Performance Portable GPU Code Generation for Matrix Multiplication

Toomas Remmelg

Thibaut Lutz

Michel Steuwer

Christophe Dubach

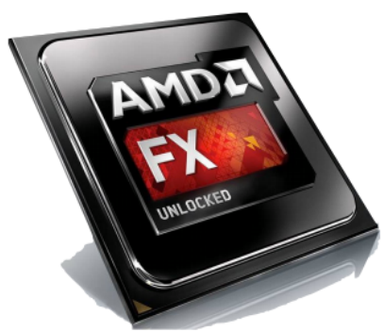
THE UNIVERSITY *of* EDINBURGH

Supported by:

Google Oracle Labs

The Problem

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

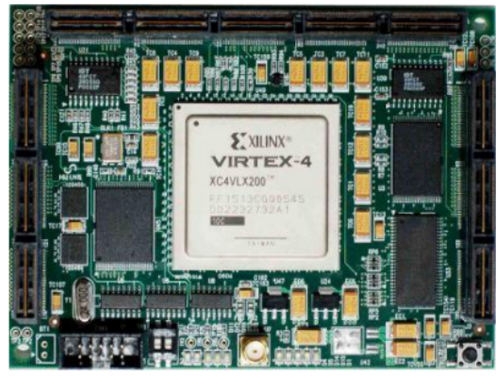


CPU

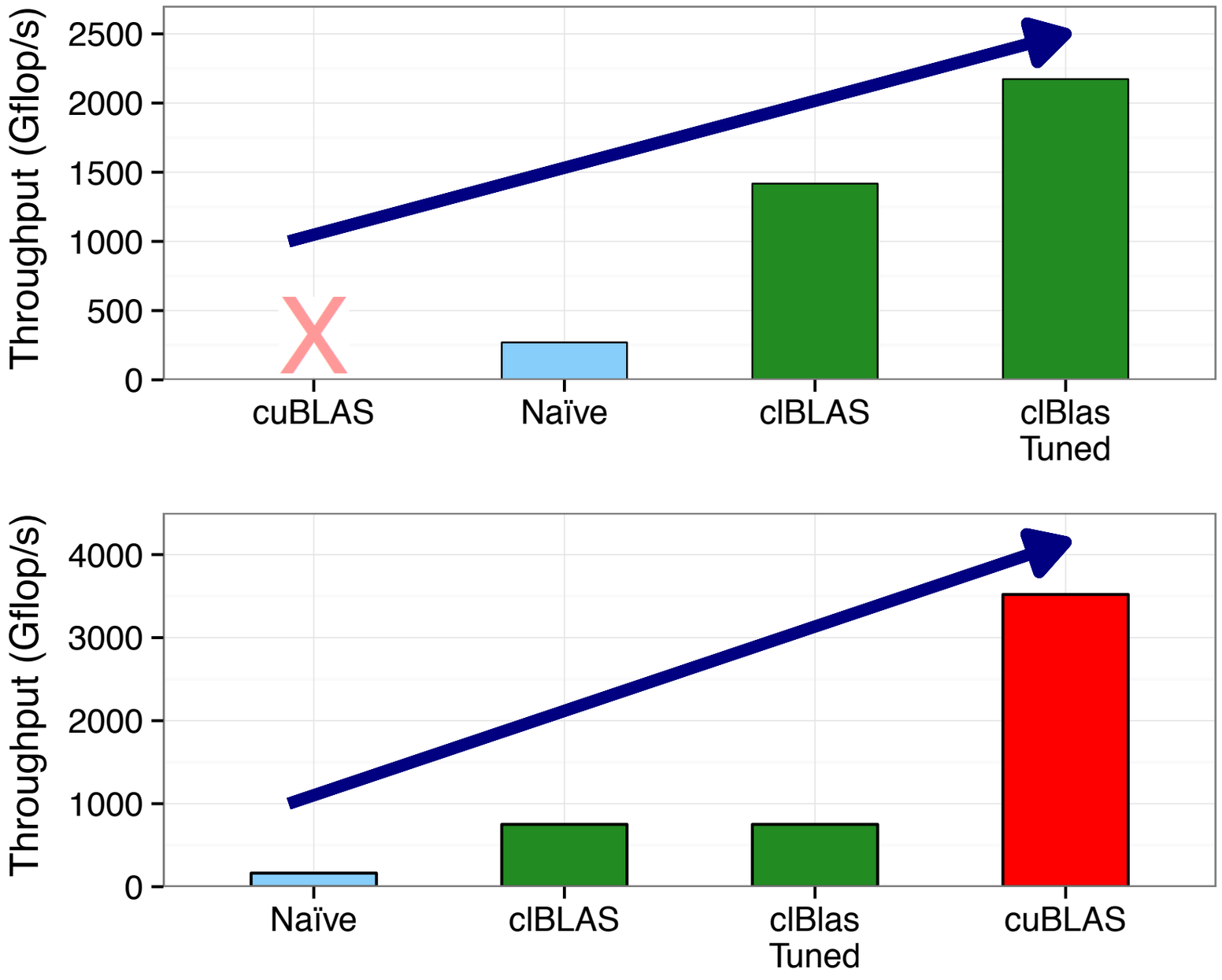
GPU

Accelerator

FPGA

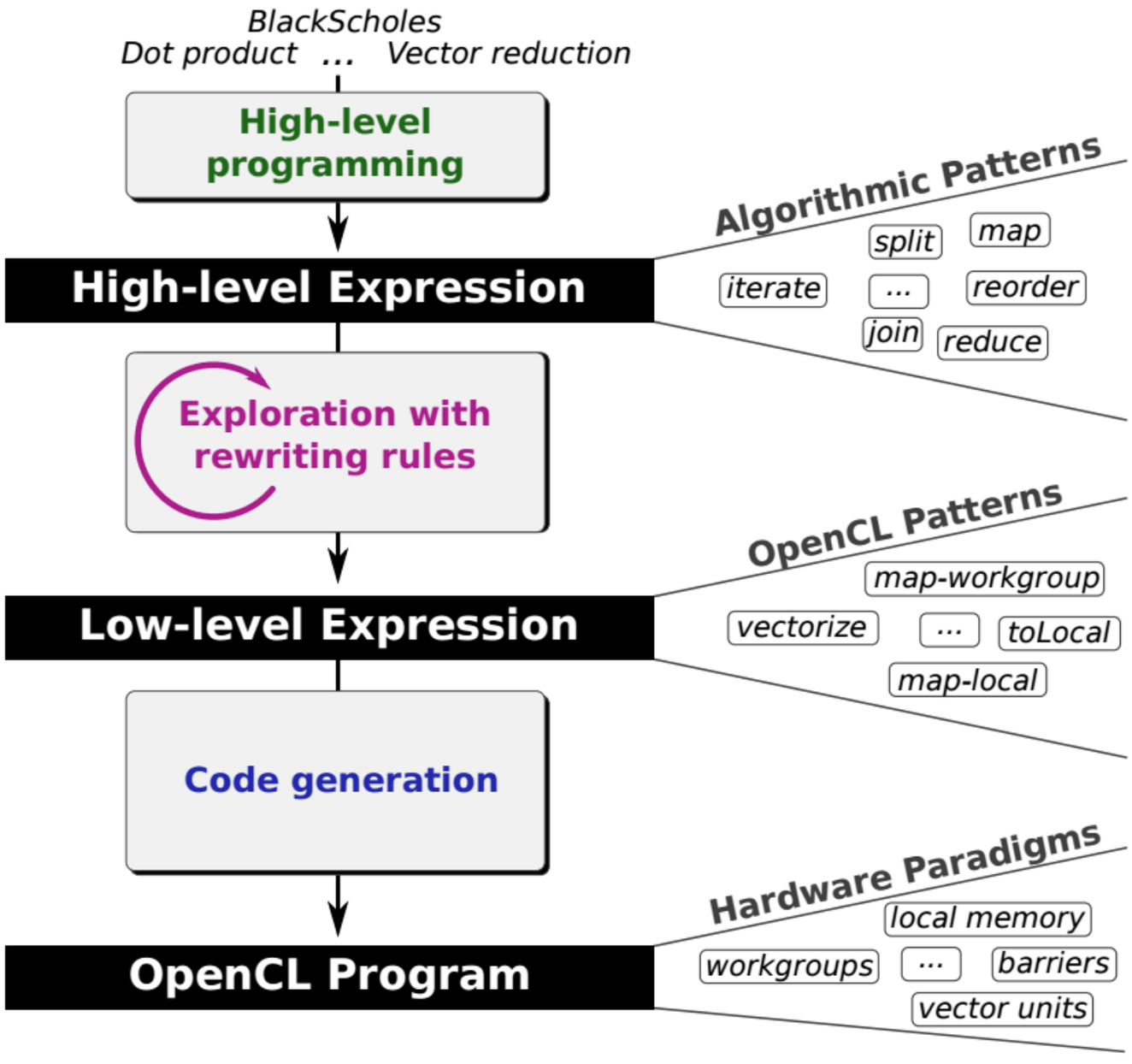


Performance Portability of Matrix Multiplication

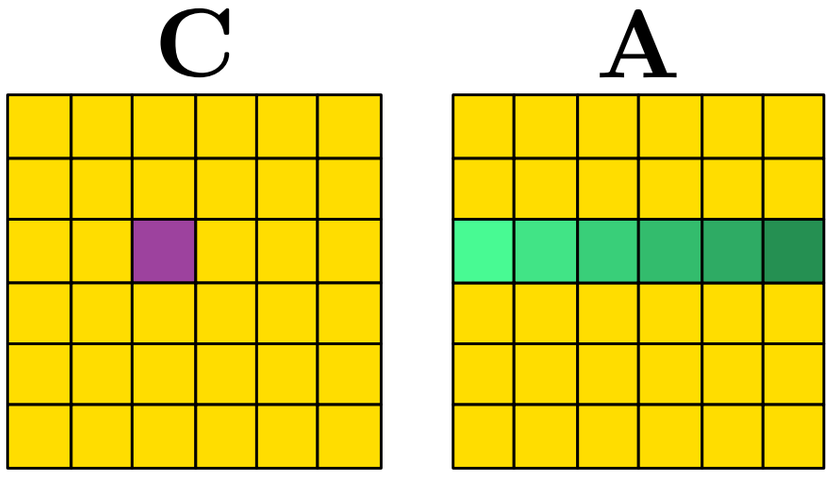


How to achieve performance portability?

Rewrite Rules



Matrix Multiplication
Expressed Functionally



Functional Representation

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

$$Map(\overrightarrow{rowA} \mapsto$$

$$Map(\overrightarrow{colB} \mapsto$$

$$DotProduct(\overrightarrow{rowA}, \overrightarrow{colB})$$

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

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

OpenCL

```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B
4    global float* C,
5    int M, int K, int N)
6  {
7    float acc = 0.0f;
8
9    for (int i = 0; i < K; i += 1)
10     acc = acc + A[id_A(glb_id_1, i)]
11       * B[id_B(i, glb_id_0)];
12
13   C[(id_C(glb_id_0, glb_id_1)] = acc;
14 }
```

THE UNIVERSITY of EDINBURGH informatics

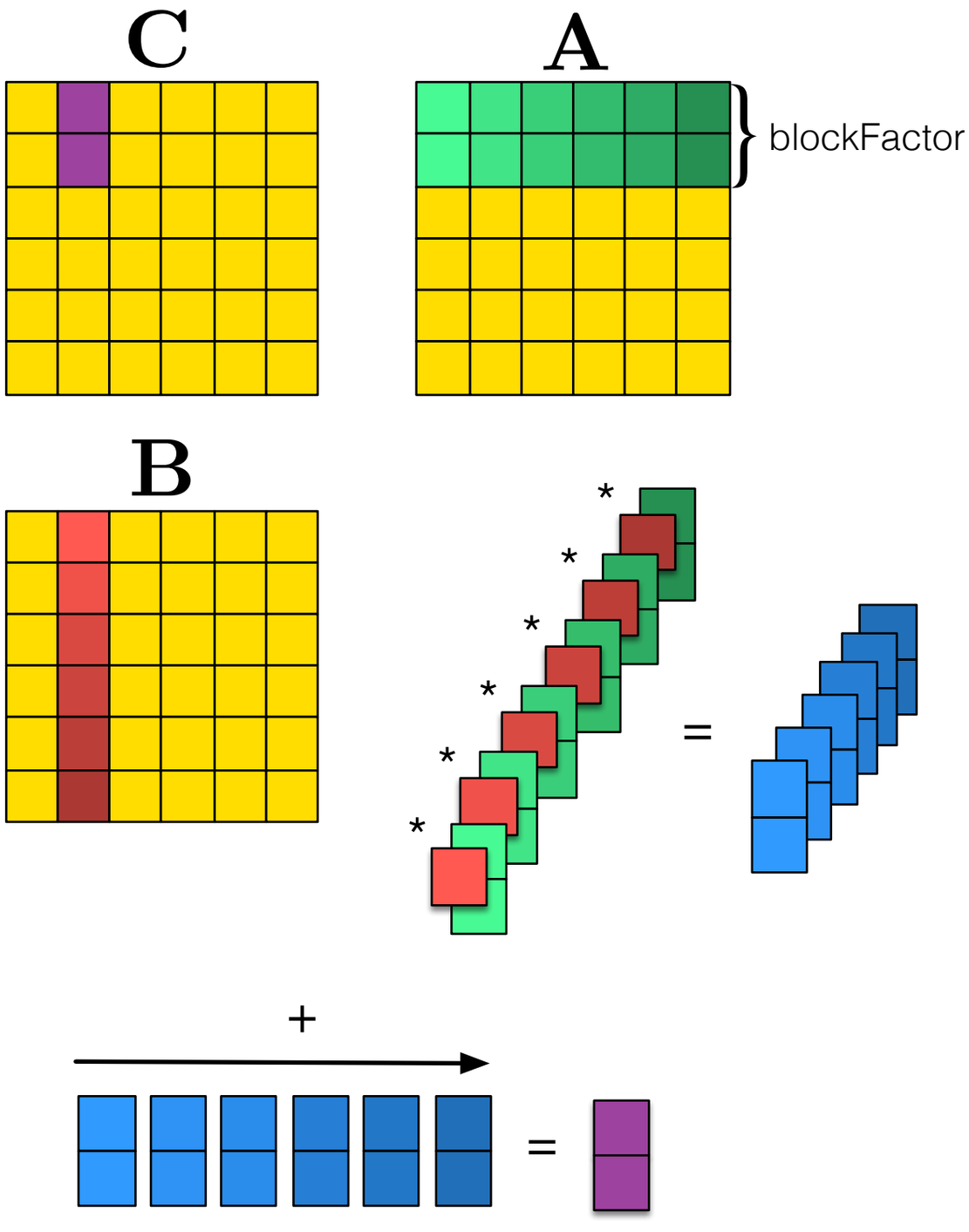
Traditional Optimisations

- **Register Blocking** Loading elements into registers and reusing them.
- **Tiling** Solving the problem by diving matrices into smaller tiles.
- **Vectorisation** Using wider vector units if available.

Why can’t this be automated by traditional compilers?

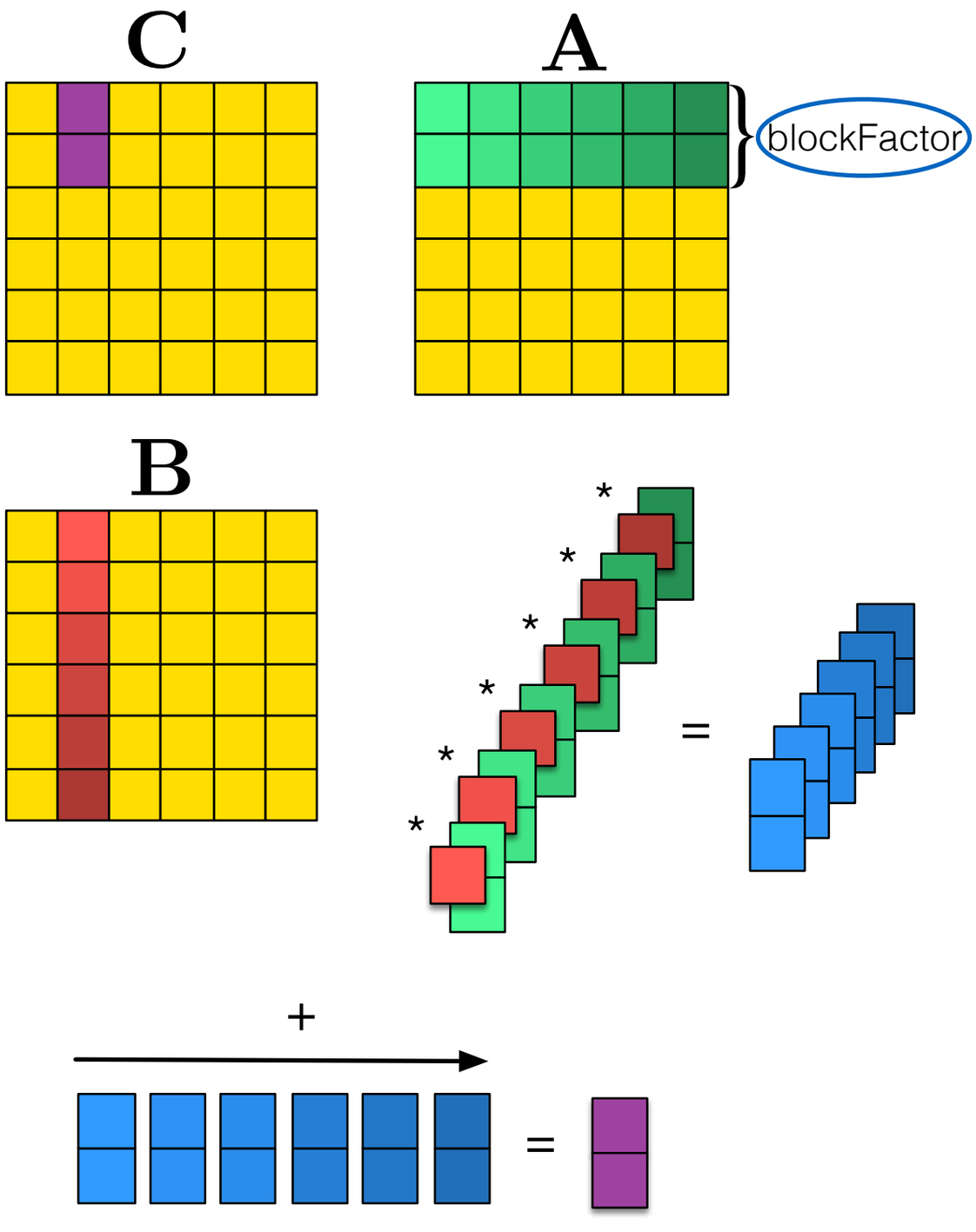
- **Complex analysis** Proving the optimisations are legal.
- **Conservative** Must always be correct.
- **No obvious defaults for parameters** Good tile and block sizes depend on hardware capabilities.

Register Blocking



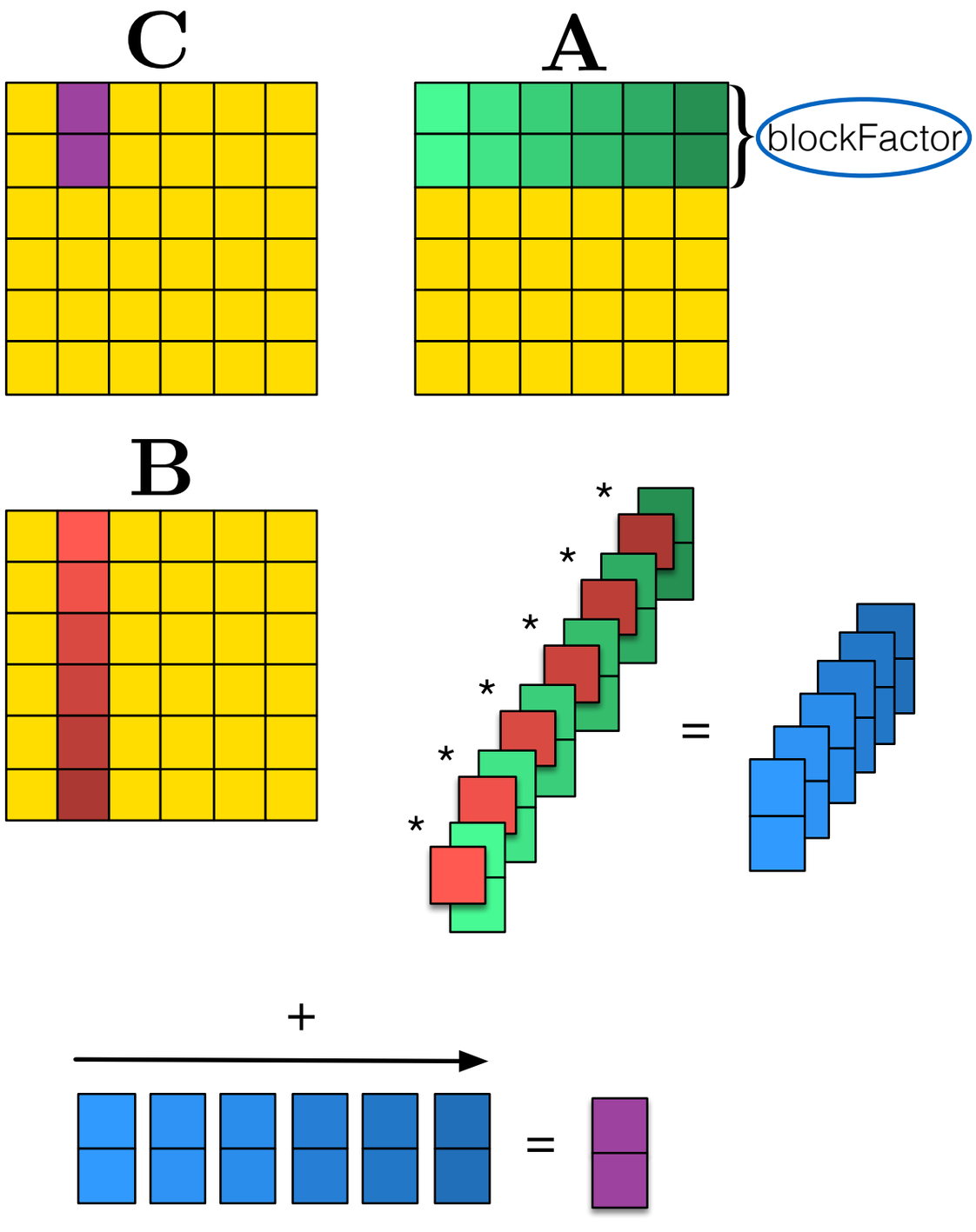

```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B,
4    global float* C, int K, int M, int N)
5  {
6    float acc[blockFactor];
7
8    for (int glb_id_1 = get_global_id(1);
9          glb_id_1 < M / blockFactor;
10         glb_id_1 += get_global_size(1)) {
11     for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
12         glb_id_0 += get_global_size(0)) {
13
14       for (int i = 0; i < K; i += 1)
15         float temp = B[i * N + glb_id_0];
16         for (int j = 0; j < blockFactor; j+= 1)
17           acc[j] +=
18             A[blockFactor * glb_id_1 * K + j * K + i]
19               * temp;
20
21       for (int j = 0; j < blockFactor; j += 1)
22         C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
23           = acc[j];
24     }
25   }
26 }
```

Register Blocking




```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B,
4    global float* C, int K, int M, int N)
5  {
6    float acc[blockFactor];
7
8    for (int glb_id_1 = get_global_id(1);
9          glb_id_1 < M / blockFactor;
10         glb_id_1 += get_global_size(1)) {
11     for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
12         glb_id_0 += get_global_size(0)) {
13
14       for (int i = 0; i < K; i += 1)
15         float temp = B[i * N + glb_id_0];
16         for (int j = 0; j < blockFactor; j+= 1)
17           acc[j] +=
18             A[blockFactor * glb_id_1 * K + j * K + i]
19               * temp;
20
21       for (int j = 0; j < blockFactor; j += 1)
22         C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
23           = acc[j];
24     }
25   }
26 }
```

Register Blocking




```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B,
4    global float* C, int K, int M, int N)
5  {
6    float acc[blockFactor];
7
8    for (int glb_id_1 = get_global_id(1);
9          glb_id_1 < M / blockFactor;
10         glb_id_1 += get_global_size(1)) {
11     for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
12         glb_id_0 += get_global_size(0)) {
13
14       for (int i = 0; i < K; i += 1)
15         float temp = B[i * N + glb_id_0];
16         for (int j = 0; j < blockFactor; j+= 1)
17           acc[j] +=
18             A[blockFactor * glb_id_1 * K + j * K + i]
19               * temp;
20
21       for (int j = 0; j < blockFactor; j += 1)
22         C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
23           = acc[j];
24     }
25   }
26 }
```

Register Blocking

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

$$Map(f) \Rightarrow Join() \circ Map(Map(f)) \circ Split(k)$$

Register Blocking

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

$$
\begin{array}{l}
Join() \circ Map(rowsA \mapsto \\
\quad Map(\overrightarrow{rowA} \mapsto \\
\quad\quad Map(\overrightarrow{colB} \mapsto \\
\quad\quad\quad Reduce(+) \circ Map(*) \\
\quad\quad\quad\quad \$\ Zip(\overrightarrow{rowA}, \overrightarrow{colB}) \\
\quad\quad) \circ Transpose()\ \$\ \mathbf{B} \\
\quad)\ \$\ rowsA \\
) \circ Split(blockFactor)\ \$\ \mathbf{A}
\end{array}
$$

$$Map(f) \Rightarrow Join() \circ Map(Map(f)) \circ Split(k)$$

Register Blocking

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

$$
\begin{aligned}
Map(a \mapsto Map(b \mapsto f(a,b))) \Rightarrow \\
Transpose() \circ Map(b \mapsto Map(a \mapsto f(a,b)))
\end{aligned}
$$

Register Blocking

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

$$
\Rightarrow
$$

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

$$
\begin{aligned}
Map(a \mapsto Map(b \mapsto f(a,b))) \Rightarrow \\
Transpose() \circ Map(b \mapsto Map(a \mapsto f(a,b)))
\end{aligned}
$$

Register Blocking

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

$$Map(f \circ g) \Rightarrow Map(f) \circ Map(g)$$

Register Blocking

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

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

$$Map(f \circ g) \Rightarrow Map(f) \circ Map(g)$$

Register Blocking

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

$$
\begin{aligned}
Map(Reduce(f)) \Rightarrow \\
Transpose() \circ Reduce(Map(f) \circ Zip())
\end{aligned}
$$

Register Blocking

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

$$
\begin{aligned}
&Join() \circ Map(rowsA \mapsto \\
&\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
&\qquad \underline{Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{next}) \mapsto} \\
&\qquad\quad Map(+) \$ Zip(\overrightarrow{acc}, \overrightarrow{next}) \\
&\qquad) \circ Transpose() \circ Map(\overrightarrow{rowA} \mapsto \\
&\qquad\quad Map(*) \$ Zip(\overrightarrow{rowA}, \overrightarrow{colB}) \\
&\qquad) \$ rowsA \\
&\quad) \circ Transpose() \$ \mathbf{B} \\
&) \circ Split(blockFactor) \$ \mathbf{A}
\end{aligned}
$$

$$
\begin{aligned}
&Map(Reduce(f)) \Rightarrow \\
&Transpose() \circ Reduce(Map(f) \circ Zip())
\end{aligned}
$$

Register Blocking

$$
\begin{aligned}
&Join() \circ Map(rowsA \mapsto \\
&\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
&\qquad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{next}) \mapsto \\
&\qquad\quad Map(+)\ \$\ Zip(\overrightarrow{acc}, \overrightarrow{next}) \\
&\qquad) \circ Transpose() \circ Map(\overrightarrow{rowA} \mapsto \\
&\qquad\quad Map(*)\$\ Zip(\overrightarrow{rowA}, \overrightarrow{colB}) \\
&\qquad) \$\ rowsA \\
&\quad) \circ Transpose()\ \$\ \mathbf{B} \\
&) \circ Split(blockFactor)\ \$\ \mathbf{A}
\end{aligned}
$$

$$
\begin{aligned}
Map(Map(f)) \Rightarrow \\
Transpose() \circ Map(Map(f)) \circ Transpose()
\end{aligned}
$$

Register Blocking

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

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

$$
\begin{aligned}
&Map(Map(f)) \Rightarrow \\
&Transpose() \circ Map(Map(f)) \circ Transpose()
\end{aligned}
$$

Register Blocking

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

$$Transpose() \circ Transpose() \Rightarrow id$$

Register Blocking

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

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

$$Transpose() \circ Transpose() \Rightarrow id$$

Register Blocking

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

$$
\begin{aligned}
Reduce(f) \circ Map(g) &\Rightarrow \\
Reduce((acc, x) &\mapsto f(acc, g(x))
\end{aligned}
$$

Register Blocking

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

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

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

Register Blocking

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

$$Map(f) \circ Map(g) \Rightarrow Map(f \circ g)$$

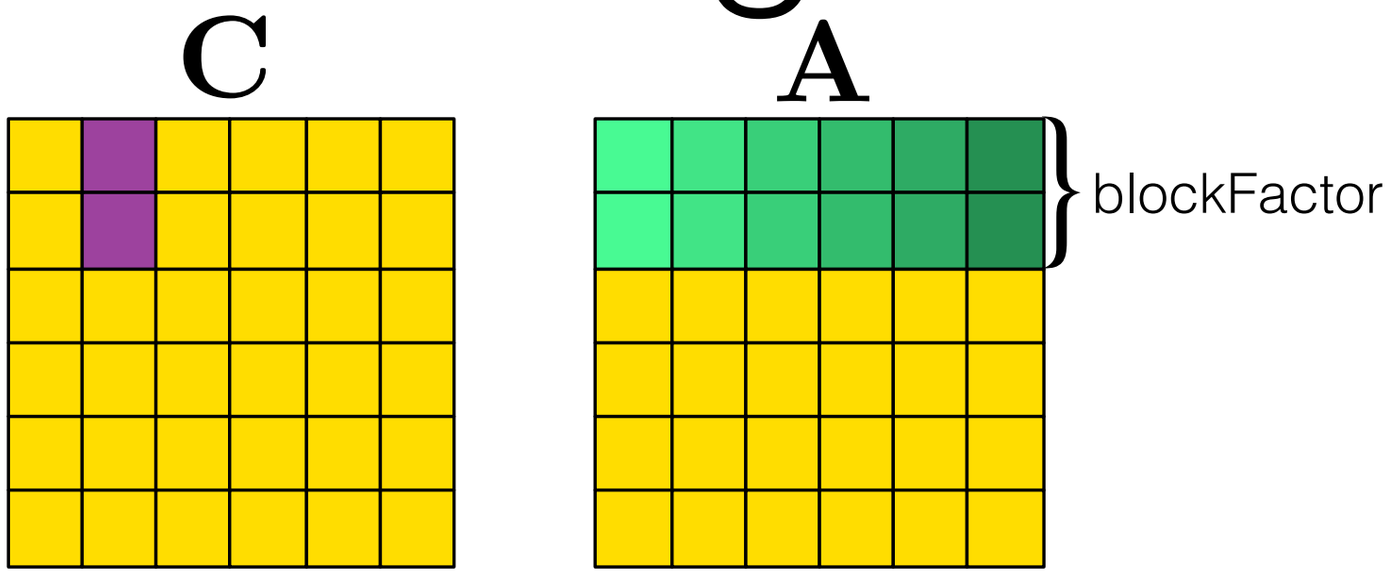
Register Blocking

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

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

$$Map(f) \circ Map(g) \Rightarrow Map(f \circ g)$$

Register Blocking



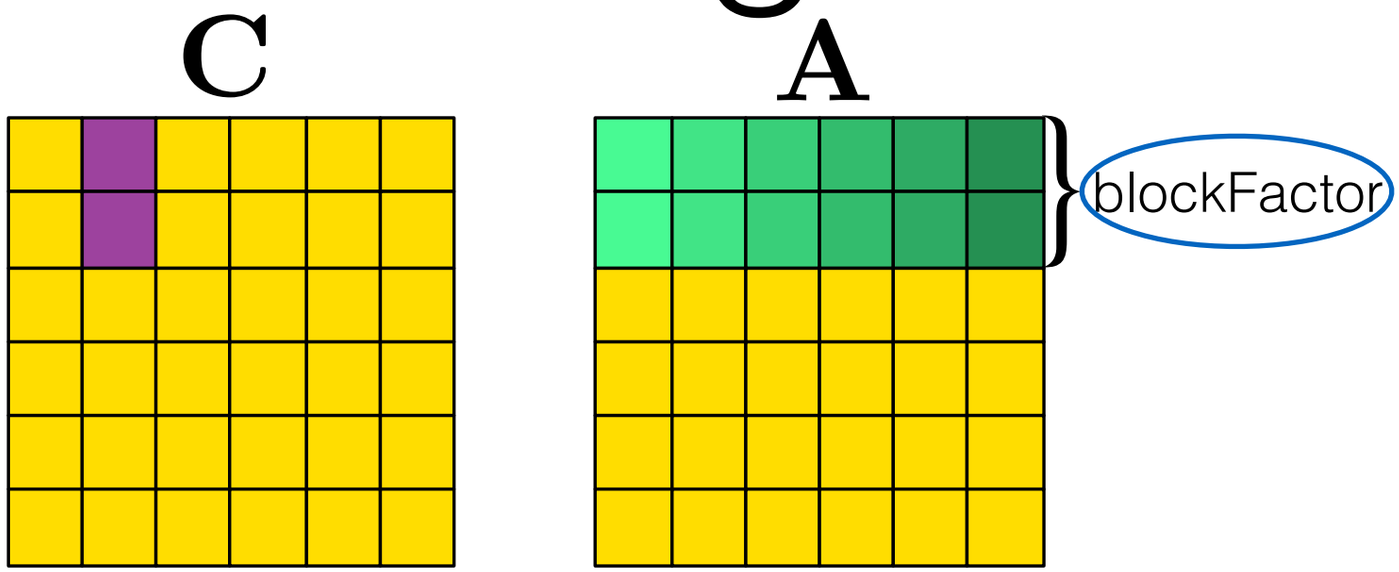
Functional Representation

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

OpenCL

```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B,
4    global float* C, int K, int M, int N)
5  {
6    float acc[blockFactor];
7
8    for (int glb_id_1 = get_global_id(1);
9          glb_id_1 < M / blockFactor;
10         glb_id_1 += get_global_size(1)) {
11     for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
12         glb_id_0 += get_global_size(0)) {
13
14       for (int i = 0; i < K; i += 1)
15         float temp = B[i * N + glb_id_0];
16         for (int j = 0; j < blockFactor; j+= 1)
17           acc[j] +=
18             A[blockFactor * glb_id_1 * K + j * K + i]
19               * temp;
20
21       for (int j = 0; j < blockFactor; j += 1)
22         C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
23           = acc[j];
24     }
25   }
26 }
```

Register Blocking



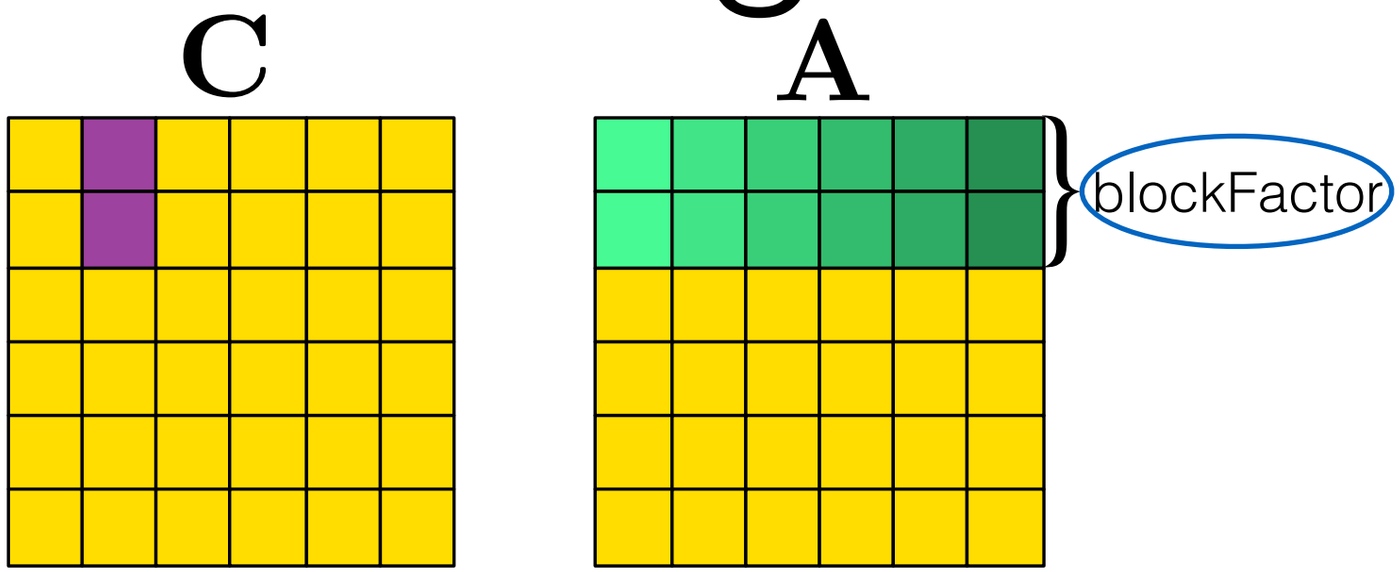
Functional Representation

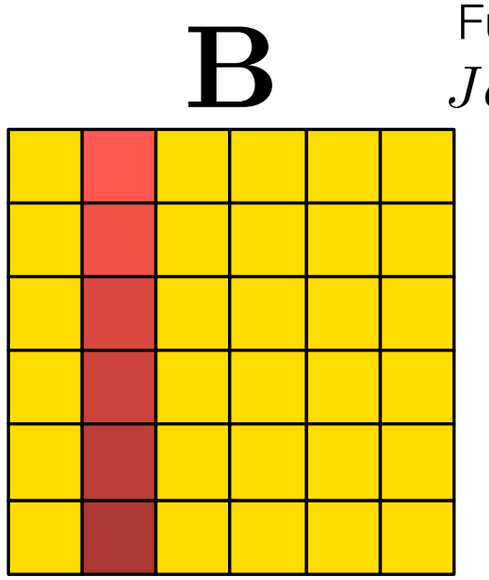
$$
\begin{aligned}
&Join() \circ Map(rowsA \mapsto \\
&\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
&\quad\quad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{pair}) \mapsto \\
&\quad\quad\quad Map(x \mapsto x\_0 + x\_1 * pair.\_1) \\
&\quad\quad\quad \$\ Zip(\overrightarrow{acc},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

OpenCL

```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B,
4    global float* C, int K, int M, int N)
5  {
6    float acc[blockFactor];
7
8    for (int glb_id_1 = get_global_id(1);
9          glb_id_1 < M / blockFactor;
10         glb_id_1 += get_global_size(1)) {
11     for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
12         glb_id_0 += get_global_size(0)) {
13
14       for (int i = 0; i < K; i += 1)
15         float temp = B[i * N + glb_id_0];
16         for (int j = 0; j < blockFactor; j+= 1)
17           acc[j] +=
18             A[blockFactor * glb_id_1 * K + j * K + i]
19               * temp;
20
21       for (int j = 0; j < blockFactor; j += 1)
22         C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
23           = acc[j];
24     }
25   }
26 }
```

Register Blocking





Functional Representation

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

OpenCL

```
1  kernel void KERNEL(
2    const global float* restrict A,
3    const global float* restrict B,
4    global float* C, int K, int M, int N)
5  {
6    float acc[blockFactor];
7
8    for (int glb_id_1 = get_global_id(1);
9          glb_id_1 < M / blockFactor;
10         glb_id_1 += get_global_size(1)) {
11     for (int glb_id_0 = get_global_id(0); glb_id_0 < N;
12         glb_id_0 += get_global_size(0)) {
13
14       for (int i = 0; i < K; i += 1)
15         float temp = B[i * N + glb_id_0];
16         for (int j = 0; j < blockFactor; j+= 1)
17           acc[j] +=
18             A[blockFactor * glb_id_1 * K + j * K + i]
19               * temp;
20
21       for (int j = 0; j < blockFactor; j += 1)
22         C[blockFactor * glb_id_1 * N + j * N + glb_id_0]
23           = acc[j];
24     }
25   }
26 }
```

Combining Optimisations

$$
\begin{aligned}
&\mathbf{A} * \mathbf{B} = \\
&Map(\overrightarrow{rowA} \mapsto \\
&\quad Map(\overrightarrow{colB} \mapsto \\
&\qquad DotProduct(\overrightarrow{rowA}, \overrightarrow{colB}) \\
&\quad) \circ Transpose()\ \$\ \mathbf{B} \\
&)\ \$\ \mathbf{A}
\end{aligned}
$$

80 rewrites

$$
\begin{aligned}
&(p239, p36 \mapsto \\
&\quad Join() \circ Map((p179 \mapsto \\
&\quad\quad Transpose() \circ Join() \circ Map((p70 \mapsto \\
&\quad\quad\quad Transpose() \circ Join() \circ Map((p20 \mapsto \\
&\quad\quad\quad\quad Transpose() \circ Map((p65 \mapsto \\
&\quad\quad\quad\quad\quad Transpose()(p65) \\
&\quad\quad\quad\quad)) \circ Transpose()(p20) \\
&\quad\quad\quad)) \circ Transpose() \circ Reduce((p75, p0 \mapsto \\
&\quad\quad\quad\quad Map((p164 \mapsto \\
&\quad\quad\quad\quad\quad Join() \circ Map((p81 \mapsto \\
&\quad\quad\quad\quad\quad\quad Reduce((p136, p90 \mapsto \\
&\quad\quad\quad\quad\quad\quad\quad Map((p163 \mapsto \\
&\quad\quad\quad\quad\quad\quad\quad\quad Get(0)(p163) + Get(1)(p163) * Get(1)(p90) \\
&\quad\quad\quad\quad\quad\quad\quad)) \circ Zip(2)(p136, Get(0)(p90)) \\
&\quad\quad\quad\quad\quad\quad))(Get(0)(p81), Zip(2)(Transpose() \circ Get(1)(p164), Get(1)(p81))) \\
&\quad\quad\quad\quad\quad)) \circ Zip(2)(Get(0)(p164), Get(1)(p0)) \\
&\quad\quad\quad\quad)) \circ Zip(2)(p75, Split(blockFactor) \circ Transpose() \circ Get(0)(p0)) \\
&\quad\quad\quad))(Zip(2)(Split(sizeK) \circ Transpose()(p179), p70)) \\
&\quad\quad)) \circ Transpose() \circ Map((p4 \mapsto \\
&\quad\quad\quad Split(sizeN) \circ Transpose()(p4) \\
&\quad\quad)) \circ Split(sizeK)(p36) \\
&\quad)) \circ Split(sizeM)(p239) \\
&)
\end{aligned}
$$

How do we apply these optimisations?

Exploration Strategy

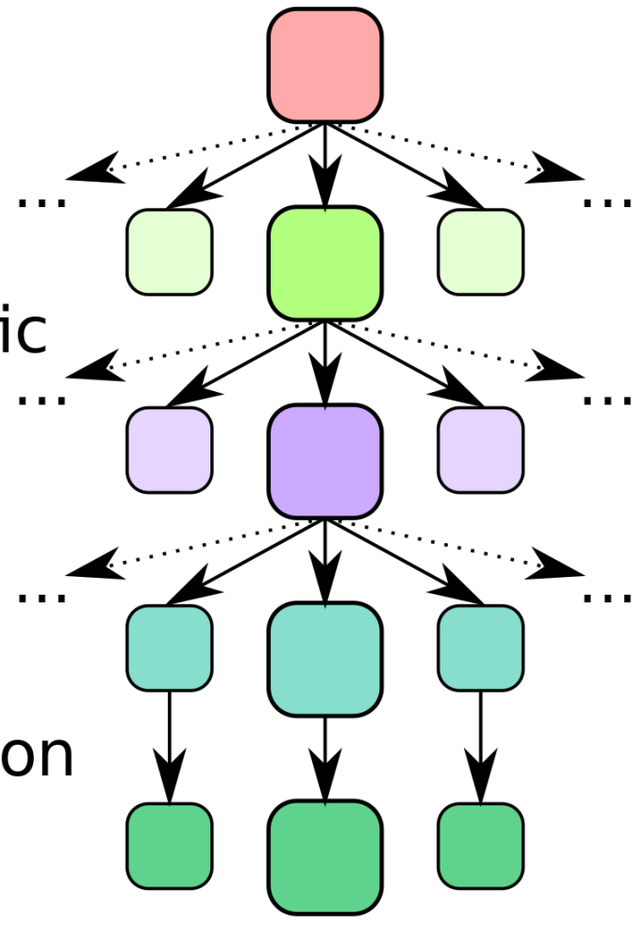
Phases:

Algorithmic Exploration

OpenCL specific Exploration

Parameter Exploration

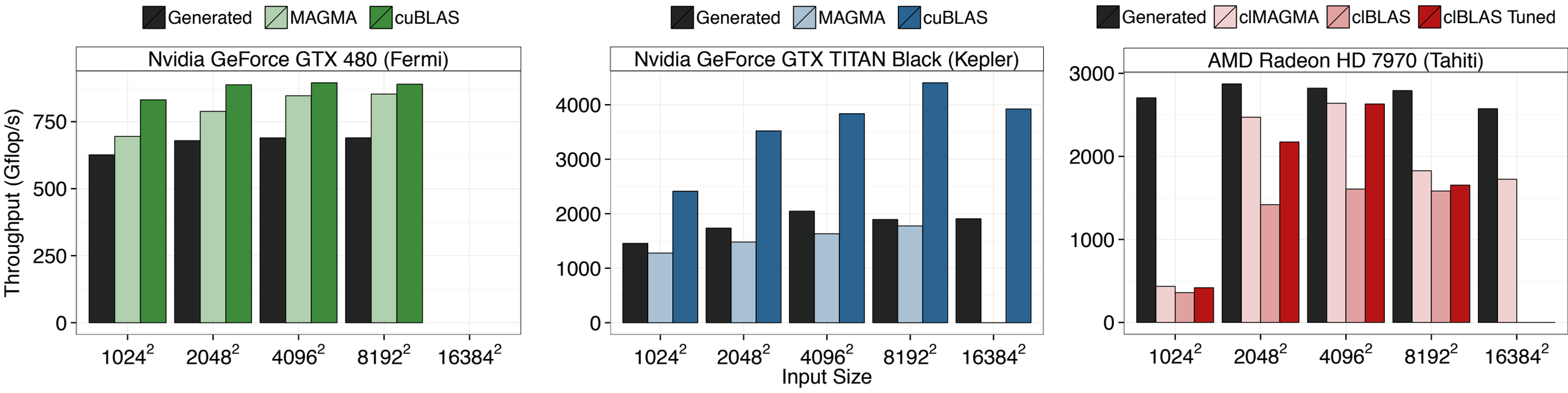
Code Generation



Program Variants:

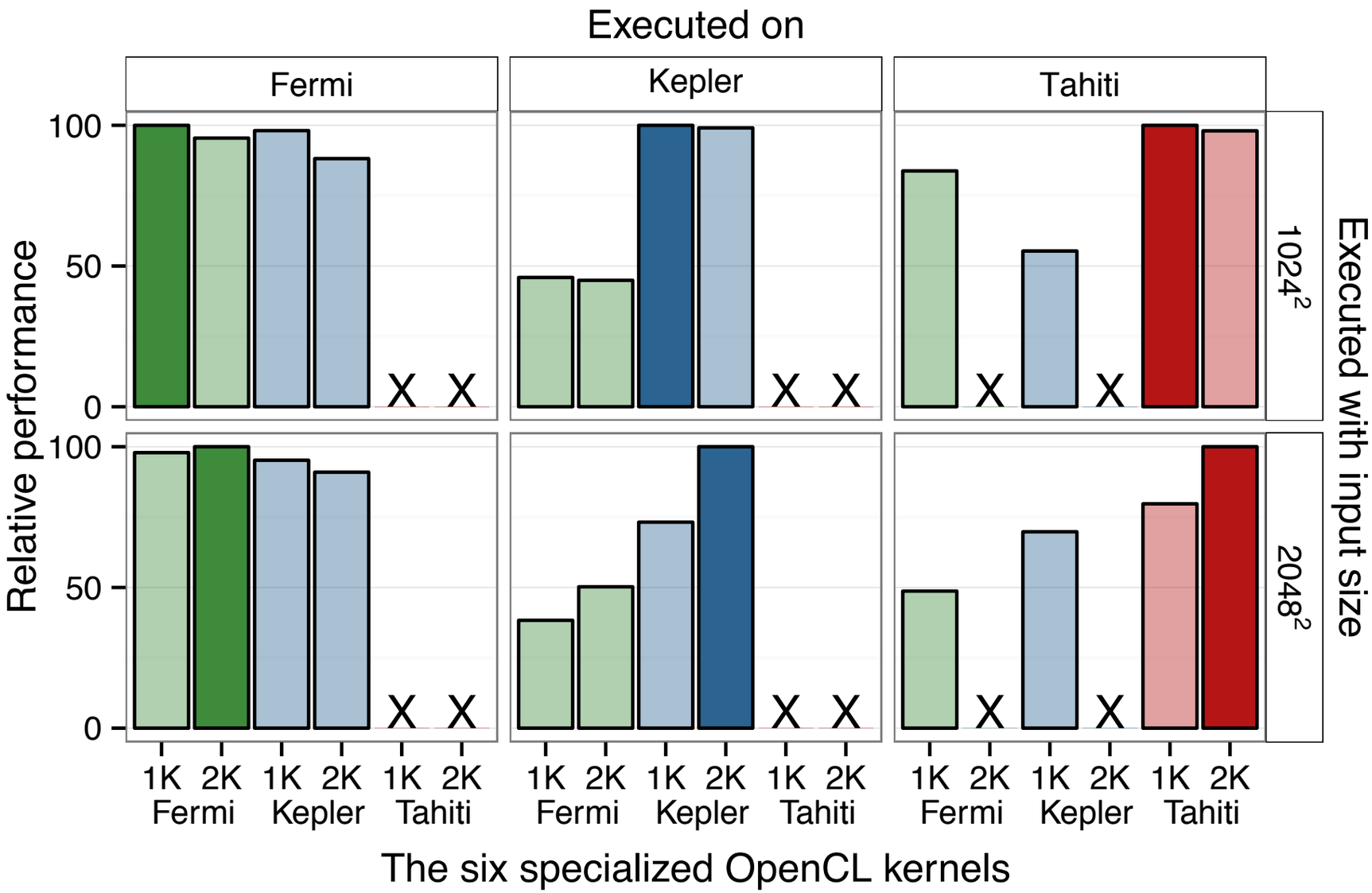
High-Level Program	1
Algorithmic Rewritten Program	8
OpenCL Specific Program	760
Fully Specialized Program	46,000
OpenCL Code	46,000

Performance Results



Performance close or better than hand-tuned MAGMA library

Performance Portability



Performance is not portable
across architectures and input sizes

Conclusion

- OpenCL code is not performance portable
- Using a functional approach along with rewrite rules we can achieve performance portability
- Performance of matrix multiplication on par with tuned OpenCL code

Toomas Remmelg - toomas.remmelg@ed.ac.uk

Supported by:

Google Oracle Labs