THE UNIVERSITY *of* EDINBURGH

# informatics

- Largest Informatics Department in the UK:
  - > 500 academic and research staff + PhD students
- Overall 6 Research Institutes

2 particular relevant for the topic of the talk:

- ICSA — Institute for Computing Systems Architecture
  - Compiler & Architecture
  - Parallel Computing
  - …
- LFCS — Laboratory for Foundations of Computer Science
  - Programming Languages and Foundations
  - Software Engineering
  - …

# The *lift* Project:

# Performance Portability via Rewrite Rules

http://www.lift-project.org

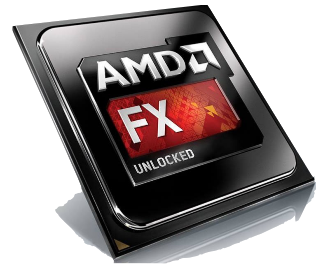

Michel Steuwer

http://homepages.inf.ed.ac.uk/msteuwer/

THE UNIVERSITY of EDINBURGH

# The Problem(s)

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



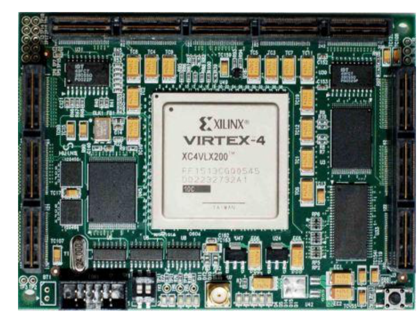


CPU

GPU

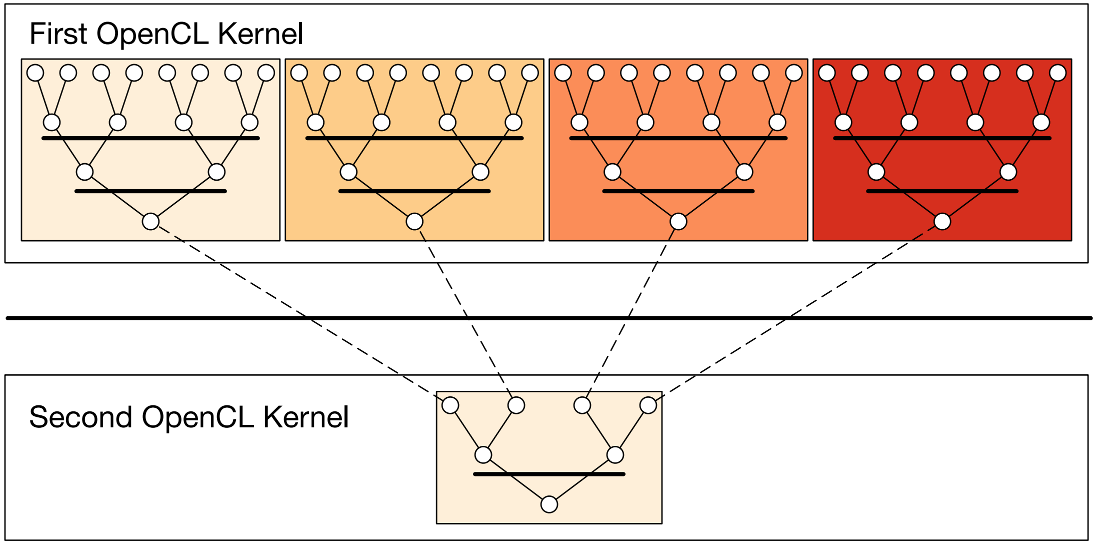

Accelerator

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];
}
```

# OpenCL Programming Model

```
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];
}
```

- Multiple *work-items* (threads) execute the same *kernel* function
- *Work-items* are organised for execution in *work-groups*

# OpenCL Programming Model

```
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];
}
```

- Multiple *work-items* (threads) execute the same *kernel* function
- *Work-items* are organised for execution in *work-groups*

# 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]; }
```

# Case Study Conclusions

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

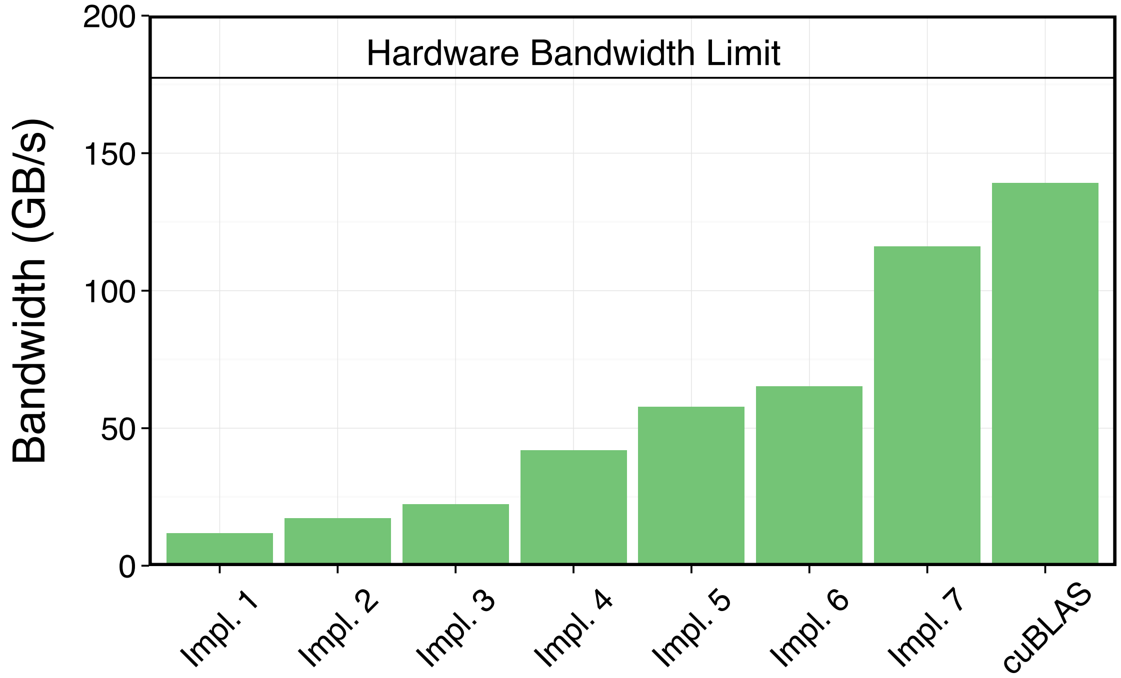
```
kernel
void reduce0(global float* g_idata,
             global float* g_odata,
             unsigned int n,
             local float* l_data) {
  unsigned int tid = get_local_id(0);
  unsigned int i   = get_global_id(0);
  l_data[tid] = (i < n) ? g_idata[i] : 0;
  barrier(CLK_LOCAL_MEM_FENCE);

  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

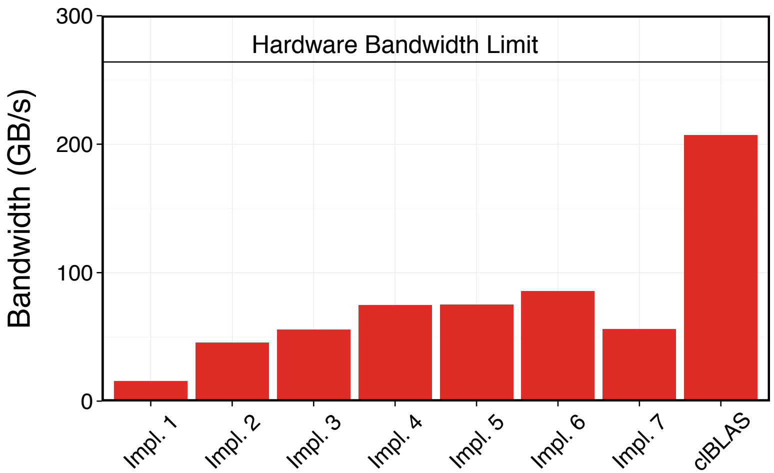
# Performance Results Nvidia



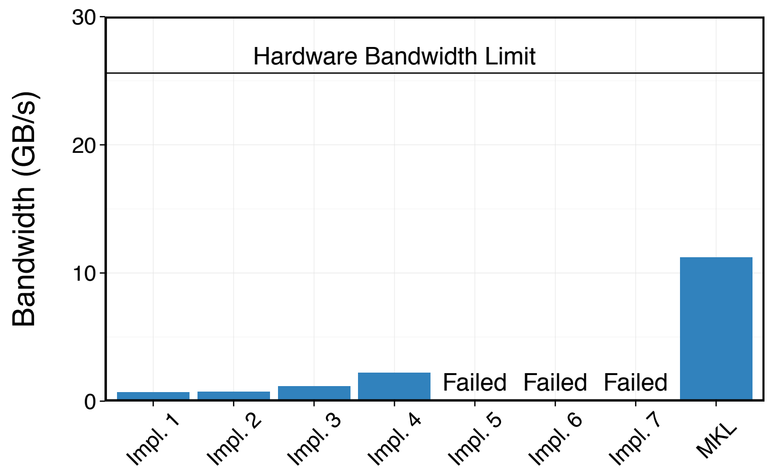

(a) Nvidia's GTX 480 GPU.

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

# Performance Results AMD and Intel




(b) AMD's HD 7970 GPU.

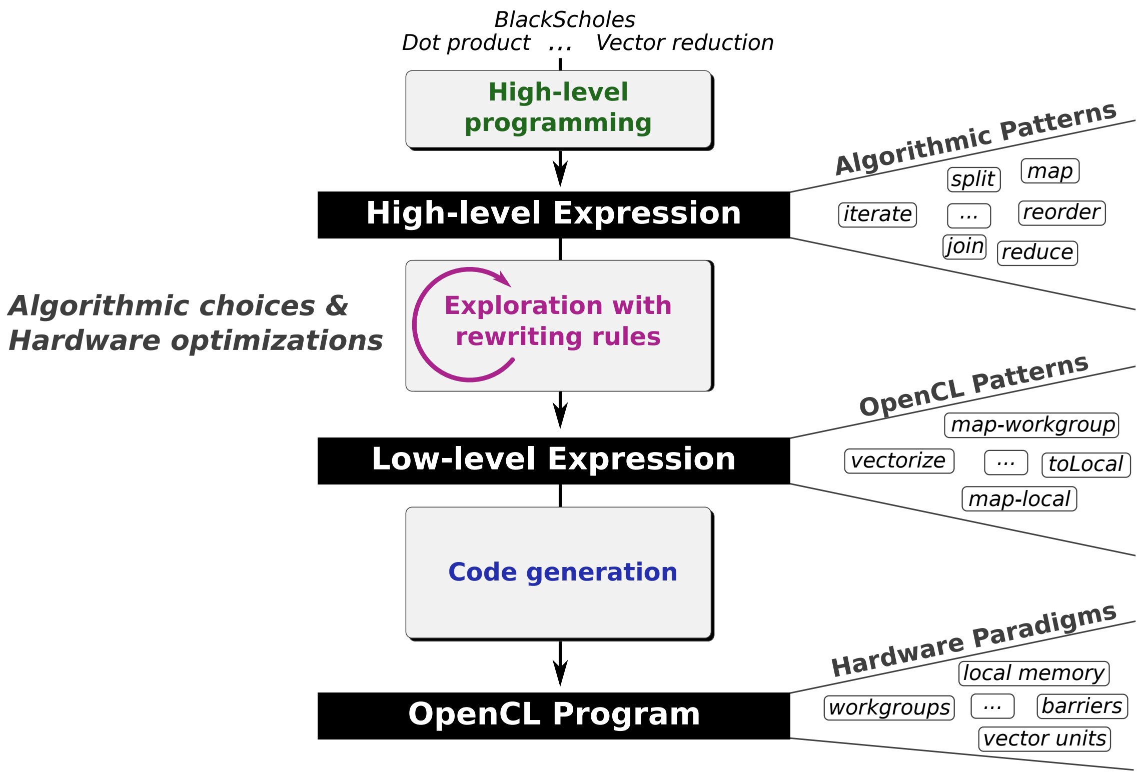



(c) Intel's E5530 dual-socket CPU.

- … unfortunately, optimisations in OpenCL are not portable!

- **Challenge**: how to achieving portable performance?

# Generating Performance Portable Code using Rewrite Rules




- **Ambition**: automatic generation of *Performance Portable* code

# Walkthrough

① $sum(vec) = reduce(+, 0, vec)$

↓ rewrite rules

② 

$$
\begin{aligned}
vecSum = \mathit{reduce} \circ \mathit{join} \circ \mathit{map\text{-}workgroup}\ ( \\
\quad \mathit{join} \circ \mathit{toGlobal}\ (\mathit{map\text{-}local}\ (\mathit{map\text{-}seq}\ \mathit{id})) \circ \mathit{split}\ 1 \circ \\
\quad \mathit{join} \circ \mathit{map\text{-}warp}\ ( \\
\qquad \mathit{join} \circ \mathit{map\text{-}lane}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 1 \circ \\
\qquad \mathit{join} \circ \mathit{map\text{-}lane}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 2 \circ \\
\qquad \mathit{join} \circ \mathit{map\text{-}lane}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 4 \circ \\
\qquad \mathit{join} \circ \mathit{map\text{-}lane}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 8 \circ \\
\qquad \mathit{join} \circ \mathit{map\text{-}lane}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 16 \circ \\
\qquad \mathit{join} \circ \mathit{map\text{-}lane}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 32 \\
\quad ) \circ \mathit{split}\ 64 \circ \\
\quad \mathit{join} \circ \mathit{map\text{-}local}\ (\mathit{reduce\text{-}seq}\ (+)\ 0) \circ \mathit{split}\ 2 \circ \mathit{reorder\text{-}stride}\ 64 \circ \\
\quad \mathit{join} \circ \mathit{toLocal}\ (\mathit{map\text{-}local}\ (\mathit{reduce\text{-}seq}\ (+)\ 0)) \circ \\
\quad \mathit{split}\ (\mathrm{blockSize}/128) \circ \mathit{reorder\text{-}stride}\ 128 \\
) \circ \mathit{split}\ \mathrm{blockSize}
\end{aligned}
$$

↗ 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];
}
```

THE UNIVERSITY *of* EDINBURGH
**informatics**

# Walkthrough

① $sum(vec) = reduce(+, 0, vec)$

rewrite rules

code generation

②

$vecSum = reduce \circ join \circ map\text{-}workgroup\ ($
$\quad join \circ toGlobal\ (map\text{-}local\ (map\text{-}seq\ id)) \circ split\ 1 \circ$
$\quad join \circ map\text{-}warp\ ($
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 1 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 2 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 4 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 8 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 16 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 32$
$\quad ) \circ split\ 64 \circ$
$\quad join \circ map\text{-}local\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 64 \circ$
$\quad join \circ toLocal\ (map\text{-}local\ (reduce\text{-}seq\ (+)\ 0)) \circ$
$\quad split\ (blockSize/128)\ \circ reorder\text{-}stride\ 128$
$) \circ split\ blockSize$

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)

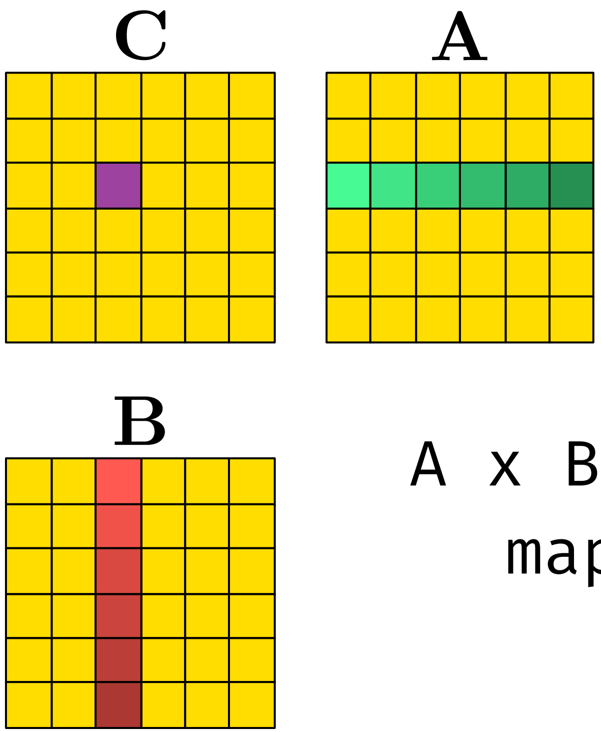
| | | | |
|---|---|---|---|
| map(f, x): | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ | $\longmapsto$ | $[f(x_1), f(x_2), f(x_3), f(x_4), f(x_5), f(x_6), f(x_7), f(x_8)]$ |
| zip(x, y): | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$<br>$[y_1, y_2, y_3, y_4, y_5, y_6, y_7, y_8]$ | $\longmapsto$ | $[(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), (x_5, y_5), (x_6, y_6), (x_7, y_7), (x_8, y_8)]$ |
| reduce(+, 0, x): | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ | $\longmapsto$ | $[x_1+x_2+x_3+x_4+x_5+x_6+x_7+x_8]$ |
| split(n, x): | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ | $\longmapsto$ | $[[x_1, x_2], [x_3, x_4], [x_5, x_6], [x_7, x_8]]$ |
| join(x): | $[[x_1, x_2], [x_3, x_4], [x_5, x_6], [x_7, x_8]]$ | $\longmapsto$ | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ |
| iterate(f, n, x): | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ | $\longmapsto$ | $f(\ldots f([x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8])\ldots)$ |
| reorder(σ, x): | $[x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8]$ | $\longmapsto$ | $[x_{\sigma(1)}, x_{\sigma(2)}, x_{\sigma(3)}, x_{\sigma(4)}, x_{\sigma(5)}, x_{\sigma(6)}, x_{\sigma(7)}, x_{\sigma(8)}]$ |

# 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) ) )
```

THE UNIVERSITY of EDINBURGH
informatics

# ① High-Level Programs




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

# Walkthrough

① $sum(vec) = reduce(+, 0, vec)$

rewrite rules ↓ code generation ↗

②

$vecSum = reduce \circ join \circ map\text{-}workgroup\ ($
$\quad join \circ toGlobal\ (map\text{-}local\ (map\text{-}seq\ id)) \circ split\ 1 \circ$
$\quad join \circ map\text{-}warp\ ($
$\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 1 \circ$
$\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 2 \circ$
$\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 4 \circ$
$\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 8 \circ$
$\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 16 \circ$
$\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 32$
$\quad ) \circ split\ 64 \circ$
$\quad join \circ map\text{-}local\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2 \circ reorder\text{-}stride\ 64 \circ$
$\quad join \circ toLocal\ (map\text{-}local\ (reduce\text{-}seq\ (+)\ 0)) \circ$
$\quad split\ (blockSize/128) \circ reorder\text{-}stride\ 128$
$) \circ split\ blockSize$

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];
}
```

THE UNIVERSITY of EDINBURGH informatics

# Walkthrough

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

rewrite rules

code generation

②

$$
\begin{aligned}
vecSum = \ & reduce \circ join \circ map\text{-}workgroup\ ( \\
& \quad join \circ toGlobal\ (map\text{-}local\ (map\text{-}seq\ id)) \circ split\ 1\ \circ \\
& \quad join \circ map\text{-}warp\ ( \\
& \qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 1\ \circ \\
& \qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 2\ \circ \\
& \qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 4\ \circ \\
& \qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 8\ \circ \\
& \qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 16\ \circ \\
& \qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 32 \\
& \quad ) \circ split\ 64\ \circ \\
& \quad join \circ map\text{-}local\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 64\ \circ \\
& \quad join \circ toLocal\ (map\text{-}local\ (reduce\text{-}seq\ (+)\ 0))\ \circ \\
& \quad split\ (\text{blockSize}/128)\ \circ reorder\text{-}stride\ 128 \\
& )\ \circ split\ \text{blockSize}
\end{aligned}
$$

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:

$$map\ f \rightarrow join \circ map\ (map\ f) \circ split\ n$$

Map fusion rule:

$$map\ f \circ map\ g \rightarrow map\ (f \circ g)$$

Reduce rules:

$$reduce\ f\ \ z \rightarrow reduce\ f\ \ z \circ reducePart\ f\ \ z$$

$$reducePart\ f\ \ z \rightarrow reducePart\ f\ \ z \circ reorder$$

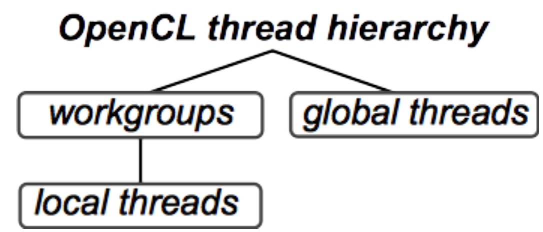
$$reducePart\ f\ \ z \rightarrow join\ \ \circ map\ (reducePart\ f\ \ z) \circ split\ n$$

$$reducePart\ f\ \ z \rightarrow iterate\ n\ (reducePart\ f\ \ z)$$

THE UNIVERSITY *of* EDINBURGH
informatics

# ② OpenCL Primitives

| **Primitive** | **OpenCL concept** |
|---|---|
| $mapGlobal$ | Work-items |
| $mapWorkgroup$<br>$mapLocal$ | Work-groups |
| $mapSeq$<br>$reduceSeq$ | Sequential implementations |
| $toLocal$ , $toGlobal$ | Memory areas |
| $mapVec$,<br>$splitVec$, $joinVec$ | Vectorisation |



# ② OpenCL Rewrite Rules

- Express low-level implementation and optimisation choices

## Map rules:

$$map\ f \rightarrow mapWorkgroup\ f \mid mapLocal\ f \mid mapGlobal\ f \mid mapSeq\ f$$

## Local/ global memory rules:

$$mapLocal\ f \rightarrow toLocal\ (mapLocal\ f)$$

$$mapLocal\ f \rightarrow toGlobal\ (mapLocal\ f)$$

## Vectorisation rule:

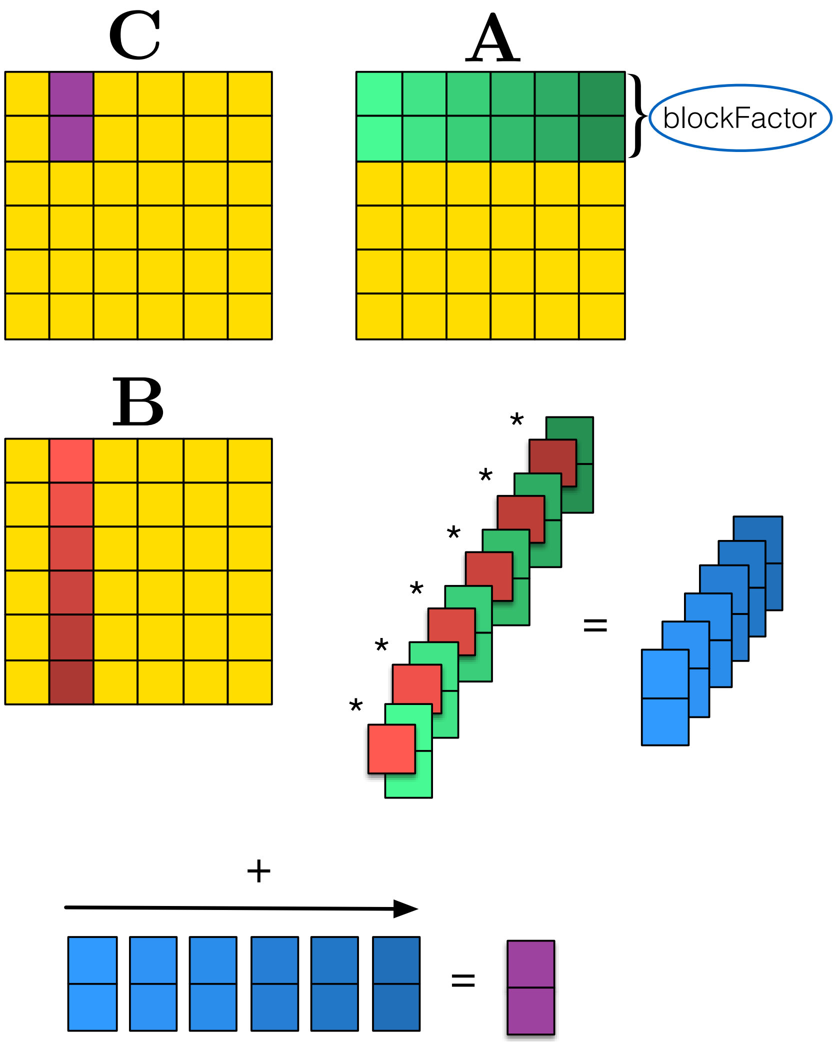
$$map\ f \rightarrow joinVec \circ map\ (mapVec\ f) \circ splitVec\ n$$

## Fusion rule:

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

THE UNIVERSITY *of* EDINBURGH informatics

# ② Optimisation Example: Register Blocking




```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B,
  global float* C, int K, int M, int N)
{
  float acc[blockFactor];

  for (int glb_id_1 = get_global_id(1);
       glb_id_1 < M / blockFactor;
       glb_id_1 += get_global_size(1)) {
    for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
       glb_id_0 += get_global_size(0)) {

      for (int i = 0; i < K; i += 1)
        float temp = B[i * N + glb_id_0];
        for (int j = 0; j < blockFactor; j+= 1)
          acc[j] +=
            A[blockFactor * glb_id_1 * K + j * K + i]
              * temp;

      for (int j = 0; j < blockFactor; j += 1)
        C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
           = acc[j];
    }
  }
}
```

# ② Register Blocking as a Macro Rule

- Optimisations are expressed as *Macro Rules:*
  - Series of Rewrites applied to achieve an optimisation goal

$$
\begin{aligned}
&registerBlocking = \\
&\quad Map(f) \Rightarrow Join() \circ Map(Map(f)) \circ Split(k) \\
&\quad Map(a \mapsto Map(b \mapsto f(a,b))) \Rightarrow Transpose() \circ Map(b \mapsto Map(a \mapsto f(a,b))) \\
&\quad Map(f \circ g) \Rightarrow Map(f) \circ Map(g) \\
&\quad Map(Reduce(f)) \Rightarrow Transpose() \circ Reduce((acc,x) \mapsto Map(f) \circ Zip(acc,x)) \\
&\quad Map(Map(f)) \Rightarrow Transpose() \circ Map(Map(f)) \circ Transpose() \\
&\quad Transpose() \circ Transpose() \Rightarrow id \\
&\quad Reduce(f) \circ Map(g) \Rightarrow Reduce((acc,x) \mapsto f(acc, g(x))) \\
&\quad Map(f) \circ Map(g) \Rightarrow Map(f \circ g)
\end{aligned}
$$

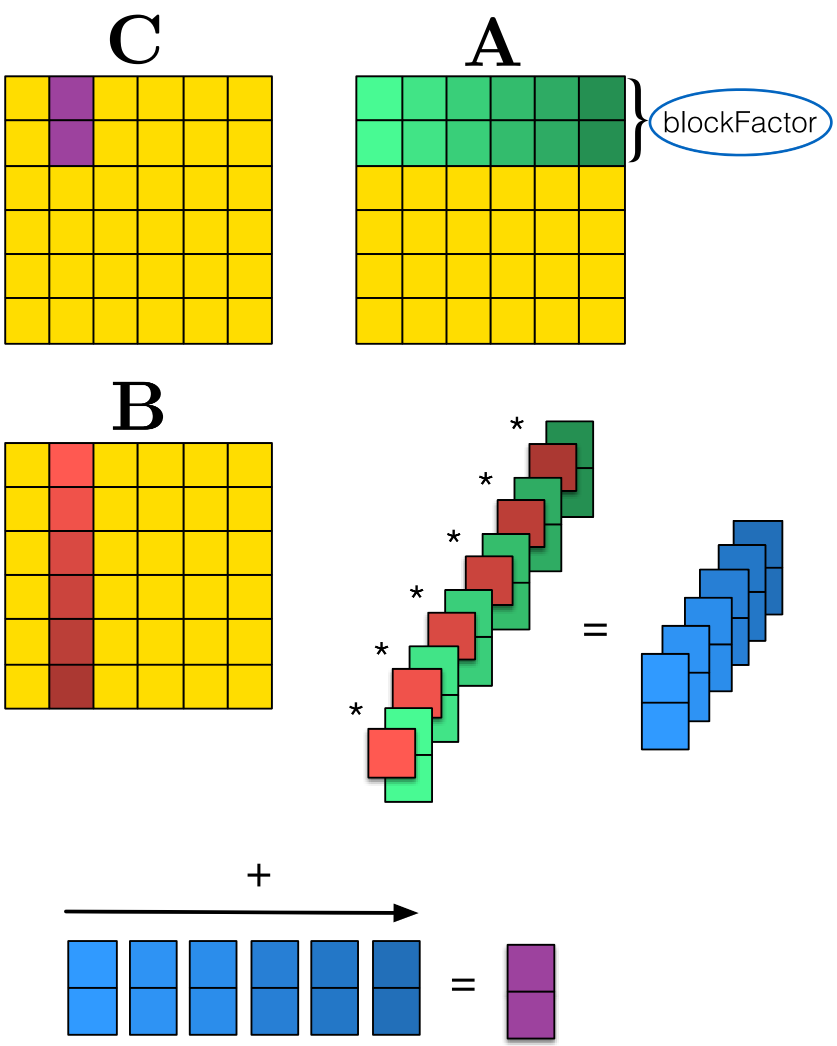
# ② Register Blocking as a Series of Rewrites

$$
\begin{aligned}
&Map(\overrightarrow{rowA} \mapsto \\
&\quad Map(\overrightarrow{colB} \mapsto \\
&\quad\quad Reduce(+) \circ Map(*) \\
&\quad\quad\quad \$\ Zip(\overrightarrow{rowA}, \overrightarrow{colB}) \\
&\quad ) \circ Transpose()\ \$\ \mathbf{B} \\
&)\ \$\ \mathbf{A}
\end{aligned}
$$

→ ... →

$$
\begin{aligned}
&Join() \circ Map(rowsA \mapsto \\
&\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
&\quad\quad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{pair}) \mapsto \\
&\quad\quad\quad Map(x \mapsto x\_0 + x\_1 * pair.\_1) \\
&\quad\quad\quad\quad \$\ Zip(\overrightarrow{acc}, pair.\_0) \\
&\quad\quad )\ \$\ Zip(Transpose()\ \$\ rowsA, \overrightarrow{colB}) \\
&\quad ) \circ Transpose()\ \$\ \mathbf{B} \\
&) \circ Split(blockFactor)\ \$\ \mathbf{A}
\end{aligned}
$$

# ② Register Blocking Functionally Expressed




$$
\begin{aligned}
& Join() \circ Map(rowsA \mapsto \\
& \quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
& \quad\quad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{pair}) \mapsto \\
& \quad\quad\quad Map(x \mapsto x\_0 + x\_1 * pair.\_1) \\
& \quad\quad\quad\quad \$\ Zip(\overrightarrow{acc}, pair.\_0) \\
& \quad\quad )\ \$\ Zip(Transpose()\ \$\ rowsA, \overrightarrow{colB}) \\
& \quad ) \circ Transpose()\ \$\ \mathbf{B} \\
& ) \circ Split(blockFactor)\ \$\ \mathbf{A}
\end{aligned}
$$

# Walkthrough

① $vecSum = reduce\ (+)\ 0$

rewrite rules

code generation

②

$$
\begin{aligned}
&vecSum = reduce \circ join \circ map\text{-}workgroup\ ( \\
&\quad join \circ toGlobal\ (map\text{-}local\ (map\text{-}seq\ id)) \circ split\ 1 \circ \\
&\quad join \circ map\text{-}warp\ ( \\
&\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 1 \circ \\
&\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 2 \circ \\
&\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 4 \circ \\
&\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 8 \circ \\
&\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 16 \circ \\
&\qquad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 32 \\
&\quad ) \circ split\ 64 \circ \\
&\quad join \circ map\text{-}local\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 64 \circ \\
&\quad join \circ toLocal\ (map\text{-}local\ (reduce\text{-}seq\ (+)\ 0)) \circ \\
&\quad split\ (\text{blockSize}/128)\ \circ reorder\text{-}stride\ 128 \\
&) \circ split\ \text{blockSize}
\end{aligned}
$$

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

code generation

② 

$vecSum = reduce \circ join \circ map\text{-}workgroup\ ($
$\quad join \circ toGlobal\ (map\text{-}local\ (map\text{-}seq\ id)) \circ split\ 1 \circ$
$\quad join \circ map\text{-}warp\ ($
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 1 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 2 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 4 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 8 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 16 \circ$
$\quad\quad join \circ map\text{-}lane\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 32$
$\quad ) \circ split\ 64 \circ$
$\quad join \circ map\text{-}local\ (reduce\text{-}seq\ (+)\ 0) \circ split\ 2\ \circ reorder\text{-}stride\ 64 \circ$
$\quad join \circ toLocal\ (map\text{-}local\ (reduce\text{-}seq\ (+)\ 0)) \circ$
$\quad split\ (blockSize/128)\ \circ reorder\text{-}stride\ 128$
$) \circ split\ blockSize$

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];
}
```

THE UNIVERSITY of EDINBURGH
informatics

# ③ Pattern based OpenCL Code Generation

- Generate OpenCL code for each OpenCL primitive

$$mapGlobal\ f\ xs \rightarrow$$

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

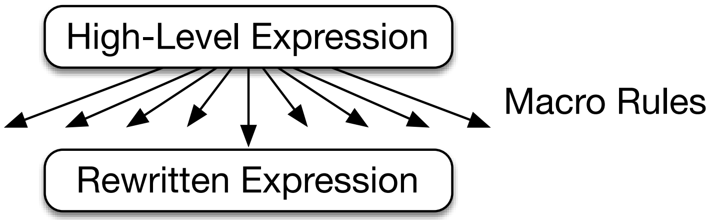
$$reduceSeq\ f\ z\ xs \rightarrow$$

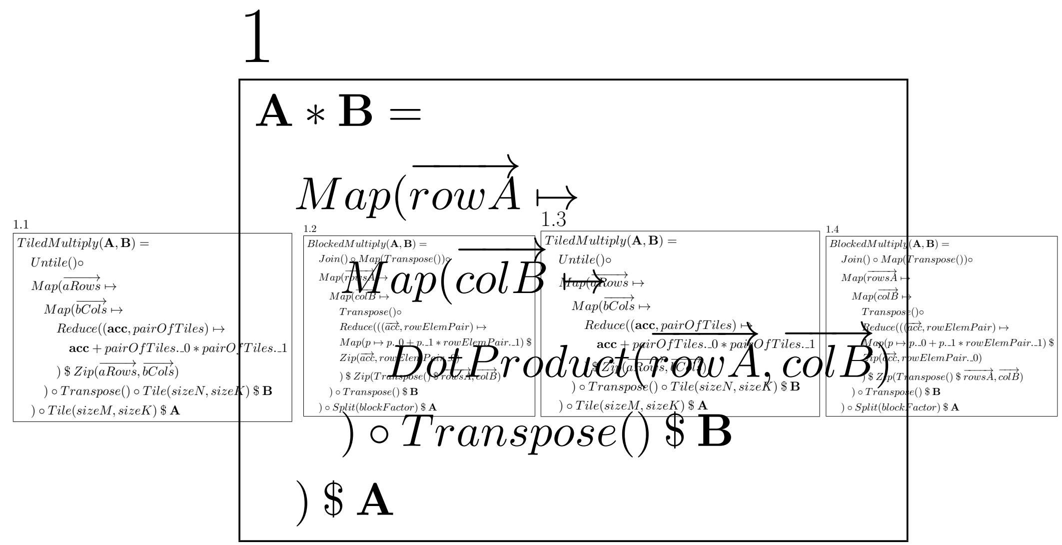
```
T acc = z;
for (int i = 0; i < n; ++i) {
  acc = f(acc, xs[i]);
}
```

$\vdots$ $\vdots$

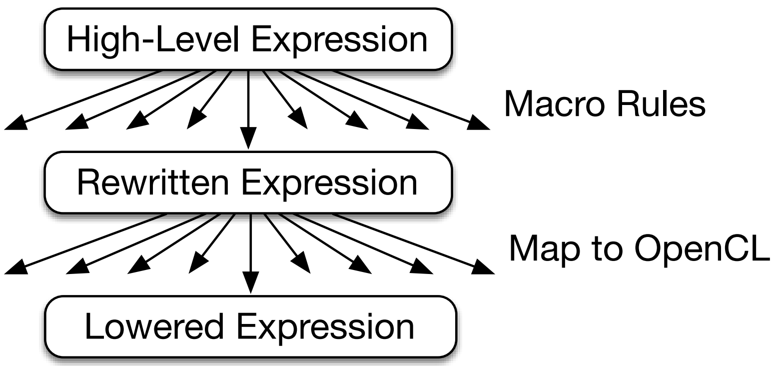
THE UNIVERSITY of EDINBURGH informatics

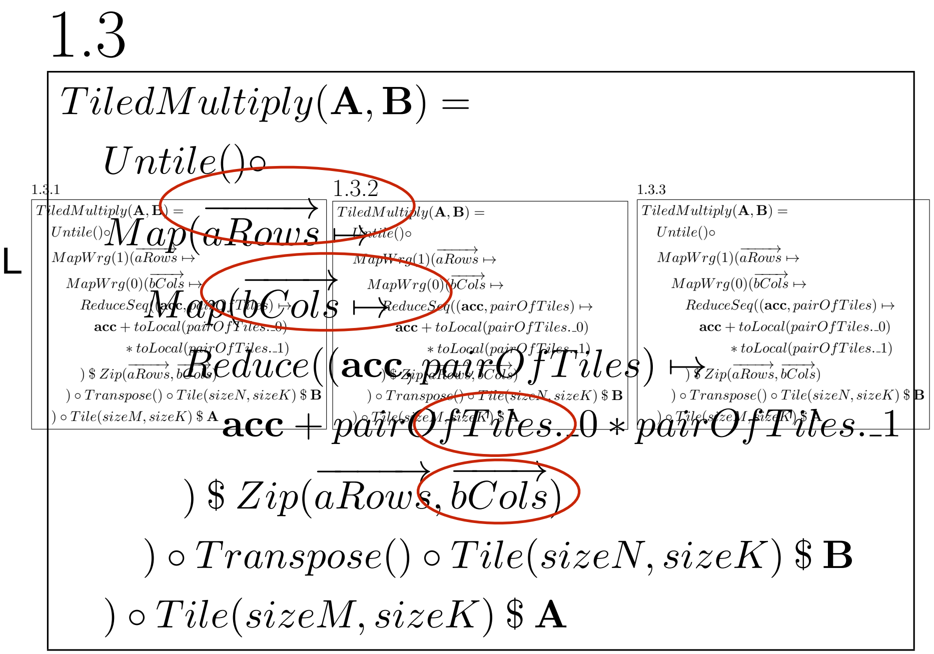
# Exploration Strategy



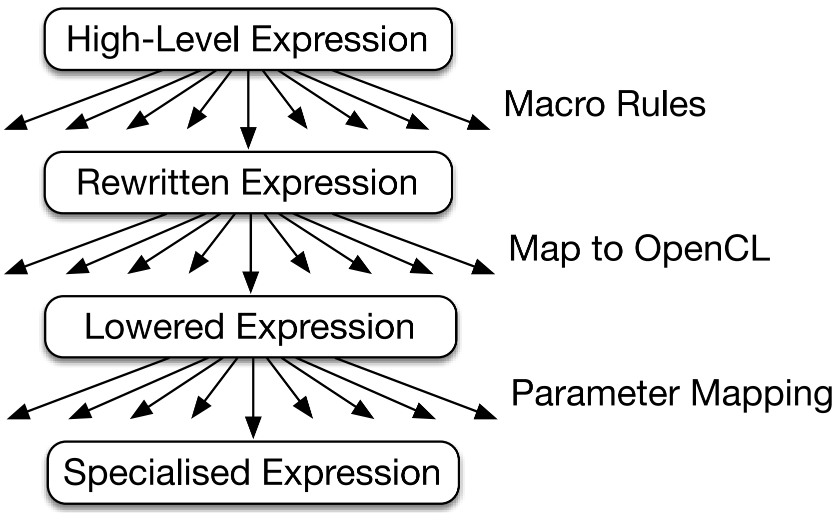



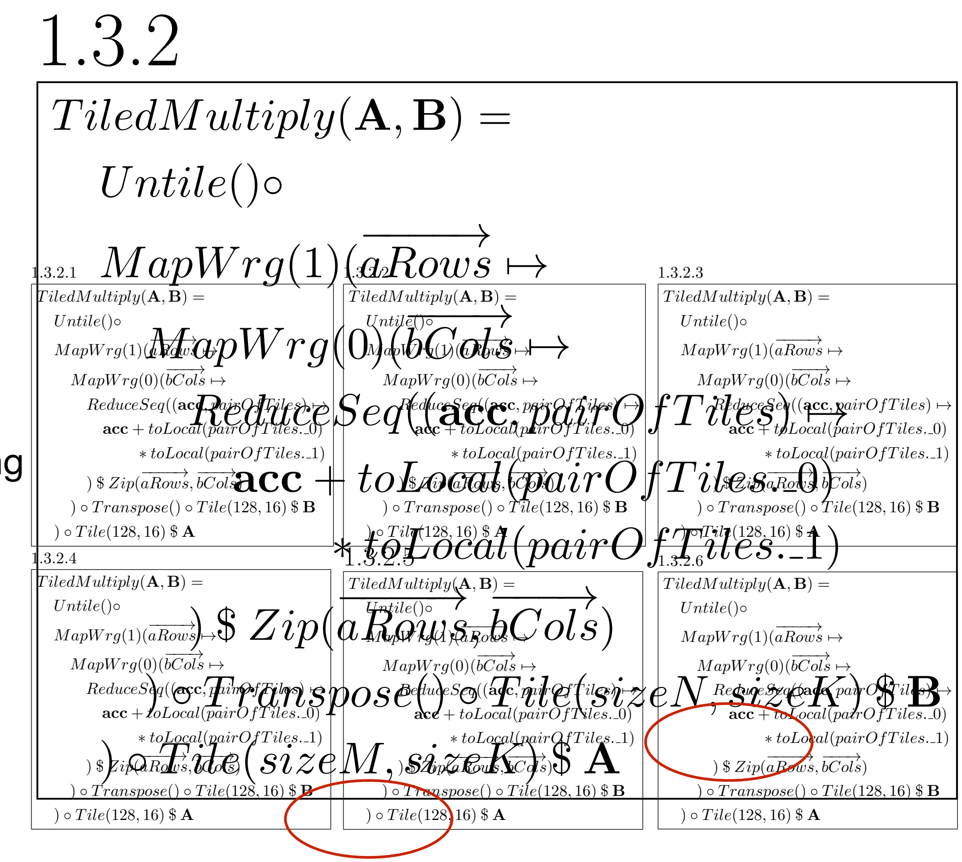
# Exploration Strategy



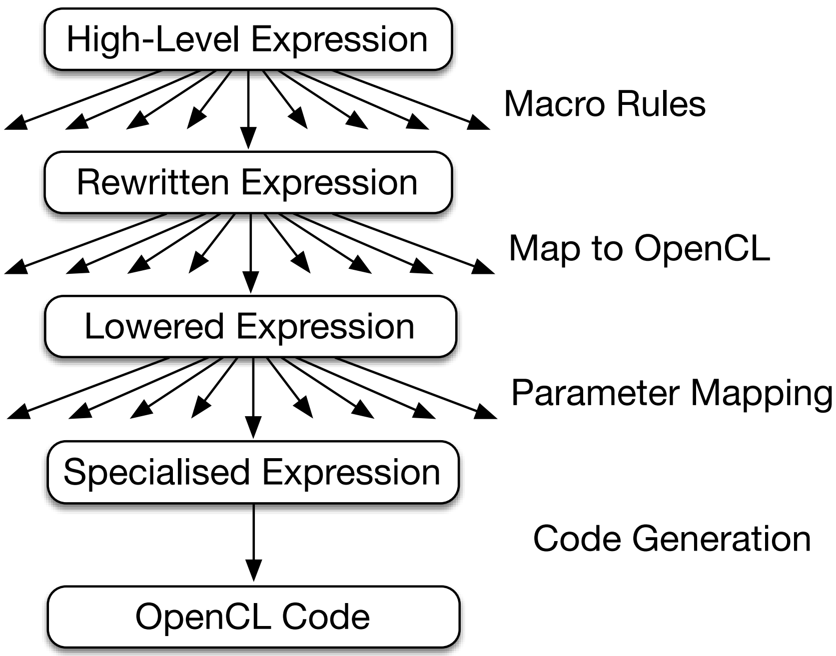



# Exploration Strategy

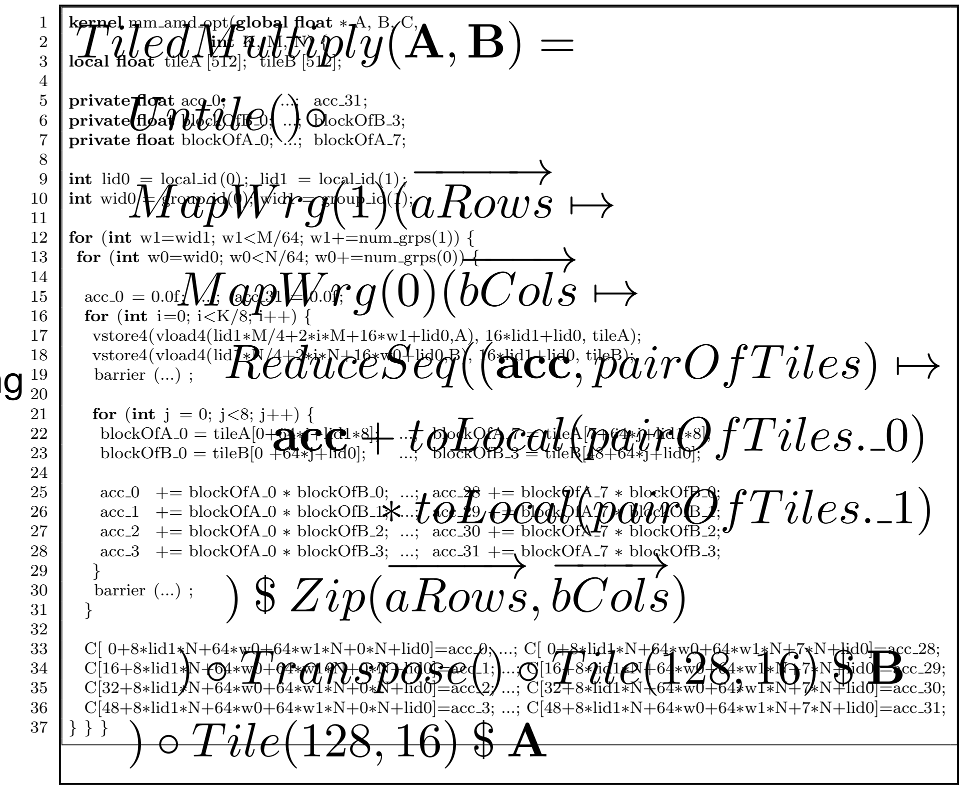


# Exploration Strategy




1.3.2.5




THE UNIVERSITY of EDINBURGH informatics

# 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

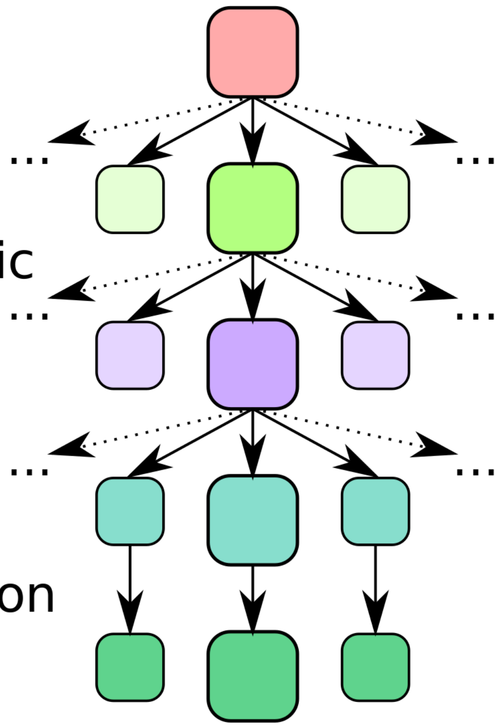
Phases:

Algorithmic Exploration

OpenCL specific Exploration

Parameter Exploration

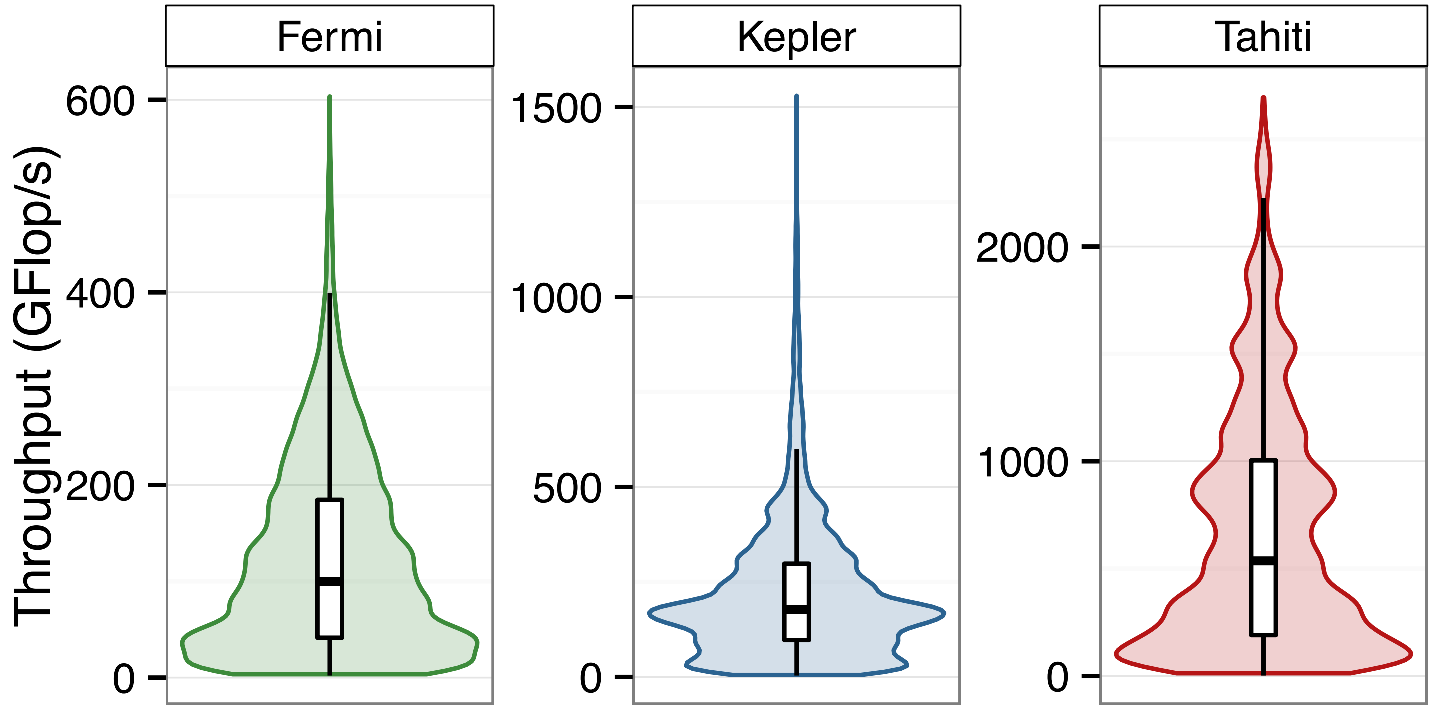
Code Generation




Program Variants:

| | |
|---|---|
| High-Level Program | 1 |
| Algorithmic Rewritten Program | 8 |
| OpenCL Specific Program | 760 |
| Fully Specialized Program | 46,000 |
| OpenCL Code | 46,000 |

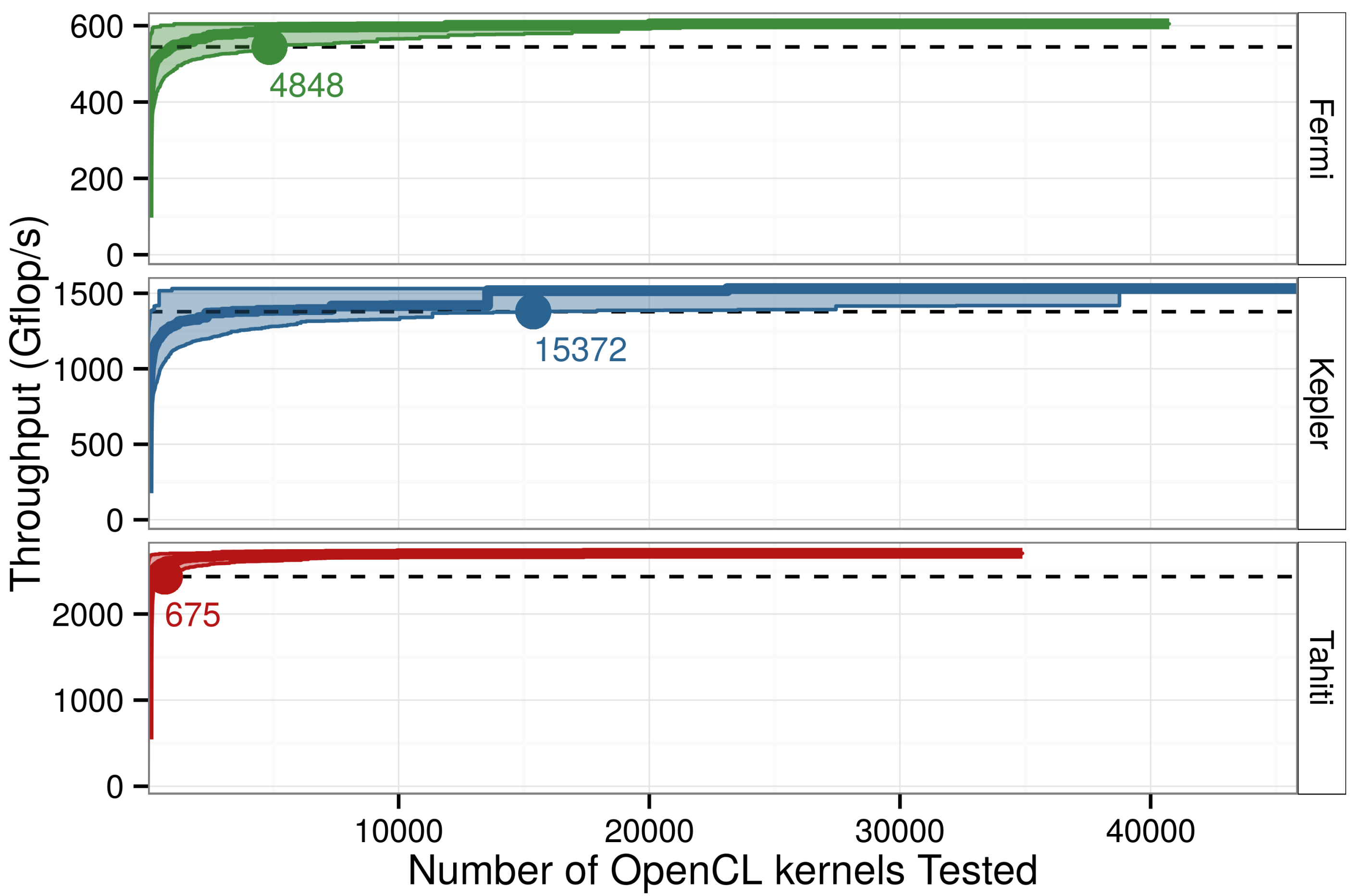
# Exploration Space for Matrix Multiplication




Only few OpenCL kernel with very good performance

THE UNIVERSITY of EDINBURGH
informatics

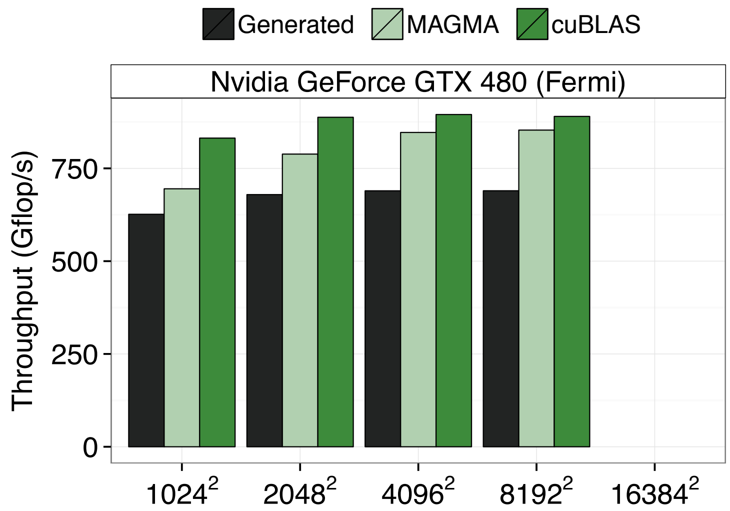
# Performance Evolution for Randomised Search

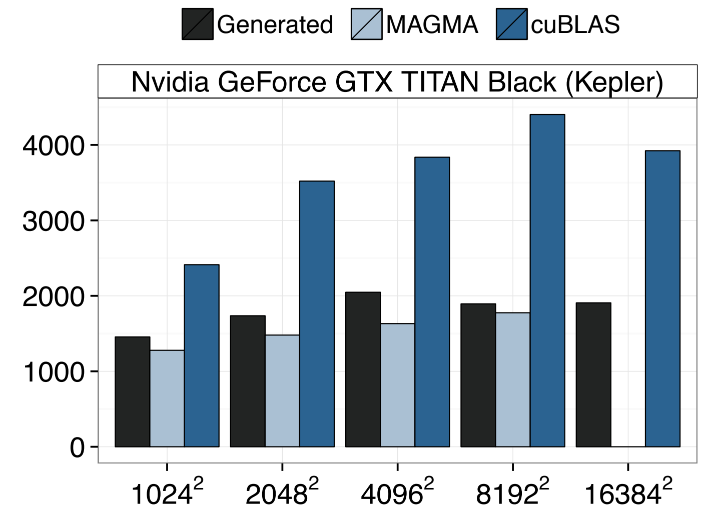


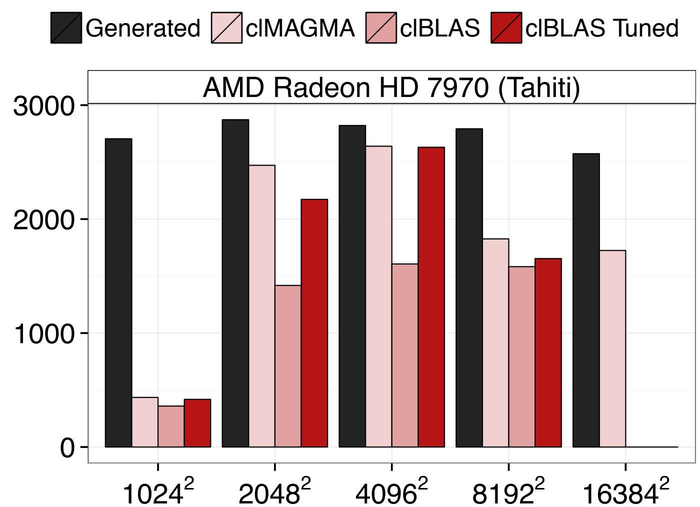

Even with a simple random search strategy one can expect to find a good performing kernel quickly

THE UNIVERSITY of EDINBURGH informatics

# Performance Results Matrix Multiplication



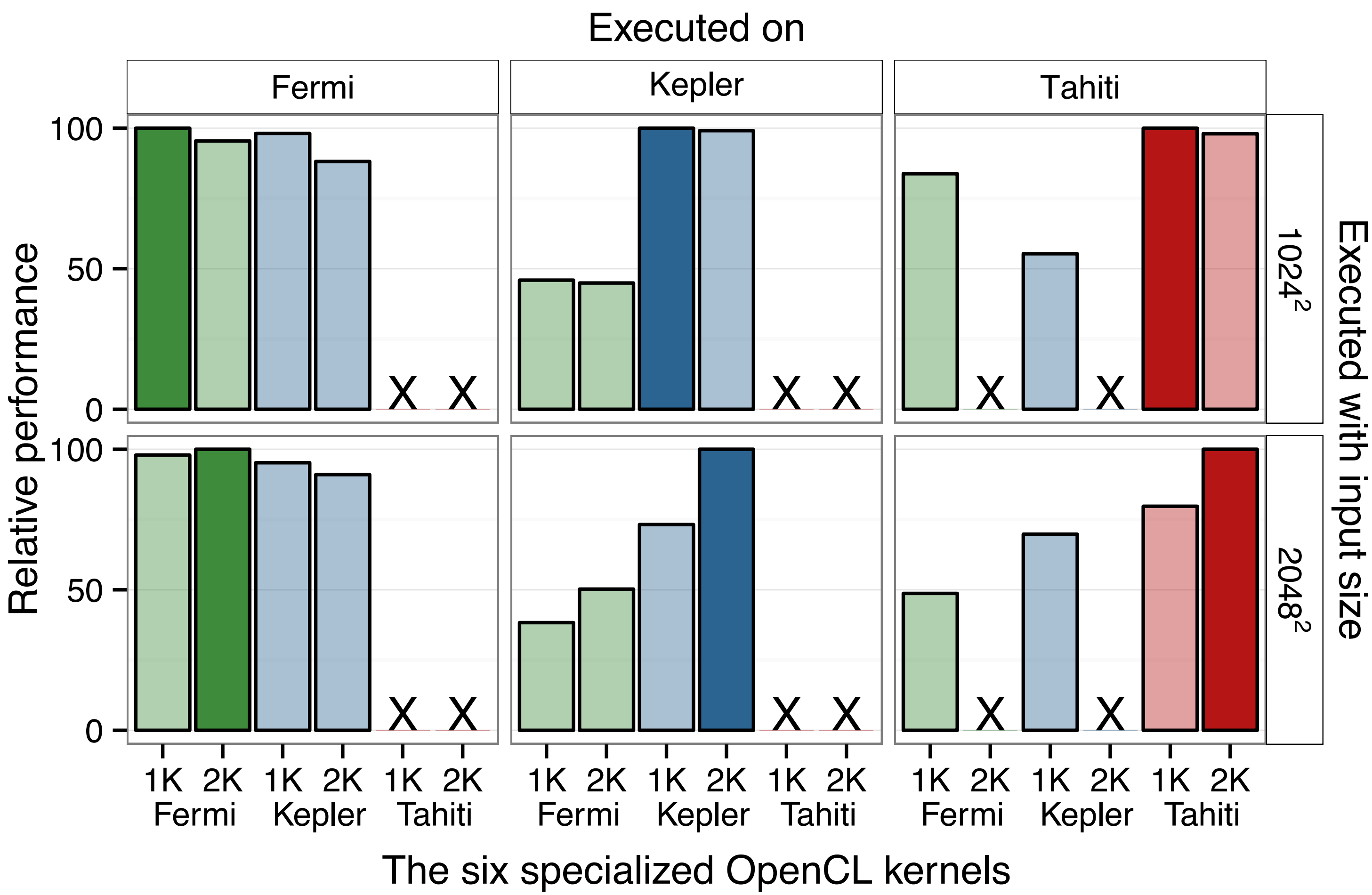




Performance close or better than hand-tuned MAGMA library

THE UNIVERSITY of EDINBURGH
informatics

# Performance Portability Matrix Multiplication




Generated kernels are specialised for device and input size

THE UNIVERSITY of EDINBURGH
informatics

# Summary

- OpenCL code is not *performance portable*
- Our approach uses
  - *portable* and functional **high-level primitives**,
  - **OpenCL-specific low-level primitives**, and
  - **rewrite-rules** to generate high *performance* code.
- Rewrite-rules define a space of possible implementations
- Performance on par with specialised, highly-tuned code

Christophe Dubach
christophe.dubach@ed.ac.uk

Michel Steuwer
michel.steuwer@ed.ac.uk

Thibaut Lutz
Now with Nvidia

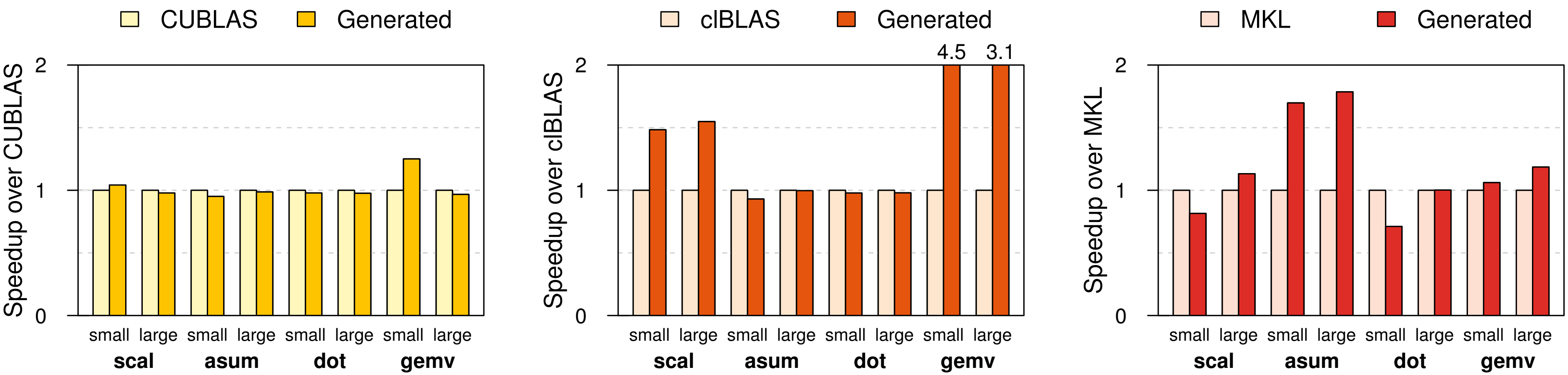
Toomas Remmelg
toomas.remmelg@ed.ac.uk

More details in the **ICFP 2015** and **GPGPU 2016** papers available at: http://www.lift-project.org

THE UNIVERSITY *of* EDINBURGH
**informatics**

supported by: Google Oracle Labs

# Performance Results more Benchmarks
## vs. Hardware-Specific Implementations

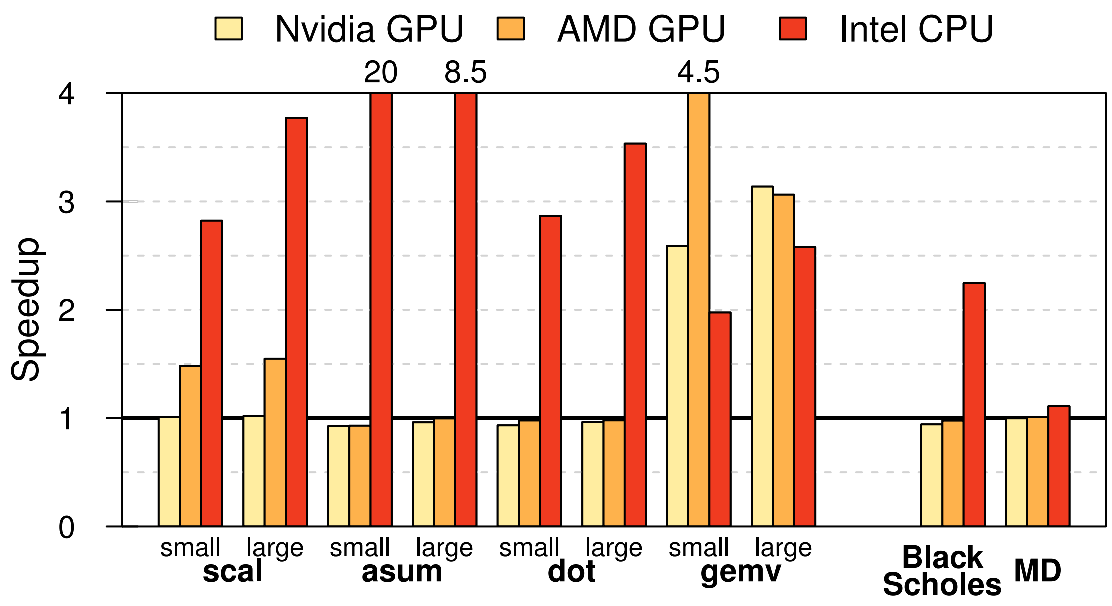



**(a)** Nvidia GPU **(b)** AMD GPU **(c)** Intel CPU

- Automatically generated code vs. expert written code
- Competitive performance vs. highly optimised implementations
- Up to **4.5x** speedup for *gemv* on AMD

THE UNIVERSITY *of* EDINBURGH
**informatics**

# Performance Results more Benchmarks
## vs. Portable Implementation




- Up to **20x** speedup on fairly simple benchmarks vs. portable clBLAS implementation

THE UNIVERSITY of EDINBURGH informatics