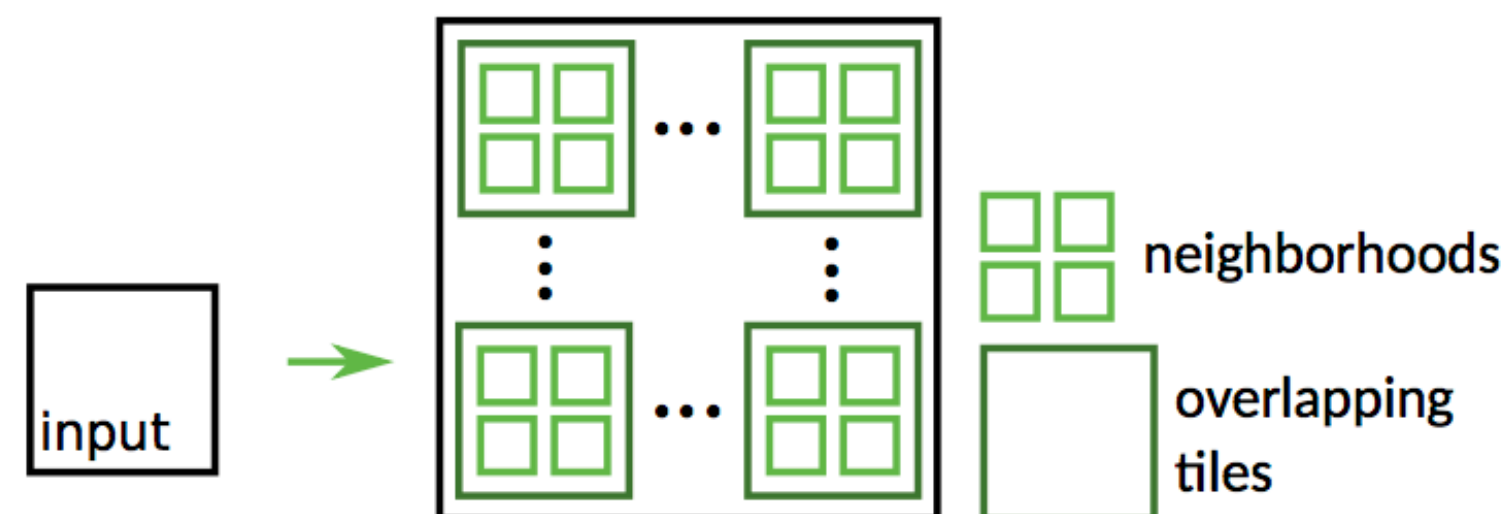


Acoustics Simulation



[Video](#)

Explore optimisations as rewrites





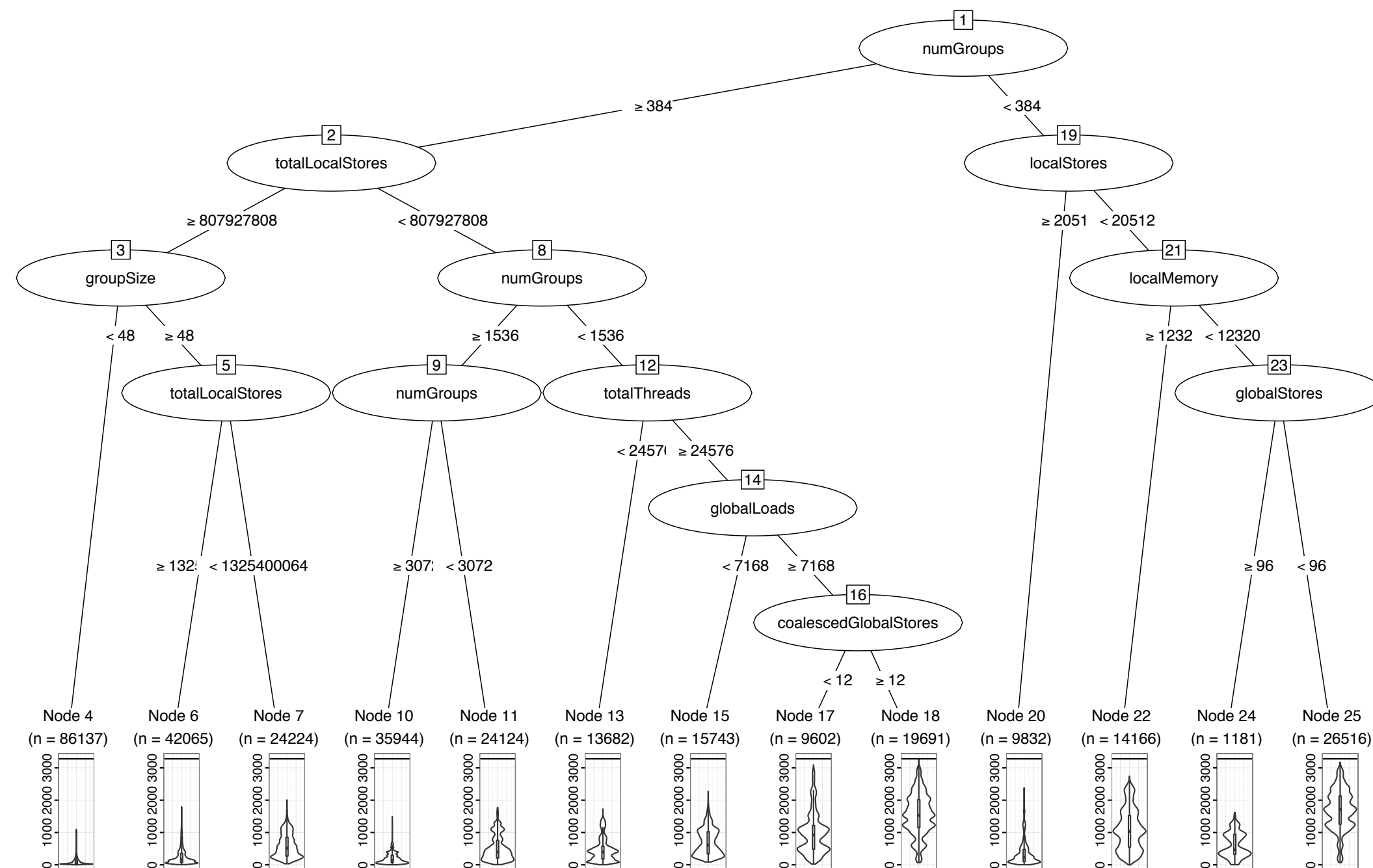
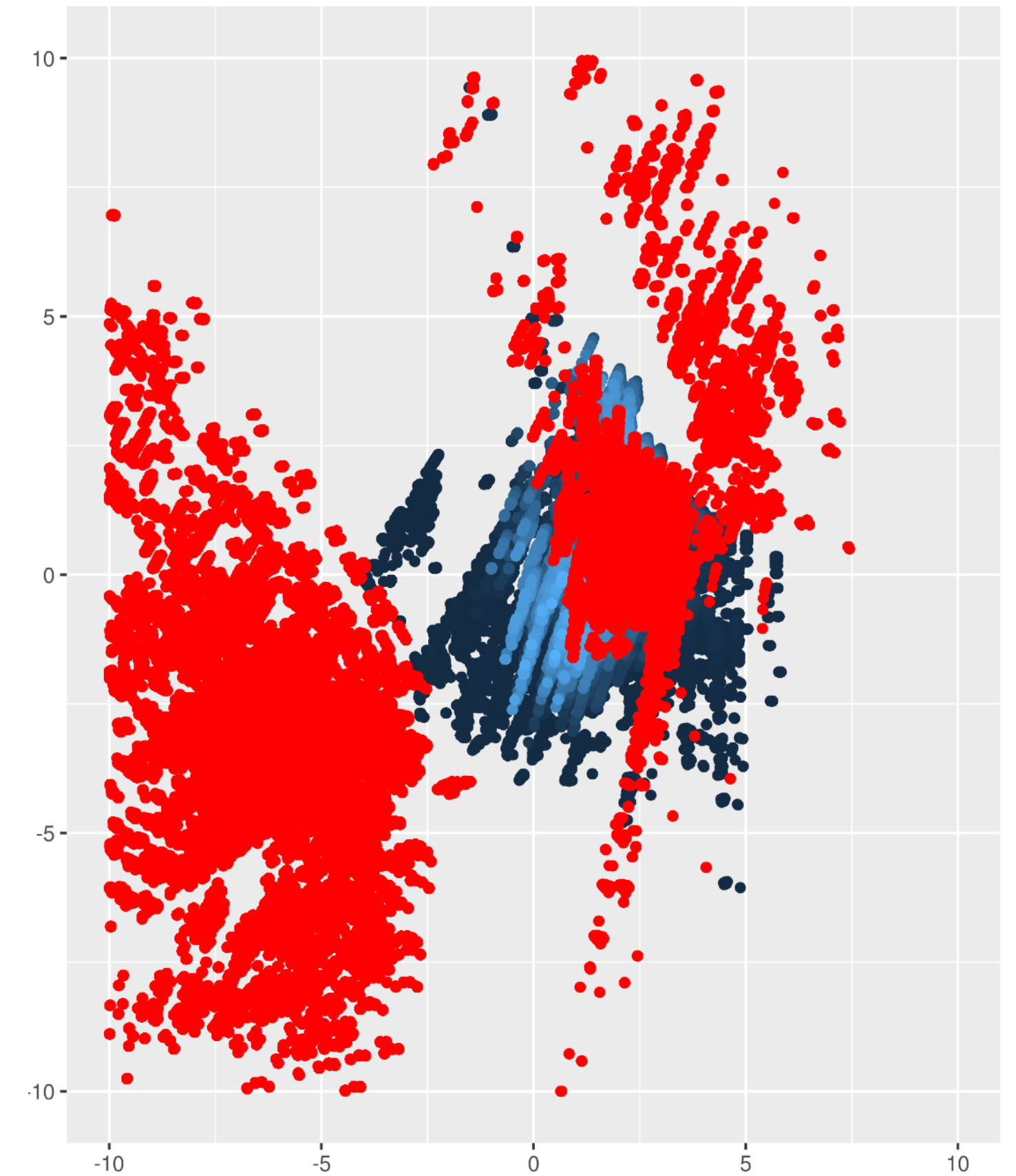
Toomas Remmelg
PhD Student
University of Edinburgh

Performance Modeling of LIFT Programs

```
until ◦ map(λ rowOfTilesA .
map(λ colOfTilesB .
toGlobal(copy2D(tileA, tileB)))
reduce(λ (tileAcc, (tileA, tileB)) .
map(map(λ as ◦ zip(tileAcc ◦
map(λ as .
map(λ bs .
reduce(λ (acc, (as, bs)) .
toLocal(copy2D(tileA)))
toLocal(copy2D(tileA)))
, zip(rowOfTilesA, colOfTilesB))
) ◦ tile(m, k, transpose
) ◦ tile(n, k, A)
```

Extract Features

Performance Model



Predictions
used to drive the
rewrite process

Data Parallel Idealised Algol as new foundation for LIFT

Collaboration with Bob Atkey (Strathclyde) and Sam Lindley (Edinburgh)

- So far LIFT rewrites only functional expressions
- Data Parallel Idealised Algol (DPIA) combines functional and imperative constructs
- Allows formal translation of functional programs into efficient parallel imperative programs
- Types separate *expressions*, *acceptors*, and *commands*:
 - *Expressions* are purely functional computations
 - *Acceptors* describe modifiable locations in memory (\cong l-values in C)
 - *Commands* are imperative actions modifying memory

Draft Paper at: <https://bentnib.org/dpia.html>

Functional

```
reduce (+) 0 (map ( $\lambda x$ . fst x * snd x) (zip xs ys))
```

```
asScalar4 (join (mapWorkgroup ( $\lambda z_{s1}$ . mapLocal ( $\lambda z_{s2}$ . reduce ( $\lambda x a$ . (fst x * snd x) + a) 0 (split 8192 zs2)) zs1) (split 8192 (zip (asVector4 xs) (asVector4 ys))))))
```

Imperative

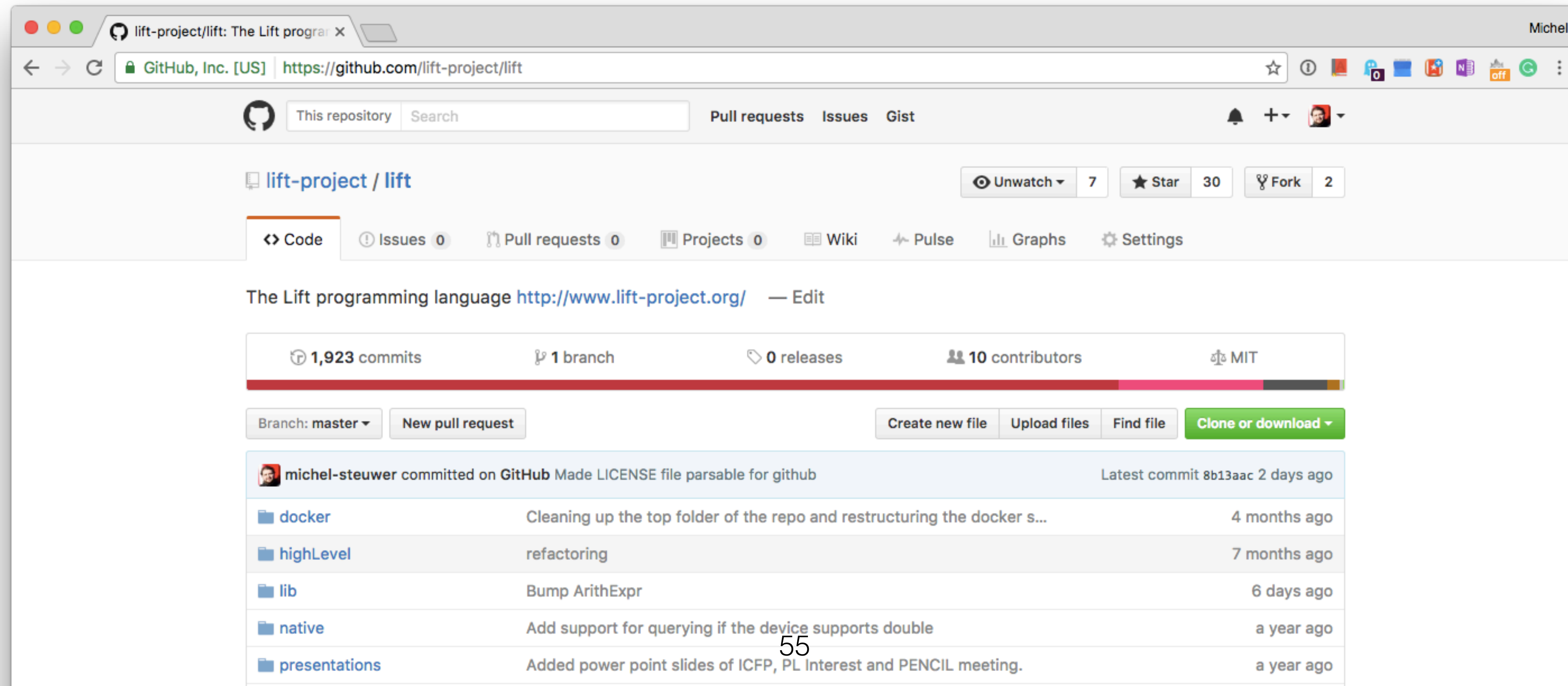
```
parforWorkgroup (N/8192) (joinAcc (N/8192) 64 (asScalarAcc4 (N/128) out)) ( $\lambda$  gid o.  
  parforLocal 64 o ( $\lambda$  lid o.  
    newPrivate num<4> accum.  
    accum.1 := 0;  
    for 2048 ( $\lambda$  i.  
      accum.1 := accum.2 +  
        (fst (idx (idx (split 2048 (idx (split (8192 * 4) (zip (asVector4 xs) (asVector4 ys))) gid)) lid) i)) *  
        (snd (idx (idx (split 2048 (idx (split (8192 * 4) (zip (asVector4 xs) (asVector4 ys))) gid)) lid) i)) );  
    out := accum.2 ))
```

```
kernel void KERNEL(global float *out, const global float *restrict xs,  
                   const global float *restrict ys, int N) {  
  for (int g_id = get_group_id(0); g_id < N / 8192; g_id += get_num_groups(0)) {  
    for (int l_id = get_local_id(0); l_id < 64; l_id += get_local_size(0)) {  
      float4 accum;  
      accum = (float4)(0.0, 0.0, 0.0, 0.0);  
      for (int i = 0; i < 2048; i += 1) {  
        accum = (accum +  
                 (vload4(((2048 * l_id) + (8192 * 4 * g_id) + i), xs) *  
                  vload4(((2048 * l_id) + (8192 * 4 * g_id) + i), ys)));  
      }  
      vstore4(accum, ((64 * g_id) + l_id), out); } } }
```

LIFT is Open-Source Software

`http://www.lift-project.org/`

`https://github.com/lift-project/lift`



The screenshot shows the GitHub repository page for lift-project/lift. The browser address bar displays the URL `https://github.com/lift-project/lift`. The repository name is lift-project / lift, with 7 Unwatch, 30 Star, and 2 Fork actions. The repository description is "The Lift programming language <http://www.lift-project.org/>". The repository statistics show 1,923 commits, 1 branch, 0 releases, 10 contributors, and MIT license. The commit history table is as follows:

Commit	Message	Time
8b13aac	Made LICENSE file parsable for github	2 days ago
	Cleaning up the top folder of the repo and restructuring the docker s...	4 months ago
	refactoring	7 months ago
	Bump ArithExpr	6 days ago
	Add support for querying if the device supports double	a year ago
	Added power point slides of ICFP, PL Interest and PENCIL meeting.	a year ago



University
of Glasgow

The LIFT Project

Performance Portable Parallel Code Generation via Rewrite Rules

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

www.lift-project.org  [@LIFTlang](https://twitter.com/LIFTlang)

**INSPIRING
PEOPLE**

#UofGWorldChangers

