THE UNIVERSITY of EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

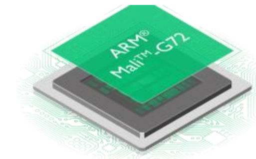
Guided Rewriting and Constraint Satisfaction for Parallel GPU Code Generation

Naums Mogers

May 9th, 2023

Parallel Architectures

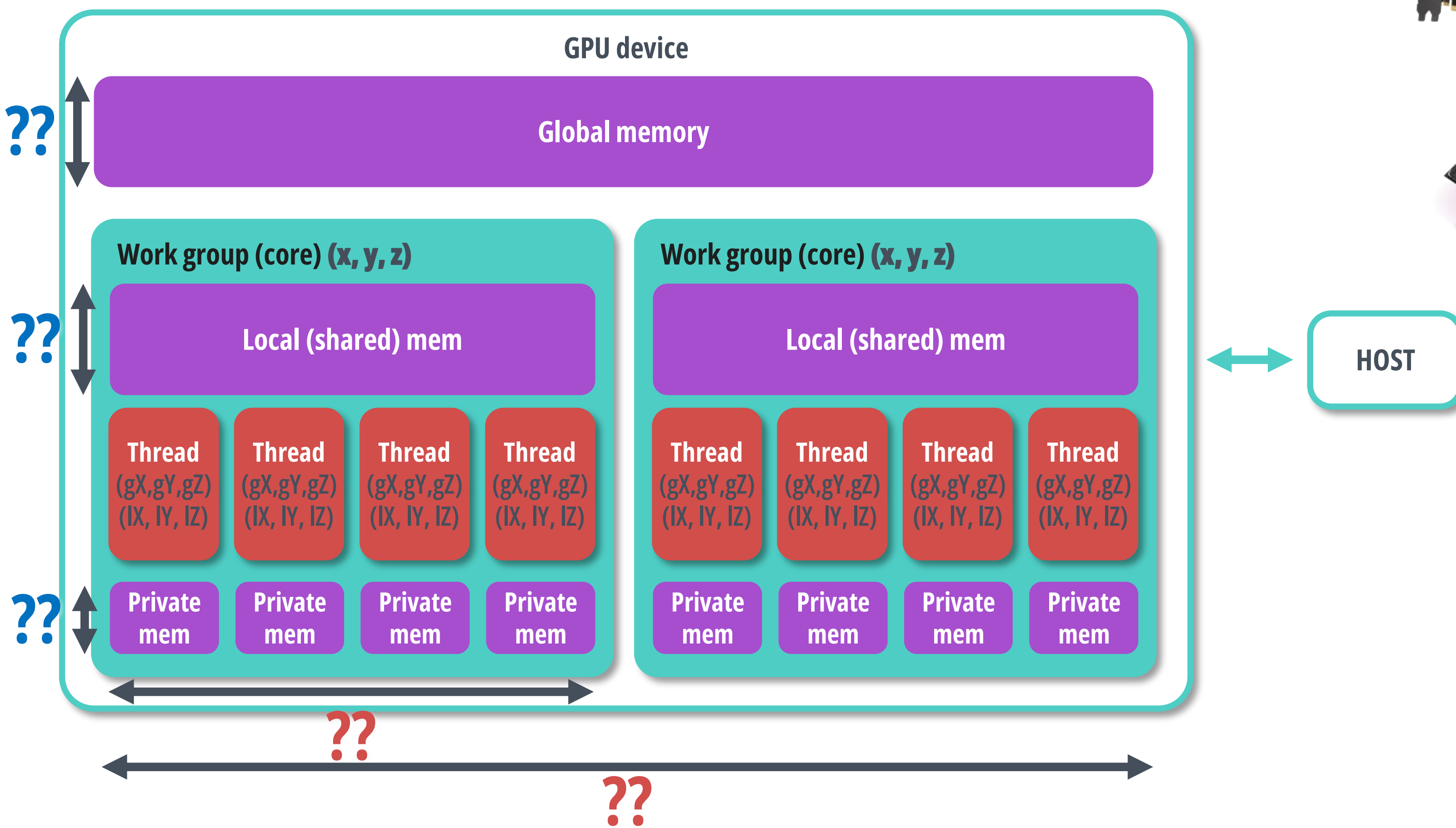
©ARM



©NVIDIA

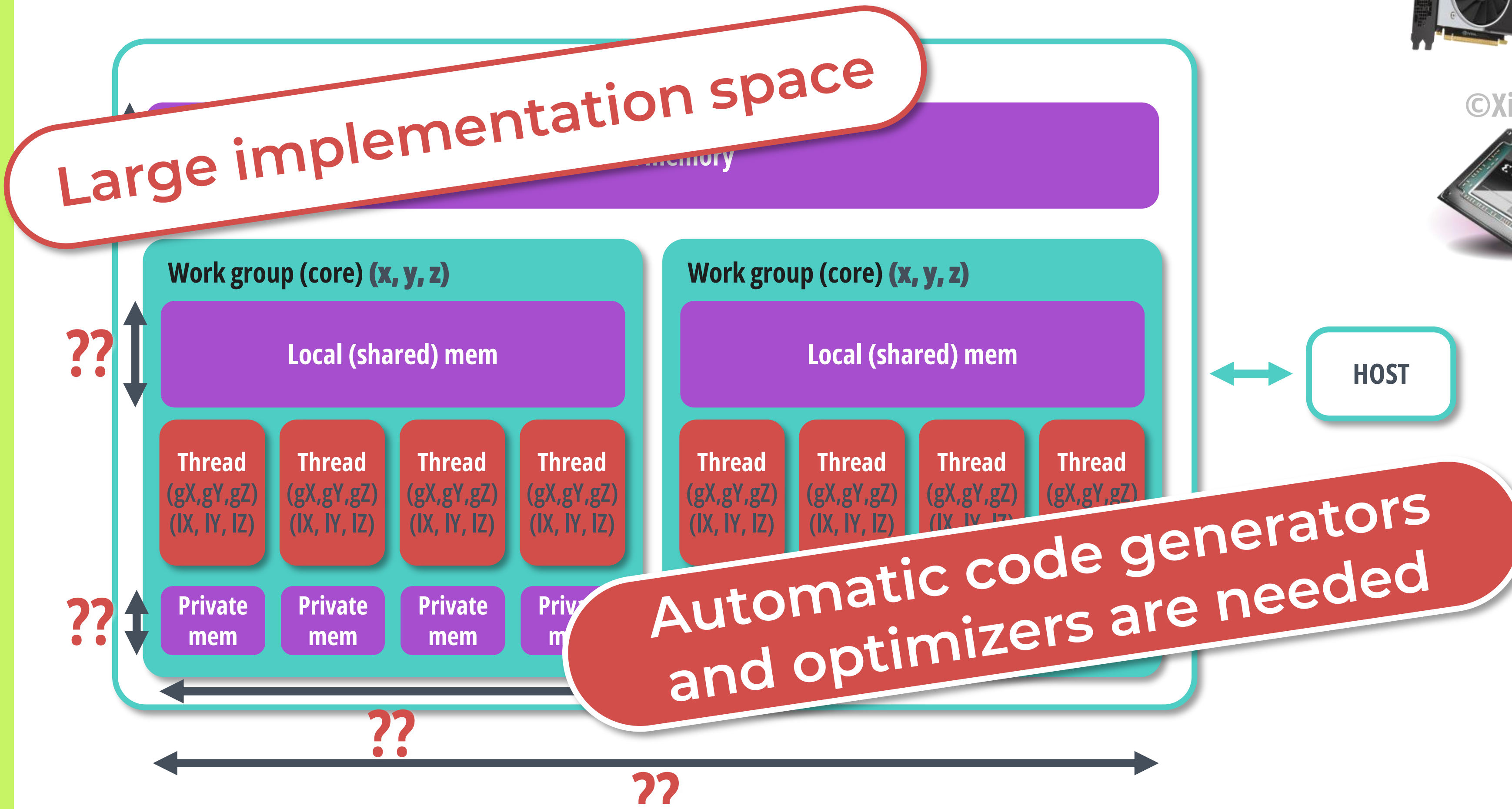
©Xilinx

Parallel architectures are hard to optimize for

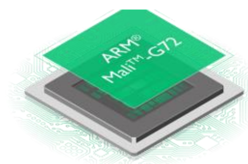


Parallel Architectures

Parallel architectures are hard to optimize for

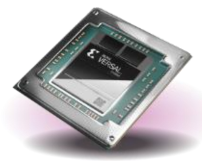


©ARM



©NVIDIA

©Xilinx



The Programmability Challenge

- Hierarchical execution and memory models
- Diverse and heterogeneous accelerator architectures

- Manual optimisation is too costly
- Heuristic optimisation strategies are over-constrained
- Automatic optimisation suffers from the search space explosion

Current Approaches

- Kernel libraries
  - Costly to maintain and extend
- User-provided schedules (*Halide*) and design choices (*PetaBricks, Tangram*)
  - Burden on the user
- Polyhedral compilation (*Tensor Comprehensions*)
  - Limited to affine loops
- Functional rewriting (*Futhark, Accelerate, Lift, RISE*)
  - Dependence on heuristics

Low-level expression close to OpenCL

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(4)) o
transpose o
split(64)
```

??

High-level expression

```
map(f)
```

Low-level expression close to OpenCL

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(4)) o
transpose o
split(64)
```

Parallelised expression

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(v)) o
transpose o
split(s)
```

High-level expression

```
map(f)
```

Chapter 4

Tuning

Chapter 4:
Functional IR for Auto-Tuning

- Shows that a functional IR can represent low-level optimisations in the context of convolution

```
def partialConv(kernelsWeights : [[[[float]inputChannels]kernelWidth]kernelHeight]numKernels,
                paddedInput    : [[[float]inputChannels]paddedInputWidth]paddedInputHeight,
                kernelStride   : (int, int))
                : [[[[[float]]windowSize/ω]σ]nWindowsInTile/σ]κ]numKernels/κ]nTilesInInput = {
  val tiledInput4D = join(slide2D(θ, tilingStride, paddedInput))
  val tiledSlidedInput5D = map(join(slide2D((kernelHeight, kernelWidth), kernelStride)), tiledInput4D)
  val windowSize = inputChannels * kernelWidth * kernelHeight
  def coalesceChunkVectorizeWindow(window : [[[float]inputChannels]kernelWidth]kernelHeight])
                                   : [[floatυ]ω]windowSize/ω = {
    val flatWindow1D = join(join(window))
    val flatCoalescedWindow1D = reorder(stridedIndex(windowSize/ω), flatWindow1D)
    val flatCoalescedChunkedWindow1D = split(ω, flatCoalescedWindow1D)
    asVector(υ, flatCoalescedChunkedWindow1D) }
  val tiledSlidedCoalescedChunkedVectorizedInput4D = map(tile4D -> split(σ, map
    coalesceChunkVectorizeWindow(window3D), tile4D)), tiledSlidedInput5D)
  val groupedCoalescedChunkedVectorizedKernelsWeights4D = split(κ, map(singleKer
    coalesceChunkVectorizeWindow(singleKernelWeights), kernelsWeights))
  mapWrg(1, inputTile3D ->
    mapWrg(0, kernelsGroupWeights3D -> transpose(
      mapLcl(1, inputWindows2D -> transpose(
        mapLcl(0, (inputWindowsChunk1D, kernelsGroupChunk2D) ->
          mapSeq(singleKernelReducedChunk -> toGlobal(singleKernelReducedC
            join(
              reduceSeq(
                init = mapSeq(toPrivate(id(Value(0, [float]κ)))),
                f = (acc, (inputsValue, kernelsGroupValue1D)) ->
                  let(inputsValuePrivate ->
                    mapSeq((accValue, singleKernelValue) ->
                      mapSeq((inputValuePrivate) ->
                        accValue + vectorize(υ, dot(inputValuePrivate, singleKernelValue)),
                        inputsValuePrivate,
                        zip(acc, kernelsGroupValue1D),
                    mapSeq(toPrivate(vectorize(υ, id(inputValue))))),
                zip(transpose(inputWindowsChunk1D), transpose(kernelsGroupChunk2D))))),
          zip(inputWindows2D, transpose(kernelsGroupWeights3D)))),
        inputTile3D)),
      groupedCoalescedChunkedVectorizedKernelsWeights4D),
    tiledSlidedCoalescedChunkedVectorizedInput4D)
```

Tiling
Coalescing
Vectorization
Loop unrolling
Kernel grouping
Thread coarsening
Memory optimization

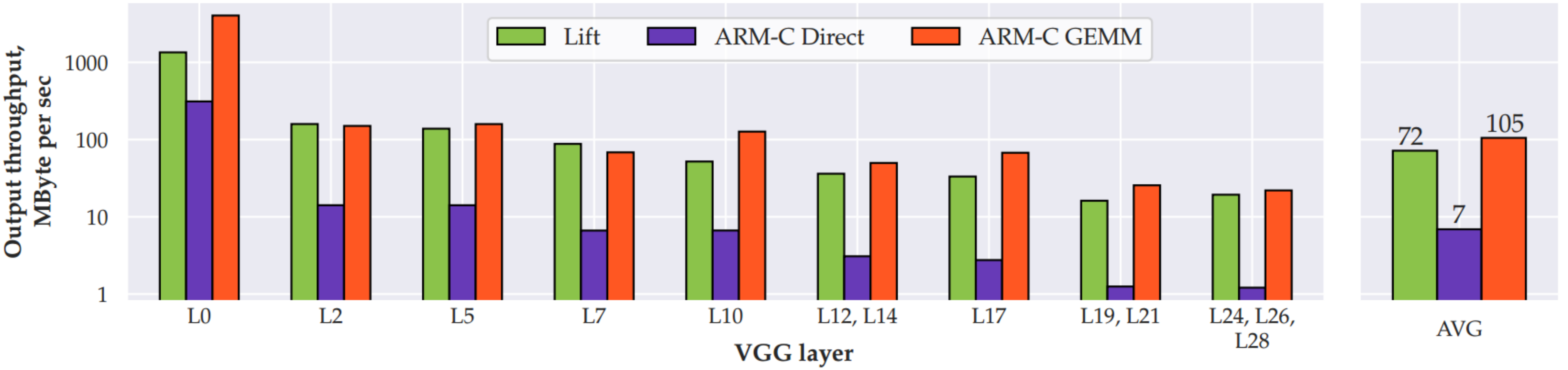
Chapter 4:
Functional IR for Auto-Tuning

- Shows that a functional IR can in the context of convolution

- Describes an auto-tuning approach which leverages strongly typed functional patterns

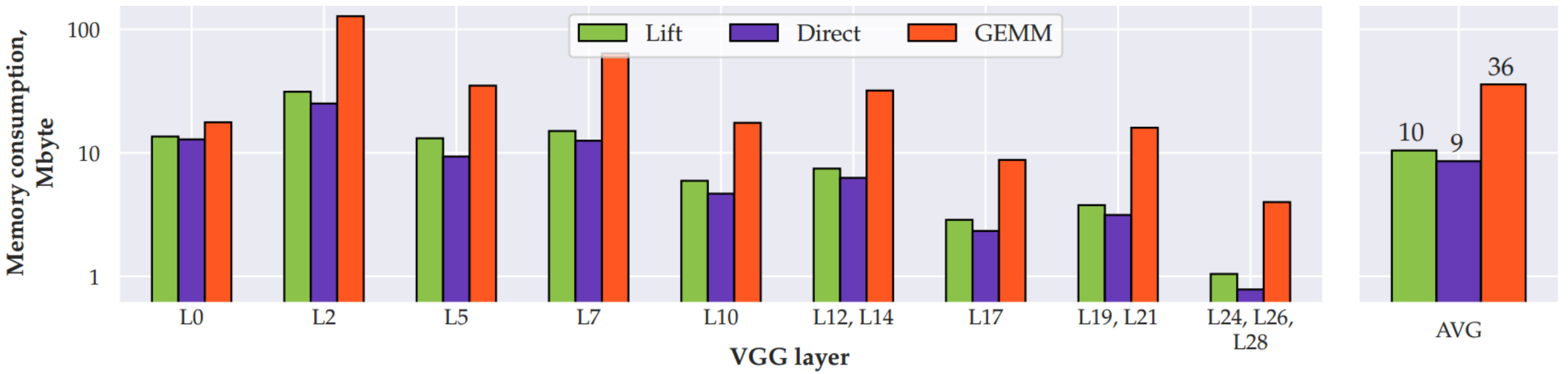
Split(s) \$ $[T]_N$	=>	$N \% s == 0$
asVector(v) \$ $[T]_N$	=>	$N \% v == 0$
Slide(len, $step$) o $[T]_N$	=>	$N >= len$
SlideStrict(len, $step$) o $[T]_N$	=>	$N >= len$ && $N \% ((N - (len - step)) / step) == 0$

Results: Throughput



- Lift kernels are:
    - Always faster than direct convolution in ARM-C **(x10 on average)**
    - In some cases, on par or better than GEMM in ARM-C **(x0.7 on average)**

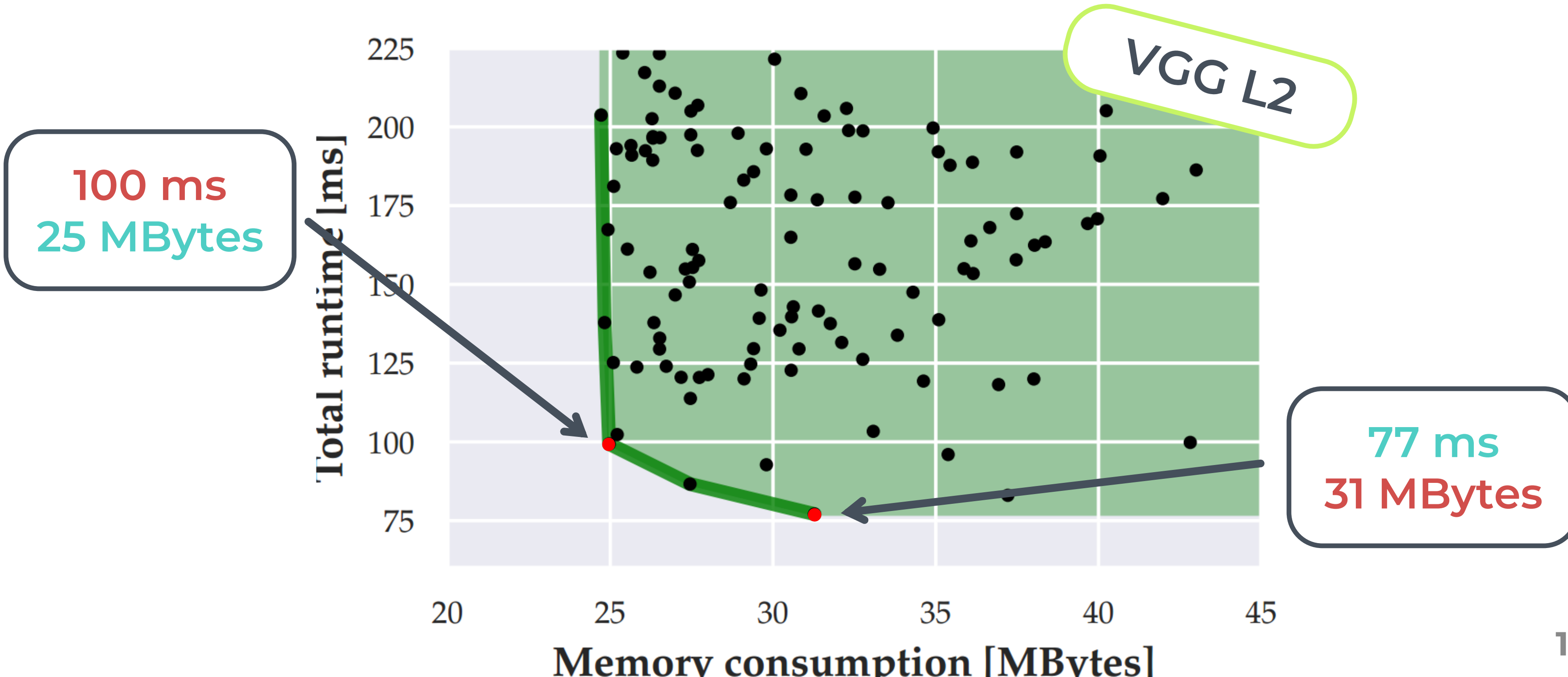
Results: Memory Consumption

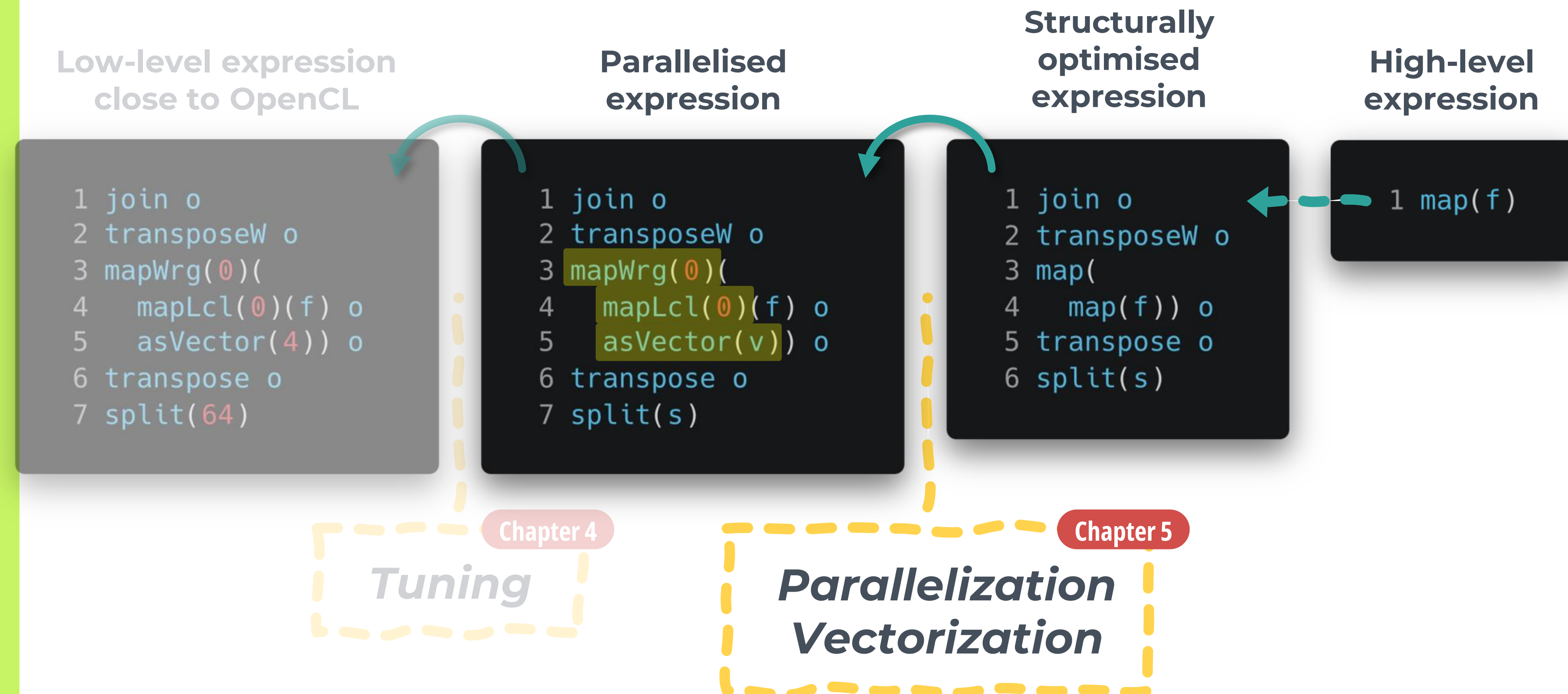


- Memory consumption of Lift kernels is:
  - On par with direct convolution in ARM-C **(x1.1 on average)**
  - Always better than GEMM in ARM-C **(x3.6 on average)**

Results: Multi-Objective Optimisation

- Shifting priorities:
  - Low memory footprint vs throughput/latency
- Search space exploration for multi-objective optimization




Low-level expression close to OpenCL
join o
transposeW o
mapWrg(0)(
mapLcl(0)(f) o
asVector(4)) o
transpose o
split(64)
Parallelised expression
join o
transposeW o
mapWrg(0)(
mapLcl(0)(f) o
asVector(v)) o
transpose o
split(s)
Structurally optimised expression
join o
transposeW o
map(
map(f)) o
transpose o
split(s)
High-level expression
map(f)
Chapter 4
Tuning
Chapter 5
Parallelization
Vectorization

Chapter 5:
Parallelism Mapping Through Constraint Satisfaction

> *I found the approach proposed by the author extremely convincing and fairly natural (to the point I'm almost surprised it wasn't proposed earlier).*

CC'22 reviewer

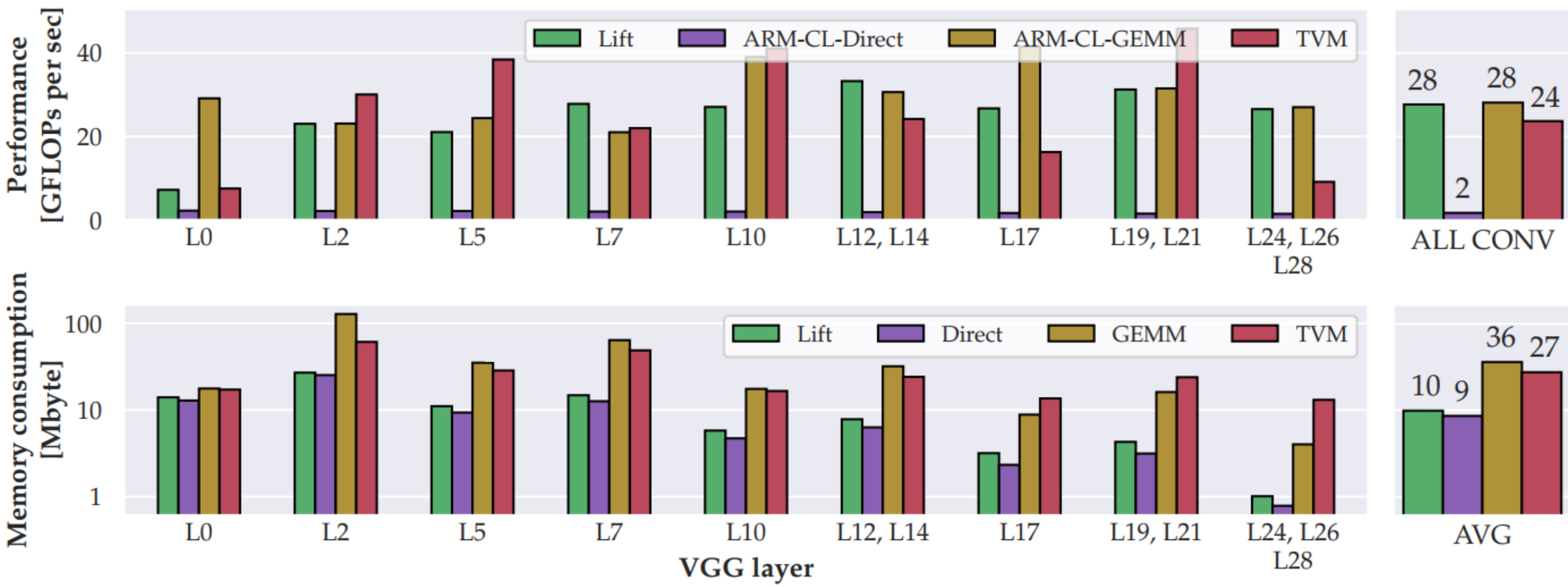
Chapter 5:
Parallelism Mapping Through Constraint Satisfaction

- Functional patterns
  - …expose parallelism
  - …express parallel restrictions succinctly
  - …aid dependency analysis for synchronization

- Chapter 5:
  - Expresses the GPU parallel programming model as arithmetic constraints on functional patterns
  - Uses a constraint solver to explore parallel mappings
  - Describes a functional IR-based barrier insertion method

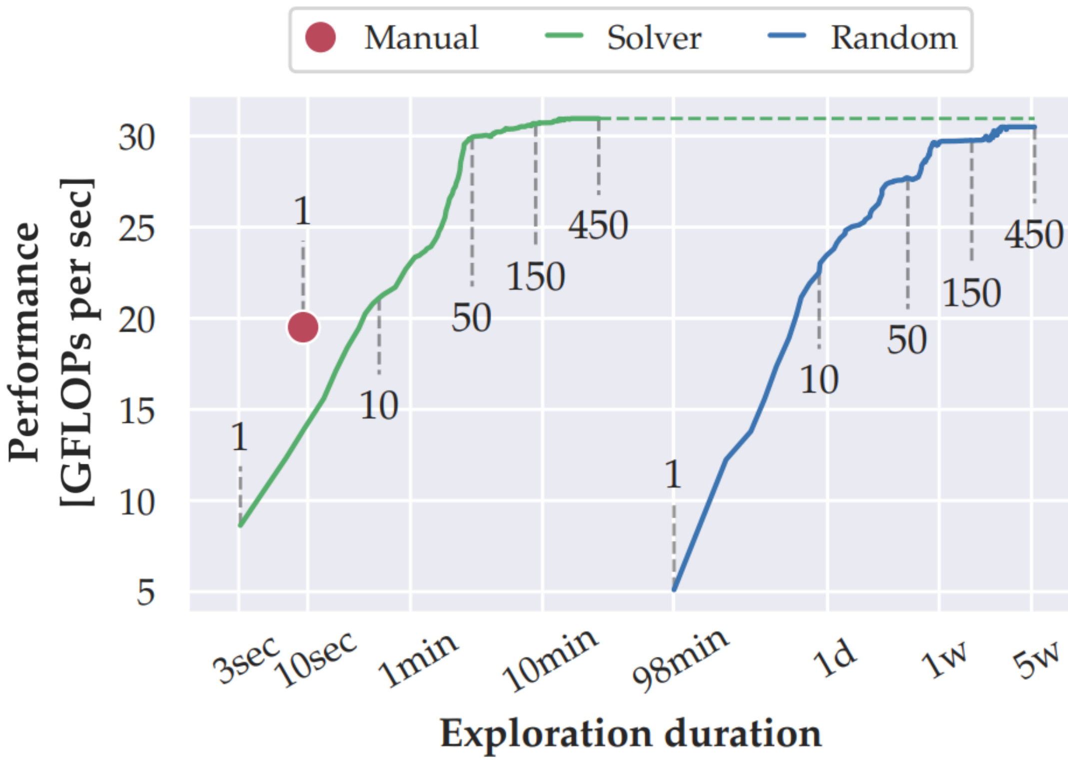
Results: Performance & Memory

VGG-16 on Mali-G72 GPU



- Stencil performance **on par** with ARM-CL GEMM and **0.86x** of TVM's
- **3.6x** less memory on average than ARM-CL GEMM
- **2.7x** less memory on average than TVM

Results: Exploration Efficiency



- Peak performance after 95 minutes
- Peaks before the random approach produces even 1 result (a bad one)

Low-level expression close to OpenCL

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(4)) o
transpose o
split(64)
```

Parallelised expression

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(v)) o
transpose o
split(s)
```

Structurally optimised expression

```
join o
transposeW o
map(
  map(f)) o
transpose o
split(s)
```

High-level expression

```
map(f)
```

Chapter 4

Tuning

Chapter 5

Parallelization
Vectorization

Chapter 6

Algorithmic optimisations
HW-specific optimisations

Chapter 6:
Towards Guided Rewriting

- Expressing design choices directly in a functional IR:
  - Decouples optimisation from code generation
  - Truncates the search to valid implementations
  - Helps the user drive rewriting loosely through composable optimisations
- Chapter 6:
  - Defines eleven rewrite points expressing a range of low-level optimisations
  - Shows how two convolution algorithms are produced from one expression

Single annotated convolution expression

```
1  def conv( inputData      :  [[[float]inChs]inW]inH,
2            kernelsWeights :  [[[[float]inChs]kerW]kerH]outChs,
3            padSize        :  (int,int,int,int),
4            kernelStride   :  (int,int) ) :  [[[float]outChs]outW]outH =
5   toHost o
6   padExtra2D( kernelFission(
7    joinSplitND( padExtra1D( kernelFission( slideWindows' ⇒
8     kernelFission( kernelsWeights' ⇒
9      oclKernel((slideWindows'', kernelsWeights'') ⇒
10      abTile( abTile(
11       tileNestedMapReduce(reduceFun = +)(
12        tileNestedMapReduce(reduceFun = +)(
13         abTile(
14          mapND2(slideWin: [T](inChs * kerW * kerH) ⇒
15           map(singleK: [T](inChs * kerW * kerH) ⇒
16            privatiseAccumulator(
17             reduce(0, +) o map(*) << zip(slideWin, singleK) )
18           ) << kernelsWeights''
19          ) << slideWindows'' )))))
20     ) << (slideWindows', kernelsWeights')
21    ) o map(joinND2) o toGPU << kernelsWeights
22   ))) o mapND2(joinND2) o
23        slideND2(kerH, kerW, kernelStride._1, kernelStride._2)
24  )) o padND2(padSize, value = 0) o toGPU << inputData
```

#	Rewrite Point	Struct and tune params	Result	Line
1	padExtra2D	**ps**=[4,0]	Extra edge padding, depadding	**6**
2	kernelFission		Input preprocessing OpenCL kernel	**6**
3	↳ optimiseDataLayout	**newDimOrder**=[0,2,1,3], **factors**=[8,48,4]	Reshaped input for coalescing	
4	↳ materialise	**targetMem**=global	Input reshaping materialised	
5	↳ splitJoinND	**tileSizes**=[1,1,4]	Parallelisable vectorisable loops	
		slideND2		
6	kernelFission		Weight preprocessing OpenCL kernel	**8**
7	↳ optimiseDataLayout	**newDimOrder**=[0,1,3,2,4], **factors**=[1,8,48,4]	Reshaped weights for coalescing	
8	↳ materialise	**targetMem**=global	Weight reshaping materialised	
9	↳ splitJoinND	**tileSizes**=[1,1,4]	Vectorisable loop nest	
10	abTile	**tileSizeA**=64, **tileSizeB**=4	Work groups mapped over input & weight tiles	**10**
11	↳ interchangeMaps		Local threads mapped over input tiles	
12	abTile	**tileSizeA**=4, **tileSizeB**=4	Work groups mapped over weight subtiles	**10**
13	tileNestedMapReduce	**tileSize**=192	Parallelisable reduction tree	**11**
14	↳ interchangeReduceND	**newReduceDim**=3, **factors**=[4,4]	Parallelisable final reduction	
15	↳ privatiseAccumulator	**resultMem**=global	Faster accumulator access	
16	↳ splitJoinND	**factors**=[3,8]	Local threads mapped over input & weight subtiles	
17	tileNestedMapReduce	**tileSize**=4	Sliding window tiled for prefetching	**12**
18	↳ privatiseAccumulator	**resultMem**=local	Shared partial reduction results	
19	abTile	**tileSizeA**=4, **tileSizeB**=4	Subtiled inputs and weights for prefetching	**13**
20	↳ materialise	**targetMem**=private	Prefetched a 4x4 tile of inputs	
21	↳ splitJoinND	**tileSizes**=[1,4,4]	Vectorisable loop	
22	↳ materialise	**targetMem**=private	Prefetched a 4x4 tile of weights	
23	↳ splitJoinND	**tileSizes**=[1,4,4]	Vectorisable loop	
24	privatiseAccumulator	**targetMem**=private	Faster accumulator access	**16**

High-performance **direct** convolution

#	Rewrite Point	Struct and tune parameters	Result	Line
		slideND2		
1	padExtra1D	**p**=6	Corner padding, depadding	**7**
2	kernelFission		Input preprocessing OpenCL kernel	**7**
3	↳ optimiseDataLayout	**newDimOrder**=[0,3,1,2,4], **factors**=[2,5,1152,4]	Reshaped input for coalescing	
4	↳ materialise	**targetMem**=global	Input reshaping materialised, im2col	
5	↳ splitJoinND	**tileSizes**=[9,32,4]	Parallelisable and vectorisable loops	
6	kernelFission		Weight preprocessing OpenCL kernel	**8**
7	↳ optimiseDataLayout	**newDimOrder**=[0,3,2,1,4], **factors**=[16,4,1152,4]	Reshaped weights for coalescing	
8	↳ materialise	**targetMem**=global	Weight reshaping materialised	
9	↳ splitJoinND	**tileSizes**=[1,1,4]	Vectorisable loop nest	
10	abTile	**tileSizeA**=5, **tileSizeB**=4	Global threads mapped over input & weight tiles	**10**
11	tileNestedMapReduce	**tileSize**=4	Each thread reduces 4 interleaved windows	**12**
12	↳ privatiseAccumulator	**resultMem**=global	Faster accumulator access	
13	abTile	**tileSizeA**=5, **tileSizeB**=4	Subtiled inputs and weights for prefetching	**13**
14	↳ materialise	**targetMem**=private	Prefetched a 5x4 tile of inputs	
15	↳ splitJoinND	**tileSizes**=[1,5,4]	Vectorisable loop	
16	↳ materialise	**targetMem**=private	Prefetched a 4x4 tile of weights	
17	↳ splitJoinND	**tileSizes**=[1,4,4]	Vectorisable loop	
18	privatiseAccumulator	**targetMem**=private	Faster accumulator access	**16**

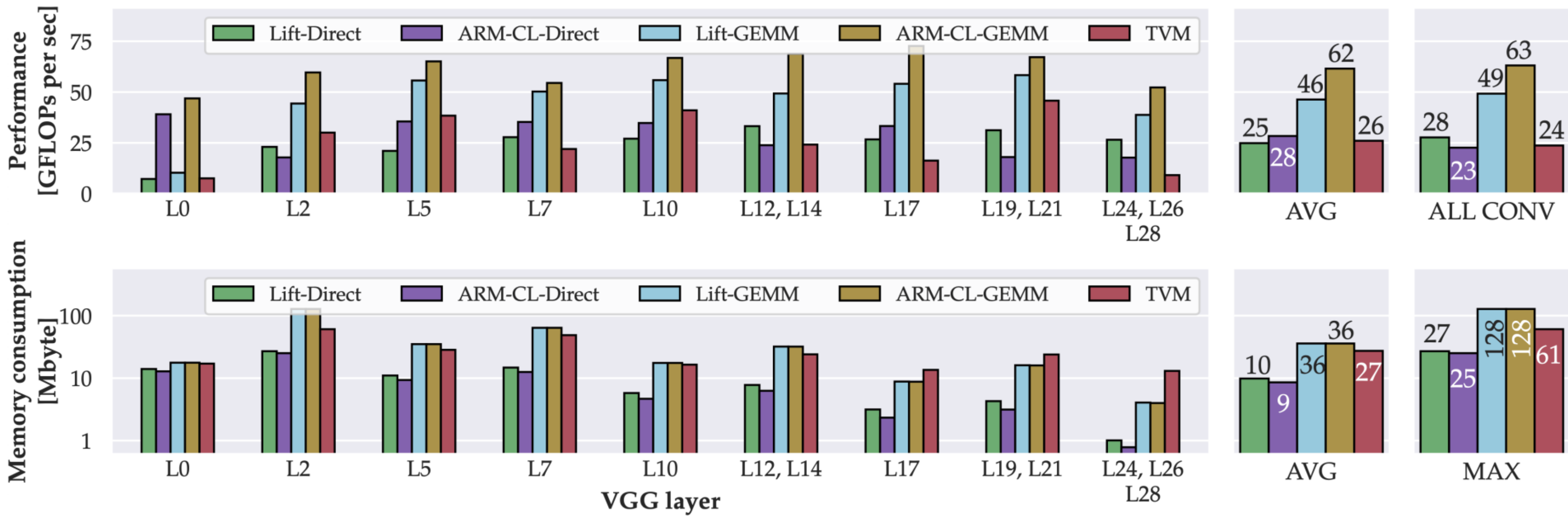
High-performance **im2col+GEMM** #1

#	Rewrite Point	Struct and tune parameters	Result	Line
		slideND2		
2	kernelFission		Input preprocessing OpenCL kernel	**7**
4	↳ materialise	**targetMem**=global	Input reshaping materialised, im2col	
5	↳ splitJoinND	**tileSizes**=[9,32,4]	Parallelisable and vectorisable loops	
6	kernelFission		Weight preprocessing OpenCL kernel	**8**
7	↳ optimiseDataLayout	**newDimOrder**=[0,3,2,1,4], **factors**=[16,4,1152,4]	Reshaped weights for coalescing	
8	↳ materialise	**targetMem**=global	Weight reshaping materialised	
9	↳ splitJoinND	**tileSizes**=[1,1,4]	Vectorisable loop nest	
10	abTile	**tileSizeA**=5, **tileSizeB**=4	Global threads mapped over input & weight tiles	**10**
11	tileNestedMapReduce	**tileSize**=4	Each thread reduces 4 interleaved windows	**12**
12	↳ privatiseAccumulator	**resultMem**=global	Faster accumulator access	
13	abTile	**tileSizeA**=5, **tileSizeB**=4	Subtiled inputs and weights for prefetching	**13**
14	↳ materialise	**targetMem**=private	Prefetched a 5x4 tile of inputs	
15	↳ splitJoinND	**tileSizes**=[1,5,4]	Vectorisable loop	
16	↳ materialise	**targetMem**=private	Prefetched a 4x4 tile of weights	
17	↳ splitJoinND	**tileSizes**=[1,4,4]	Vectorisable loop	
18	privatiseAccumulator	**targetMem**=private	Faster accumulator access	**16**

High-performance **im2col+GEMM** #2

Results: Performance & Memory

VGG-16 on Mali-G72 GPU



- Using three Lift-generated kernels:
  - Stencil performance **on par** with ARM-CL and TVM
  - Lift-GEMM **outperforms** TVM and **0.77x** of ARM-CL-GEMM
  - Lift-GEMM uses 33% more memory than TVM on average

Low-level expression close to OpenCL

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(4)) o
transpose o
split(64)
```

Parallelised expression

```
join o
transposeW o
mapWrg(0)(
  mapLcl(0)(f) o
  asVector(v)) o
transpose o
split(s)
```

Structurally optimised expression

```
join o
transposeW o
map(
  map(f)) o
transpose o
split(s)
```

High-level expression

```
map(f)
```

Chapter 4

Tuning

Chapter 5

Parallelization
Vectorization

Chapter 6

Algorithmic optimisations
HW-specific optimisations

Auto-Tuning

Convolution: state of the art

- Express as General Matrix Multiplication (GEMM) using ***im2col***
- High-performance libraries provide fast solutions on most devices

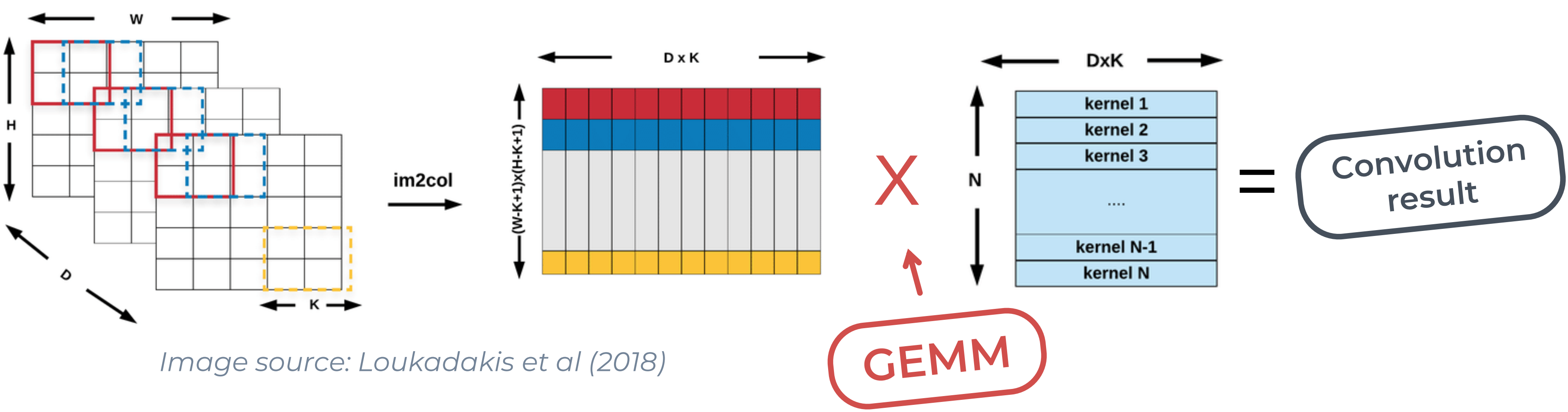
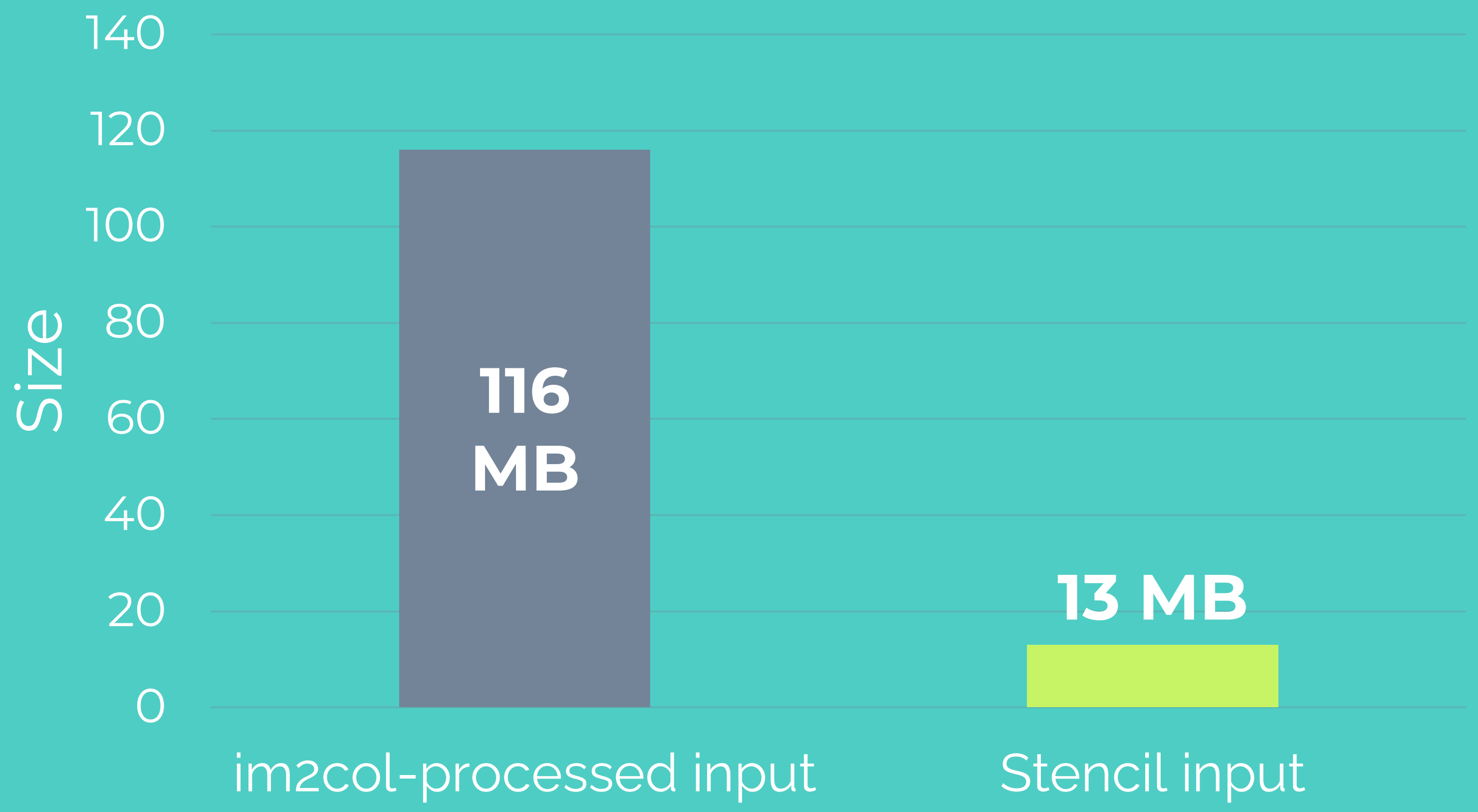


Image source: Loukadakis et al (2018)

- im2col increases memory consumption
  - This is a problem for resource-constrained platforms

Input image size in the two methods (2nd layer of VGG)



Constraints In Convolution

- Tuning parameter constraints depend on the applied rewrites

- Too many possible constraints to enumerate
  - Tile size has to be a factor of spatial input size
  - Vector length has to be a factor of a window size
  - Buffer size must not exceed device limit
  - …etc

Experimental Setup

- Benchmark:
  - **VGG-16**
- OpenCL kernel generation:
  - **1000 random candidates per layer**
- Comparison:
  - **Autotuned ARM Compute Library kernels**
- Platform:
  - **ARM Mali-G72 (12 cores) mobile GPU with Kirin 970 SoC**

Parallelism Mapping

Parallelization Is Hard

```
for i in 1 to I do
  for j in 1 to J do
    for k in 1 to K do
      shared_buf[j][k] = f( input[i][j][k] );

  for k in 1 to K do
    for j in 1 to J do
      output[i][j][k] = g( shared_buf[j][k] );
```

Transposed!

Parallelization Is Hard

```
for i in global_id(0) to I do
  for j in 1 to J do
    for k in 1 to K do
      shared_buf[j][k] = f( input[i][j][k] );

  for k in 1 to K do
    for j in 1 to J do
      output[i][j][k] = g( shared_buf[j][k] );
```

Under-saturated cores

Parallelization Is **Hard**

```
for i in global_id(1) to I do
  for j in global_id(0) to J do
    for k in 1 to K do
      shared_buf[j][k] = f( input[i][j][k] );

  for k in global_id(0) to K do
    for j in 1 to J do
      output[i][j][k] = g( shared_buf[j][k] );
```

Different threads

Cannot synchronize

Parallelization Is Hard

```
for i in 1 to I do
  for j in group_id(0) to J do
    for k in local_id(0) to K do
      shared_buf[j][k] = f( input[i][j][k] );

  for k in group_id(0) to K do
    for j in local_id(0) to J do
      output[i][j][k] = g( shared_buf[j][k] );
```

Different thread blocks

Out-of-scope reads

Parallelization Is Hard

```
for i in group_id(0) to I do
  for j in local_id(0) to J do
    for k in 1 to K do
      shared_buf[j][k] = f( input[i][j][k] );
  barrier();
  for k in local_id(0) to K do
    for j in 1 to J do
      output[i][j][k] = g( shared_buf[j][k] );
```

Synchronization overhead

Parallelization Is Hard

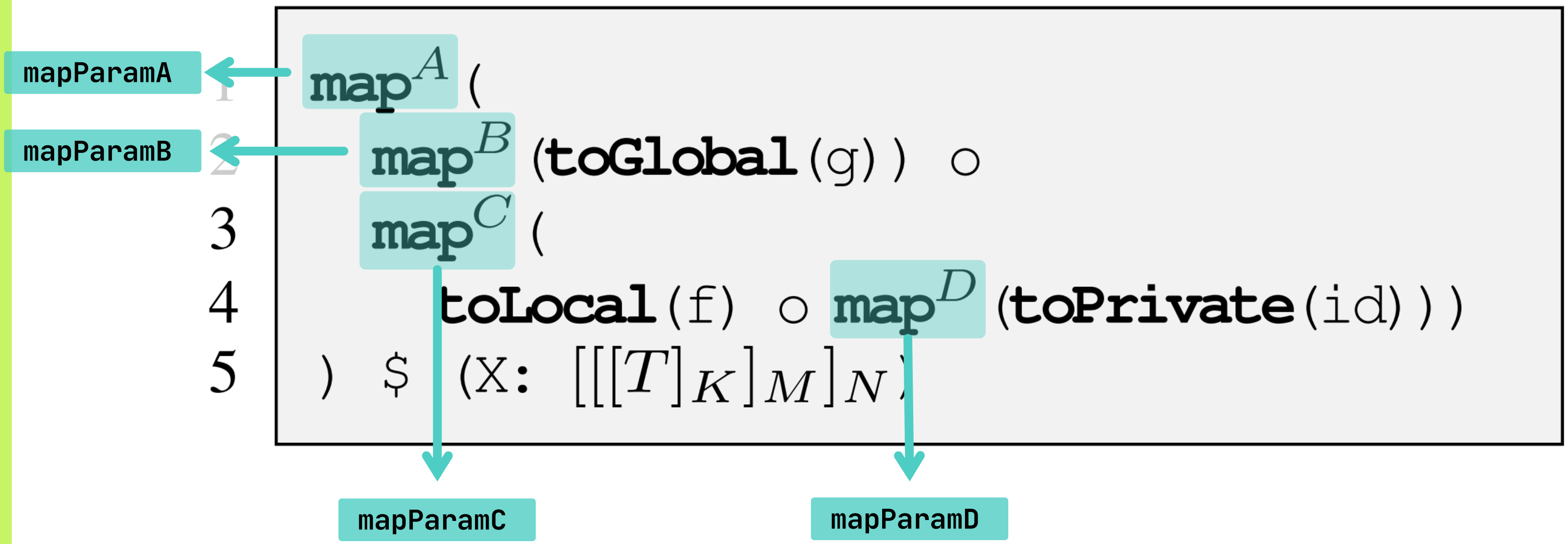
```
for i in group_id(0) to I do
  for j in local_id(0) to J do
    for k in 1 to K do
      shared_buf[j][k] = f( input[i][j][k] );

  for k in 1 to K do
    for j in local_id(0) to J do
      output[i][j][k] = g( shared_buf[j][k] );
```

Same indices

Might be optimal

Parallelization:
Scheduling Parameters



- Associate a parameter with each `map`
- Encode scheduling choices as integers
- Model parallelization restrictions as integer constraints

Parallelization:
Encoding of Choices

Code value	Map transformation
0	`mapSeq`
1	Fused with the outer `map`
10, 11, 12	`mapLcl` in dimension 0, 1 or 2 respectively
20, 21, 22	`mapWrg` in dimension 0, 1 or 2 respectively
30, 31, 32	`mapGlb` in dimension 0, 1 or 2 respectively

Constraint:
Private Memory Scope

IF: accesses private memory
THEN: cannot be parallel

```
map^A (
  map^B (toGlobal(g)) ∘
  map^C (
    toLocal(f) ∘ map^D (toPrivate(id)))
) $ (X: [[[T]_K]_M]_N)
```

Private memory access constraint. `maps` that consume or produce private memory cannot be parallelized because private memory is restricted in scope to a single thread.

Constraint:

Private Memory Scope

IF: accesses private memory
THEN: cannot be parallel

$$
\begin{array}{l}
\mathbf{map}^{A}\ (\\
\quad \mathbf{map}^{B}\ (\mathbf{toGlobal}(\mathtt{g}))\ \circ\\
\quad \mathbf{map}^{C}\ (\\
\quad\quad \mathbf{toLocal}(\mathtt{f})\ \circ\ \mathbf{map}^{D}\ (\mathbf{toPrivate}(\mathtt{id})))\\
)\ \$\ (\mathtt{X}\colon\ [[[T]_K]_M]_N)
\end{array}
$$

Private memory access constraint. **maps** that consume or produce private memory cannot be parallelized because private memory is restricted in scope to a single thread.

$\forall \mathtt{m} : \mathtt{m.usesPrivateMemory}$

GEN CONSTRAINT: $\mathtt{mapEncoding(m)}/10 < 1$

Code value	Map transformation
0	mapSeq
1	Fused with the outer map
10, 11, 12	mapLcl in dimension 0, 1 or 2
20, 21, 22	mapWrg in dimension 0, 1 or 2
30, 31, 32	mapGlb in dimension 0, 1 or 2

Constraint:
Private Memory Scope

$$
\begin{array}{l}
\mathbf{map}^{A}\ (\\
\quad \mathbf{map}^{B}\ (\mathbf{toGlobal}(\mathtt{g}))\ \circ\\
\quad \mathbf{map}^{C}\ (\\
\quad\quad \mathbf{toLocal}(\mathtt{f})\ \circ\ \mathbf{map}^{D}\ (\mathbf{toPrivate}(\mathtt{id})))\\
)\ \$\ (\mathtt{X}\!:\ [[[T]_K]_M]_N)
\end{array}
$$

$(\mathtt{mapEncoding}(\mathtt{mapParamD})/10 < 1)$

Constraint:
Shared Memory Scope

```
map^A (
  map^B (toGlobal(g)) ∘
  map^C (
    toLocal(f) ∘ map^D (toPrivate(id)))
) $ (X: [[[T]_K]_M]_N)
```

$(\texttt{mapEncoding(mapParamD)}/10 < 1)$

map^C is local or sequential $\wedge$ **map^A** is a mapWrg

Constraint:
Shared Memory Scope

$$
\begin{aligned}
&\mathtt{map}^{A}\,(\\
&\quad \mathtt{map}^{B}\,(\mathtt{toGlobal}(\mathtt{g}))\ \circ\\
&\quad \mathtt{map}^{C}\,(\\
&\qquad \mathtt{toLocal}(\mathtt{f})\ \circ\ \mathtt{map}^{D}\,(\mathtt{toPrivate}(\mathtt{id})))\\
&)\ \$\ (\mathtt{X}:\ [[[T]_K]_M]_N)
\end{aligned}
$$

$(\mathtt{mapEncoding}(\mathtt{mapParamD})/10 < 1)$

$(\mathtt{mapEncoding}(\mathtt{mapParamC})/10 < 2) \wedge \mathtt{mapEncoding}(\mathtt{mapParamA}) = 20 + \mathtt{w}$

Constraint:
Shared Memory Scope

```
map^A (
   map^B (toGlobal(g)) ∘
   map^C (
      toLocal(f) ∘ map^D (toPrivate(id)))
) $ (X: [[[T]_K]_M]_N)
```

$(\texttt{mapEncoding(mapParamD)}/10 < 1)$

$(\texttt{mapEncoding(mapParamC)}/10 < 2) \wedge \texttt{mapEncoding(mapParamA)} = 20 + \texttt{w}$

$(\texttt{mapEncoding(mapParamB)}/10 < 2) \wedge \texttt{mapEncoding(mapParamA)} = 20 + \texttt{w}$

Constraint:
Hierarchical Parallelism

$$
\begin{array}{l}
\boxed{\mathbf{map}^{A}}\ (\\
\quad \boxed{\mathbf{map}^{B}}\ (\mathbf{toGlobal}(\texttt{g}))\ \circ\\
\quad \mathbf{map}^{C}\ (\\
\qquad \mathbf{toLocal}(\texttt{f})\ \circ\ \mathbf{map}^{D}\ (\mathbf{toPrivate}(\texttt{id})))\\
)\ \$\ \ (\texttt{X:}\ [[[T]_K]_M]_N)
\end{array}
$$

$$(\texttt{mapEncoding(mapParamD)}/10 < 1)$$

$$(\texttt{mapEncoding(mapParamC)}/10 < 2)\ \wedge\ \texttt{mapEncoding(mapParamA)} = 20 + \texttt{w}$$

$$(\texttt{mapEncoding(mapParamB)}/10 < 2)\ \wedge\ \texttt{mapEncoding(mapParamA)} = 20 + \texttt{w}$$

map^{A} is not parallel $\vee$ **map^{A}** and **map^{B}** are not parallelized in the same way

Constraint:
Hierarchical Parallelism

```
map^A (
  map^B (toGlobal(g)) ∘
  map^C (
    toLocal(f) ∘ map^D (toPrivate(id)))
) $ (X: [[[T]_K]_M]_N)
```

$$(\texttt{mapEncoding(mapParamD)}/10 < 1)$$
$$(\texttt{mapEncoding(mapParamC)}/10 < 2) \wedge \texttt{mapEncoding(mapParamA)} = 20 + \texttt{w}$$
$$(\texttt{mapEncoding(mapParamB)}/10 < 2) \wedge \texttt{mapEncoding(mapParamA)} = 20 + \texttt{w}$$
$$(\texttt{mapEncoding(mapParamA)}/10 < 1) \vee (\texttt{mapEncoding(mapParamA)} \neq \texttt{mapEncoding(mapParamB)})$$
$$(\texttt{mapEncoding(mapParamA)}/10 < 1) \vee (\texttt{mapEncoding(mapParamA)} \neq \texttt{mapEncoding(mapParamC)})$$
$$\neg((\texttt{mapEncoding(mapParamA)}/10 = 1) \wedge (\texttt{mapEncoding(mapParamB)}/10 = 2) \wedge (\texttt{mapEncoding(mapParamB)}\%10 = \texttt{mapEncoding(mapParamA)}\%10))$$

+10 more hierarchical parallelism constraints

Constraint Satisfaction

```
map^A (
  map^B (toGlobal(g)) ∘
  map^C (
    toLocal(f) ∘ map^D (toPrivate(id)
) $ (X: [[[T]_K]_M]_N)
```

```
mapParamA = 20
mapParamB = 10
mapParamC = 10
mapParamD = 0
```

$(\underbrace{\texttt{mapEncoding(mapParamD)}}_{\mathbf{0}}/10 < 1)$

$(\underbrace{\texttt{mapEncoding(mapParamC)}}_{\mathbf{10}}/10 < 2) \wedge \underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}} = 20 + \mathtt{w}$

$(\underbrace{\texttt{mapEncoding(mapParamB)}}_{\mathbf{10}}/10 < 2) \wedge \underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}} = 20 + \mathtt{w}$

$(\underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}}/10 < 1) \vee (\underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}} \neq \underbrace{\texttt{mapEncoding(mapParamB)}}_{\mathbf{10}})$

$(\underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}}/10 < 1) \vee (\underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}} \neq \underbrace{\texttt{mapEncoding(mapParamC)}}_{\mathbf{10}})$

$\neg((\underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}}/10 = 1) \wedge (\underbrace{\texttt{mapEncoding(mapParamB)}}_{\mathbf{10}}/10 = 2) \wedge$
$(\underbrace{\texttt{mapEncoding(mapParamB)}}_{\mathbf{10}}\%10 = \underbrace{\texttt{mapEncoding(mapParamA)}}_{\mathbf{20}}\%10))$

+10 more hierarchical parallelism constraints

Constraint Satisfaction

```
mapWrg(0) (
  mapLcl(0) (toGlobal(g)) ∘
  mapLcl(0) (
    toLocal(f) ∘ mapSeq(toPrivate(i
) $ (X: [[[T]K]M]N)
```

```
mapParamA = 20
mapParamB = 10
mapParamC = 10
mapParamD = 0
```

(mapEncoding **0** (mapParamD) $/10 < 1)$

(mapEncoding **10** (mapParamC) $/10 < 2) \wedge$ mapEncoding **20** (mapParamA) $= 20 + \mathtt{w}$

(mapEncoding **10** (mapParamB) $/10 < 2) \wedge$ mapEncoding **20** (mapParamA) $= 20 + \mathtt{w}$

(mapEncoding **20** (mapParamA) $/10 < 1) \vee$ (mapEncoding **20** (mapParamA) $\neq$ mapEncoding **10** (mapParamB))

(mapEncoding **20** (mapParamA) $/10 < 1) \vee$ (mapEncoding **20** (mapParamA) $\neq$ mapEncoding **10** (mapParamC))

$\neg(($mapEncoding **20** (mapParamA) $/10 = 1) \wedge$ (mapEncoding **10** (mapParamB) $/10 = 2) \wedge$

(mapEncoding **10** (mapParamB) $\%10 =$ mapEncoding **20** (mapParamA) $\%10))$

+10 more hierarchical parallelism constraints

Heuristics

Sequential Map Fusion Heuristic

Perfectly nested sequential `maps` can always be fused to reduce search space

$$\forall \texttt{Chain} \in \texttt{MapNestingChains},\ \forall \texttt{m1} \in \texttt{Chain},\ \forall \texttt{m2} \in \texttt{Chain},$$

$$\texttt{m2.perfectlyNestedIn(m1)}$$

$$\textbf{GEN CONSTRAINT:}\ \neg((\texttt{mapEncoding(m1)} = 0) \wedge (\texttt{mapEncoding(m2)} = 0))$$

Code value	Map transformation
0	`mapSeq`
1	Fused with the outer `map`
10, 11, 12	`mapLcl` in dimension 0, 1 or 2 respectively
20, 21, 22	`mapWrg` in dimension 0, 1 or 2 respectively
30, 31, 32	`mapGlb` in dimension 0, 1 or 2 respectively

```
      )) $ p2096
    )) $ p27462
  )) o Map((p62981 ->
    TransposeW() o Map((p5669 ->
      TransposeW() $ p5669
    )) $ p62981
  )) o Map((p38465 ->
    Map((p4026 ->
      Map((p31009 ->
        (p17262 ->
          (p13965 ->
            Join() o Map((p57787 ->
              Split(v_tileHeight_13) o Split(v_tileWidth_12) o Map((p17309 ->
                Map((p43233 ->
                  toGlobal((p36987 ->
                    id $ p36987
                  )) $ p43233
                )) o ReduceSeq((p52739, p34159 ->
                  add $ (p52739, p34159)
                )) $ (toPrivate((p49062 ->
                  id $ p49062
                )) $ 0.0f, p17309)
              )) o Join() o Join() $ p57787
            )) o Map((p42460 ->
              Map((p49891 ->
                Map((p19231 ->
                  Transpose() $ p19231
                )) $ p49891
              )) $ p42460
            )) o Map((p9193 ->
              Map((p3082 ->
                Transpose() $ p3082
              )) $ p9193
            )) o Map((p41734 ->
              Transpose() $ p41734
            )) o Transpose() o Map((p53541 ->
              Transpose() $ p53541
            )) o Map((p32600 ->
              Map((p58028 ->
                Transpose() $ p58028
```

```
      )) $ p2096
    )) $ p27462
  )) o Map((p62981 ->
    TransposeW() o Map((p5669 ->
      TransposeW() $ p5669
    )) $ p62981
  )) o Map((p38465 ->
    Map((p4026 ->
      Map((p31009 ->
        (p17262 ->
          (p13965 ->
            Join() o Map((p57787 ->
              Split(v_tileHeight_13) o Split(v_tileWidth_12) o Map((p17309 ->
                Map((p43233 ->
                  toGlobal((p36987 ->
                    id $ p36987
                  )) $ p43233
                )) o ReduceSeq((p52739, p34159 ->
                  add $ (p52739, p34159)
                )) $ (toPrivate((p49062 ->
                  id $ p49062
                )) $ 0.0f, p17309)
              )) o Join() o Join() $ p57787
            )) o Map((p42460 ->
              Map((p49891 ->
                Map((p19231 ->
                  Transpose() $ p19231
                )) $ p49891
              )) $ p42460
            )) o Map((p9193 ->
              Map((p3082 ->
                Transpose() $ p3082
              )) $ p9193
            )) o Map((p41734 ->
              Transpose() $ p41734
            )) o Transpose() o Map((p53541 ->
              Transpose() $ p53541
            )) o Map((p32600 ->
              Map((p58028 ->
                Transpose() $ p58028
```

mapParam0
mapParam1
mapParam2
mapParam3
mapParam4
mapParam5
mapParam6

• • •

50

```
      )) $ p2096
    )) $ p27462
  )) o Map((p62981 ->
    TransposeW() o Map((p5669 ->
      TransposeW() $ p5669
    )) $ p62981
  )) o MapWrg(2,(p38465 ->
    MapWrg(0,(p4026 ->
      MapSeq((p31009 ->
        (p17262 ->
          (p13965 ->
            Join() o MapLcl(2,(p57787 ->
              Split(v_tileHeight_13) o Split(v_tileWidth_12) o MapLcl(0,(p17309
                MapSeq((p43233 ->
                  toGlobal((p36987 ->
                    id $ p36987
                  )) $ p43233
                )) o ReduceSeq((p52739, p34159 ->
                  add $ (p52739, p34159)
                )) $ (toPrivate((p49062 ->
                  id $ p49062
                )) $ 0.0f, p17309)
              )) o Join() o Join() $ p57787
            )) o Map((p42460 ->
              Map((p49891 ->
                Map((p19231 ->
                  Transpose() $ p19231
                )) $ p49891
              )) $ p42460
            )) o Map((p9193 ->
              Map((p3082 ->
                Transpose() $ p3082
              )) $ p9193
            )) o Map((p41734 ->
              Transpose() $ p41734
            )) o Transpose() o Map((p53541 ->
              Transpose() $ p53541
            )) o Map((p32600 ->
              Map((p58028 ->
                Transpose() $ p58028
```


51

Guided Rewriting

```
Map((p23724 ->
  TransposeW() $ p23724
)) o TransposeW() o Map((p53064 ->
  Map((p59387 ->
    Join() $ p59387
  )) $ p53064
)) o Join() o Map((p22048 ->
  Map((p3291 ->
    Map((p36687 ->
      Join() $ p36687
    )) o Join() o Map((p38111 ->
      TransposeW() $ p38111
    )) $ p3291
  )) $ p22048
)) o Map((p14415 ->
  TransposeW() o Map((p3478 ->
    TransposeW() $ p3478
  )) $ p14415
)) o Map((p8637 ->
  Map((p56646 ->
    Map((p60274 ->
      (p44931 ->
        (p16296 ->
          Join() o Map((p61748 ->
            Map((p3056 ->
              Map((p10788 ->
                Join() $ p10788
              )) o Join() o Map((p2536 ->
                TransposeW() $ p2536
              )) $ p3056
            )) $ p61748
          )) o Map((p44436 ->
            TransposeW() o Map((p27595 ->
              TransposeW() $ p27595
            )) $ p44436
          )) o Map((p59960 ->
            Map((p8566 ->
              Map((p32269 ->
                (p63116 ->
                  (p62040 ->
                    (p26213 ->
                      (p23403 ->
                        Join() o TransposeW() o Map((p24164 ->
                          TransposeW() $ p24164
                        )) o Map((p11838 ->
                          Map((p62211 ->
                            TransposeW() $ p62211
                          )) $ p11838
                        )) o Map((p38162 ->
                          Map((p37670 ->
                            Map((p34694 ->
                              Map((p11272 ->
                                Map((p63724 ->
                                  toGlobal((p53305 ->
                              )) $ p11272
                            )) o ReduceSeq((p50867, p35848 ->
                              Map((p13698 ->
                                add $ (Get(0) $ p13698, Get(1) $ p13698)
                              )) o Zip() $ (p50867, p35848)
                            )) $ (Map((p29056 ->
                              toPrivate((p64395 ->
                                id $ p64395
                              )) $ p29056
                            )) $ 0.0f, p34694)
                          )) $ p37670
                        )) $ p38162
                      )) o Map((p11356 ->
                        Map((p61280 ->
                          Transpose() $ p61280
                        )) $ p11356
                      )) o Map((p27341 ->
                        Transpose() $ p27341
                      )) o Transpose() o Map((p3491 ->
                        p3491
                      )) o Join() o Map((p23870 ->
                        Map((p12189 ->
                          (p18636 ->
                            Join() o Map((p18649 ->
                              Map((p26353 ->
                                Map((p8803 ->
                                  Map((p19946 ->
                                    Map((p18234 ->
                                      toLocal((p55370 ->
                                        id $ p55370
                                      )) $ p18234
                                    )) $ p19946
                                  )) $ p8803
                                )) $ p26353
                              )) $ p18649
                            )) o ReduceSeq((p38106, p46524 ->
                              Map((p25181 ->
                                Map((p31204 ->
                                  Map((p3344 ->
                                    Map((p40326 ->
                                      add $ (Get(0) $ p40326, Get(1) $ p40326)
                                    )) $ p3344
                                  )) $ p31204
                                )) $ p25181
                              )) o Split(v_seqWindowsPerThreadY_21) o
                              Split(v_seqWindowsPerThreadX_20) o Split(1) o
                              Zip() $ (Join() o Join() o Join() $ p38106,
                              Join() o Join() o Join() o Join() o Map((p54992 ->
                                Map((p7921 ->
                                  Map((p750 ->
                                    Join() $ p750
                                  )) o Join() o Map((p23842 ->
                                    TransposeW() $ p23842
                                  )) $ p7921
                                )) $ p54992
                              )) o Map((p20319 ->
                                TransposeW() o Map((p9345 ->
                        )) o Map((p28717 ->
                          Map((p2203 ->
                            Map((p257 ->
                              (p38712 ->
                                (p53378 ->
                                  Map((p52451 ->
                                    Map((p26784 ->
                                      Map((p58124 ->
                                        ReduceSeq((p59290, p5770 ->
                                          add $ (p59290, mult $ (
                                            Get(0) $ p5770, Get(1) $ p5770))
                                        )) $ (toPrivate((p9979 ->
                                          id $ p9979
                                        )) $ 0.0f, Zip() $ (Join() o Join() $
                                        p58124, Join() o Join() $ p52451))
                                      )) $ p26784
                                    )) $ p53378
                                  )) $ p38712
                                ) o Map((p7268 ->
                                  Map((p15357 ->
                                    Transpose() o Map((p5313 ->
                                      Transpose() $ p5313
                                    )) $ p15357
                                  )) $ p7268
                                )) o Map((p16531 ->
                                  Transpose() o Map((p10152 ->
                                    Transpose() $ p10152
                                  )) $ p16531
                                )) o Map((p42971 ->
                                  Map((p48512 ->
                                    Map((p26122 ->
                                      Map((p52770 ->
                                        Map((p42776 ->
                                          toPrivate((p60560 ->
                                            id $ p60560
                                          )) $ p42776
                                        )) $ p52770
                                      )) $ p26122
                                    )) $ p48512
                                  )) $ p42971
                                )) o Map((p54387 ->
                                  Map((p36304 ->
                                    Transpose() o Map((p30379 ->
                                      Transpose() $ p30379
                                    )) $ p36304
                                  )) $ p54387
                                )) o Map((p37721 ->
                                  Transpose() o Map((p36420 ->
                                    Transpose() $ p36420
                                  )) $ p37721
                                )) $ p257
                              ) o Map((p57134 ->
                                Transpose() o Map((p10654 ->
                                  Transpose() $ p10654
                                )) $ p57134
                              )) o Map((p58070 ->
```

Rewrite points expanded

x3

Guided Rewriting: Tiling

- When two nested ND-maps iterate over different buffers, tiles can be prefetched and reused

```
mapXD(a =>
  mapYD(b =>
    f(a, b),
  B),
A)
```

⟺

```
mapXD(aTile =>
  mapYD(bTile =>

    let(aTileCached =>
      let(bTileCached =>

        f(aTileCached, bTileCached),

        mapYD(
          toPrivate(id(memIdB))), bTile),
      mapXD(
        toPrivate(id(memIdA))), aTile),

    transpose(..,
      splitYD(bTileSizes, B))),

  transpose(..,
    splitXD(aTileSizes, A)))
```

Reusage

Prefetching

Tiling

Memory access pattern optimization

Guided Rewriting: Tiling

- When two nested ND-maps iterate over different buffers, tiles can be prefetched and reused

```
mapXD(a =>
  mapYD(b =>
    f(a, b),
  B),
A)
```

⟺

```
mapXD(aTile =>
  mapYD(bTile =>

    let(aTileCached =>
      let(bTileCached =>

        f(aTileCached, bTileCached),

        mapYD(
          toPrivate(id(memIdB))), bTile),
        mapXD(
          toPrivate(id(memIdA))), aTile),

    transpose(..,
      splitYD(bTileSizes, B))),

  transpose(..,
    splitXD(aTileSizes, A)))
```

Rewrite params: whether to prefetch and in which memory

Tuning params: ND tile sizes

Rewrite params: whether to transform tile data layout for coalescing