A Functional Approach to Performance Portable GPU Code Generation

A Case Study on Matrix Multiplication

Toomas Remmelg, Thibaut Lutz, Michel Steuwer,
Christophe Dubach

THE UNIVERSITY
of EDINBURGH

The Problem

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

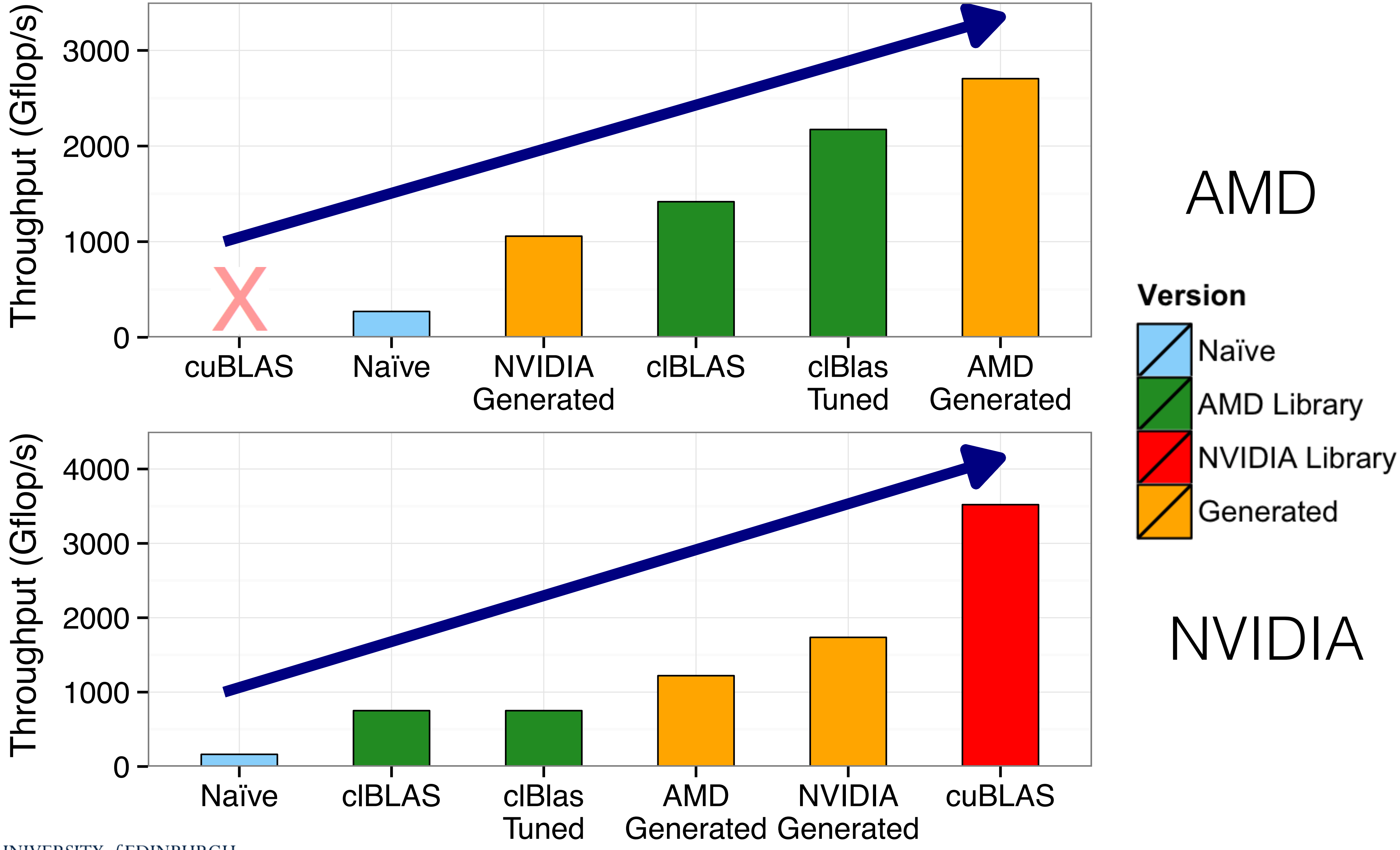
CPU

GPU

Accelerator

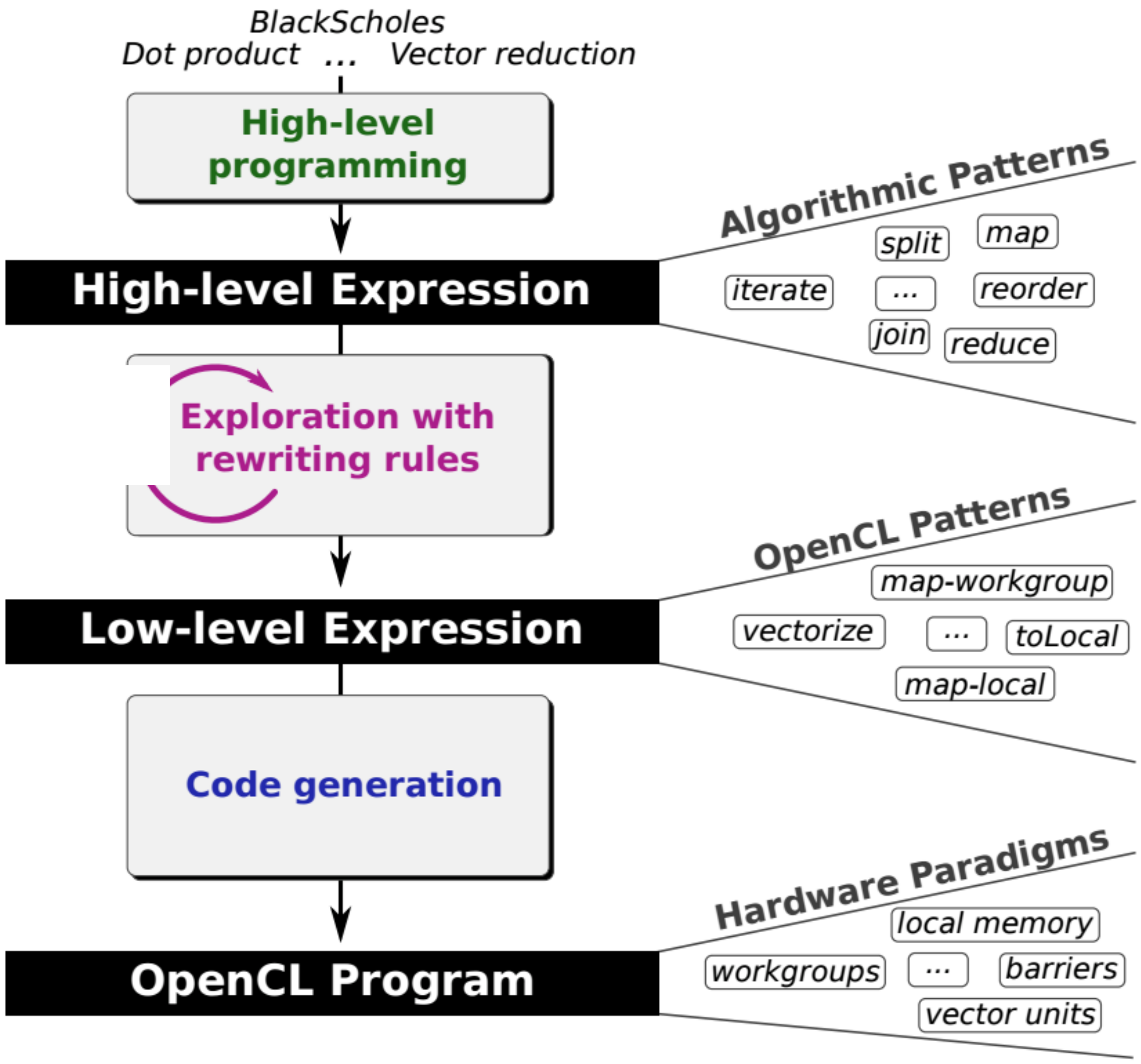
FPGA

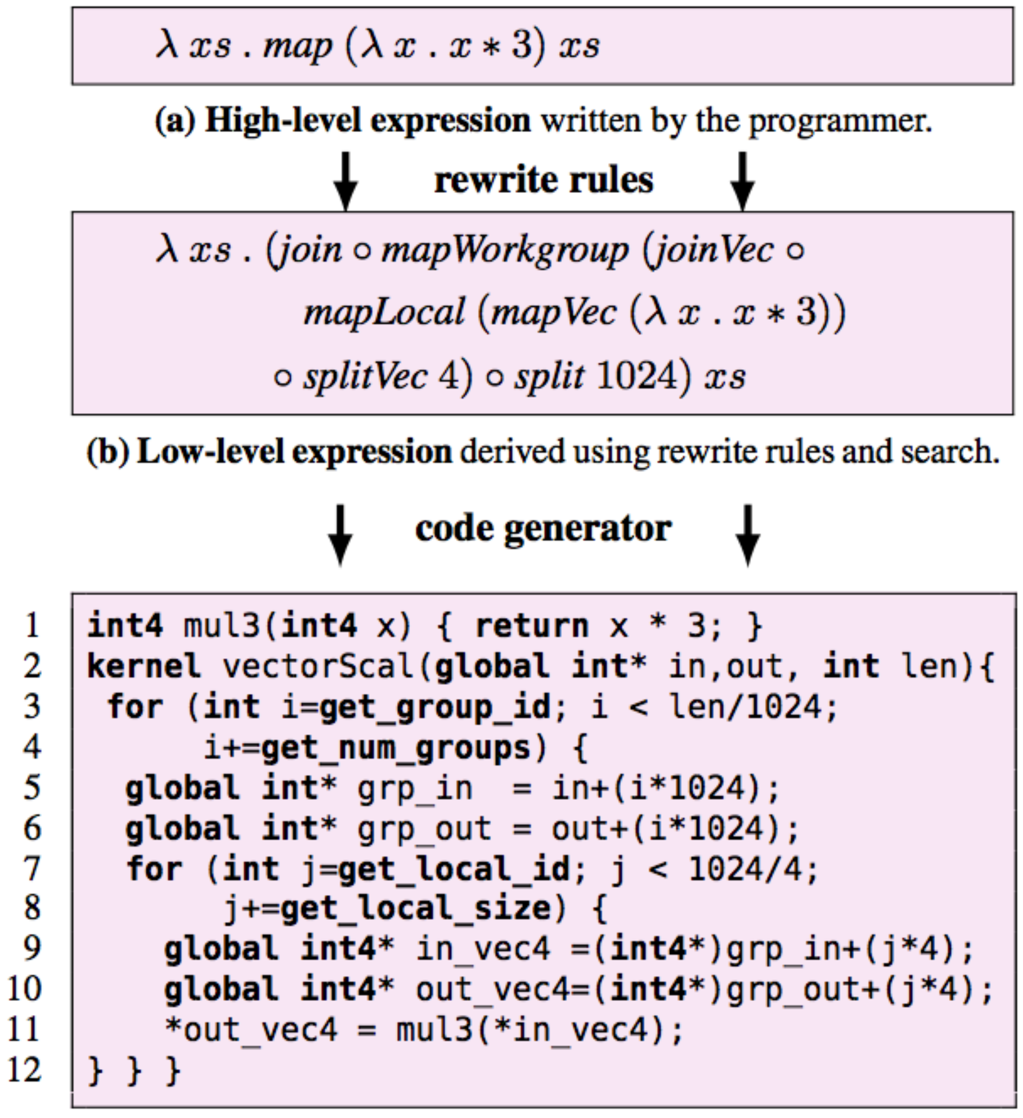
Performance Portability of Matrix Multiplication



THE UNIVERSITY *of* EDINBURGH
informatics

Generating Performance Portable Code Using Rewrite Rules





$\lambda\, xs\, .\, map\ (\lambda\, x\, .\, x * 3)\ xs$

(a) High-level expression written by the programmer.

rewrite rules

$\lambda\, xs\, .\, (join \circ mapWorkgroup\ (joinVec \circ mapLocal\ (mapVec\ (\lambda\, x\, .\, x * 3)) \circ splitVec\ 4) \circ split\ 1024)\ xs$

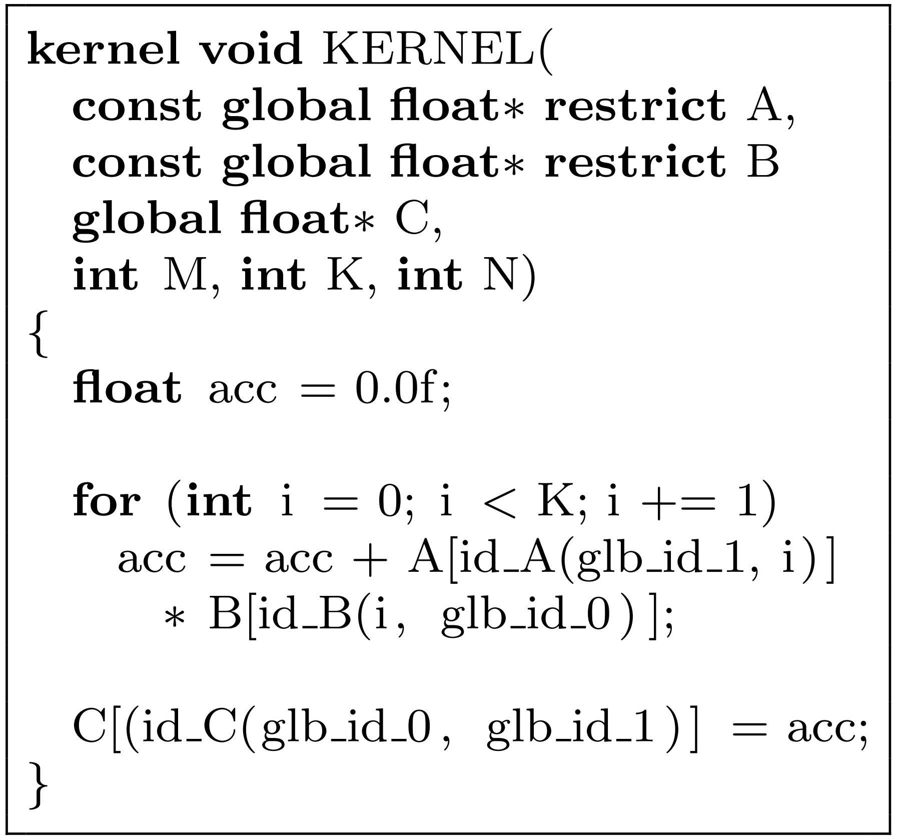
(b) Low-level expression derived using rewrite rules and search.

code generator

```
int4 mul3(int4 x) { return x * 3; }
kernel vectorScal(global int* in,out, int len){
 for (int i=get_group_id; i < len/1024;
      i+=get_num_groups) {
  global int* grp_in  = in+(i*1024);
  global int* grp_out = out+(i*1024);
  for (int j=get_local_id; j < 1024/4;
       j+=get_local_size) {
    global int4* in_vec4 =(int4*)grp_in+(j*4);
    global int4* out_vec4=(int4*)grp_out+(j*4);
    *out_vec4 = mul3(*in_vec4);
} } }
```

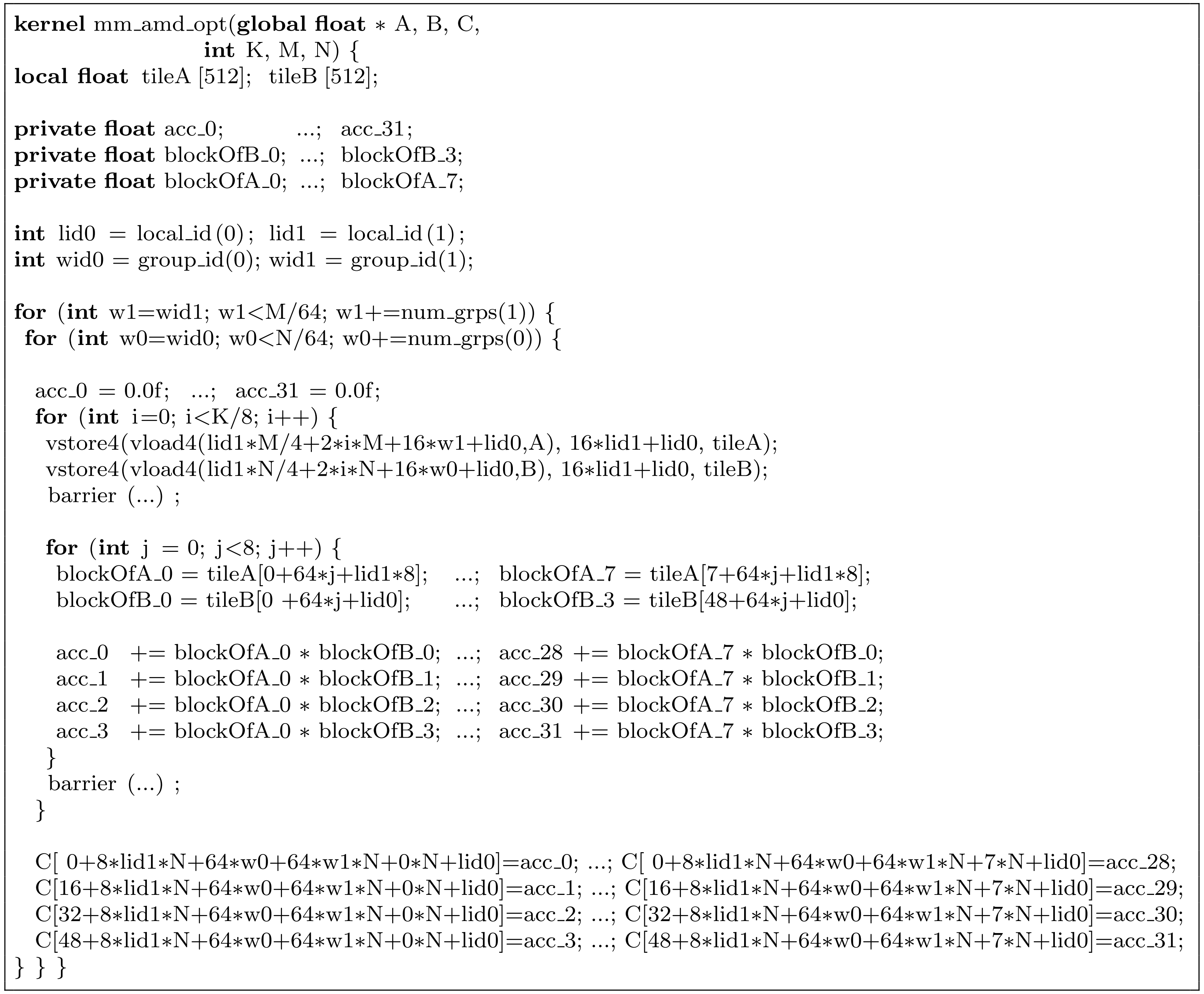
(c) OpenCL program produced by our code generator.

Matrix Multiplication in OpenCL



```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B
  global float* C,
  int M, int K, int N)
{
  float acc = 0.0f;

  for (int i = 0; i < K; i += 1)
    acc = acc + A[id_A(glb_id_1, i)]
      * B[id_B(i, glb_id_0)];

  C[(id_C(glb_id_0, glb_id_1)] = acc;
}
```

Naïve

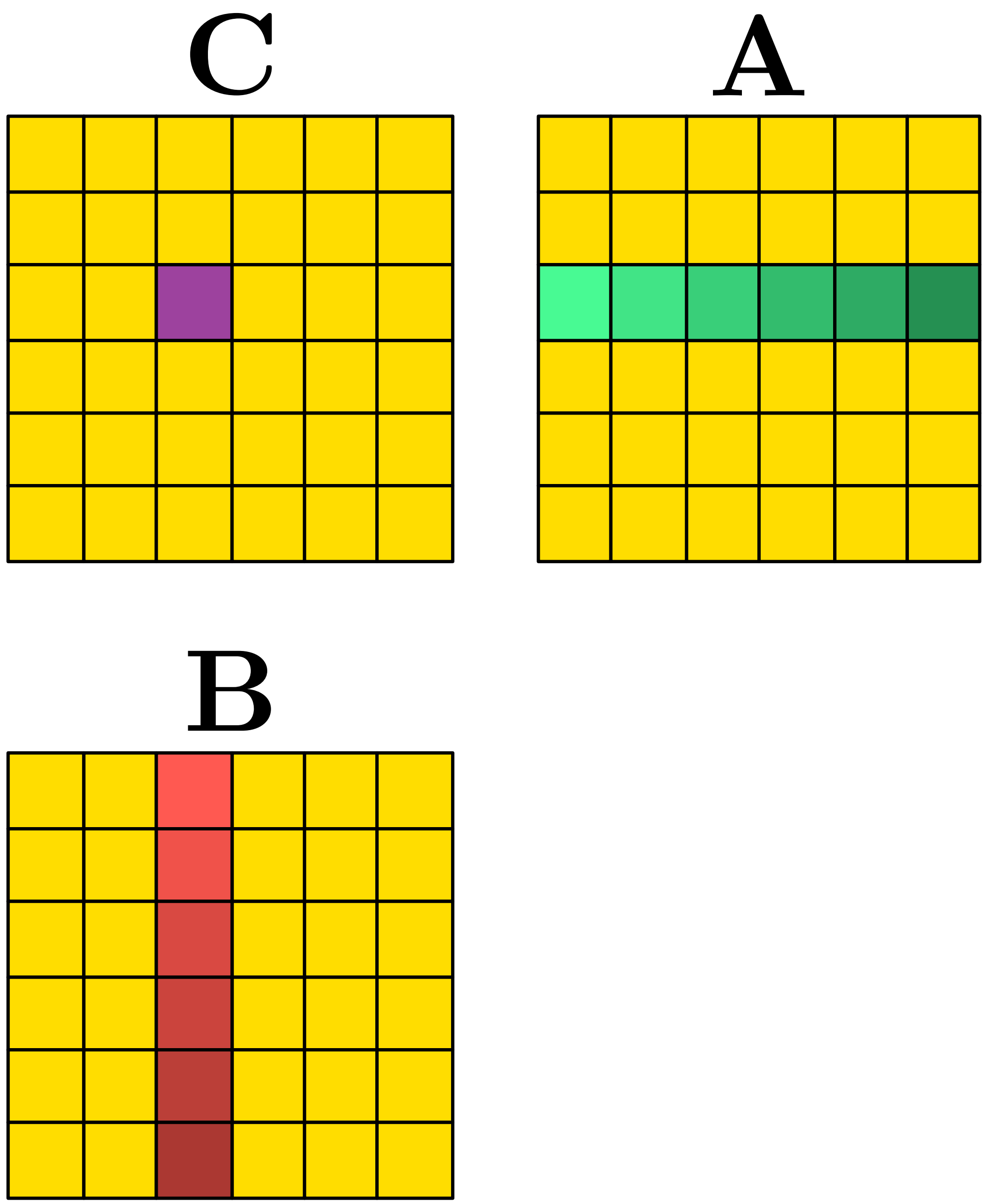


```
kernel mm_amd_opt(global float * A, B, C,
                  int K, M, N) {
local float tileA [512];  tileB [512];

private float acc_0;        ...;  acc_31;
private float blockOfB_0;  ...;  blockOfB_3;
private float blockOfA_0;  ...;  blockOfA_7;

int lid0 = local_id(0); lid1 = local_id(1);
int wid0 = group_id(0); wid1 = group_id(1);

for (int w1=wid1; w1<M/64; w1+=num_grps(1)) {
 for (int w0=wid0; w0<N/64; w0+=num_grps(0)) {

  acc_0 = 0.0f;  ...;  acc_31 = 0.0f;
  for (int i=0; i<K/8; i++) {
   vstore4(vload4(lid1*M/4+2*i*M+16*w1+lid0,A), 16*lid1+lid0, tileA);
   vstore4(vload4(lid1*N/4+2*i*N+16*w0+lid0,B), 16*lid1+lid0, tileB);
   barrier (...) ;

   for (int j = 0; j<8; j++) {
    blockOfA_0 = tileA[0+64*j+lid1*8];   ...;  blockOfA_7 = tileA[7+64*j+lid1*8];
    blockOfB_0 = tileB[0 +64*j+lid0];     ...;  blockOfB_3 = tileB[48+64*j+lid0];

    acc_0  += blockOfA_0 * blockOfB_0;  ...;  acc_28 += blockOfA_7 * blockOfB_0;
    acc_1  += blockOfA_0 * blockOfB_1;  ...;  acc_29 += blockOfA_7 * blockOfB_1;
    acc_2  += blockOfA_0 * blockOfB_2;  ...;  acc_30 += blockOfA_7 * blockOfB_2;
    acc_3  += blockOfA_0 * blockOfB_3;  ...;  acc_31 += blockOfA_7 * blockOfB_3;
   }
   barrier (...) ;
  }

  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;
  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;
  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;
  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;
} } }
```

Generated for AMD

THE UNIVERSITY *of* EDINBURGH
informatics

Functional Definition of Matrix Multiplication



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

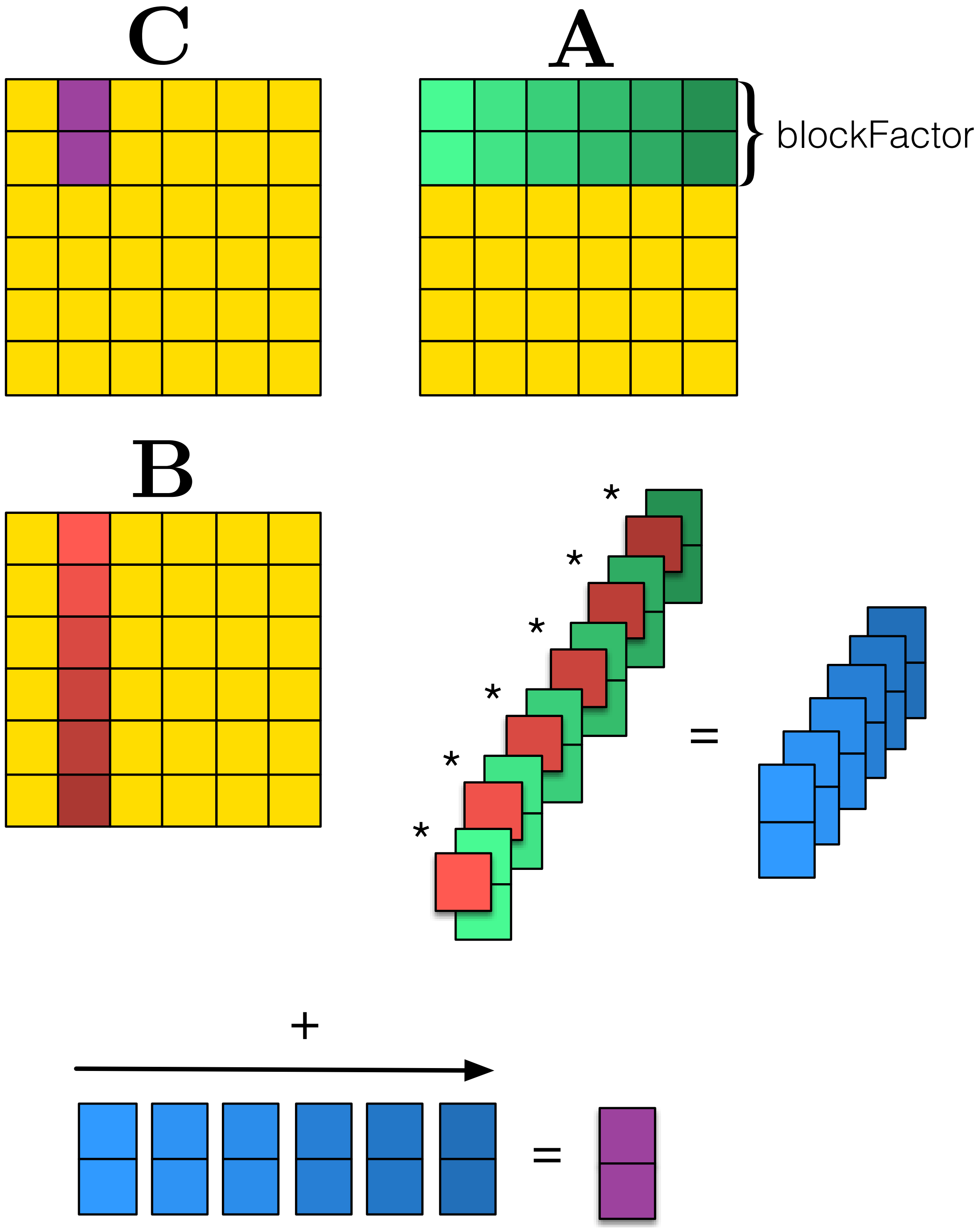
Traditional Optimisations

- **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?

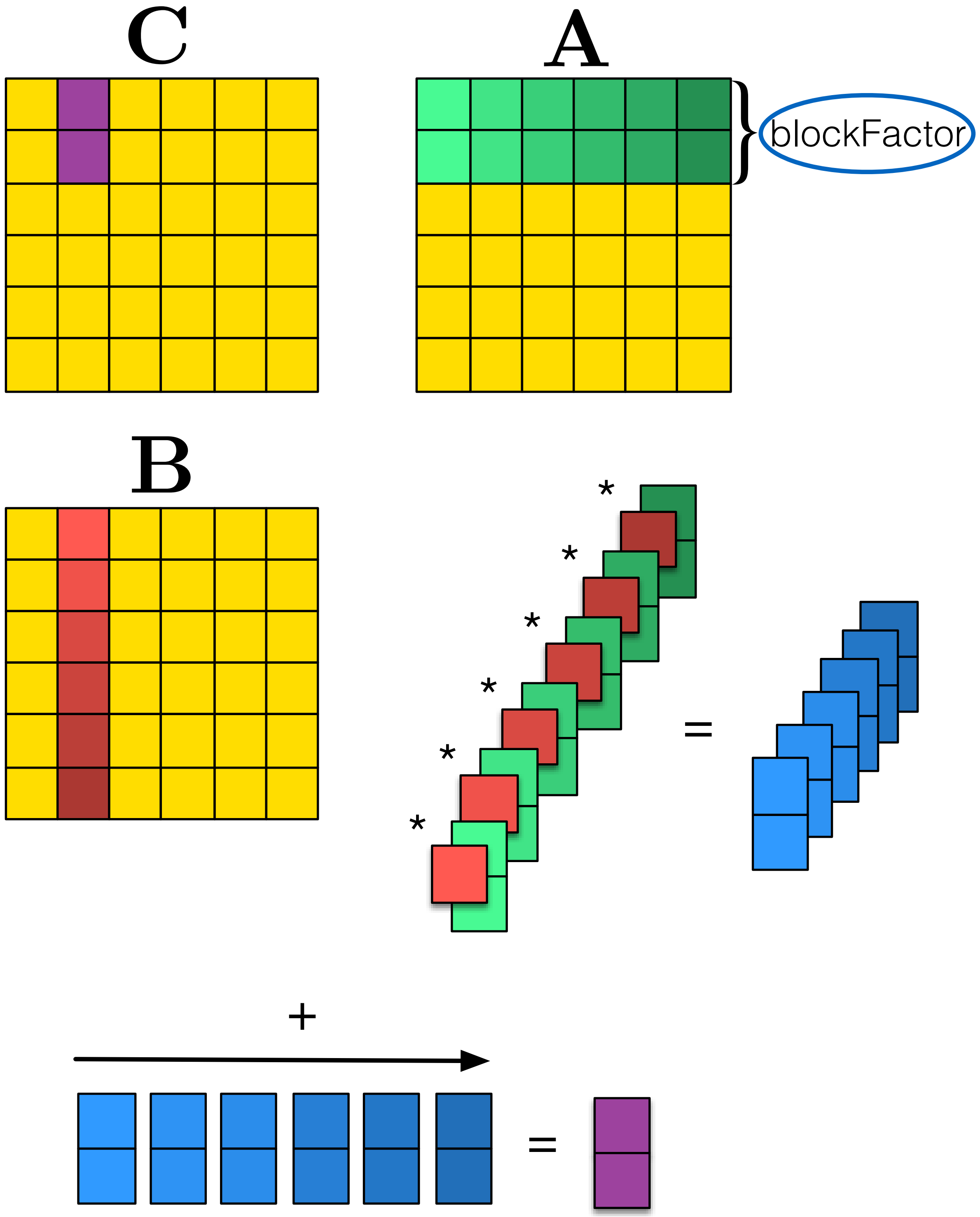
- **Complex analysis** Proving the optimisations are legal.
- **Conservative** Must always be correct.
- **No obvious defaults for parameters** Good tile and block sizes depend on hardware capabilities.

Register Blocking



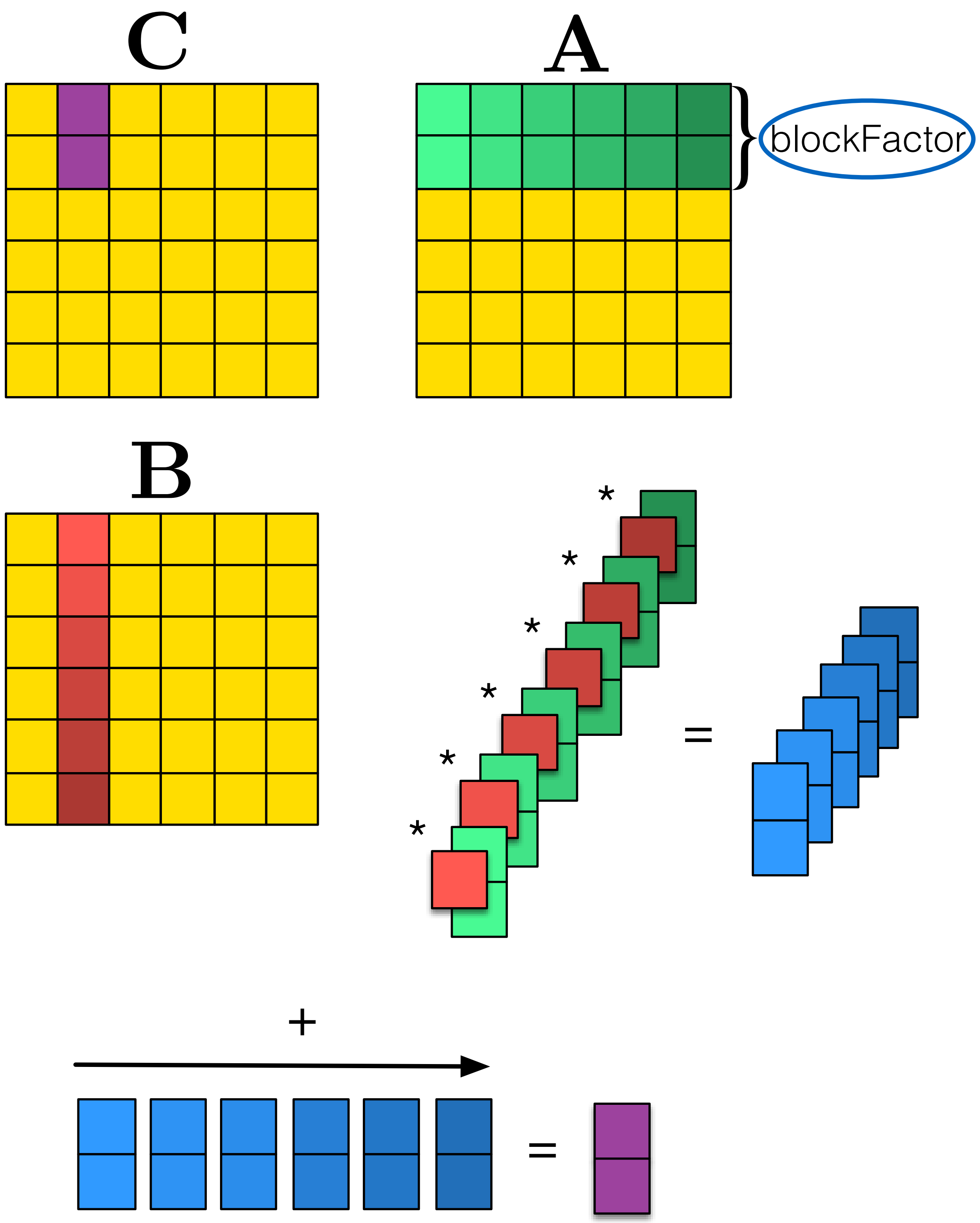
```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B,
  global float* C, int K, int M, int N)
{
  float acc[blockFactor];

  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];
    }
  }
}
```

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];
    }
  }
}
```

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];
    }
  }
}
```

Register Blocking as a Series of Rewrites

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

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

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

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

Register Blocking

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

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

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

Register Blocking

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

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

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

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

$$Map(Reduce(f)) \Rightarrow$$
$$Transpose() \circ Reduce(Map(f) \circ Zip())$$

Register Blocking

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

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

$$
Map(Reduce(f)) \Rightarrow
$$

$$
Transpose() \circ Reduce(Map(f) \circ Zip())
$$

Register Blocking

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

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

Register Blocking

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

$$\Rightarrow$$

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

$$\mathit{Map}(\mathit{Map}(f)) \Rightarrow$$

$$\mathit{Transpose}() \circ \mathit{Map}(\mathit{Map}(f)) \circ \mathit{Transpose}()$$

Register Blocking

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

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

Register Blocking

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

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

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

Register Blocking

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

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

Register Blocking

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

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

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

Register Blocking

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

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

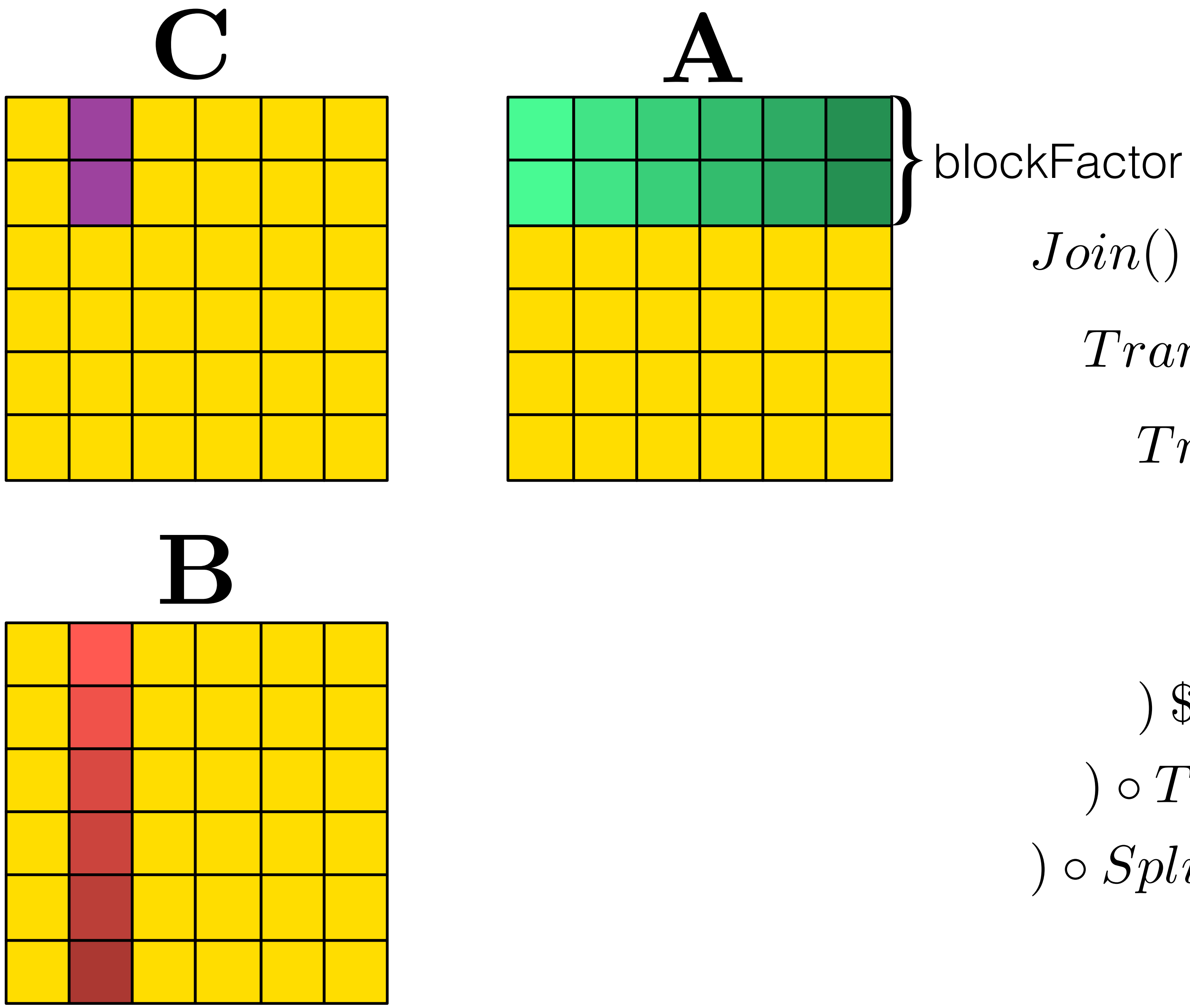
Register Blocking

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

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

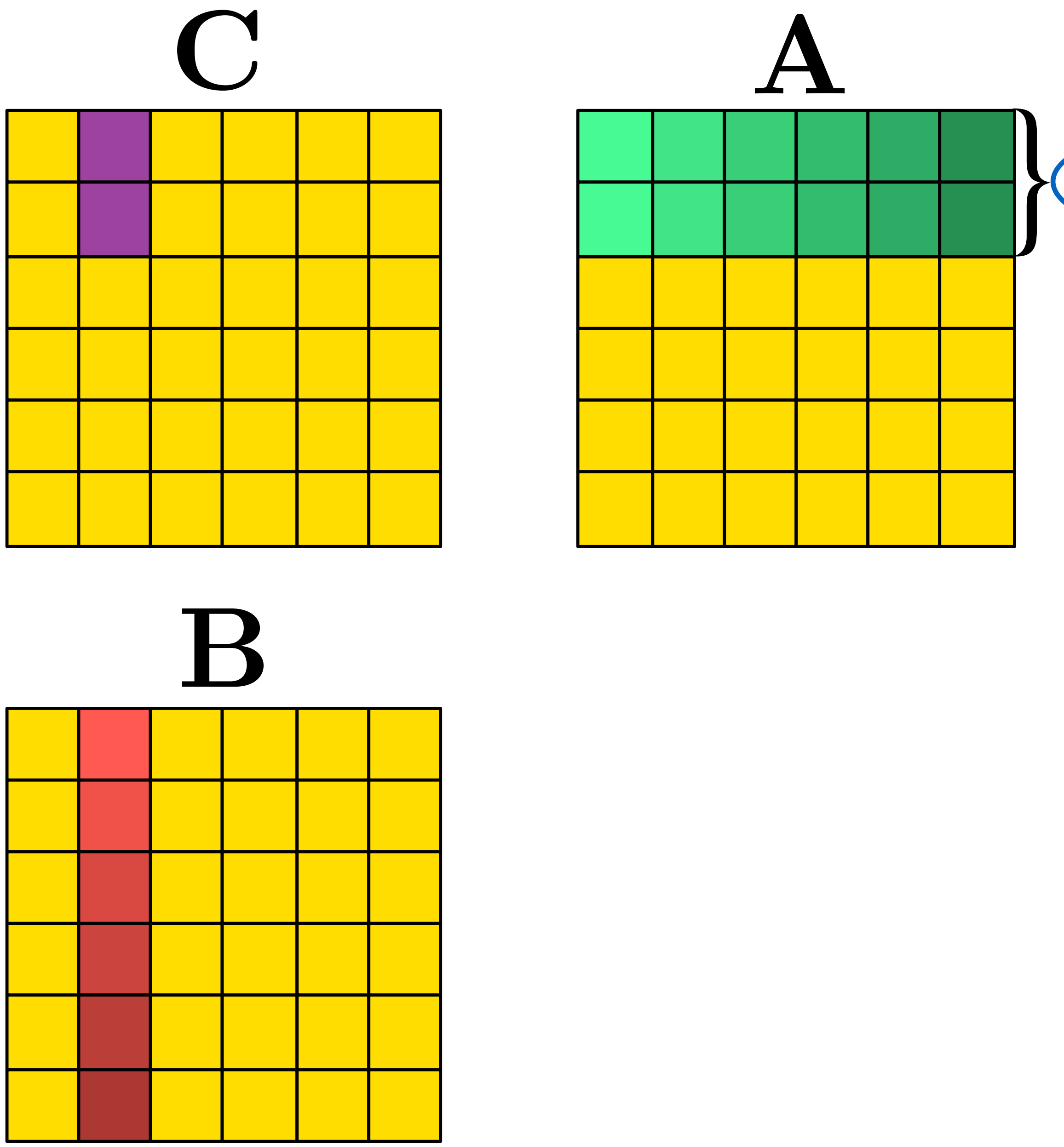
$$Map(f) \circ Map(g) \Rightarrow Map(f \circ g)$$

Register Blocking



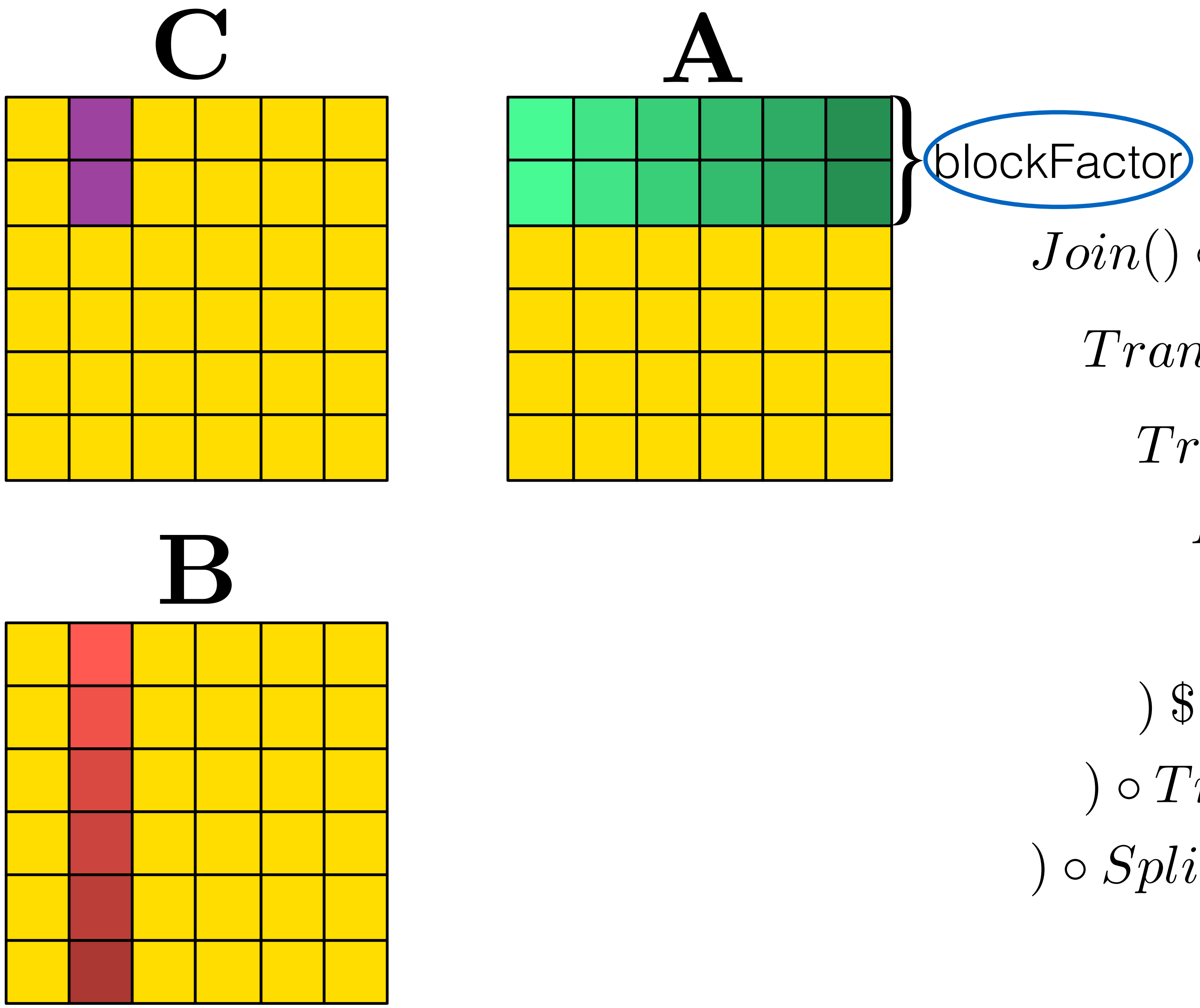
$$
\begin{array}{l}
Join() \circ Map(rowsA \mapsto \\
\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
\quad\quad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{pair}) \mapsto \\
\quad\quad\quad Map(x \mapsto x\_0 + x\_1 * pair.\_1) \\
\quad\quad\quad\quad \$\ Zip(\overrightarrow{acc},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

Register Blocking



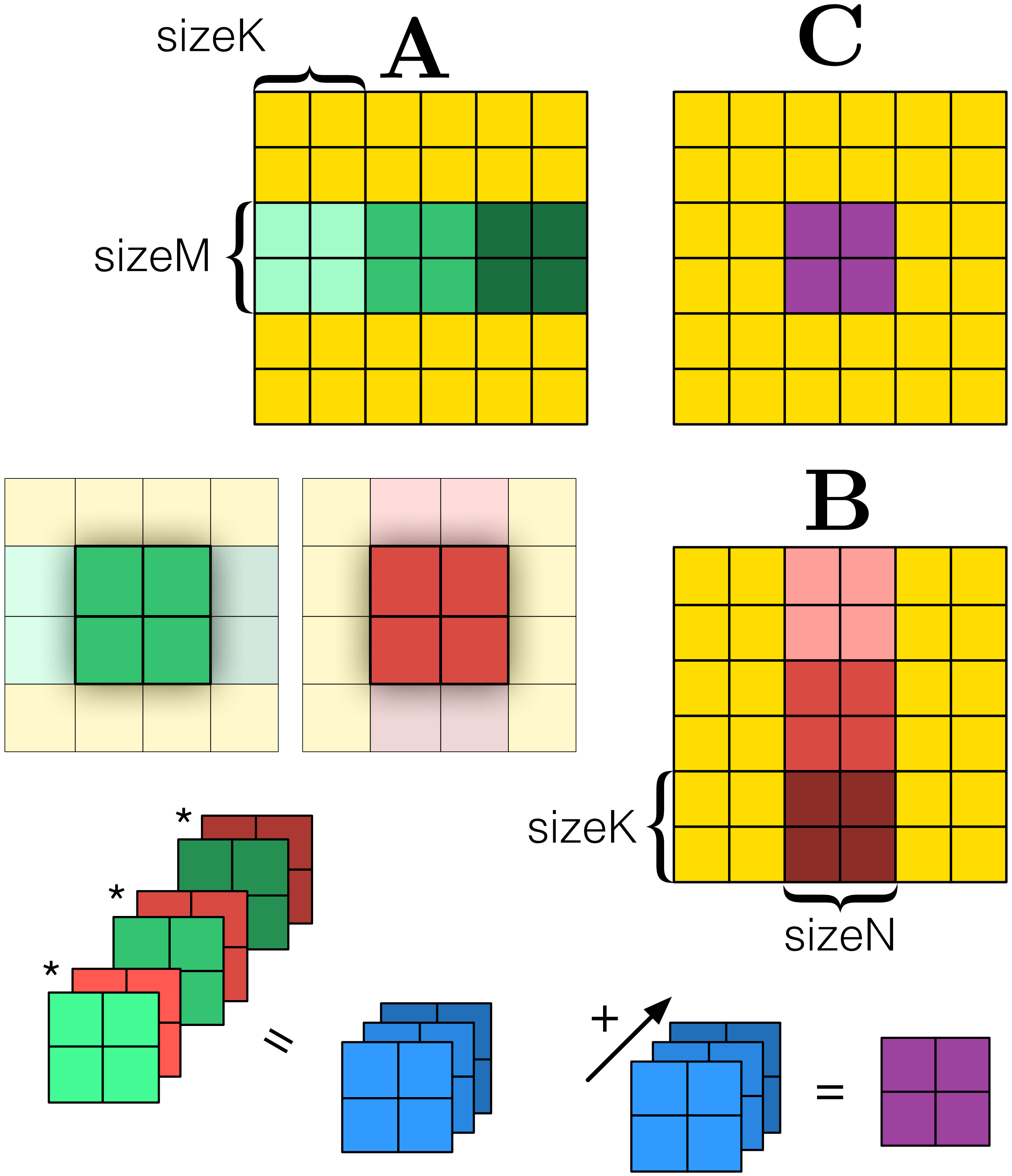
$$\begin{array}{l}
Join() \circ Map(rowsA \mapsto \\
\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
\quad\quad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{pair}) \mapsto \\
\quad\quad\quad Map(x \mapsto x\_0 + x\_1 * pair.\_1) \\
\quad\quad\quad\quad \$\ Zip(\overrightarrow{acc},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

Register Blocking



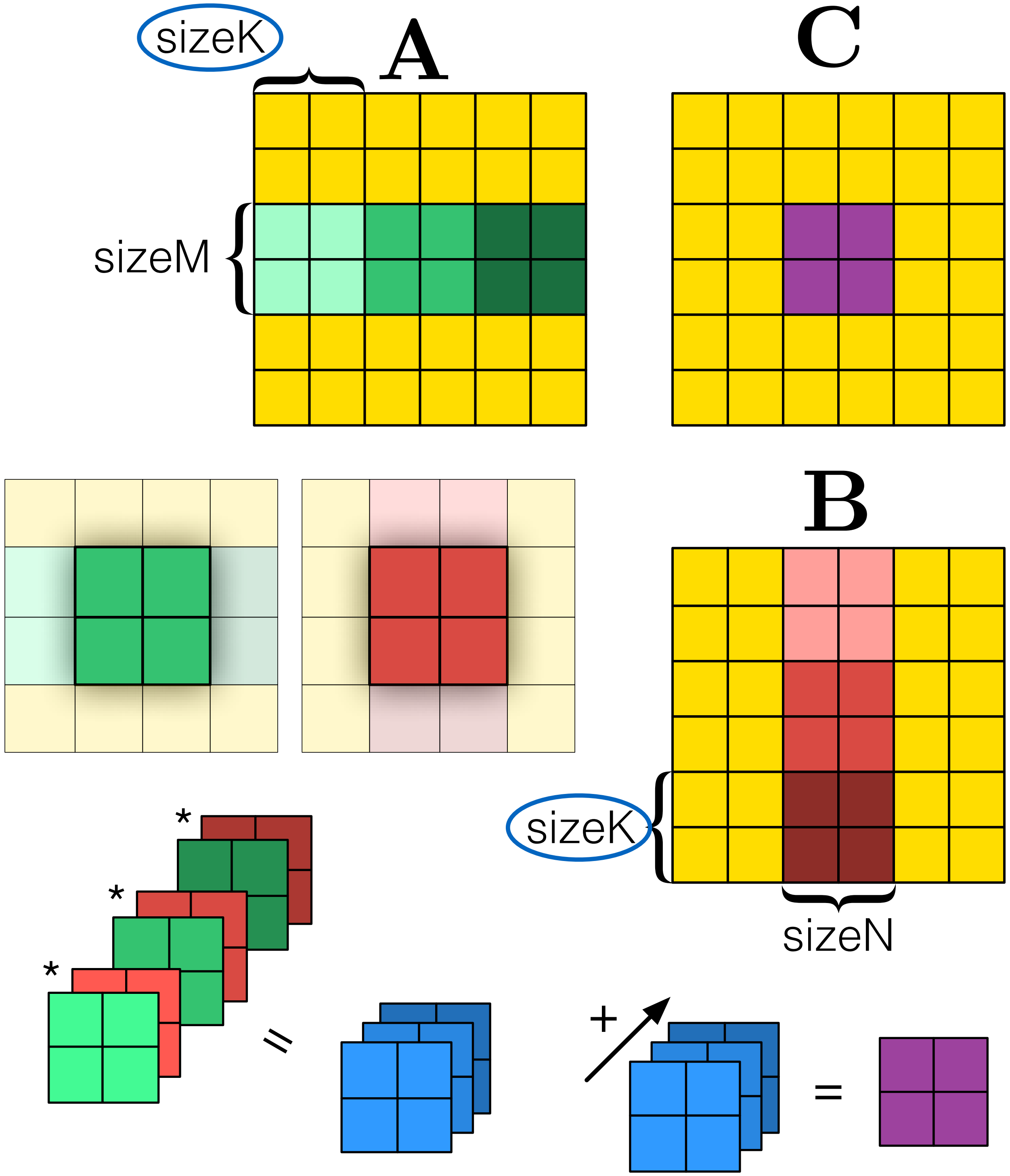
$$
\begin{aligned}
&Join() \circ Map(rowsA \mapsto \\
&\quad Transpose() \circ Map(\overrightarrow{colB} \mapsto \\
&\qquad Transpose() \circ Reduce((\overrightarrow{acc}, \overrightarrow{pair}) \mapsto \\
&\qquad\quad Map(x \mapsto x\_0 + x\_1 * pair.\_1) \\
&\qquad\qquad \$\ Zip(\overrightarrow{acc},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

Tiling



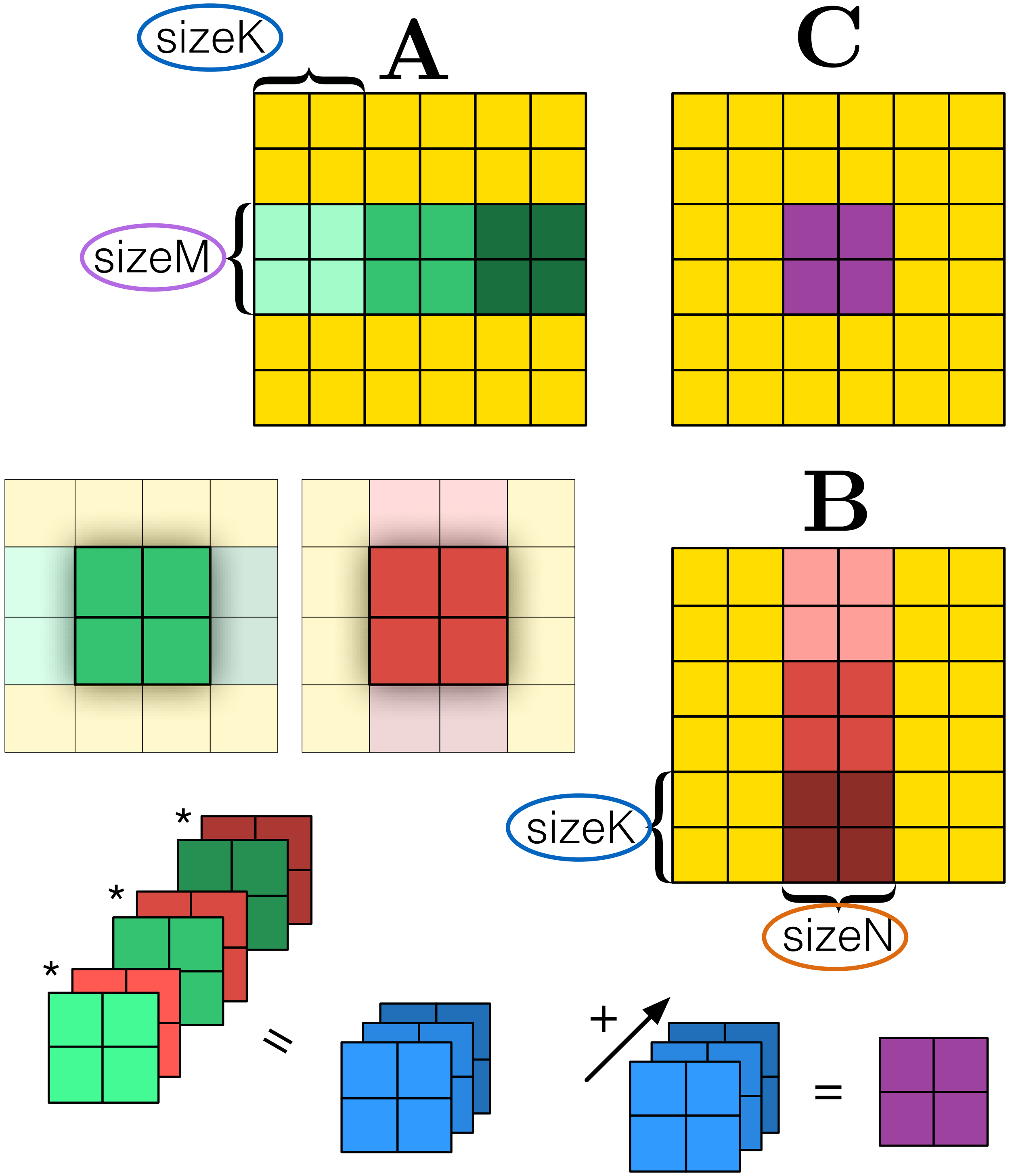
```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B,
  global float* C,
  int M, int K, int N)
{
  local float  A_lcl[sizeM*sizeK];
  local float  B_lcl[sizeK*sizeN];
  float acc = 0.0f;

  for (int i = 0; i < K / sizeK; i += 1) {
    A_lcl[l_idA( lcl_id_0 ,  lcl_id_1 )] =
      A[idA(i,  lcl_id_0 ,  lcl_id_1 ,  grp_id_0)];
    B_lcl[l_idB( lcl_id_0 ,  lcl_id_1 )] =
      B[idB(grp_id_1,  lcl_id_0 ,  lcl_id_1 ,  i)];

    barrier(CLK_LOCAL_MEM_FENCE);

    for (int j = 0; j < sizeK; j += 1)
      acc += A_lcl[l_idA(j,  lcl_id_1 )]
           * B_lcl[l_idB( lcl_id_0 ,  j)];

    barrier(CLK_LOCAL_MEM_FENCE);
  }
  C[idC(grp_id_0,  lcl_id_1 ,  grp_id_1,  lcl_id_0 )] = acc;
}
```

Tiling



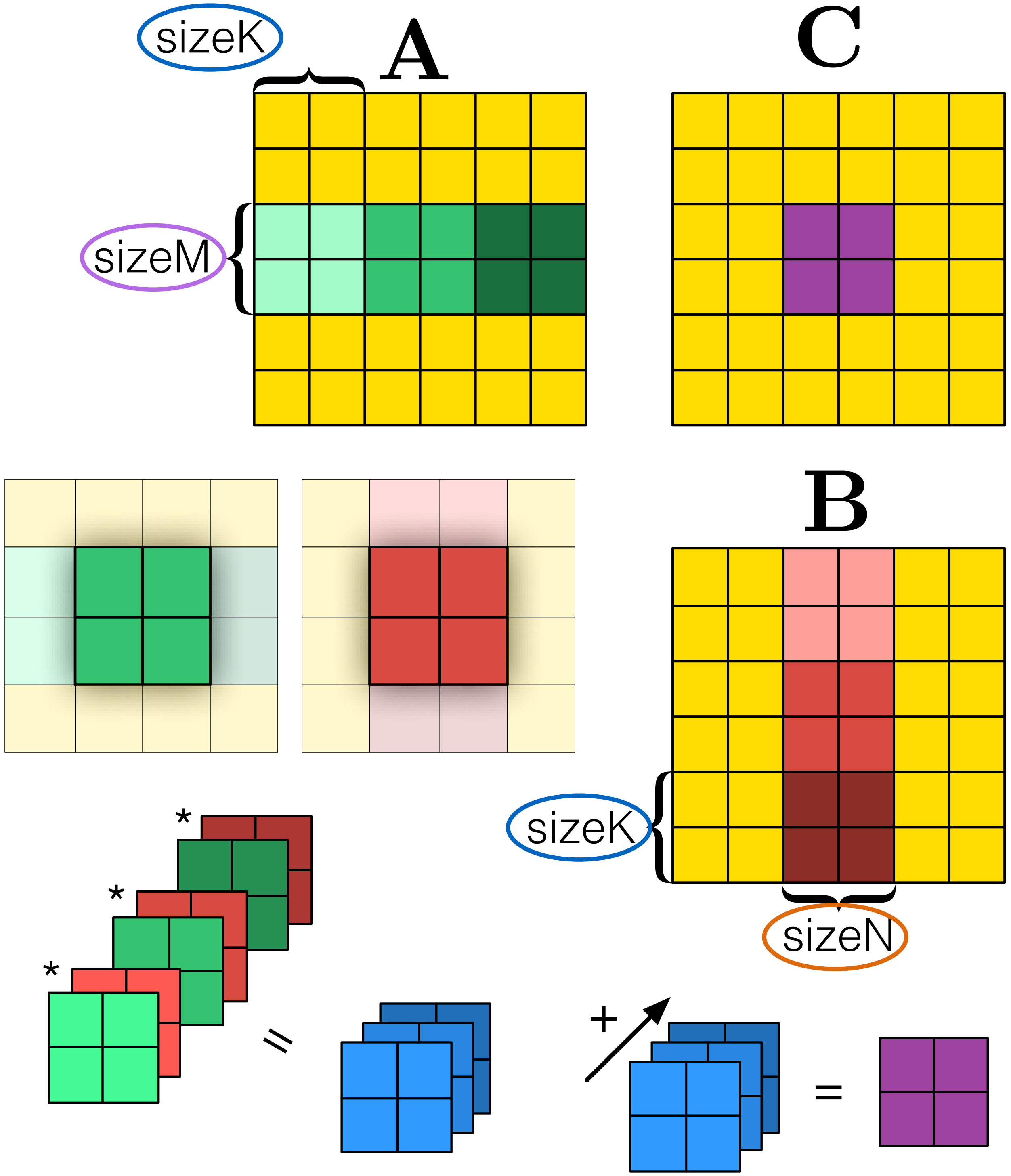
```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B,
  global float* C,
  int M, int K, int N)
{
  local float  A_lcl[sizeM*sizeK];
  local float  B_lcl[sizeK*sizeN];
  float acc = 0.0f;

  for (int i = 0; i < K / sizeK; i += 1) {
    A_lcl[l_idA( lcl_id_0 ,  lcl_id_1 )]  =
      A[idA(i,  lcl_id_0 ,  lcl_id_1 ,  grp_id_0)];
    B_lcl[l_idB( lcl_id_0 ,  lcl_id_1 )]  =
      B[idB(grp_id_1,  lcl_id_0 ,  lcl_id_1 ,  i)];

    barrier(CLK_LOCAL_MEM_FENCE);

    for (int j = 0; j < sizeK; j += 1)
      acc += A_lcl[l_idA(j,  lcl_id_1 )]
           * B_lcl[l_idB( lcl_id_0 ,  j)];

    barrier(CLK_LOCAL_MEM_FENCE);
  }
  C[idC(grp_id_0,  lcl_id_1 ,  grp_id_1,  lcl_id_0 )]  = acc;
}
```

Tiling



```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B,
  global float* C,
  int M, int K, int N)
{
  local float  A_lcl[sizeM*sizeK];
  local float  B_lcl[sizeK*sizeN];
  float acc = 0.0f;

  for (int i = 0; i < K / sizeK; i += 1) {
    A_lcl[l_idA( lcl_id_0 ,  lcl_id_1 )]  =
      A[idA(i,  lcl_id_0 ,  lcl_id_1 ,  grp_id_0)];
    B_lcl[l_idB( lcl_id_0 ,  lcl_id_1 )]  =
      B[idB(grp_id_1,  lcl_id_0 ,  lcl_id_1 ,  i)];

    barrier(CLK_LOCAL_MEM_FENCE);

    for (int j = 0; j < sizeK; j += 1)
      acc += A_lcl[l_idA(j,  lcl_id_1 )]
           * B_lcl[l_idB( lcl_id_0 ,  j)];

    barrier(CLK_LOCAL_MEM_FENCE);
  }
  C[idC(grp_id_0,  lcl_id_1 ,  grp_id_1,  lcl_id_0 )]  = acc;
}
```

Tiling



```
kernel void KERNEL(
  const global float* restrict A,
  const global float* restrict B,
  global float* C,
  int M, int K, int N)
{
  local float  A_lcl[sizeM*sizeK];
  local float  B_lcl[sizeK*sizeN];
  float acc = 0.0f;

  for (int i = 0; i < K / sizeK; i += 1) {
    A_lcl[l_idA( lcl_id_0 ,  lcl_id_1 )] =
      A[idA(i,  lcl_id_0 ,  lcl_id_1 ,  grp_id_0)];
    B_lcl[l_idB( lcl_id_0 ,  lcl_id_1 )] =
      B[idB(grp_id_1,  lcl_id_0 ,  lcl_id_1 ,  i)];

    barrier(CLK_LOCAL_MEM_FENCE);

    for (int j = 0; j < sizeK; j += 1)
      acc += A_lcl[l_idA(j,  lcl_id_1 )]
           * B_lcl[l_idB( lcl_id_0 ,  j)];

    barrier(CLK_LOCAL_MEM_FENCE);
  }
  C[idC(grp_id_0,  lcl_id_1 ,  grp_id_1,  lcl_id_0 )] = acc;
}
```

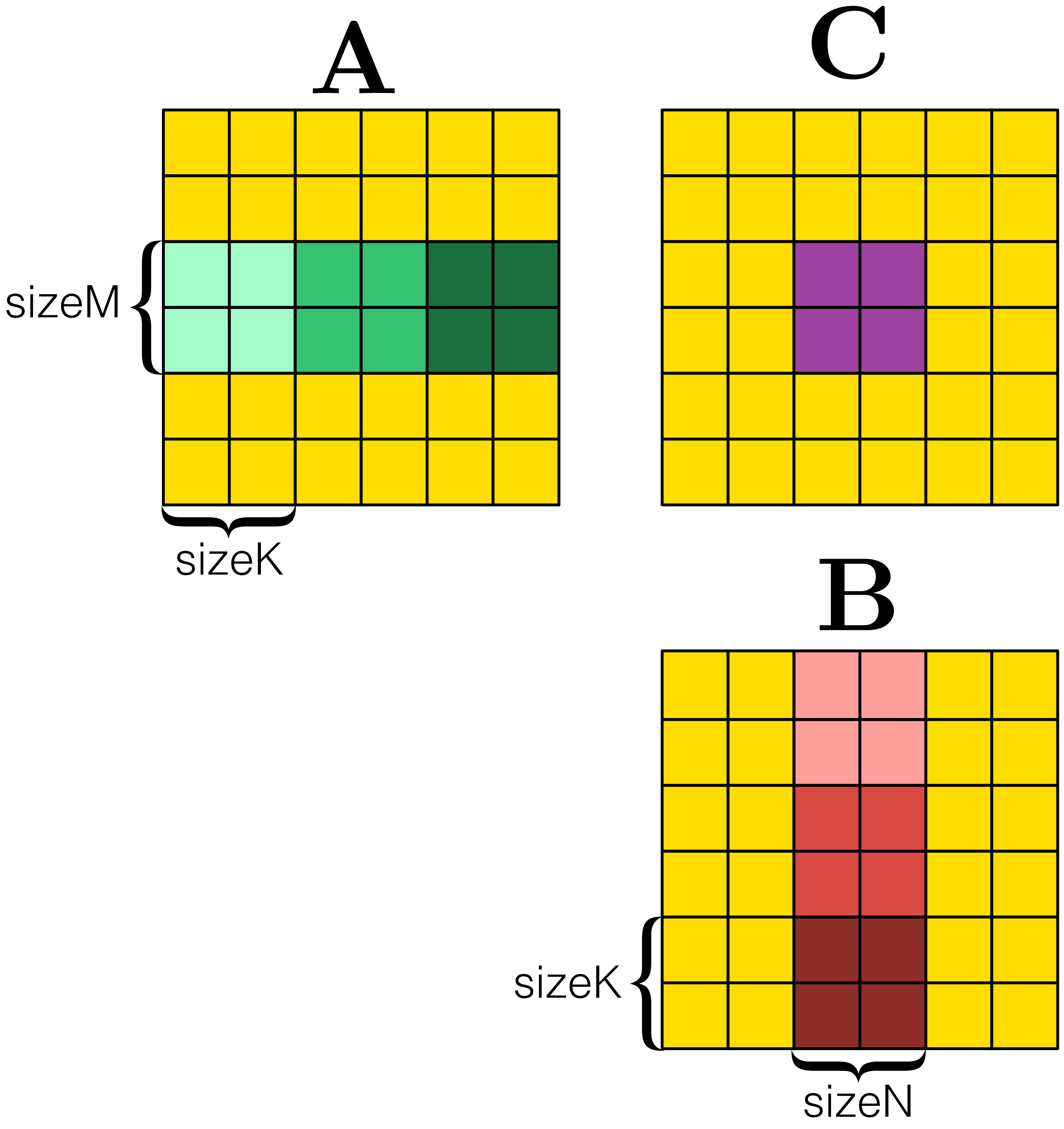
Tiling

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

Rewriting

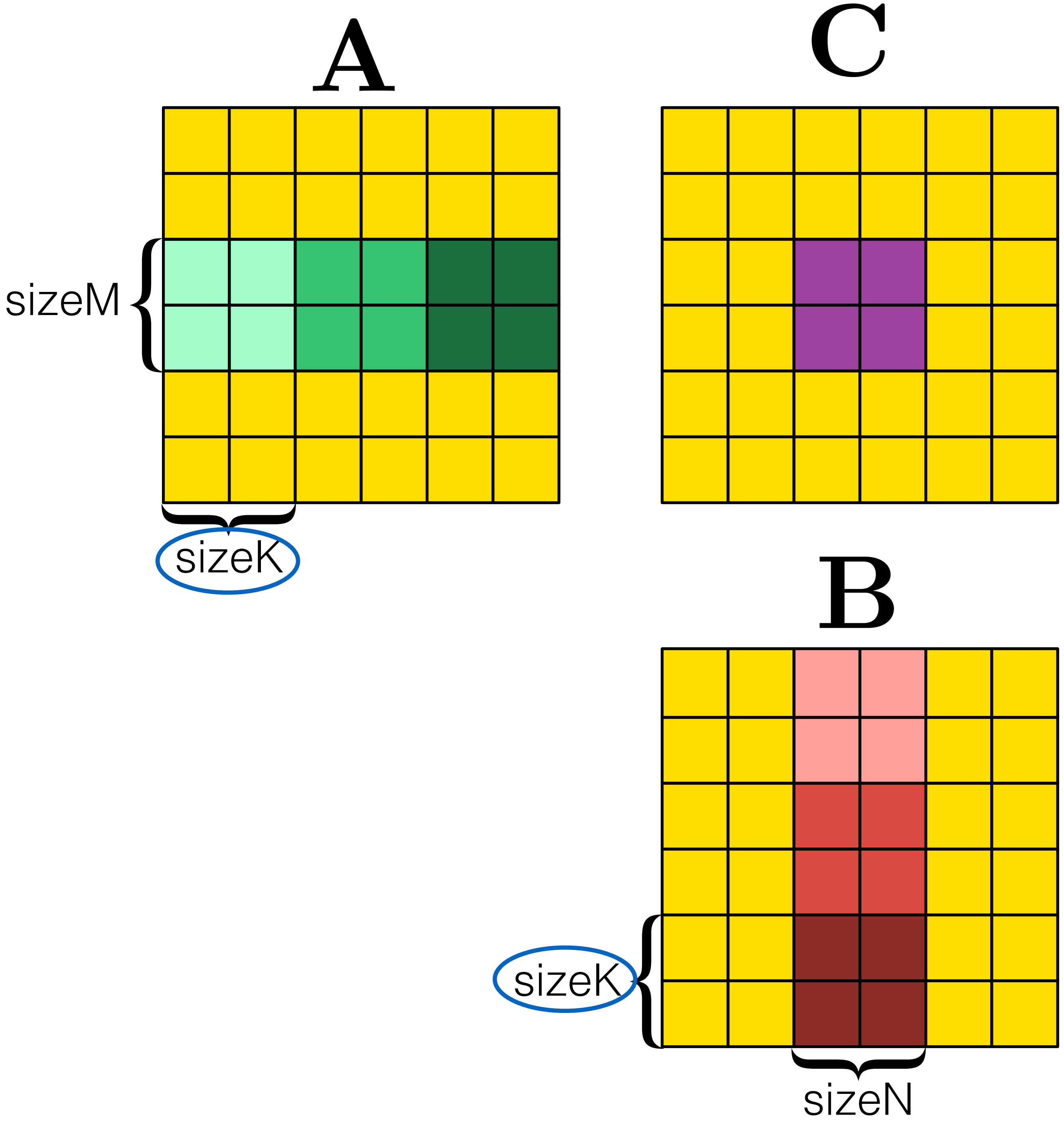
$$
\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad Map(\overrightarrow{aRows} \mapsto \\
\quad\quad Map(\overrightarrow{bCols} \mapsto \\
\quad\quad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
\quad\quad\quad\quad \mathbf{acc} + pairOfTiles.\_0 * \\
\quad\quad\quad\quad\quad Transpose() \circ pairOfTiles.\_1 \\
\quad\quad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\quad\quad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{array}
$$

Tiling



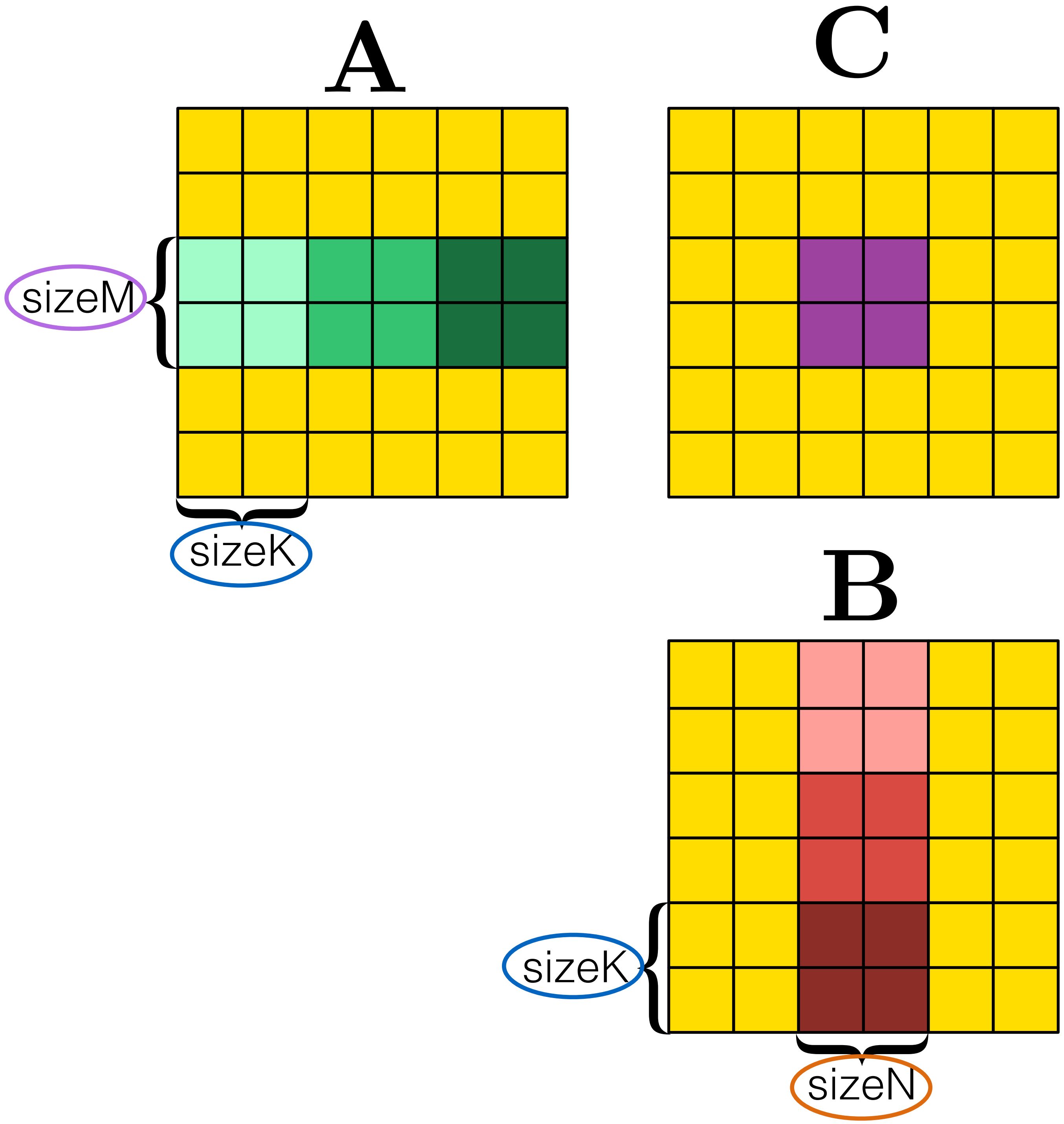
$$
\begin{array}{l}
\mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\
\quad \mathit{Untile}() \circ \\
\quad \mathit{Map}(\overrightarrow{\mathit{aRows}} \mapsto \\
\qquad \mathit{Map}(\overrightarrow{\mathit{bCols}} \mapsto \\
\qquad\quad \mathit{Reduce}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\
\qquad\qquad \mathbf{acc} + \mathit{pairOfTiles}.\_0 * \\
\qquad\qquad\quad \mathit{Transpose}() \circ \mathit{pairOfTiles}.\_1 \\
\qquad\quad) \,\$\, \mathit{Zip}(\overrightarrow{\mathit{aRows}}, \overrightarrow{\mathit{bCols}}) \\
\qquad) \circ \mathit{Transpose}() \circ \mathit{Tile}(\mathit{sizeN}, \mathit{sizeK}) \,\$\, \mathbf{B} \\
\quad) \circ \mathit{Tile}(\mathit{sizeM}, \mathit{sizeK}) \,\$\, \mathbf{A}
\end{array}
$$

Tiling



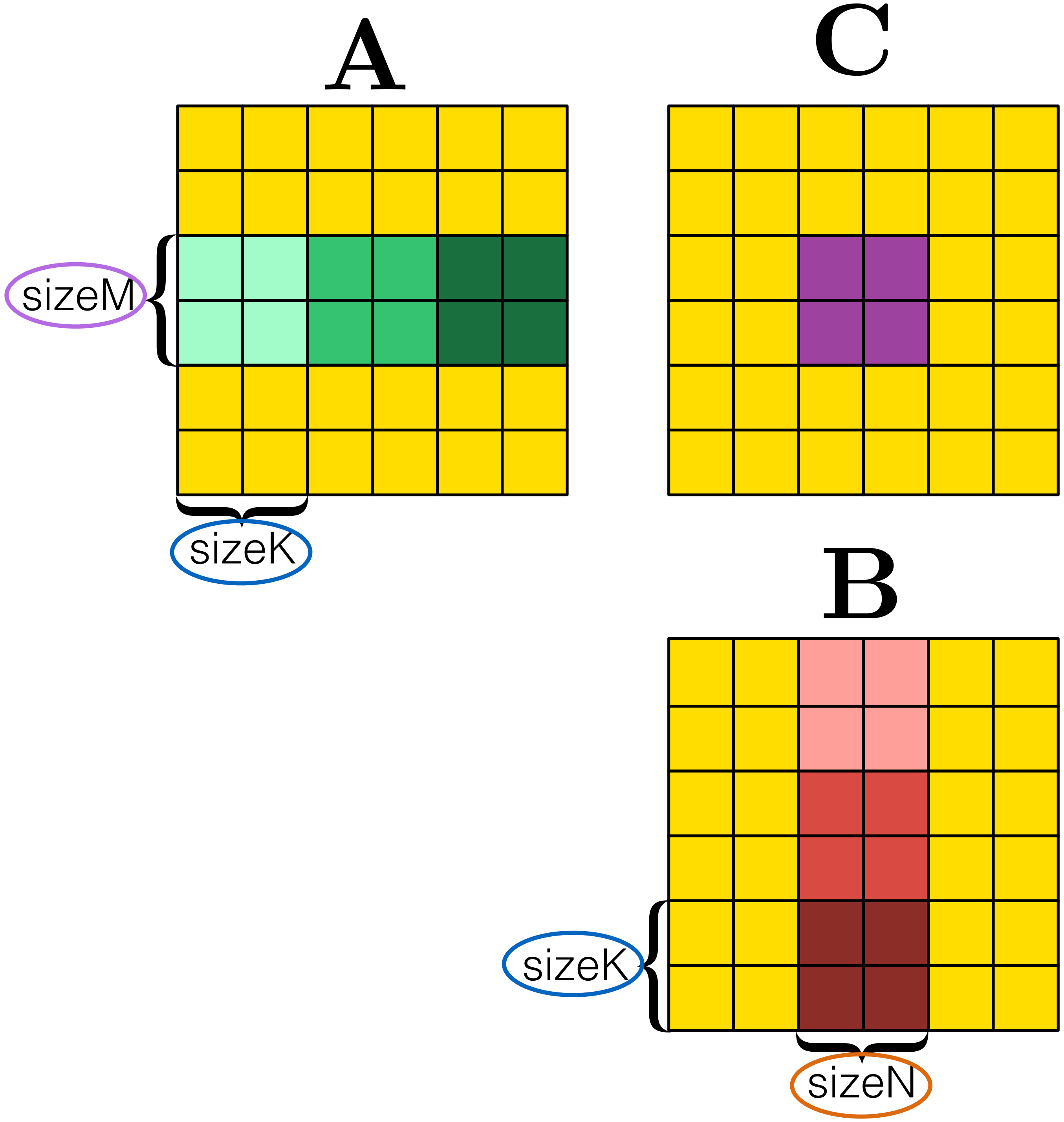
$$
\begin{aligned}
&TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
&\quad Untile() \circ \\
&\quad Map(\overrightarrow{aRows} \mapsto \\
&\qquad Map(\overrightarrow{bCols} \mapsto \\
&\qquad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
&\qquad\qquad \mathbf{acc} + pairOfTiles.\_0 * \\
&\qquad\qquad\quad Transpose() \circ pairOfTiles.\_1 \\
&\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
&\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
&\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{aligned}
$$

Tiling



$$
\begin{aligned}
& \mathit{TiledMultiply}(\mathbf{A}, \mathbf{B}) = \\
& \quad \mathit{Untile}() \circ \\
& \quad \mathit{Map}(\overrightarrow{aRows} \mapsto \\
& \qquad \mathit{Map}(\overrightarrow{bCols} \mapsto \\
& \qquad\quad \mathit{Reduce}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\
& \qquad\qquad \mathbf{acc} + \mathit{pairOfTiles}.\_0 * \\
& \qquad\qquad\quad \mathit{Transpose}() \circ \mathit{pairOfTiles}.\_1 \\
& \qquad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
& \qquad) \circ \mathit{Transpose}() \circ \mathit{Tile}(sizeN, sizeK) \ \$ \ \mathbf{B} \\
& \quad) \circ \mathit{Tile}(sizeM, sizeK) \ \$ \ \mathbf{A}
\end{aligned}
$$

Tiling



$$
\begin{aligned}
&TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
&\quad Untile() \circ \\
&\quad Map(\overrightarrow{aRows} \mapsto \\
&\qquad Map(\overrightarrow{bCols} \mapsto \\
&\qquad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
&\qquad\qquad \mathbf{acc} + pairOfTiles.\_0 * \\
&\qquad\qquad\quad Transpose() \circ pairOfTiles.\_1 \\
&\qquad\quad) \ \$ \ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
&\qquad) \circ Transpose() \circ Tile(sizeN, sizeK) \ \$ \ \mathbf{B} \\
&\quad) \circ Tile(sizeM, sizeK) \ \$ \ \mathbf{A}
\end{aligned}
$$

Combining Optimisations

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

80 rewrites

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

Combining Optimisations

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

Tiling

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

Combining Optimisations

$$
\begin{array}{l}
\mathbf{A} * \mathbf{B} = \\
\quad \mathit{Map}(\overrightarrow{\mathit{rowA}} \mapsto \\
\quad\quad \mathit{Map}(\overrightarrow{\mathit{colB}} \mapsto \\
\quad\quad\quad \mathit{DotProduct}(\overrightarrow{\mathit{rowA}}, \overrightarrow{\mathit{colB}}) \\
\quad\quad) \circ \mathit{Transpose}() \,\$\, \mathbf{B} \\
\quad) \,\$\, \mathbf{A}
\end{array}
$$

Blocking

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

Combining Optimisations

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

→ 80 rewrites

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

Macro Rules

- A *macro rule* is a rewrite rule that has a particular goal and can apply different rewrite rules to achieve it.

- Examples:

  - 1D Register Blocking, 2D Register Blocking

  - Tiling

  - Map-Map Interchange

Map-Map Interchange Macro Rule

$$
\begin{array}{l}
Map(a \mapsto \\
\quad Map(b \mapsto \\
\qquad f(a,b)) \ \$ \ B \\
) \ \$ \ A \\
\\
\qquad\quad \Downarrow \\
\\
Transpose() \circ \\
Map(b \mapsto \\
\quad Map(a \mapsto \\
\qquad f(a,b)) \ \$ \ A \\
) \ \$ \ B
\end{array}
$$

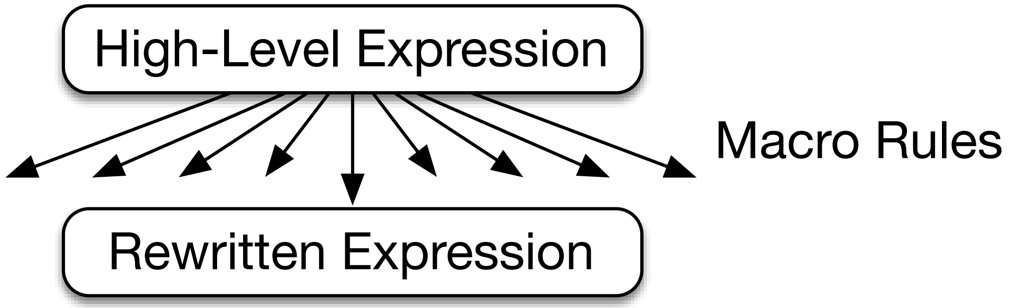
$$
\begin{array}{l}
Map(a \mapsto \\
\quad Map(b \mapsto \\
\qquad f(b)) \ \$ \ a \\
) \ \$ \ A \\
\\
\qquad\quad \Downarrow \\
\\
Transpose() \circ \\
Map(b \mapsto \\
\quad Map(a \mapsto \\
\qquad f(a)) \ \$ \ b \\
) \circ Transpose() \ \$ \ A
\end{array}
$$

$$
\begin{array}{l}
Map(a \mapsto \\
\quad Map(b \mapsto \\
\qquad f(b.\_0, b.\_1) \\
\quad) \ \$ \ Zip(a, c) \\
) \ \$ \ A \\
\\
\qquad\quad \Downarrow \\
\\
Transpose() \circ \\
Map(b \mapsto \\
\quad Map(a \mapsto \\
\qquad f(a, b.\_1) \\
\quad) \ \$ \ b.\_0 \\
) \ \$ \ Zip(\\
Transpose() \$ A, \\
c)
\end{array}
$$

$$
\begin{array}{l}
Map(a \mapsto \\
\quad Map(b \mapsto \\
\qquad f(b, a.\_1) \\
\quad) \ \$ \ a.\_0 \\
) \ \$ \ Zip(A, c) \\
\\
\qquad\quad \Downarrow \\
\\
Transpose() \circ \\
Map(b \mapsto \\
\quad Map(a \mapsto \\
\qquad f(a.\_0, a.\_1) \\
\quad) \ \$ \ Zip(b.\_0, c) \\
) \circ Transpose() \ \$ \ A
\end{array}
$$

THE UNIVERSITY *of* EDINBURGH
informatics

Exploration Strategy

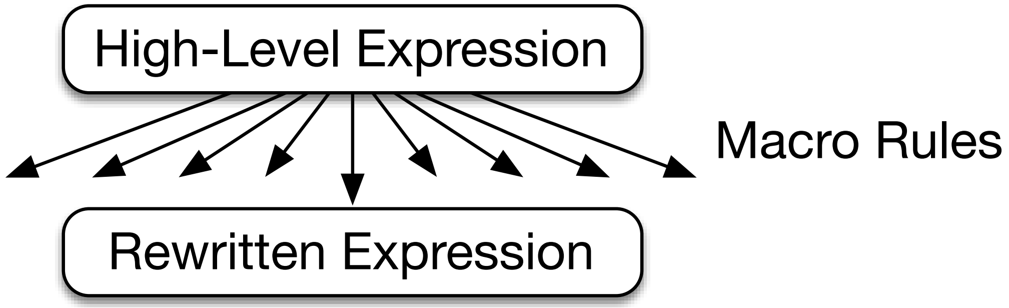


1

$$\boxed{\begin{array}{l} \mathbf{A} * \mathbf{B} = \\ \quad Map(\overrightarrow{rowA} \mapsto \\ \quad\quad Map(\overrightarrow{colB} \mapsto \\ \quad\quad\quad DotProduct(\overrightarrow{rowA}, \overrightarrow{colB}) \\ \quad\quad) \circ Transpose() \; \$ \; \mathbf{B} \\ \quad) \; \$ \; \mathbf{A} \end{array}}$$

THE UNIVERSITY *of* EDINBURGH
informatics

Exploration Strategy

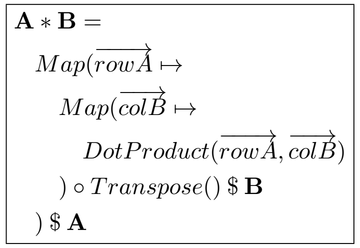


High-Level Expression

Macro Rules

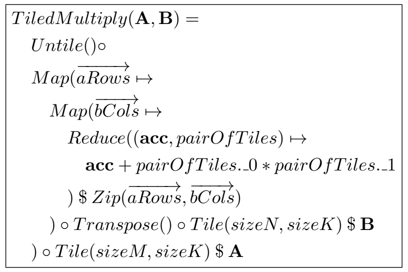
Rewritten Expression

1



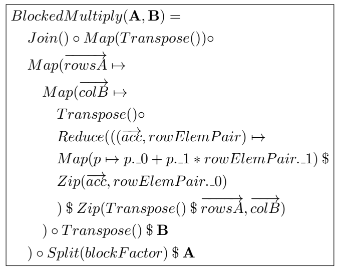
$$\begin{array}{l}
\mathbf{A} * \mathbf{B} = \\
\quad Map(\overrightarrow{rowA} \mapsto \\
\quad\quad Map(\overrightarrow{colB} \mapsto \\
\quad\quad\quad DotProduct(\overrightarrow{rowA}, \overrightarrow{colB}) \\
\quad\quad) \circ Transpose() \ \$ \ \mathbf{B} \\
\quad) \ \$ \ \mathbf{A}
\end{array}$$

1.1



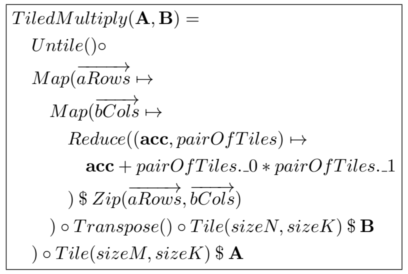
$$\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad Map(\overrightarrow{aRows} \mapsto \\
\quad\quad Map(\overrightarrow{bCols} \mapsto \\
\quad\quad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
\quad\quad\quad\quad \mathbf{acc} + pairOfTiles.\_0 * pairOfTiles.\_1 \\
\quad\quad\quad) \ \$ \ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\quad\quad) \circ Transpose() \circ Tile(sizeN, sizeK) \ \$ \ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK) \ \$ \ \mathbf{A}
\end{array}$$

1.2



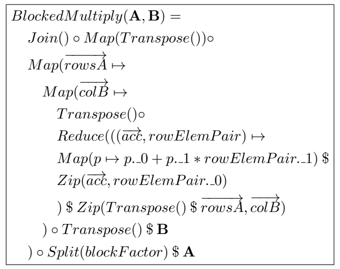
$$\begin{array}{l}
BlockedMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Join() \circ Map(Transpose()) \circ \\
\quad Map(\overrightarrow{rowsA} \mapsto \\
\quad\quad Map(\overrightarrow{colB} \mapsto \\
\quad\quad\quad Transpose() \circ \\
\quad\quad\quad Reduce(((\overrightarrow{acc}, rowElemPair) \mapsto \\
\quad\quad\quad Map(p \mapsto p.\_0 + p.\_1 * rowElemPair.\_1) \ \$ \\
\quad\quad\quad Zip(\overrightarrow{acc}, rowElemPair.\_0) \\
\quad\quad\quad) \ \$ \ Zip(Transpose() \ \$ \ \overrightarrow{rowsA}, \overrightarrow{colB}) \\
\quad\quad) \circ Transpose() \ \$ \ \mathbf{B} \\
\quad) \circ Split(blockFactor) \ \$ \ \mathbf{A}
\end{array}$$

1.3



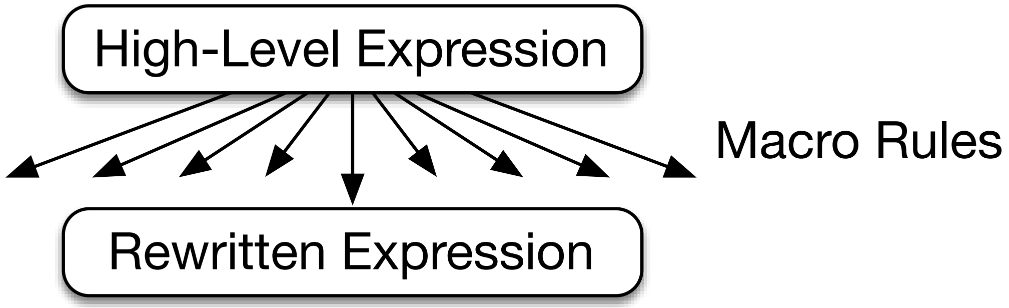
$$\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad Map(\overrightarrow{aRows} \mapsto \\
\quad\quad Map(\overrightarrow{bCols} \mapsto \\
\quad\quad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
\quad\quad\quad\quad \mathbf{acc} + pairOfTiles.\_0 * pairOfTiles.\_1 \\
\quad\quad\quad) \ \$ \ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\quad\quad) \circ Transpose() \circ Tile(sizeN, sizeK) \ \$ \ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK) \ \$ \ \mathbf{A}
\end{array}$$

1.4



$$\begin{array}{l}
BlockedMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Join() \circ Map(Transpose()) \circ \\
\quad Map(\overrightarrow{rowsA} \mapsto \\
\quad\quad Map(\overrightarrow{colB} \mapsto \\
\quad\quad\quad Transpose() \circ \\
\quad\quad\quad Reduce(((\overrightarrow{acc}, rowElemPair) \mapsto \\
\quad\quad\quad Map(p \mapsto p.\_0 + p.\_1 * rowElemPair.\_1) \ \$ \\
\quad\quad\quad Zip(\overrightarrow{acc}, rowElemPair.\_0) \\
\quad\quad\quad) \ \$ \ Zip(Transpose() \ \$ \ \overrightarrow{rowsA}, \overrightarrow{colB}) \\
\quad\quad) \circ Transpose() \ \$ \ \mathbf{B} \\
\quad) \circ Split(blockFactor) \ \$ \ \mathbf{A}
\end{array}$$

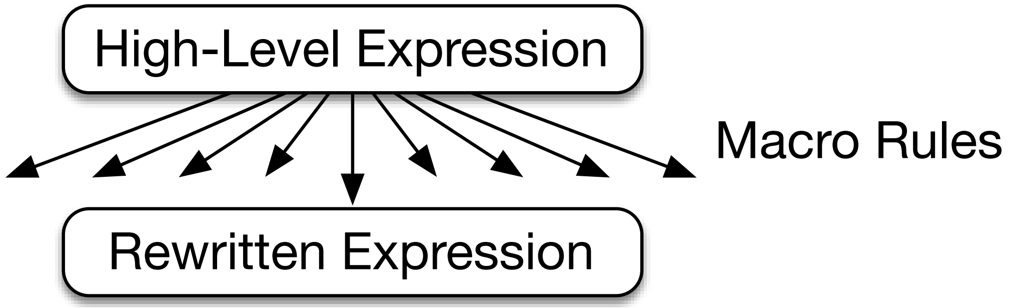
Exploration Strategy



1.3

$$
\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad Map(\overrightarrow{aRows} \mapsto \\
\qquad Map(\overrightarrow{bCols} \mapsto \\
\qquad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + pairOfTiles.\_0 * pairOfTiles.\_1 \\
\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{array}
$$

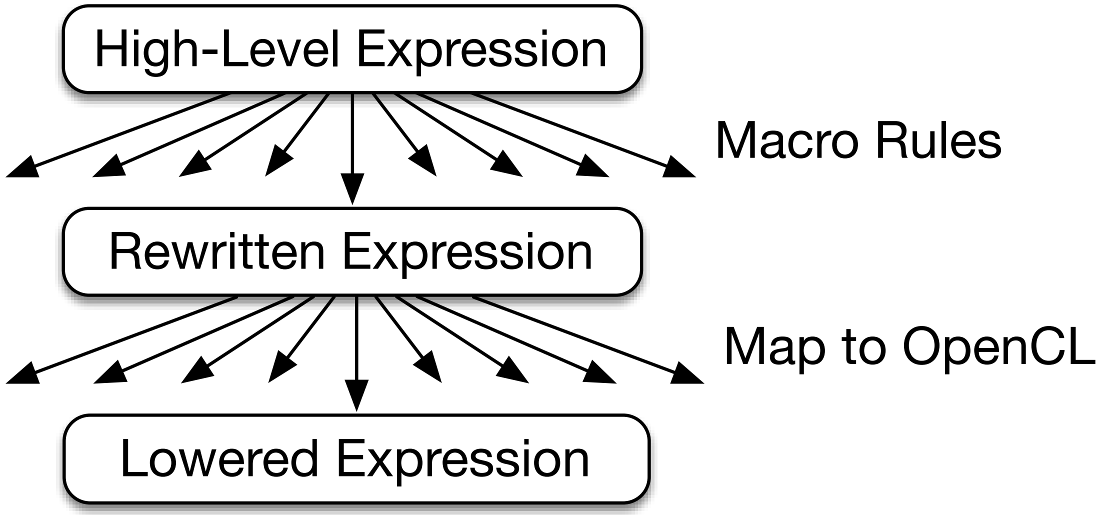
Exploration Strategy



1.3

$$\boxed{\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad Map(\overrightarrow{aRows} \mapsto \\
\qquad Map(\overrightarrow{bCols} \mapsto \\
\qquad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + pairOfTiles.\_0 * pairOfTiles.\_1 \\
\qquad\quad) \ \$ \ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK) \ \$ \ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK) \ \$ \ \mathbf{A}
\end{array}}$$

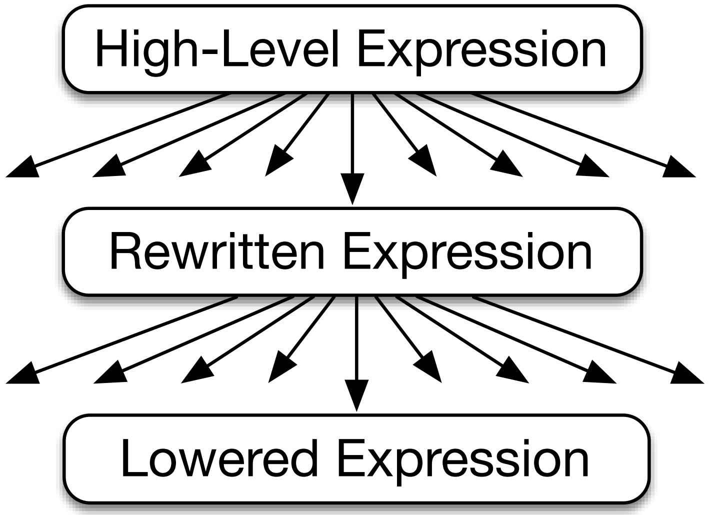
Exploration Strategy



1.3

$$\boxed{\begin{aligned}
&TiledMultiply(\mathbf{A},\mathbf{B}) = \\
&\quad Untile() \circ \\
&\quad Map(\overrightarrow{aRows} \mapsto \\
&\qquad Map(\overrightarrow{bCols} \mapsto \\
&\qquad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
&\qquad\qquad \mathbf{acc} + pairOfTiles.\_0 * pairOfTiles.\_1 \\
&\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
&\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
&\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{aligned}}$$

Exploration Strategy



1.3

$$\begin{array}{l}
TiledMultiply(\mathbf{A},\mathbf{B}) = \\
\quad Untile()\circ \\
\quad Map(\overrightarrow{aRows} \mapsto \\
\qquad Map(\overrightarrow{bCols} \mapsto \\
\qquad\quad Reduce((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + pairOfTiles.\_0 * pairOfTiles.\_1 \\
\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{array}$$

1.3.1

$$\begin{array}{l}
TiledMultiply(\mathbf{A},\mathbf{B}) = \\
\quad Untile()\circ \\
\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
\qquad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
\qquad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
\qquad\qquad\qquad * toLocal(pairOfTiles.\_1) \\
\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{array}$$

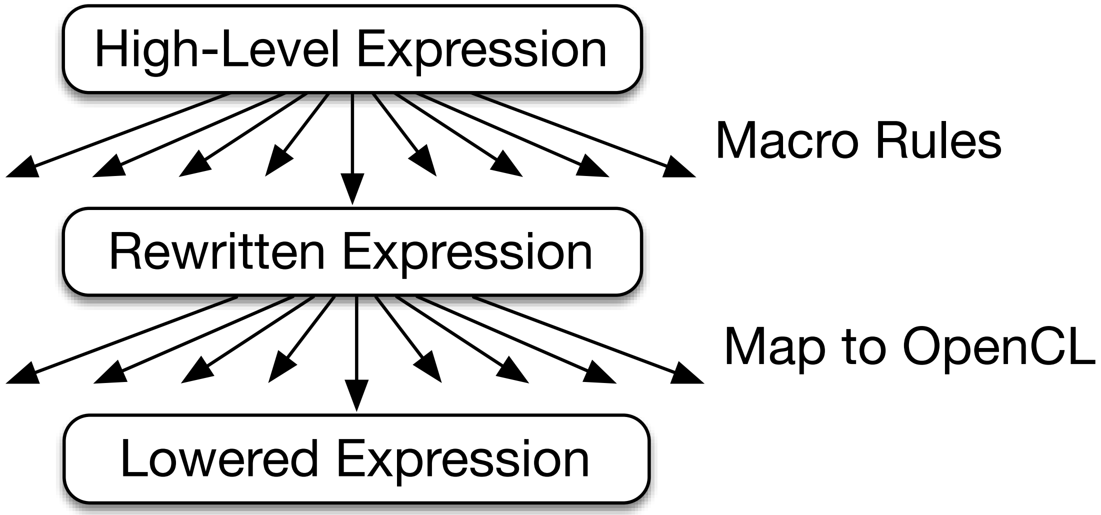
1.3.2

$$\begin{array}{l}
TiledMultiply(\mathbf{A},\mathbf{B}) = \\
\quad Untile()\circ \\
\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
\qquad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
\qquad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
\qquad\qquad\qquad * toLocal(pairOfTiles.\_1) \\
\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{array}$$

1.3.3

$$\begin{array}{l}
TiledMultiply(\mathbf{A},\mathbf{B}) = \\
\quad Untile()\circ \\
\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
\qquad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
\qquad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
\qquad\qquad\qquad * toLocal(pairOfTiles.\_1) \\
\qquad\quad)\ \$\ Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK)\ \$\ \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK)\ \$\ \mathbf{A}
\end{array}$$

Exploration Strategy



High-Level Expression

Macro Rules

Rewritten Expression

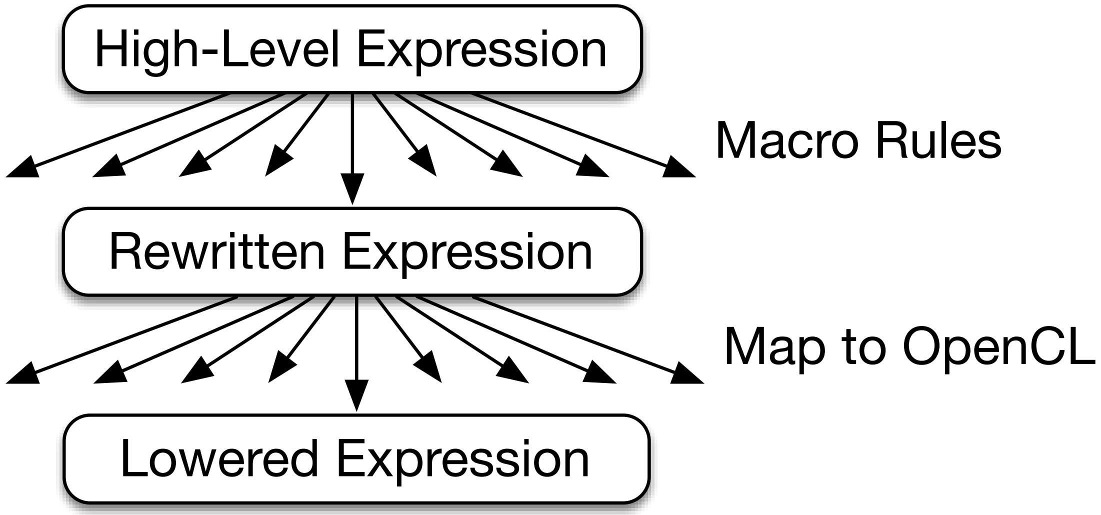
Map to OpenCL

Lowered Expression

1.3.2

$$
\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
\quad\quad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
\quad\quad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
\quad\quad\quad\quad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
\quad\quad\quad\quad\quad * toLocal(pairOfTiles.\_1) \\
\quad\quad\quad) \; \$ \; Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\quad\quad) \circ Transpose() \circ Tile(sizeN, sizeK) \; \$ \; \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK) \; \$ \; \mathbf{A}
\end{array}
$$

Exploration Strategy



High-Level Expression

Macro Rules

Rewritten Expression

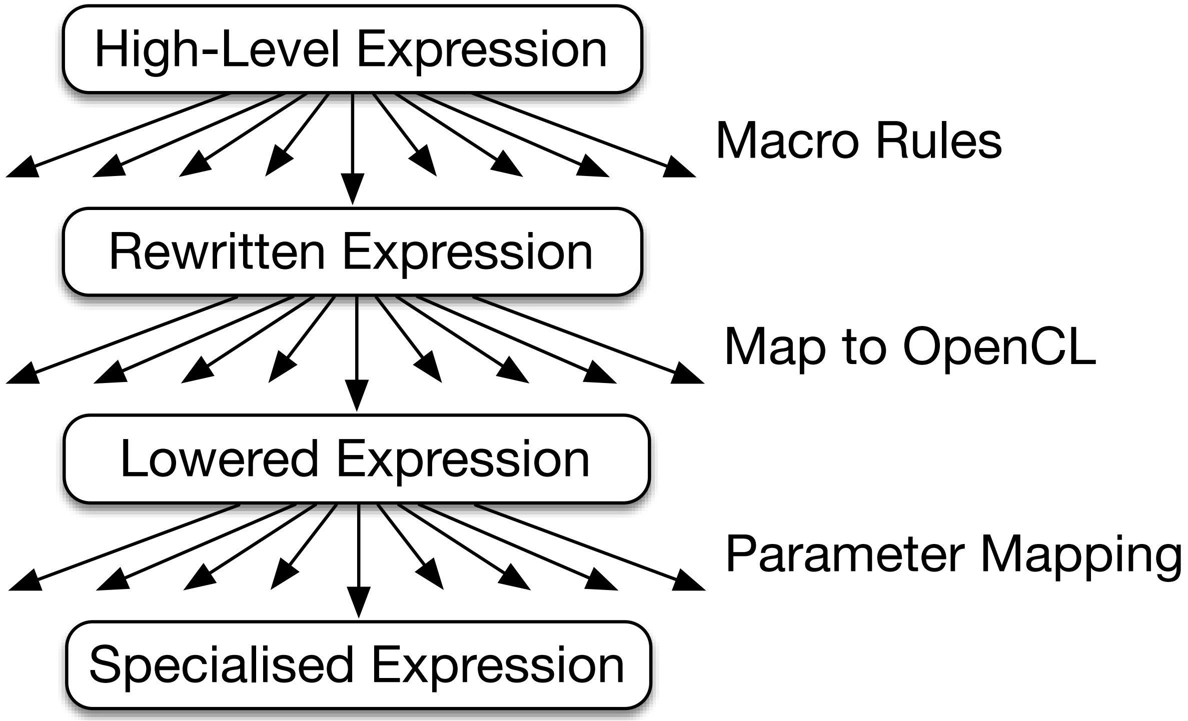
Map to OpenCL

Lowered Expression

1.3.2

$$
\boxed{
\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
\quad\quad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
\quad\quad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
\quad\quad\quad\quad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
\quad\quad\quad\quad\quad\quad * \, toLocal(pairOfTiles.\_1) \\
\quad\quad\quad) \; \$ \; Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\quad\quad) \circ Transpose() \circ Tile(sizeN, sizeK) \; \$ \; \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK) \; \$ \; \mathbf{A}
\end{array}
}
$$

Exploration Strategy



High-Level Expression

Macro Rules

Rewritten Expression

Map to OpenCL

Lowered Expression

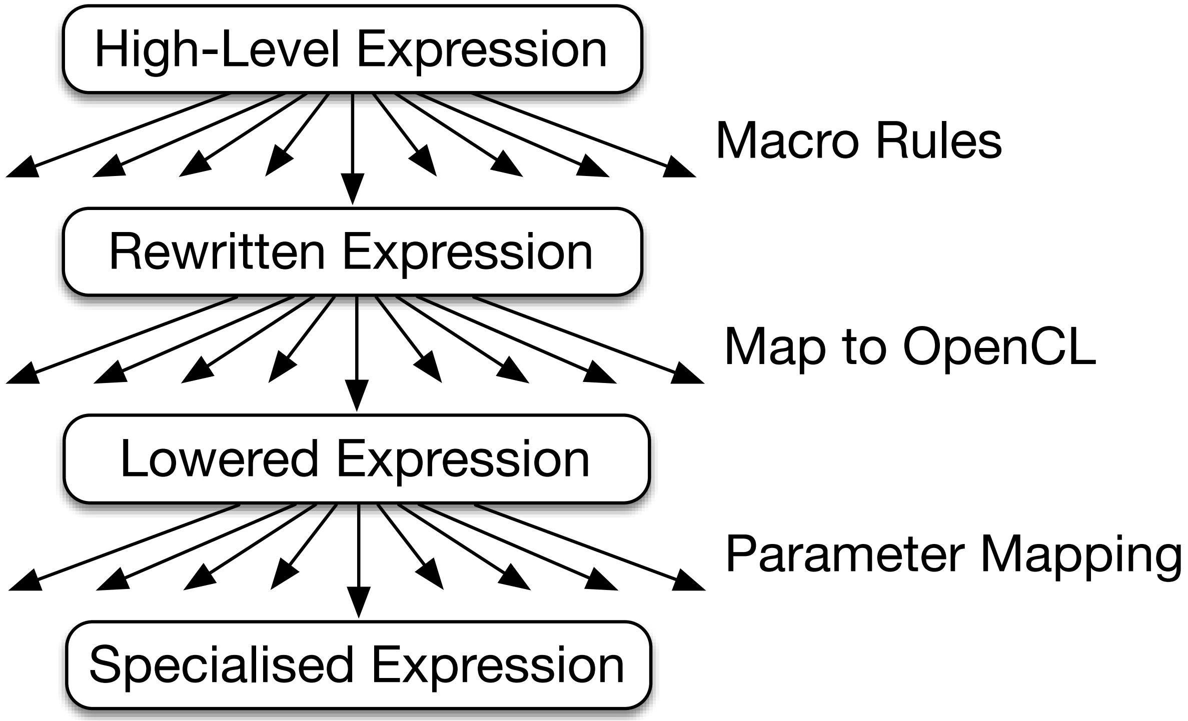
Parameter Mapping

Specialised Expression

1.3.2

$$
\boxed{
\begin{array}{l}
TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
\quad Untile() \circ \\
\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
\qquad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
\qquad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
\qquad\qquad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
\qquad\qquad\qquad * \, toLocal(pairOfTiles.\_1) \\
\qquad\quad) \; \$ \; Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
\qquad) \circ Transpose() \circ Tile(sizeN, sizeK) \; \$ \; \mathbf{B} \\
\quad) \circ Tile(sizeM, sizeK) \; \$ \; \mathbf{A}
\end{array}
}
$$

Exploration Strategy



High-Level Expression

Macro Rules

Rewritten Expression

Map to OpenCL

Lowered Expression

Parameter Mapping

Specialised Expression

1.3.2

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(sizeN, sizeK) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(sizeM, sizeK) \ \$ \ \mathbf{A} \end{array}$$

1.3.2.1

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{A} \end{array}$$

1.3.2.2

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{A} \end{array}$$

1.3.2.3

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{A} \end{array}$$

1.3.2.4

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{A} \end{array}$$

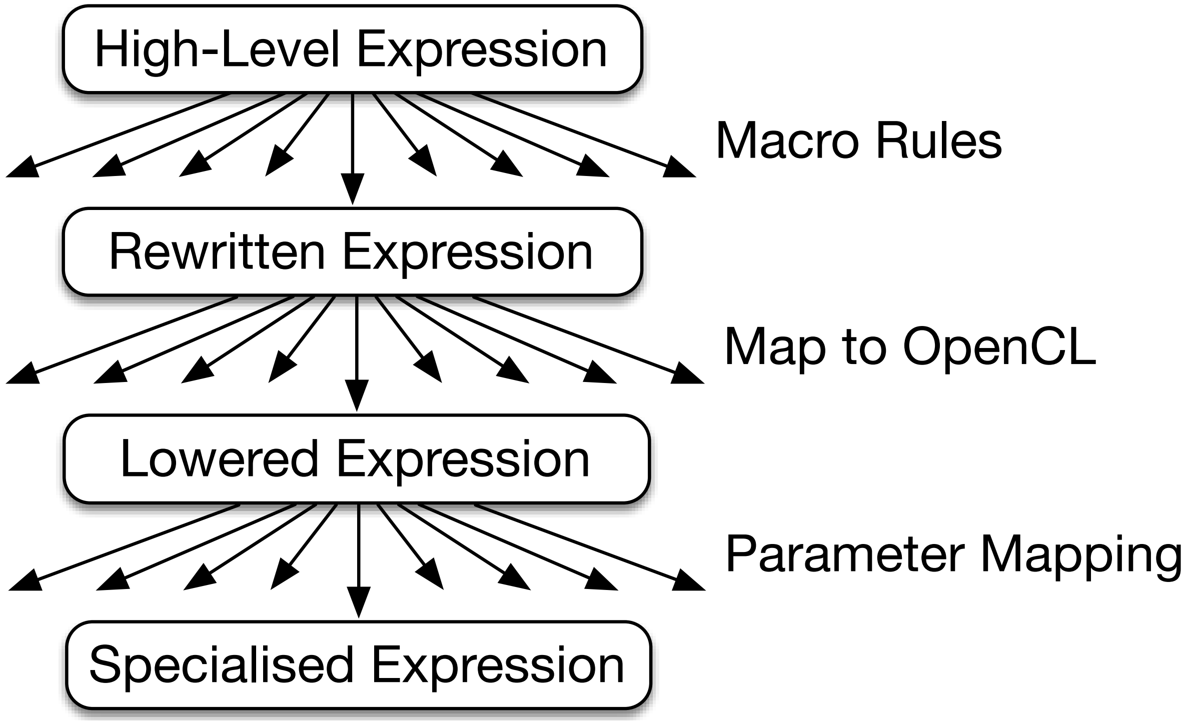
1.3.2.5

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{A} \end{array}$$

1.3.2.6

$$\begin{array}{l} \mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\ \quad \mathit{Untile}() \circ \\ \quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\ \quad\quad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\ \quad\quad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\ \quad\quad\quad\quad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\ \quad\quad\quad\quad\quad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\ \quad\quad\quad) \ \$ \ \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\ \quad\quad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{B} \\ \quad) \circ \mathit{Tile}(128, 16) \ \$ \ \mathbf{A} \end{array}$$

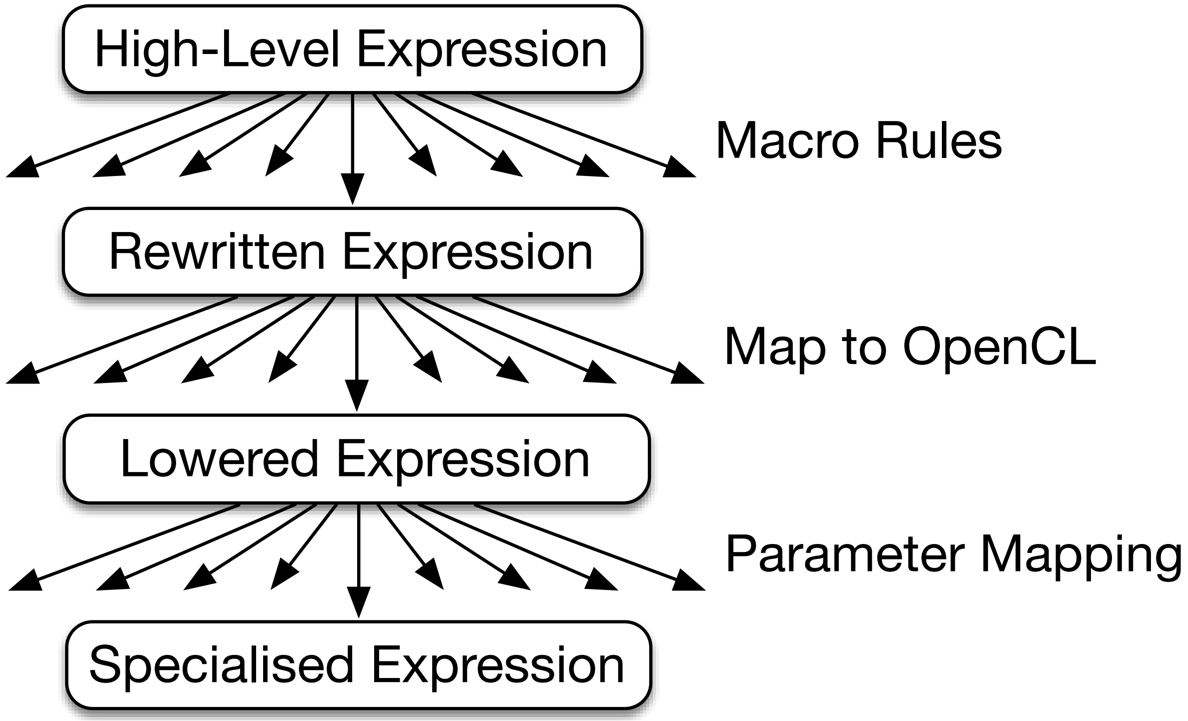
Exploration Strategy



1.3.2.5

$$
\begin{aligned}
&\mathit{TiledMultiply}(\mathbf{A},\mathbf{B}) = \\
&\quad \mathit{Untile}() \circ \\
&\quad \mathit{MapWrg}(1)(\overrightarrow{aRows} \mapsto \\
&\qquad \mathit{MapWrg}(0)(\overrightarrow{bCols} \mapsto \\
&\qquad\quad \mathit{ReduceSeq}((\mathbf{acc}, \mathit{pairOfTiles}) \mapsto \\
&\qquad\qquad \mathbf{acc} + \mathit{toLocal}(\mathit{pairOfTiles}.\_0) \\
&\qquad\qquad\qquad * \mathit{toLocal}(\mathit{pairOfTiles}.\_1) \\
&\qquad\quad) \mathbin{\$} \mathit{Zip}(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
&\qquad) \circ \mathit{Transpose}() \circ \mathit{Tile}(128, 16) \mathbin{\$} \mathbf{B} \\
&\quad) \circ \mathit{Tile}(128, 16) \mathbin{\$} \mathbf{A}
\end{aligned}
$$

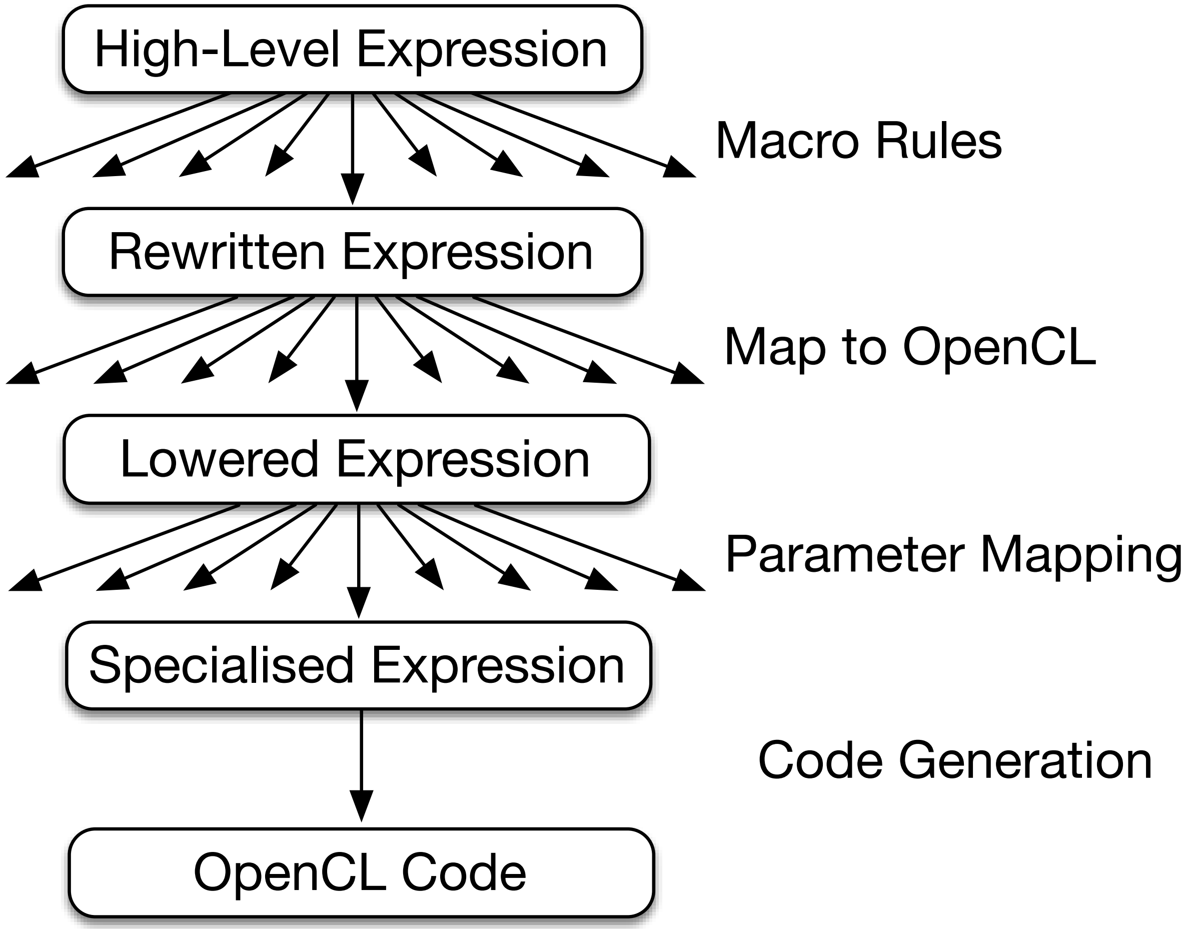
Exploration Strategy



1.3.2.5

$$\begin{aligned}
&TiledMultiply(\mathbf{A}, \mathbf{B}) = \\
&\quad Untile() \circ \\
&\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
&\qquad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
&\qquad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
&\qquad\qquad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
&\qquad\qquad\qquad * \, toLocal(pairOfTiles.\_1) \\
&\qquad\quad) \,\$\, Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
&\qquad) \circ Transpose() \circ Tile(128, 16) \,\$\, \mathbf{B} \\
&\quad) \circ Tile(128, 16) \,\$\, \mathbf{A}
\end{aligned}$$

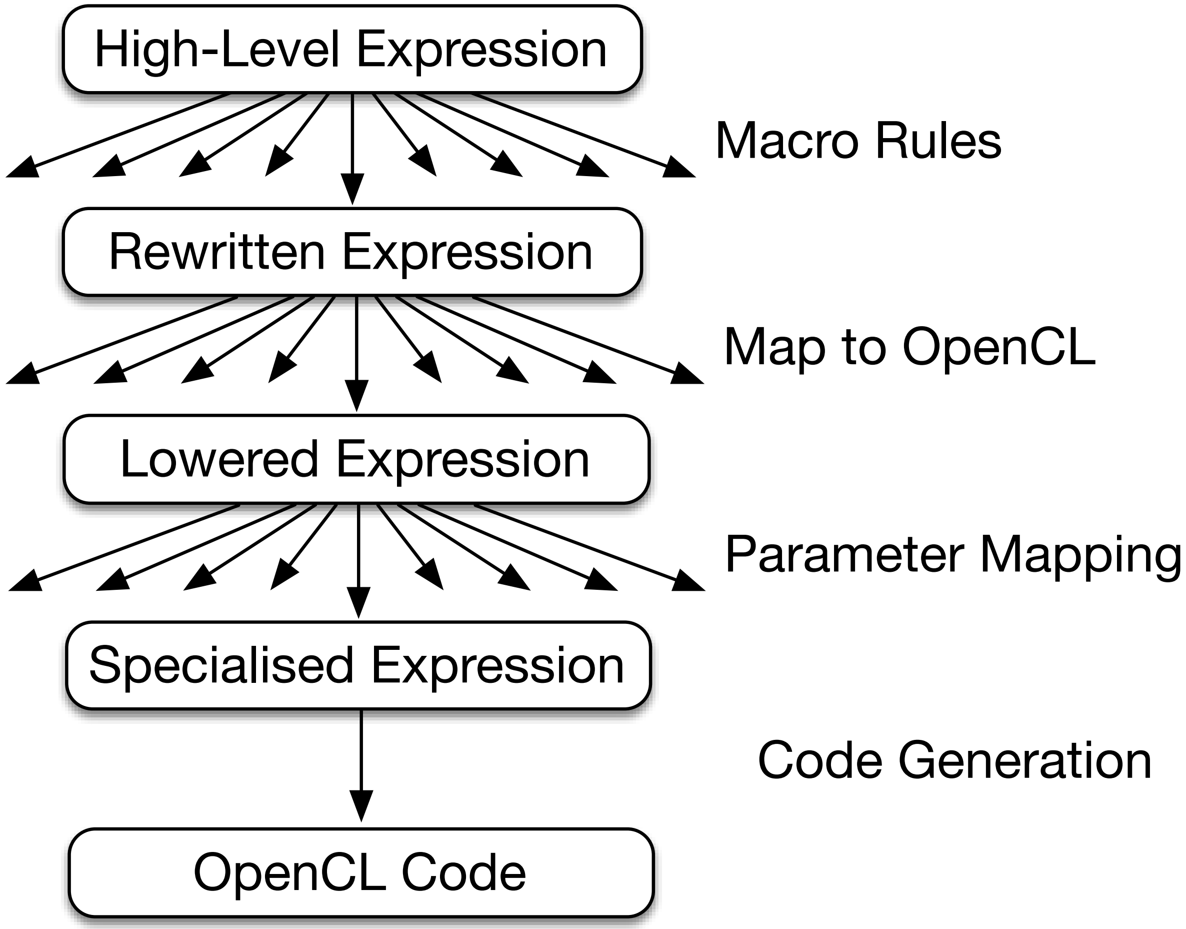
Exploration Strategy

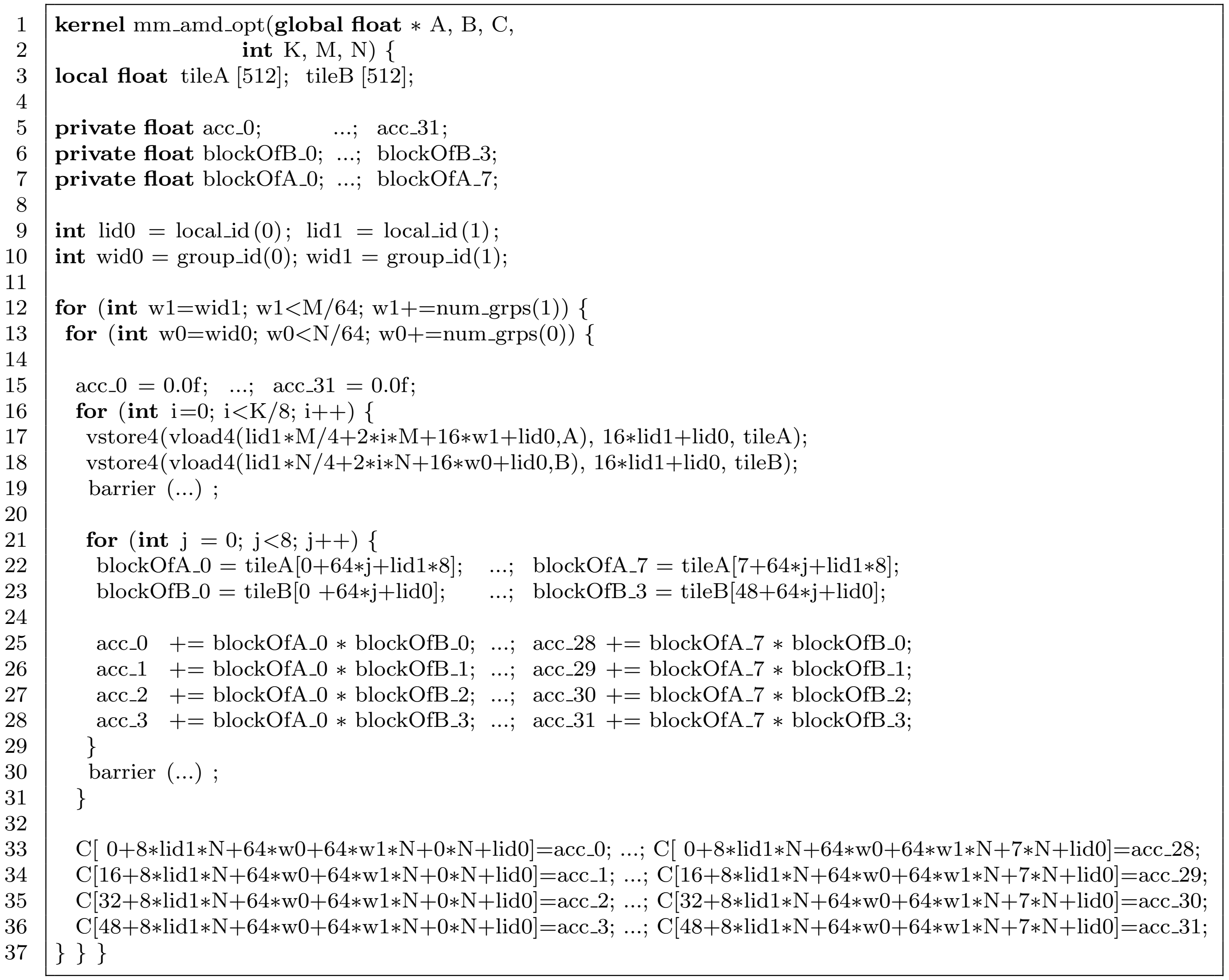


1.3.2.5

$$
\begin{aligned}
&TiledMultiply(\mathbf{A},\mathbf{B}) = \\
&\quad Untile() \circ \\
&\quad MapWrg(1)(\overrightarrow{aRows} \mapsto \\
&\qquad MapWrg(0)(\overrightarrow{bCols} \mapsto \\
&\qquad\quad ReduceSeq((\mathbf{acc}, pairOfTiles) \mapsto \\
&\qquad\qquad \mathbf{acc} + toLocal(pairOfTiles.\_0) \\
&\qquad\qquad\qquad * \, toLocal(pairOfTiles.\_1) \\
&\qquad\quad) \; \$ \; Zip(\overrightarrow{aRows}, \overrightarrow{bCols}) \\
&\qquad) \circ Transpose() \circ Tile(128, 16) \; \$ \; \mathbf{B} \\
&\quad) \circ Tile(128, 16) \; \$ \; \mathbf{A}
\end{aligned}
$$

Exploration Strategy





```
kernel mm_amd_opt(global float * A, B, C,
                  int K, M, N) {
local float tileA [512]; tileB [512];

private float acc_0;       ...; acc_31;
private float blockOfB_0; ...; blockOfB_3;
private float blockOfA_0; ...; blockOfA_7;

int lid0 = local_id(0); lid1 = local_id(1);
int wid0 = group_id(0); wid1 = group_id(1);

for (int w1=wid1; w1<M/64; w1+=num_grps(1)) {
 for (int w0=wid0; w0<N/64; w0+=num_grps(0)) {

  acc_0 = 0.0f; ...; acc_31 = 0.0f;
  for (int i=0; i<K/8; i++) {
   vstore4(vload4(lid1*M/4+2*i*M+16*w1+lid0,A), 16*lid1+lid0, tileA);
   vstore4(vload4(lid1*N/4+2*i*N+16*w0+lid0,B), 16*lid1+lid0, tileB);
   barrier (...) ;

   for (int j = 0; j<8; j++) {
    blockOfA_0 = tileA[0+64*j+lid1*8];  ...; blockOfA_7 = tileA[7+64*j+lid1*8];
    blockOfB_0 = tileB[0 +64*j+lid0];   ...; blockOfB_3 = tileB[48+64*j+lid0];

    acc_0  += blockOfA_0 * blockOfB_0; ...; acc_28 += blockOfA_7 * blockOfB_0;
    acc_1  += blockOfA_0 * blockOfB_1; ...; acc_29 += blockOfA_7 * blockOfB_1;
    acc_2  += blockOfA_0 * blockOfB_2; ...; acc_30 += blockOfA_7 * blockOfB_2;
    acc_3  += blockOfA_0 * blockOfB_3; ...; acc_31 += blockOfA_7 * blockOfB_3;
   }
   barrier (...) ;
  }

  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;
  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;
  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;
  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;
} } }
```

Heuristics

For Macro Rules:

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

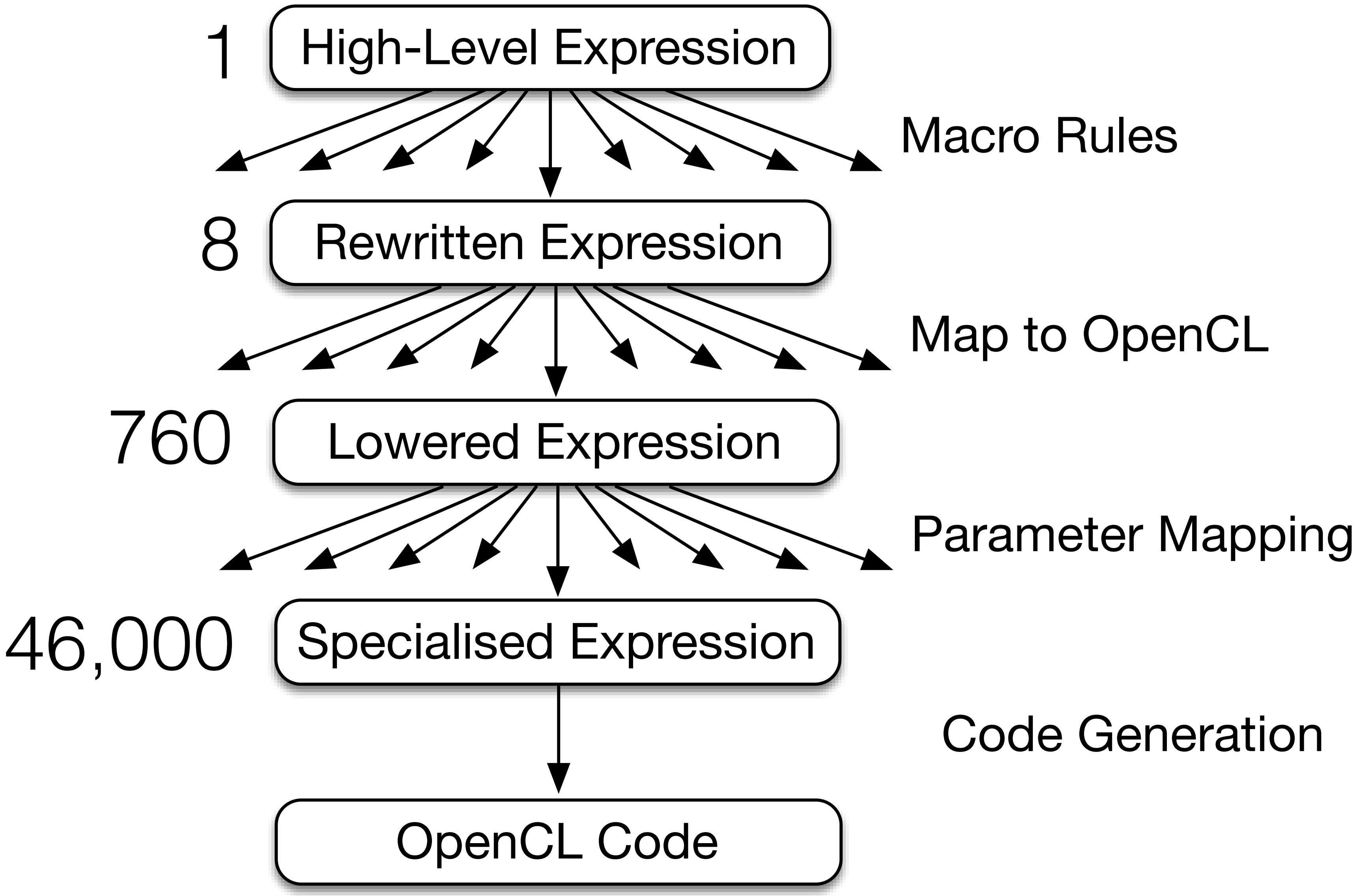
For Map to OpenCL:

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

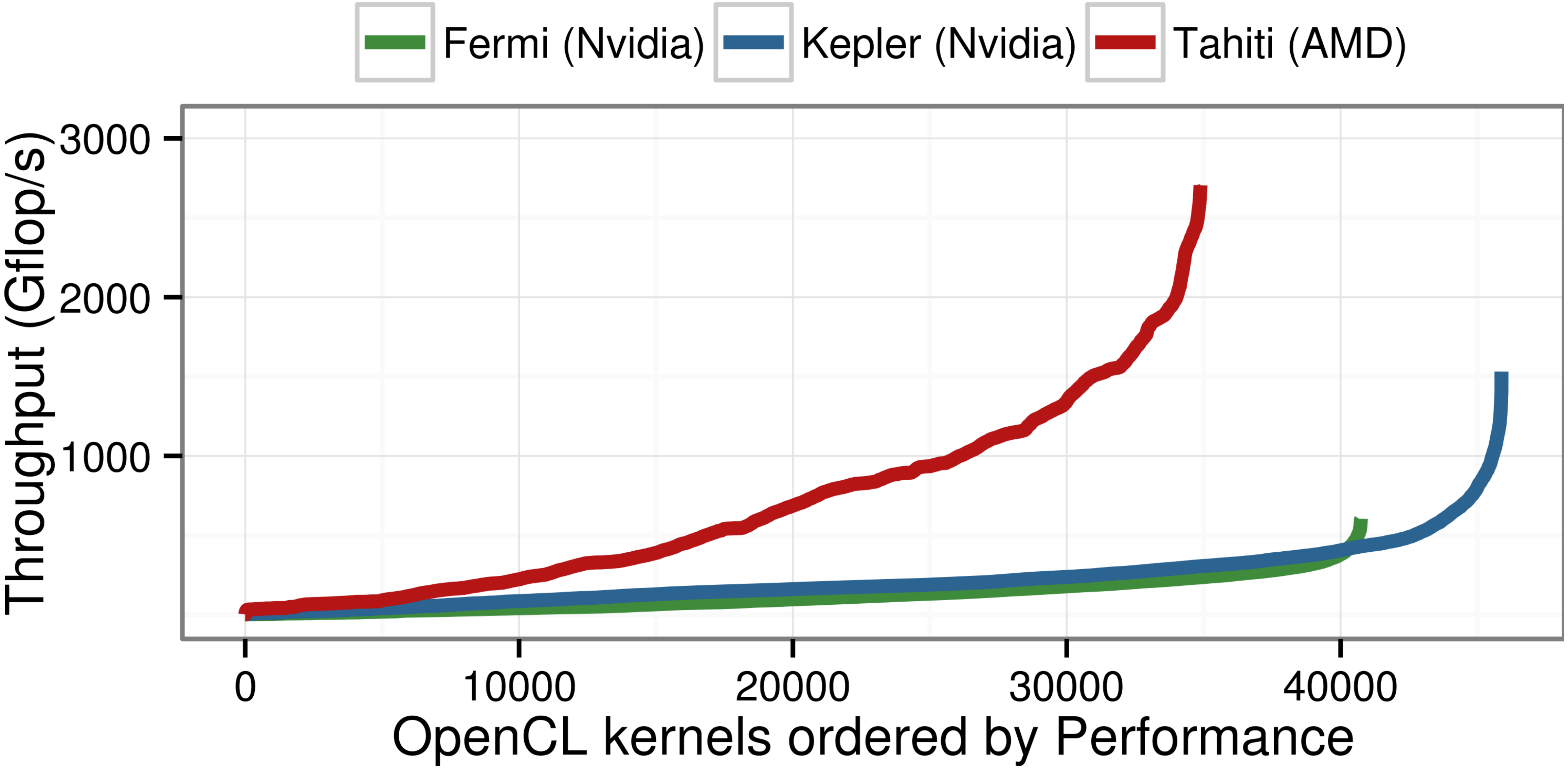
For Parameter Mapping:

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

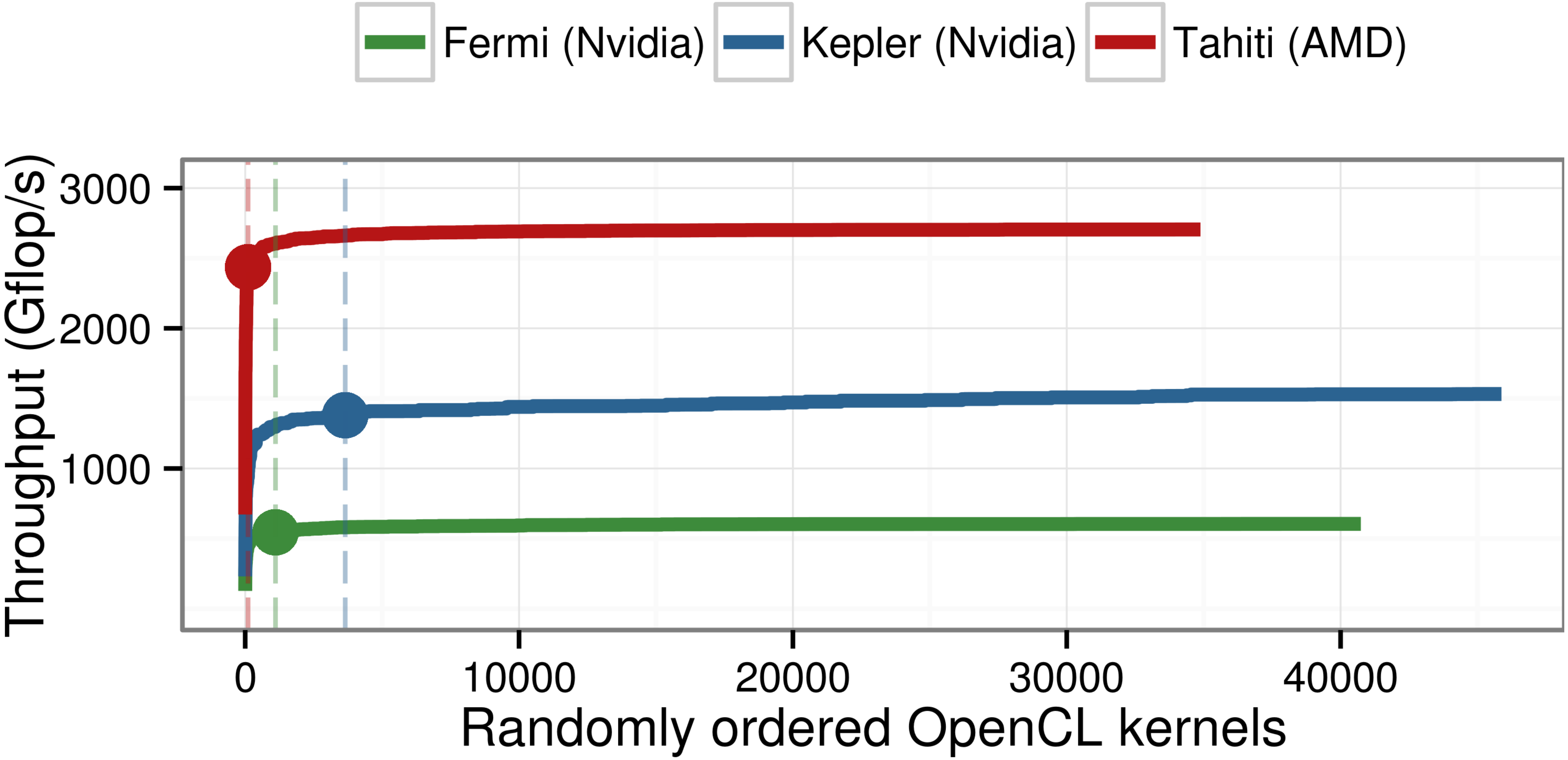
Exploration in Numbers



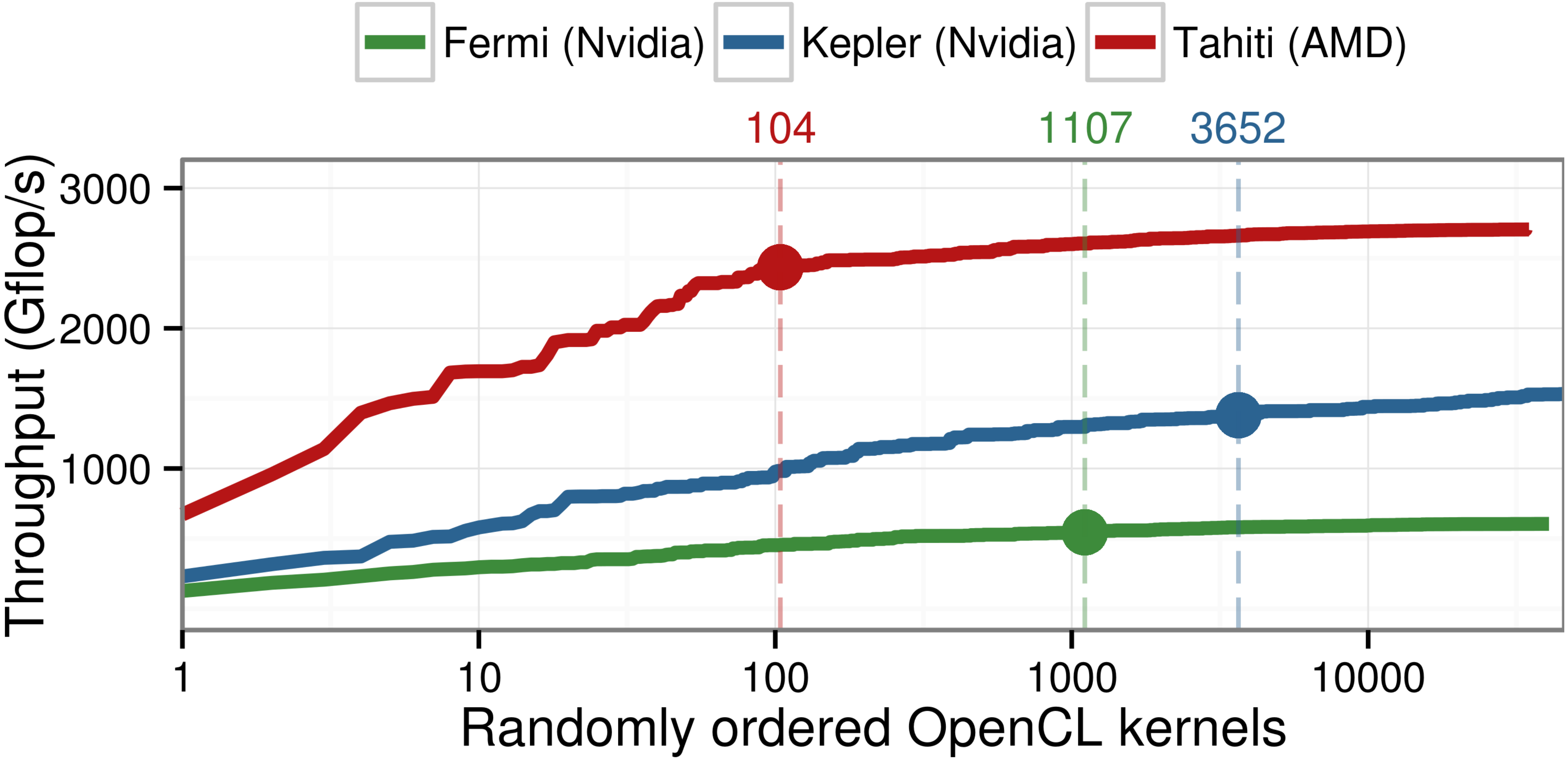
Exploration Space



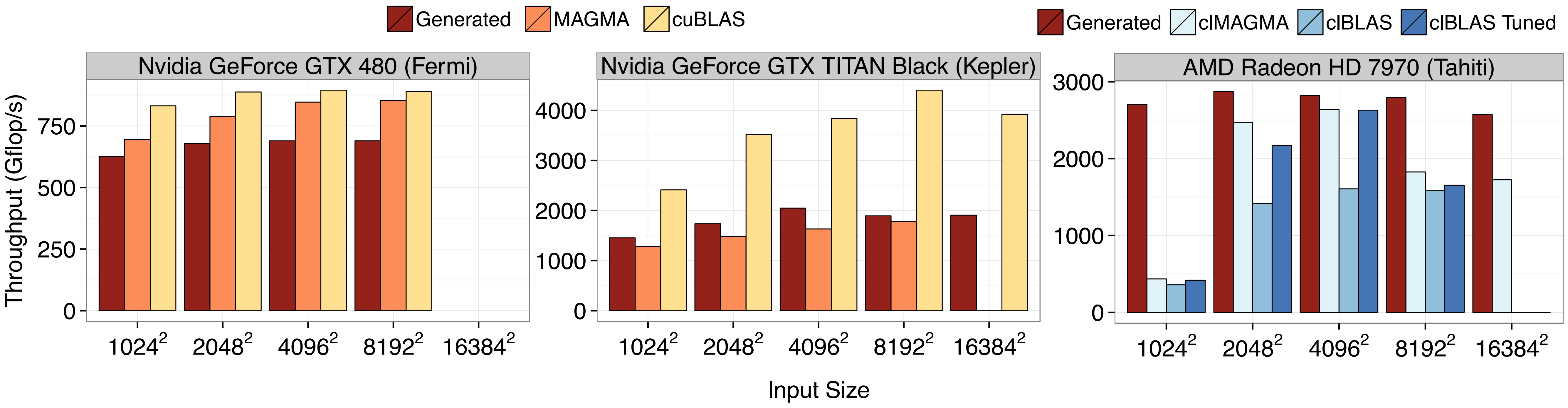
Performance Evolution



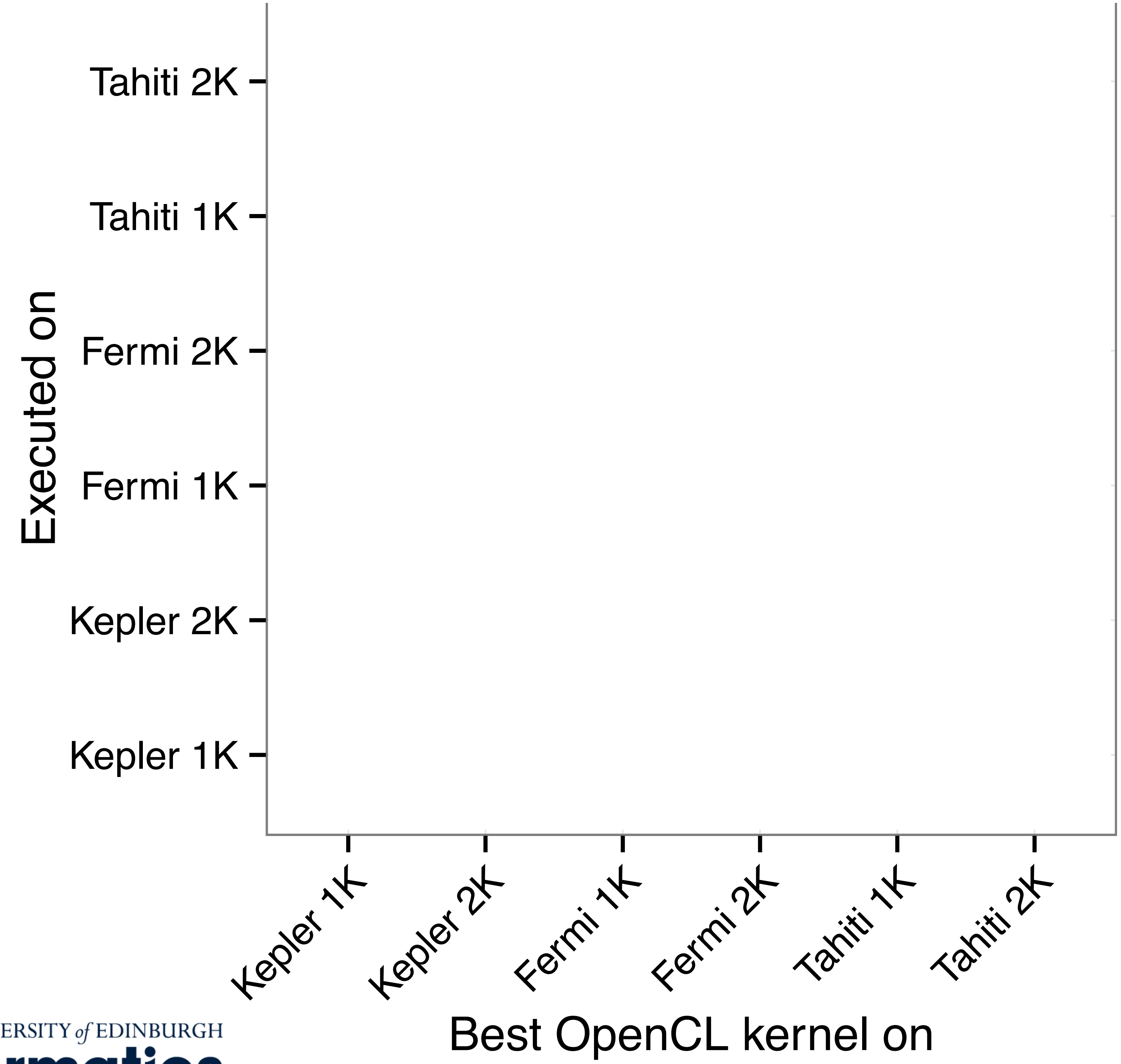
Performance Evolution



Performance Results

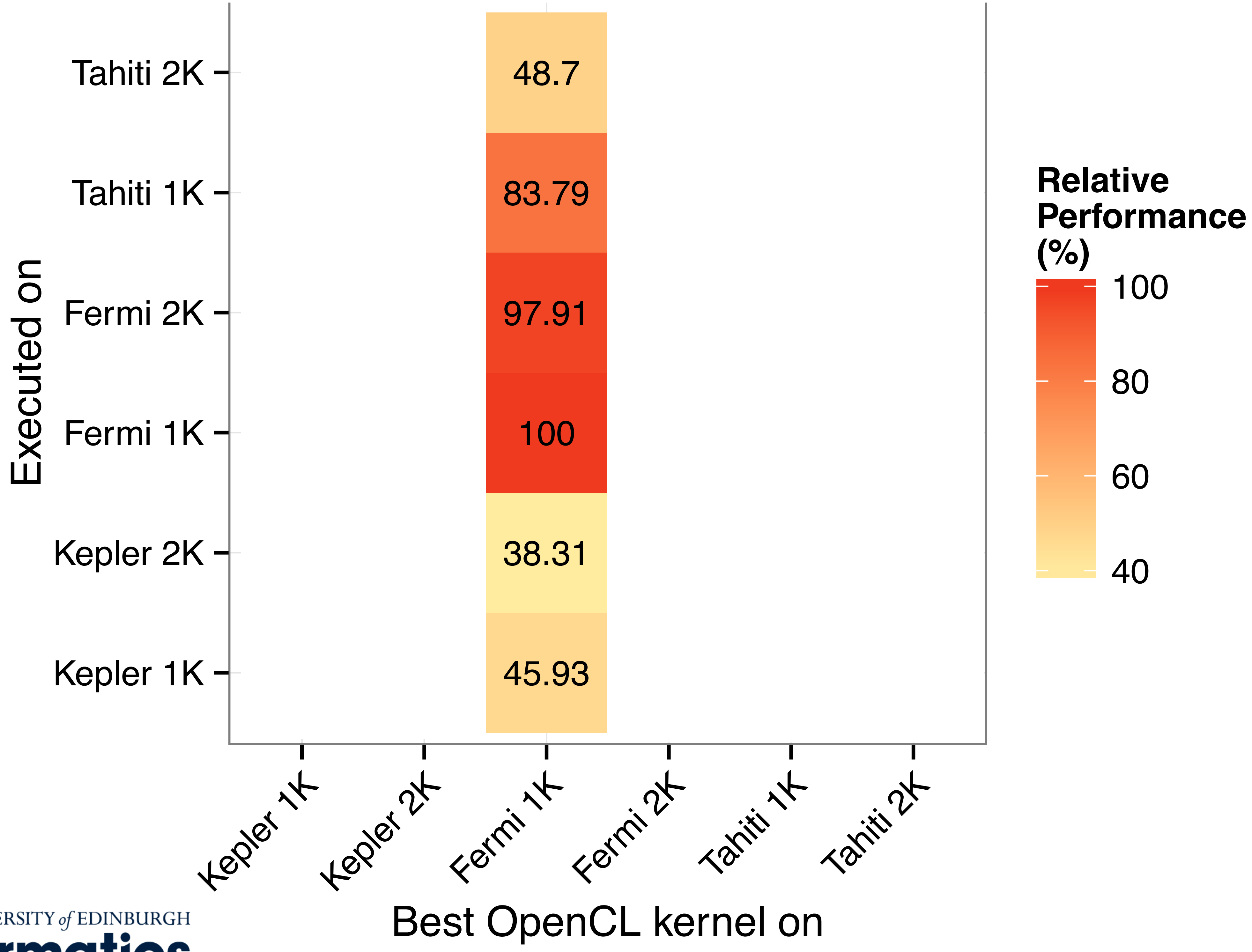


Performance Portability



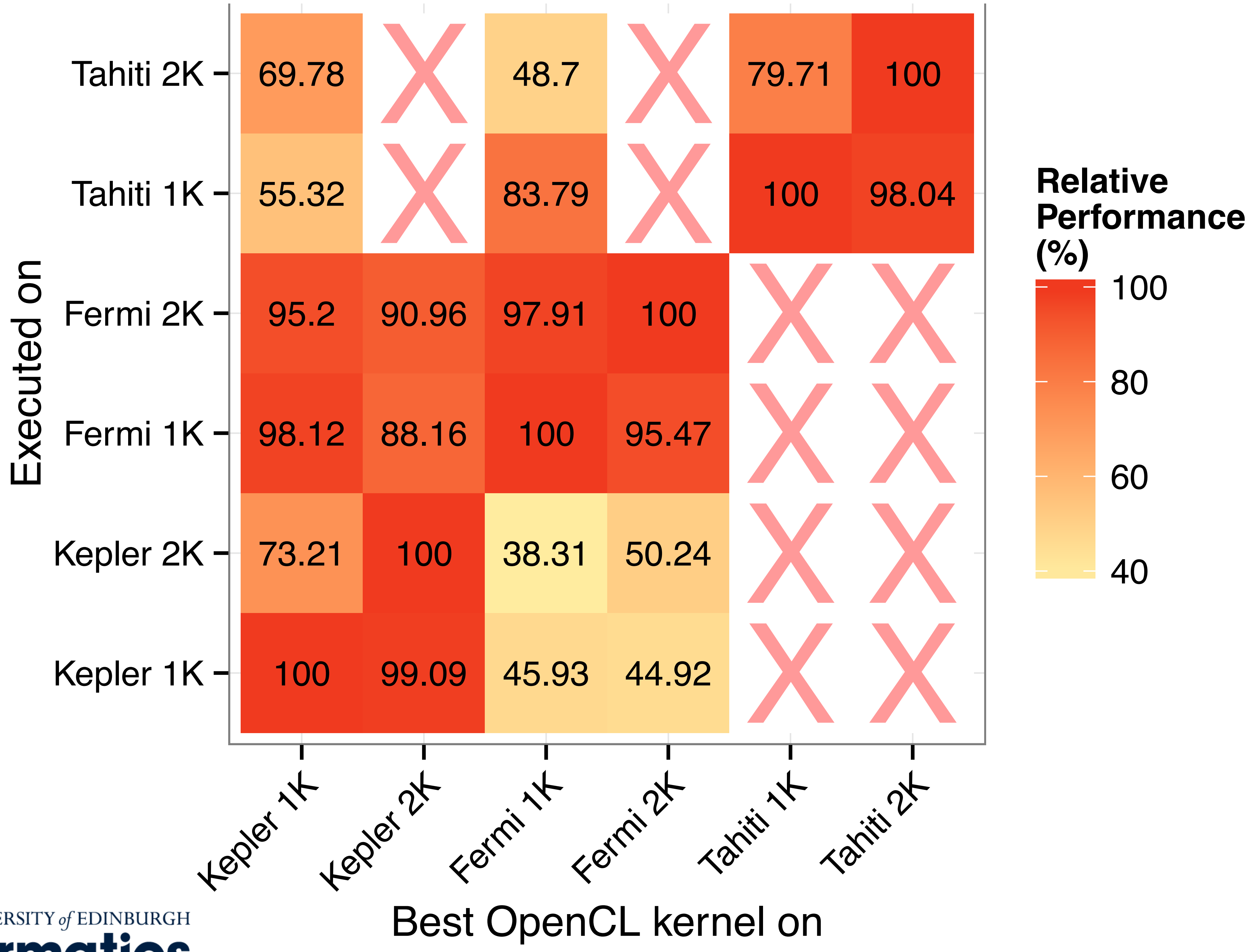
THE UNIVERSITY *of* EDINBURGH
informatics

Performance Portability



THE UNIVERSITY of EDINBURGH
informatics

Performance Portability



Executed on	Kepler 1K	Kepler 2K	Fermi 1K	Fermi 2K	Tahiti 1K	Tahiti 2K
Tahiti 2K	69.78	X	48.7	X	79.71	100
Tahiti 1K	55.32	X	83.79	X	100	98.04
Fermi 2K	95.2	90.96	97.91	100	X	X
Fermi 1K	98.12	88.16	100	95.47	X	X
Kepler 2K	73.21	100	38.31	50.24	X	X
Kepler 1K	100	99.09	45.93	44.92	X	X

Conclusion

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

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

Supported by:

Google Oracle Labs

THE UNIVERSITY of EDINBURGH
informatics