THE UNIVERSITY of EDINBURGH
informatics

EPSRC Centre for Doctoral Training in
Pervasive Parallelism

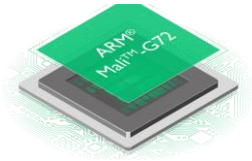
Guided Rewriting and Constraint Satisfaction for Parallel GPU Code Generation

Naums Mogers

May 9th, 2023

Parallel Architectures

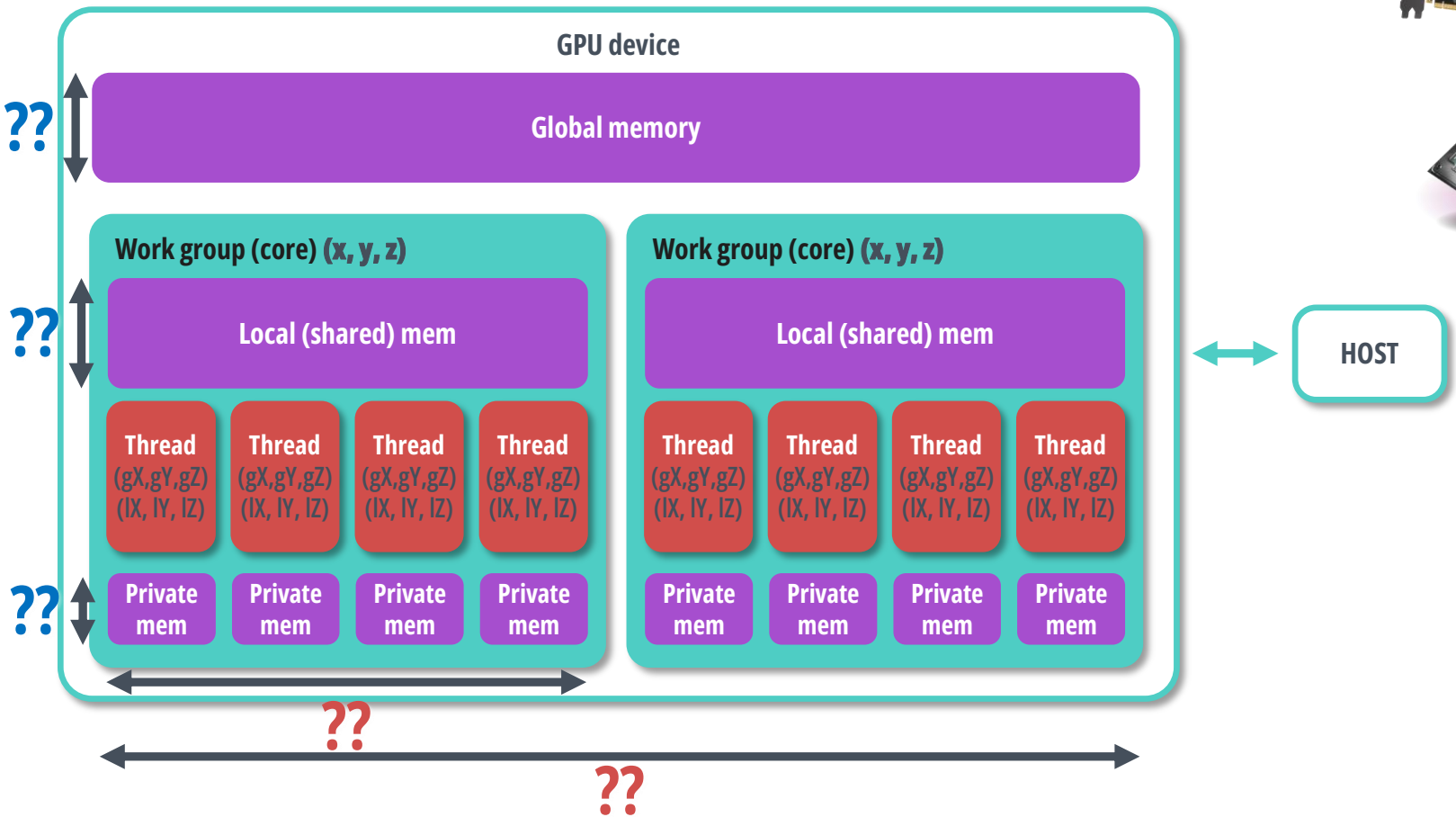
©ARM



©NVIDIA

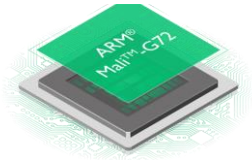
©Xilinx

Parallel architectures are hard to optimize for



Parallel Architectures

©ARM



©NVIDIA

©Xilinx

Parallel architectures are hard to optimize for

Large implementation space

Automatic code generators and optimizers are needed

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

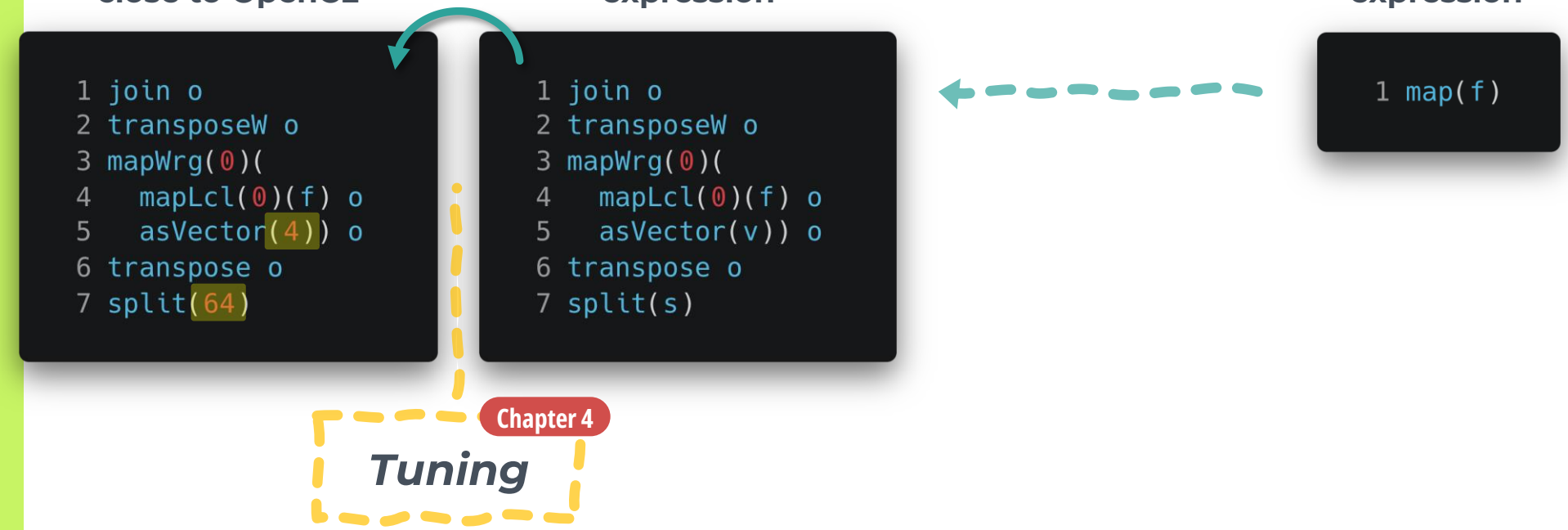
- Kernel libraries
  - Costly to maintain and extend
- User-provided schedules (*Halide*) and design choices (*PetaBricks*, *Tangram*)
  - Burden on the user
- Polyhedral compilation (*Tensor Comprehensions*)
  - Limited to affine loops
- Functional rewriting (*Futhark*, *Accelerate*, *Lift*, *RISE*)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

Low-level expression close to OpenCL

```
1 join o
2 transposeW o
3 mapWrg(0)(
4   mapLcl(0)(f) o
5   asVector(4)) o
6 transpose o
7 split(64)
```

High-level expression

```
1 map(f)
```


Low-level expression close to OpenCL
Parallelised expression
High-level expression
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(4)) o
6 transpose o
7 split(64)
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(v)) o
6 transpose o
7 split(s)
1 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])
                                   : [[float_v]_ω]_windowSize/ω = {
    val flatWindow1D = join(join(window))
    val flatCoalescedWindow1D = reorder(striddenIndex(windowSize/ω), flatWindow1D)
    val flatCoalescedChunkedWindow1D = split(ω, flatCoalescedWindow1D)
    asVector(v, flatCoalescedChunkedWindow1D) }
  val tiledSlidedCoalescedChunkedVectorizedInput4D = map(tile4D -> split(σ, map
    coalesceChunkVectorizeWindow(window3D), tile4D)), tiledSlidedInput5D)
  val groupedCoalescedChunkedVectorizedKernelsWeights4D = split(κ, map(singleKernel
    coalesceChunkVectorizeWindow(singleKernelWeights), kernelsWeights))
  mapWrg(1, inputTile3D ->
    mapWrg(0, kernelsGroupWeights3D -> transpose(
      mapLcl(1, inputWindows2D -> transpose(
        mapLcl(0, (inputWindowsChunk1D, kernelsGroupChunk2D) ->
          mapSeq(singleKernelReducedChunk -> toGlobal(singleKernelReduced
            join(
              reduceSeq(
                init = mapSeq(toPrivate(id(Value(0, [float]_κ)))),
                f = (acc, (inputsValue, kernelsGroupValue1D)) ->
                  let(inputsValuePrivate ->
                    mapSeq((accValue, singleKernelValue) ->
                      mapSeq((inputValuePrivate) ->
                        accValue + vectorize(v, dot(inputValuePrivate, singleKernelValue)),
                      inputsValuePrivate,
                      zip(acc, kernelsGroupValue1D),
                    mapSeq(toPrivate(vectorize(v, 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

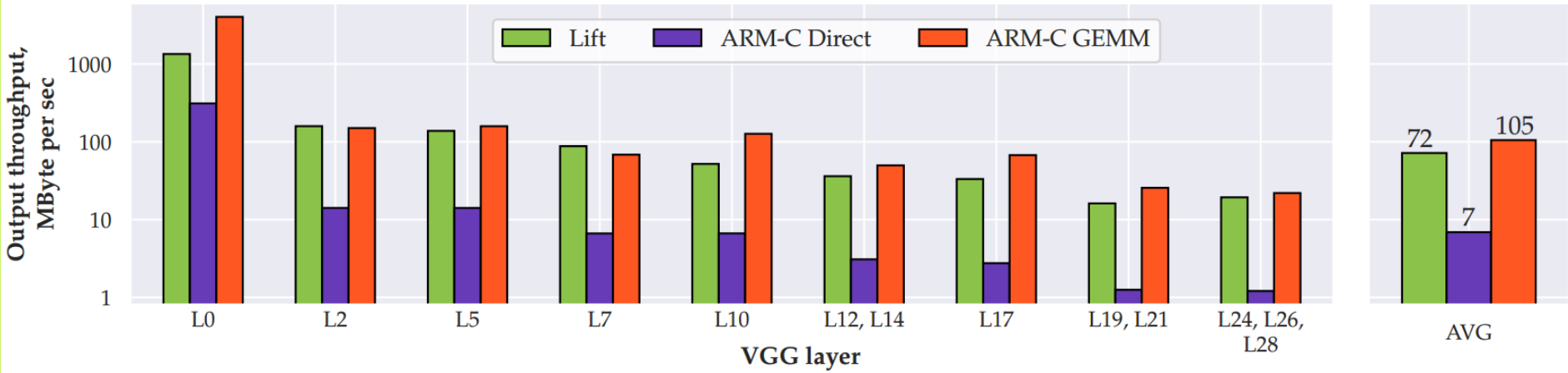
Chapter 4:
Functional IR for Auto-Tuning

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

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

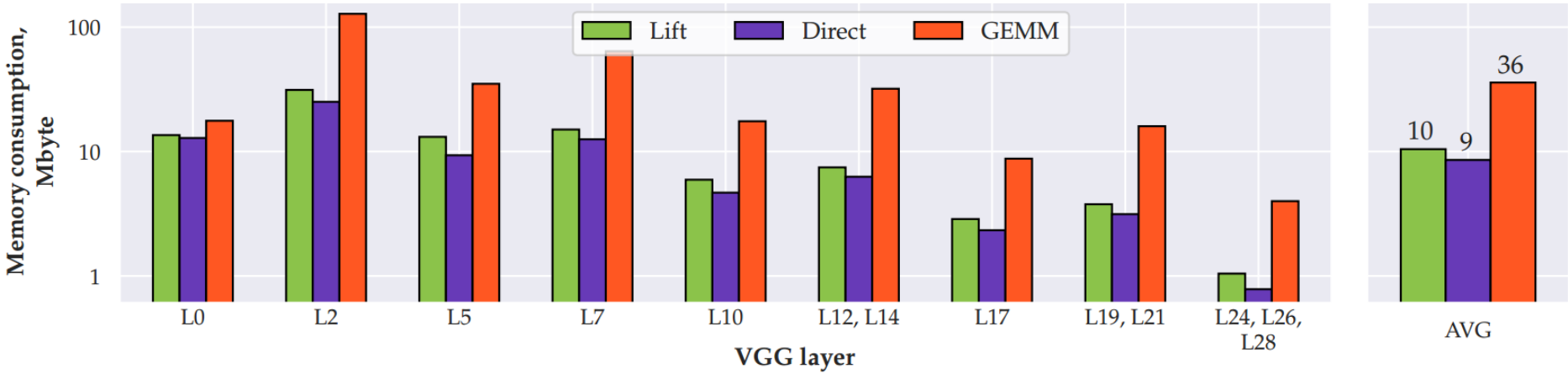
$\text{Split}(s)\ \$\ [T]_N$	=>	$N \% s == 0$
$\text{asVector}(v)\ \$\ [T]_N$	=>	$N \% v == 0$
$\text{Slide}(len, step)\ \circ\ [T]_N$	=>	$N >= len$
$\text{SlideStrict}(len, step)\ \circ\ [T]_N$	=>	$N >= len\ \&\&$ $N \% ((N - (len - step)) / step) == 0$

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)**

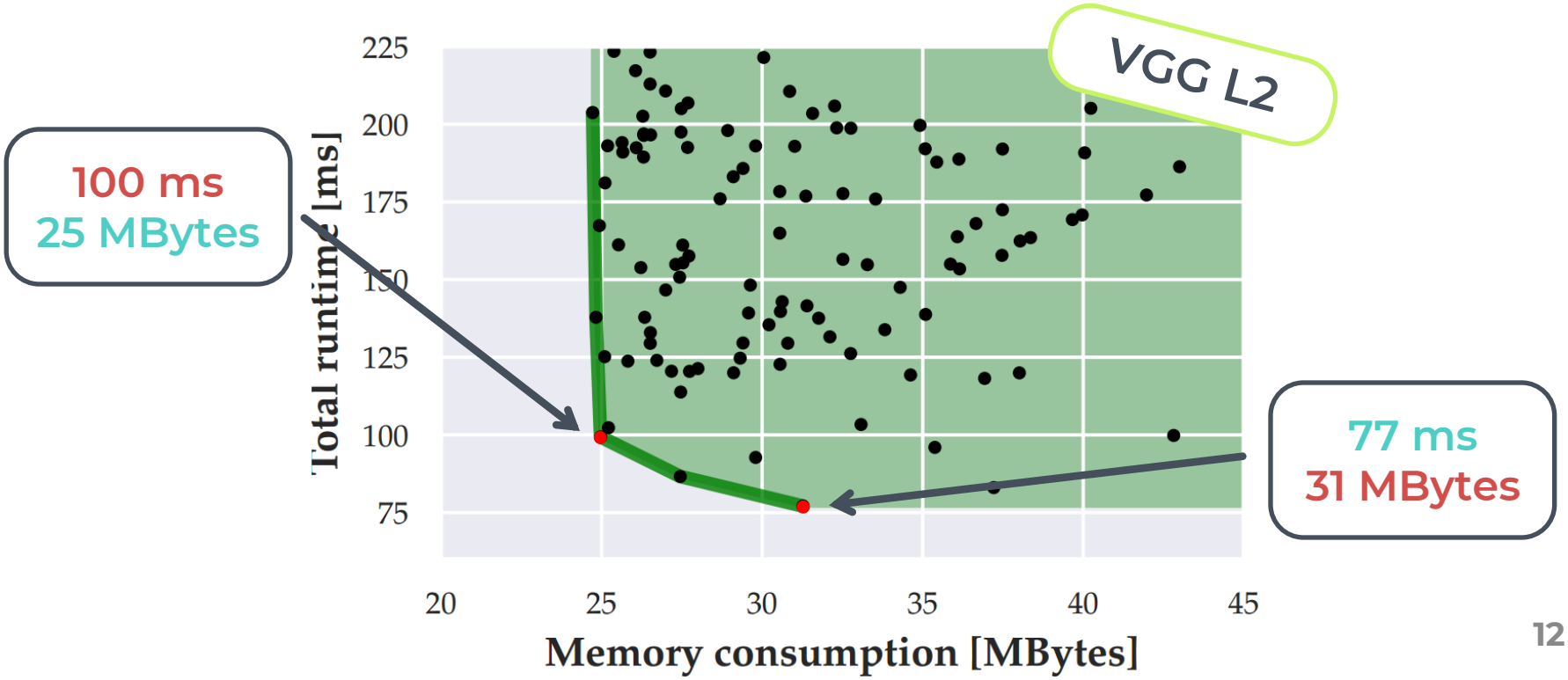
Results: Memory Consumption

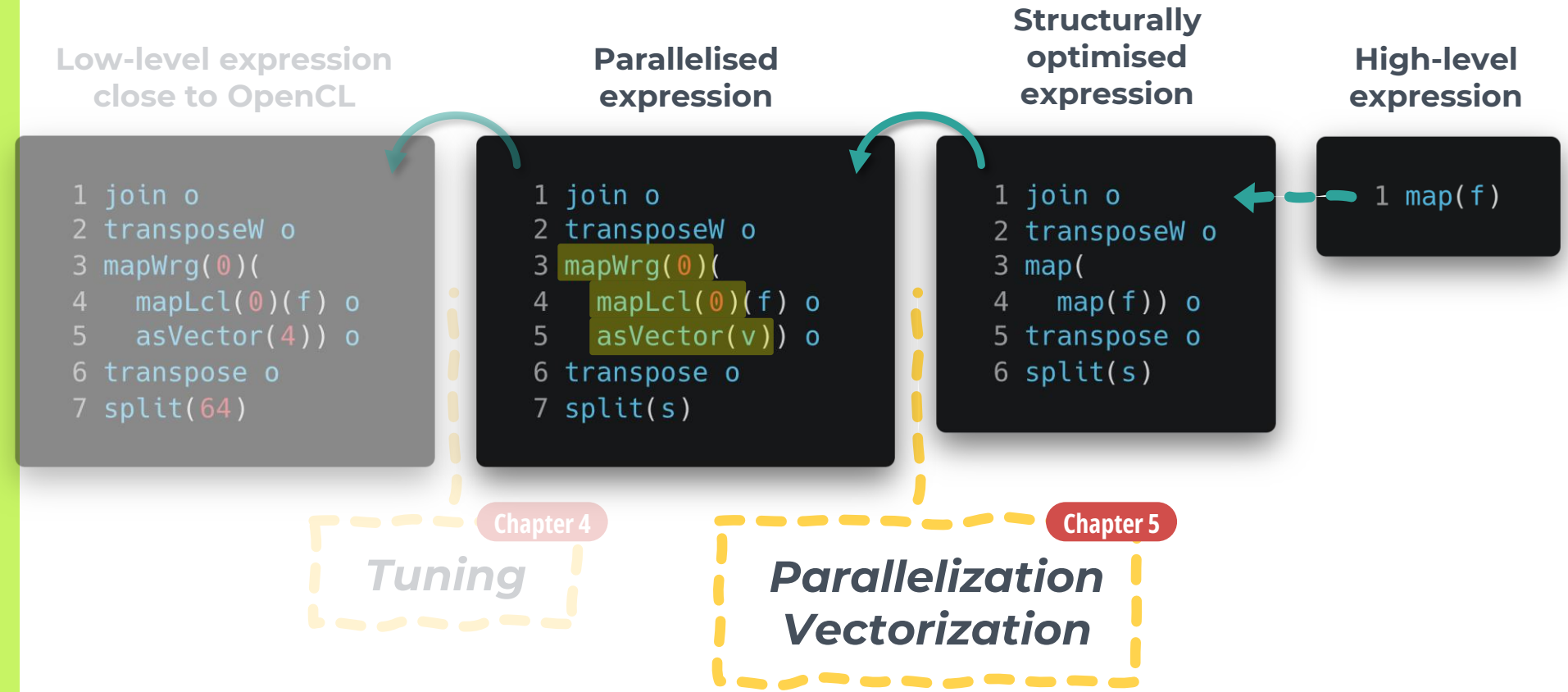


- Memory consumption of Lift kernels is:
  - On par with direct convolution in ARM-C **(x1.1 on average)**
  - Always better than GEMM in ARM-C **(x3.6 on average)**

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
Parallelised expression
Structurally optimised expression
High-level expression
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(4)) o
6 transpose o
7 split(64)
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(v)) o
6 transpose o
7 split(s)
1 join o
2 transposeW o
3 map(
4 map(f)) o
5 transpose o
6 split(s)
1 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).

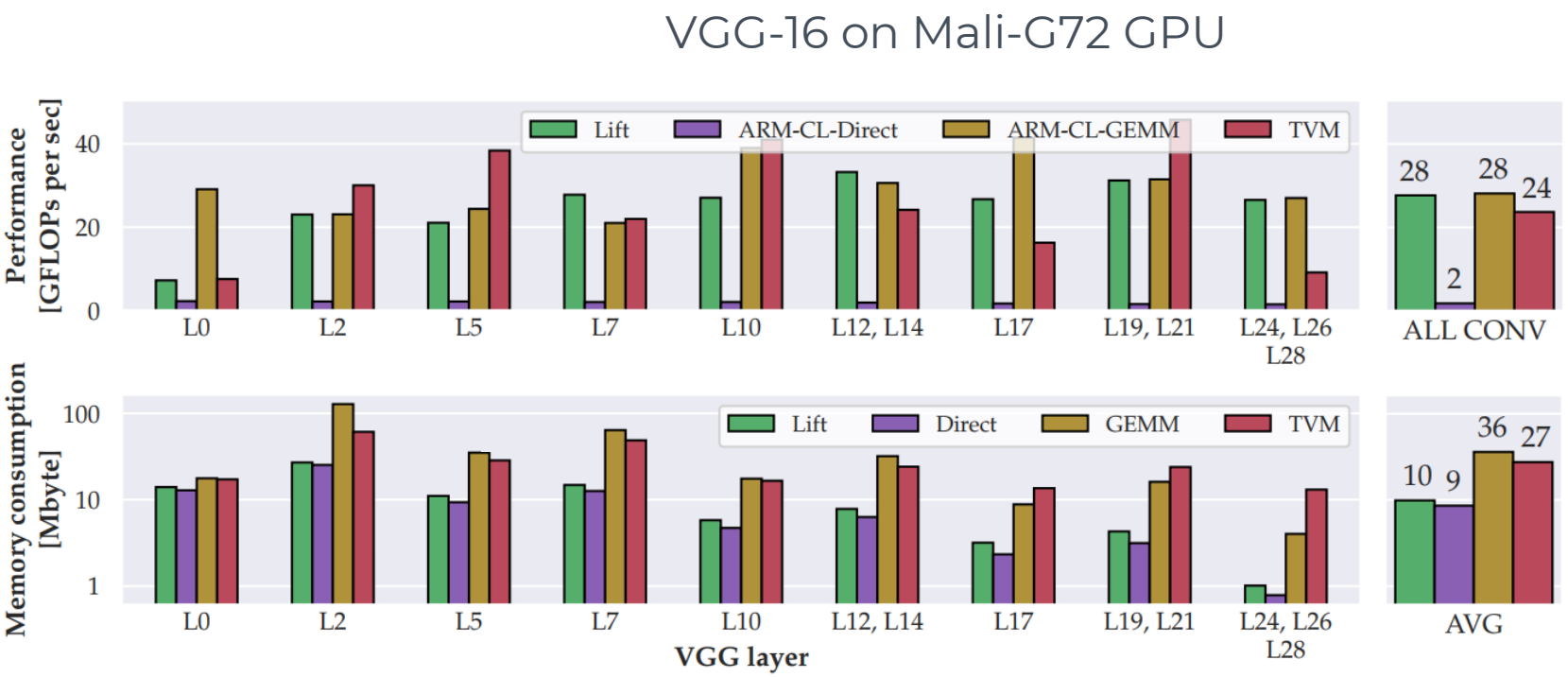
CC'22 reviewer

Chapter 5:
Parallelism Mapping Through Constraint Satisfaction

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

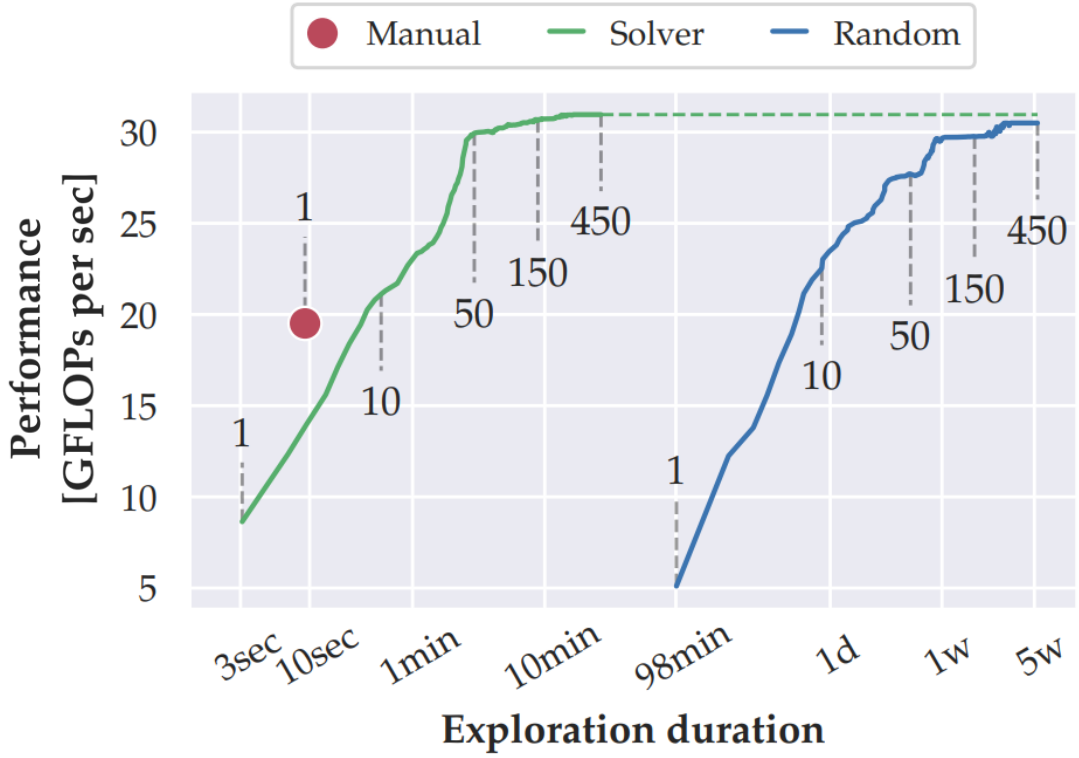
- Chapter 5:
  - Expresses the GPU parallel programming model as arithmetic constraints on functional patterns
  - Uses a constraint solver to explore parallel mappings
  - Describes a functional IR-based barrier insertion method

Results: Performance & Memory

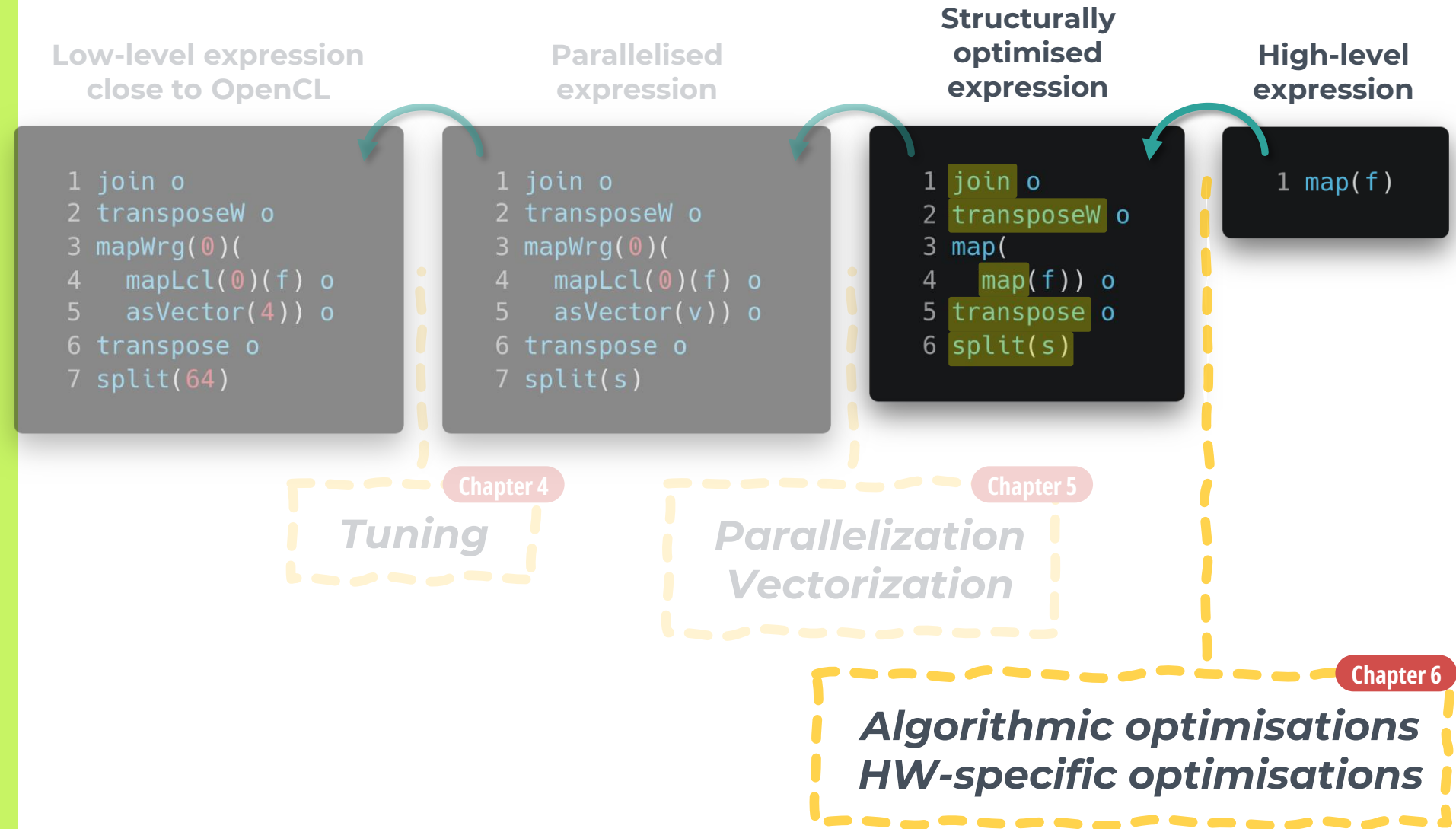


- Stencil performance **on par** with ARM-CL GEMM and **0.86x** of TVM's
- **3.6x** less memory on average than ARM-CL GEMM
- **2.7x** less memory on average than TVM

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
Parallelised expression
Structurally optimised expression
High-level expression
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(4)) o
6 transpose o
7 split(64)
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(v)) o
6 transpose o
7 split(s)
1 join o
2 transposeW o
3 map(
4 map(f)) o
5 transpose o
6 split(s)
1 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

```
def conv( inputData      :  [[[float]_inCh]_inW]_inH,
          kernelsWeights :  [[[[float]_inCh]_kerW]_kerH]_outCh,
          padSize        :  (int,int,int,int),
          kernelStride   :  (int,int) ) :  [[[float]_outCh]_outW]_outH =
toHost o
padExtra2D( kernelFission(
  joinSplitND( padExtra1D( kernelFission( slideWindows' ⇒
    kernelFission( kernelsWeights' ⇒
      oclKernel((slideWindows'', kernelsWeights'') ⇒
        abTile( abTile(
          tileNestedMapReduce(reduceFun = +)(
            tileNestedMapReduce(reduceFun = +)(
              abTile(
                mapND_2(slideWin: [T]_(inChs * kerW * kerH) ⇒
                  map(singleK: [T]_(inChs * kerW * kerH) ⇒
                    privatiseAccumulator(
                      reduce(0, +) o map(*) << zip(slideWin, singleK) )
                  ) << kernelsWeights''
                ) << slideWindows'' )))))
        ) << (slideWindows', kernelsWeights')
    ) o map(joinND_2) o toGPU << kernelsWeights
  ))) o mapND_2(joinND_2) o
      slideND_2(kerH, kerW, kernelStride._1, kernelStride._2)
)) o padND_2(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	
		slideND$_2$		
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

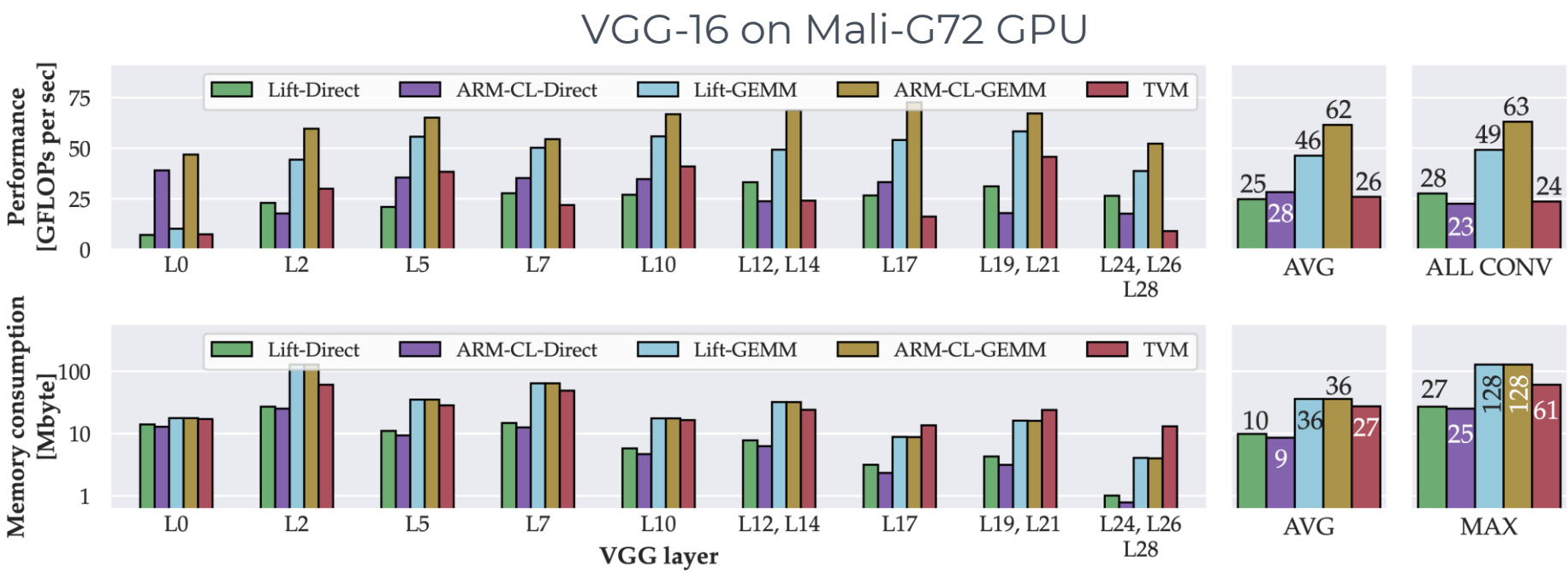
#	Rewrite Point	Struct and tune parameters	Result	Line
		slideND$_2$		
1	padExtra1D	**p**=6	Corner padding, depadding	**7**
2	kernelFission		Input preprocessing OpenCL kernel	**7**
3	↳ optimiseDataLayout	**newDimOrder**=[0,3,1,2,4], **factors**=[2,5,1152,4]	Reshaped input for coalescing	
4	↳ materialise	**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

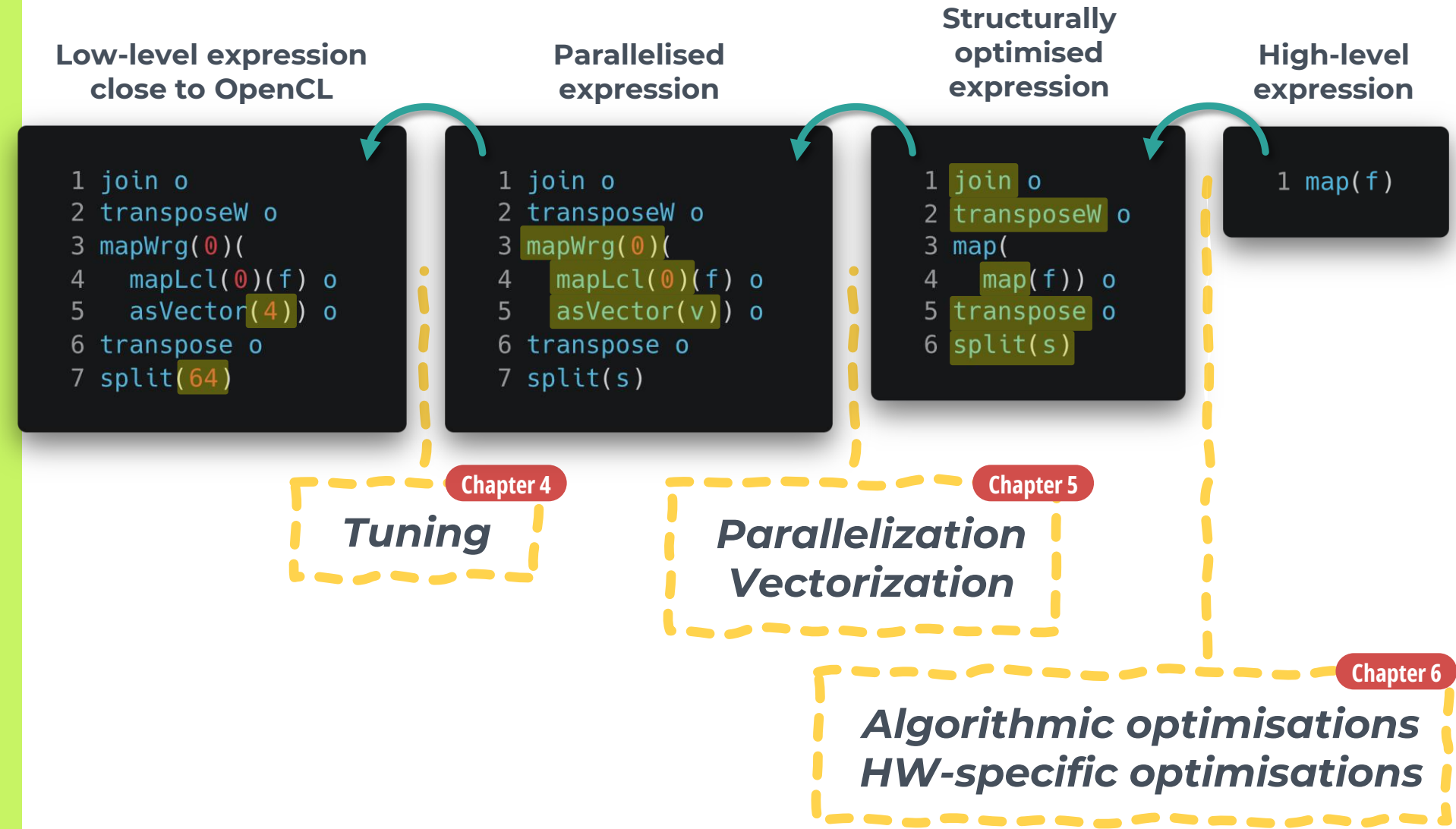
#	Rewrite Point	Struct and tune parameters	Result	Line
		slideND$_2$		
2	kernelFission		Input preprocessing OpenCL kernel	**7**
4	↳ materialise	**targetMem**=global	Input reshaping materialised, im2col	
5	↳ splitJoinND	**tileSizes**=[9,32,4]	Parallelisable and vectorisable loops	
6	kernelFission		Weight 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



- 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
Parallelised expression
Structurally optimised expression
High-level expression
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(4)) o
6 transpose o
7 split(64)
1 join o
2 transposeW o
3 mapWrg(0)(
4 mapLcl(0)(f) o
5 asVector(v)) o
6 transpose o
7 split(s)
1 join o
2 transposeW o
3 map(
4 map(f)) o
5 transpose o
6 split(s)
1 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

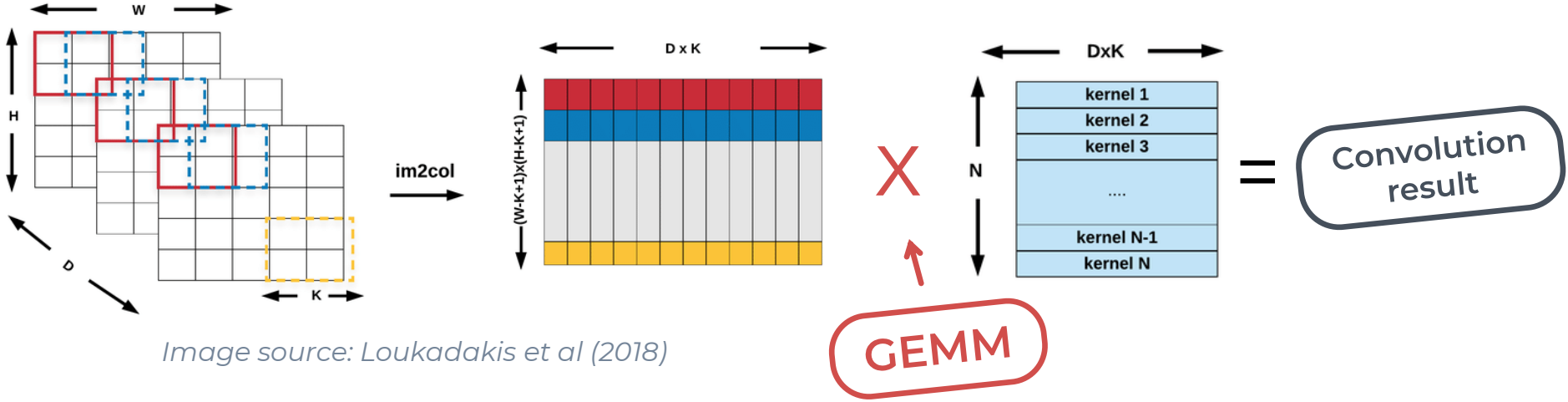
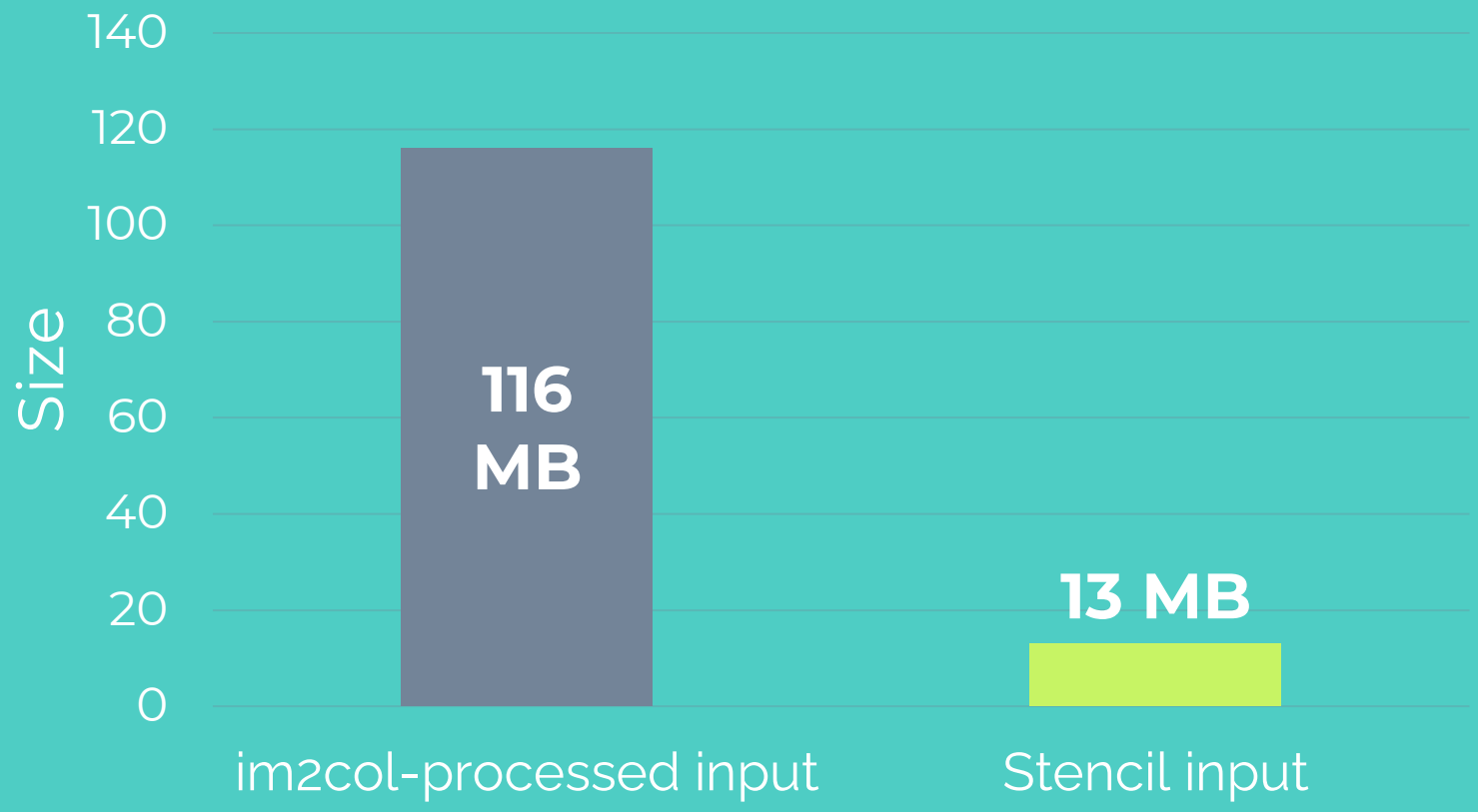


Image source: Loukadakis et al (2018)

- 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

```
1  for i in 1 to I do
2    for j in 1 to J do
3      for k in 1 to K do
4        shared_buf[j][k] = f( input[i][j][k] );
5
6    for k in 1 to K do
7      for j in 1 to J do
8        output[i][j][k] = g( shared_buf[j][k] );
```

Transposed!

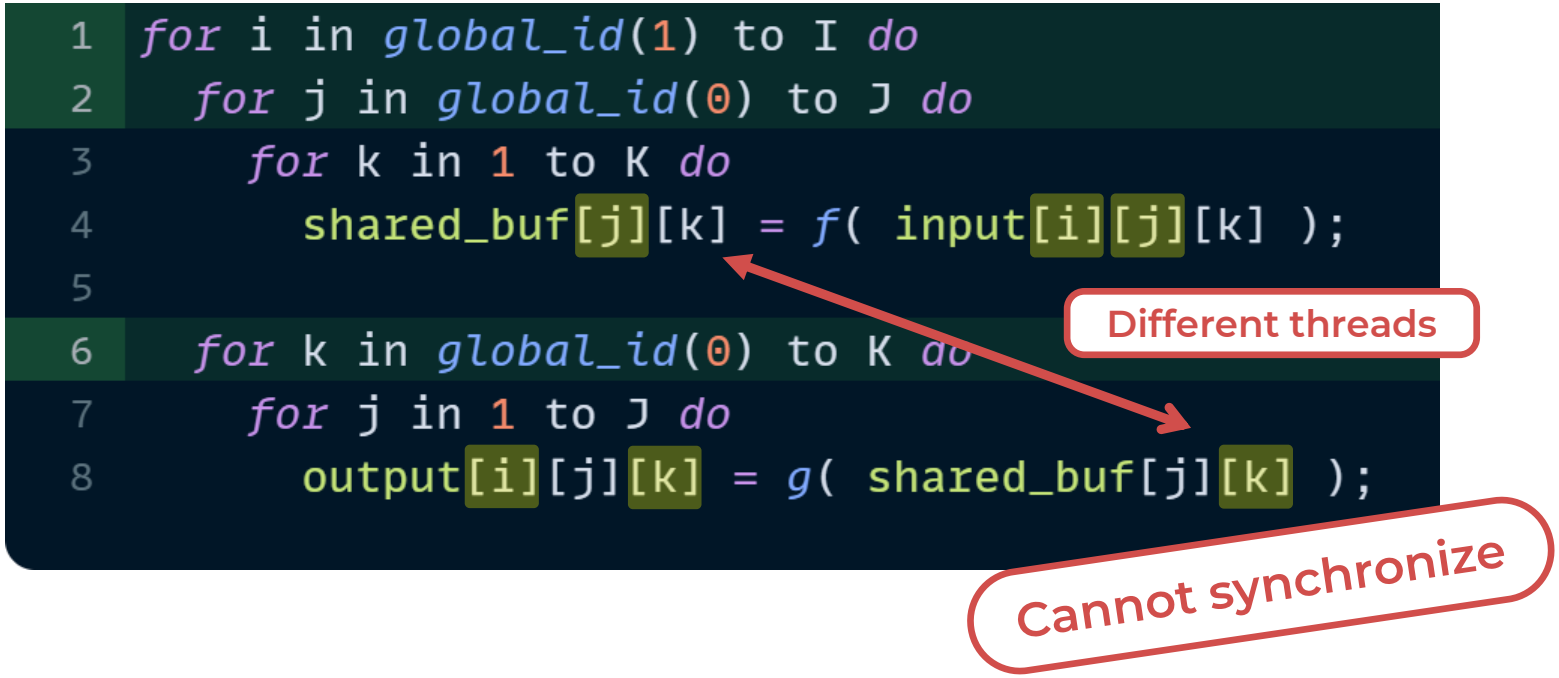
Parallelization Is Hard

```
1  for i in global_id(0) to I do
2    for j in 1 to J do
3      for k in 1 to K do
4        shared_buf[j][k] = f( input[i][j][k] );
5
6    for k in 1 to K do
7      for j in 1 to J do
8        output[i][j][k] = g( shared_buf[j][k] );
```

Under-saturated cores

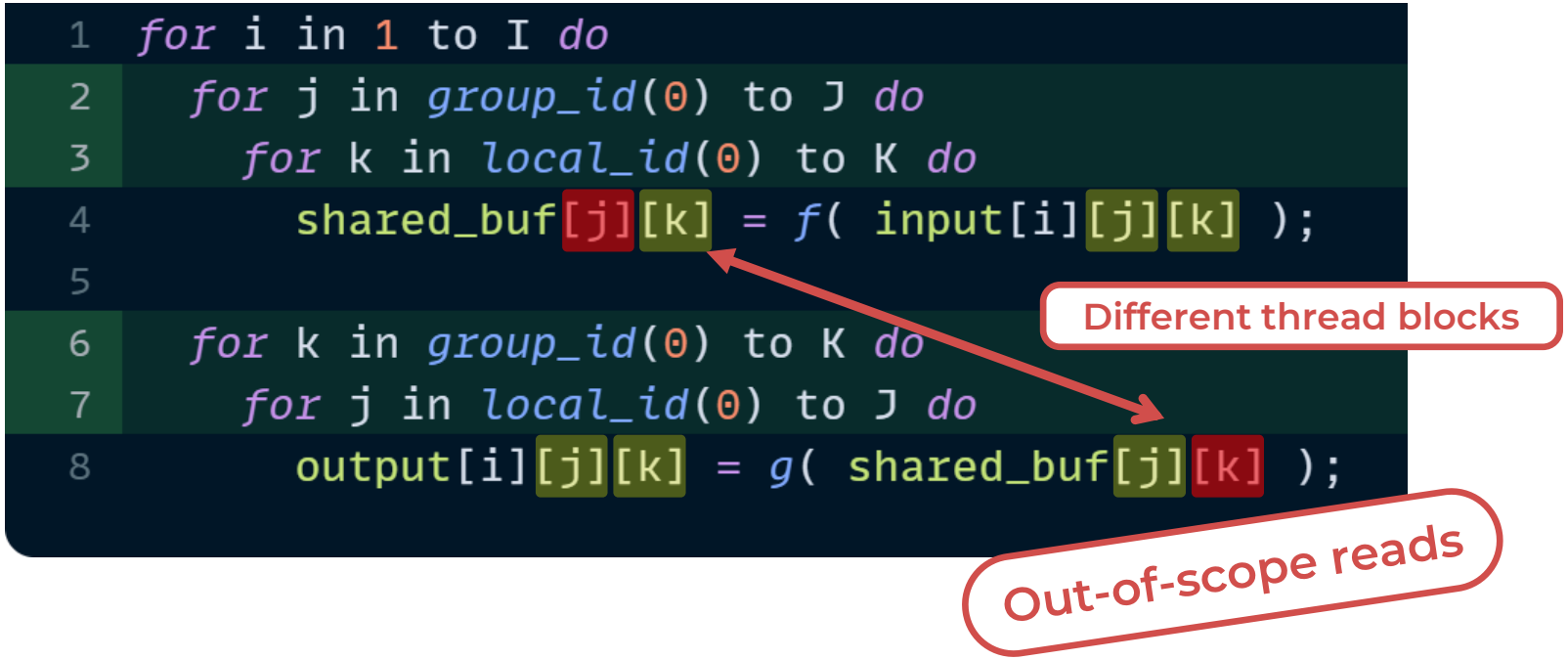
Parallelization Is Hard

```
1  for i in global_id(1) to I do
2    for j in global_id(0) to J do
3      for k in 1 to K do
4        shared_buf[j][k] = f( input[i][j][k] );
5
6    for k in global_id(0) to K do
7      for j in 1 to J do
8        output[i][j][k] = g( shared_buf[j][k] );
```



Parallelization Is Hard

```
1  for i in 1 to I do
2    for j in group_id(0) to J do
3      for k in local_id(0) to K do
4        shared_buf[j][k] = f( input[i][j][k] );
5
6    for k in group_id(0) to K do
7      for j in local_id(0) to J do
8        output[i][j][k] = g( shared_buf[j][k] );
```



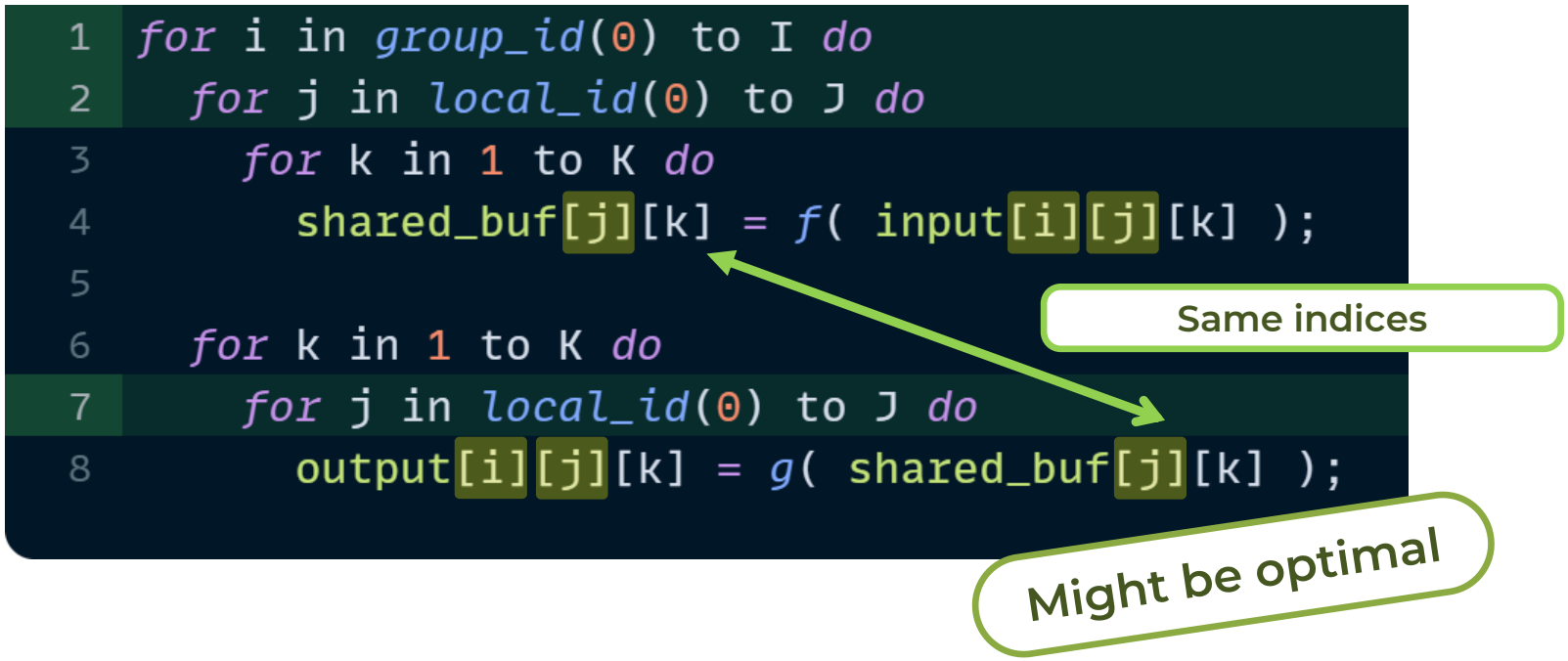
Parallelization Is Hard

```
1  for i in group_id(0) to I do
2    for j in local_id(0) to J do
3      for k in 1 to K do
4        shared_buf[j][k] = f( input[i][j][k] );
5    barrier();
6    for k in local_id(0) to K do
7      for j in 1 to J do
8        output[i][j][k] = g( shared_buf[j][k] );
```

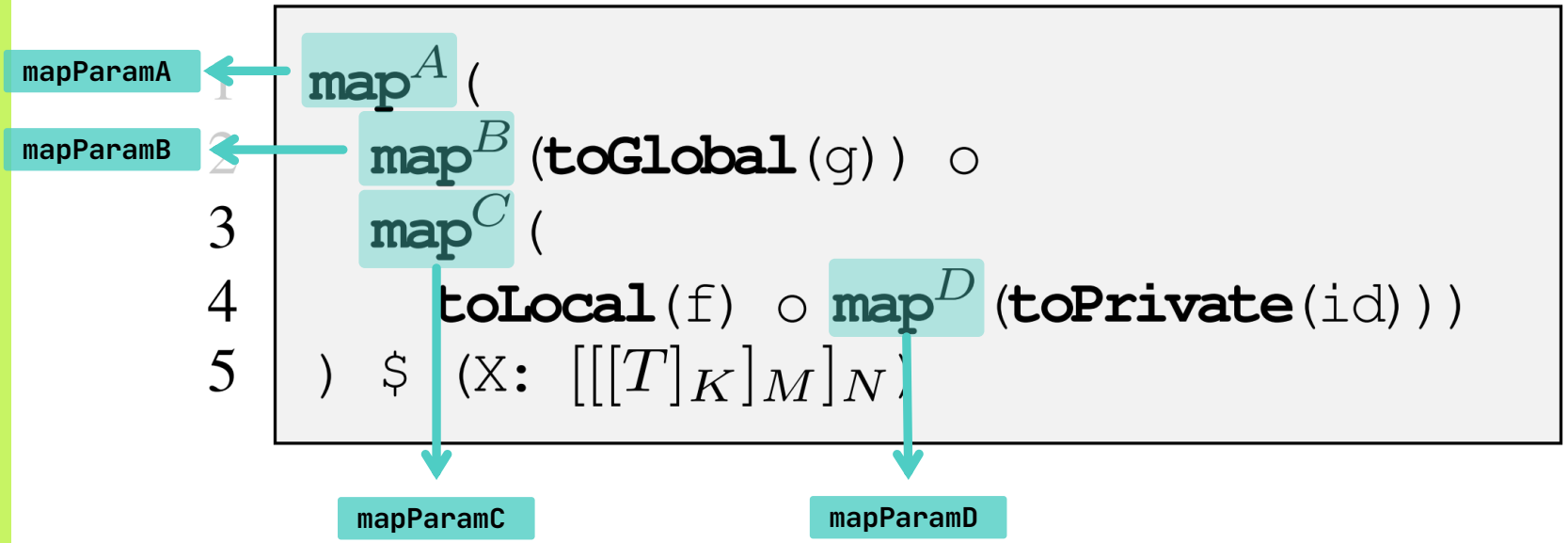
Synchronization overhead

Parallelization Is Hard

```
1  for i in group_id(0) to I do
2    for j in local_id(0) to J do
3      for k in 1 to K do
4        shared_buf[j][k] = f( input[i][j][k] );
5
6    for k in 1 to K do
7      for j in local_id(0) to J do
8        output[i][j][k] = g( shared_buf[j][k] );
```



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

```
1  map^A (
2    map^B (toGlobal(g)) ∘
3    map^C (
4      toLocal(f) ∘ map^D (toPrivate(id))
5  ) $ (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

```
1  map^A (
2    map^B (toGlobal(g)) ∘
3    map^C (
4      toLocal(f) ∘ map^D (toPrivate(id))
5  ) $ (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.

```
∀m : m.usesPrivateMemory
```

GEN CONSTRAINT: `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

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

(mapEncoding(mapParamD) /10 < 1)

Constraint:
Shared Memory Scope

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

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

map^C is local or sequential ∧ **map^A** is a mapWrg

Constraint:
Shared Memory Scope

```
1  map^A (
2     map^B (toGlobal(g)) ∘
3     map^C (
4        toLocal(f) ∘ map^D (toPrivate(id)))
5  ) $ (x: [[[T]_K]_M]_N)
```

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

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

Constraint:
Shared Memory Scope

```
1  map^A (
2    map^B (toGlobal(g)) ∘
3    map^C (
4      toLocal(f) ∘ map^D (toPrivate(id)))
5  ) $ (x: [[[T]_K]_M]_N)
```

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

Constraint:
Hierarchical Parallelism

```
1  map^A (
2     map^B (toGlobal(g)) ∘
3     map^C (
4        toLocal(f) ∘ map^D (toPrivate(id)))
5  ) $ (x: [[[T]_K]_M]_N)
```

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

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

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

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

Constraint:
Hierarchical Parallelism

```
1  map^A (
2    map^B (toGlobal(g)) ∘
3    map^C (
4      toLocal(f) ∘ map^D (toPrivate(id)))
5  ) $ (x: [[[T]_K]_M]_N)
```

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

+10 more hierarchical parallelism constraints

Constraint Satisfaction

```
1  map^A (
2    map^B (toGlobal(g)) ∘
3    map^C (
4      toLocal(f) ∘ map^D (toPrivate(id)
5  ) $ (x: [[[T]_K]_M]_N)
```

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

$$(\texttt{mapEncoding}_{\texttt{0}}(\texttt{mapParamD}) \,/10 < 1)$$
$$(\texttt{mapEncoding}_{\texttt{10}}(\texttt{mapParamC}) \,/10 < 2) \wedge \texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) = 20 + \texttt{w}$$
$$(\texttt{mapEncoding}_{\texttt{10}}(\texttt{mapParamB}) \,/10 < 2) \wedge \texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) = 20 + \texttt{w}$$
$$(\texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) \,/10 < 1) \vee (\texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) \neq \texttt{mapEncoding}_{\texttt{10}}(\texttt{mapParamB}))$$
$$(\texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) \,/10 < 1) \vee (\texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) \neq \texttt{mapEncoding}_{\texttt{10}}(\texttt{mapParamC}))$$
$$\neg((\texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) \,/10 = 1) \wedge (\texttt{mapEncoding}_{\texttt{10}}(\texttt{mapParamB}) \,/10 = 2) \wedge (\texttt{mapEncoding}_{\texttt{10}}(\texttt{mapParamB}) \,\%10 = \texttt{mapEncoding}_{\texttt{20}}(\texttt{mapParamA}) \,\%10))$$

+10 more hierarchical parallelism constraints

Constraint Satisfaction

```
1  mapWrg(0)(
2    mapLcl(0)(toGlobal(g)) ∘
3    mapLcl(0)(
4      toLocal(f) ∘ mapSeq(toPrivate(i
5  ) $ (X: [[[T]_K]_M]_N)
```

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

$$
\begin{aligned}
&(\text{mapEncoding}_{0}(\text{mapParamD}) / 10 < 1) \\
&(\text{mapEncoding}_{10}(\text{mapParamC}) / 10 < 2) \wedge \text{mapEncoding}_{20}(\text{mapParamA}) = 20 + w \\
&(\text{mapEncoding}_{10}(\text{mapParamB}) / 10 < 2) \wedge \text{mapEncoding}_{20}(\text{mapParamA}) = 20 + w \\
&(\text{mapEncoding}_{20}(\text{mapParamA}) / 10 < 1) \vee (\text{mapEncoding}_{20}(\text{mapParamA}) \neq \text{mapEncoding}_{10}(\text{mapParamB})) \\
&(\text{mapEncoding}_{20}(\text{mapParamA}) / 10 < 1) \vee (\text{mapEncoding}_{20}(\text{mapParamA}) \neq \text{mapEncoding}_{10}(\text{mapParamC})) \\
&\neg((\text{mapEncoding}_{20}(\text{mapParamA}) / 10 = 1) \wedge (\text{mapEncoding}_{10}(\text{mapParamB}) / 10 = 2) \wedge \\
&\quad (\text{mapEncoding}_{10}(\text{mapParamB}) \% 10 = \text{mapEncoding}_{20}(\text{mapParamA}) \% 10))
\end{aligned}
$$

+10 more hierarchical parallelism constraints

Heuristics

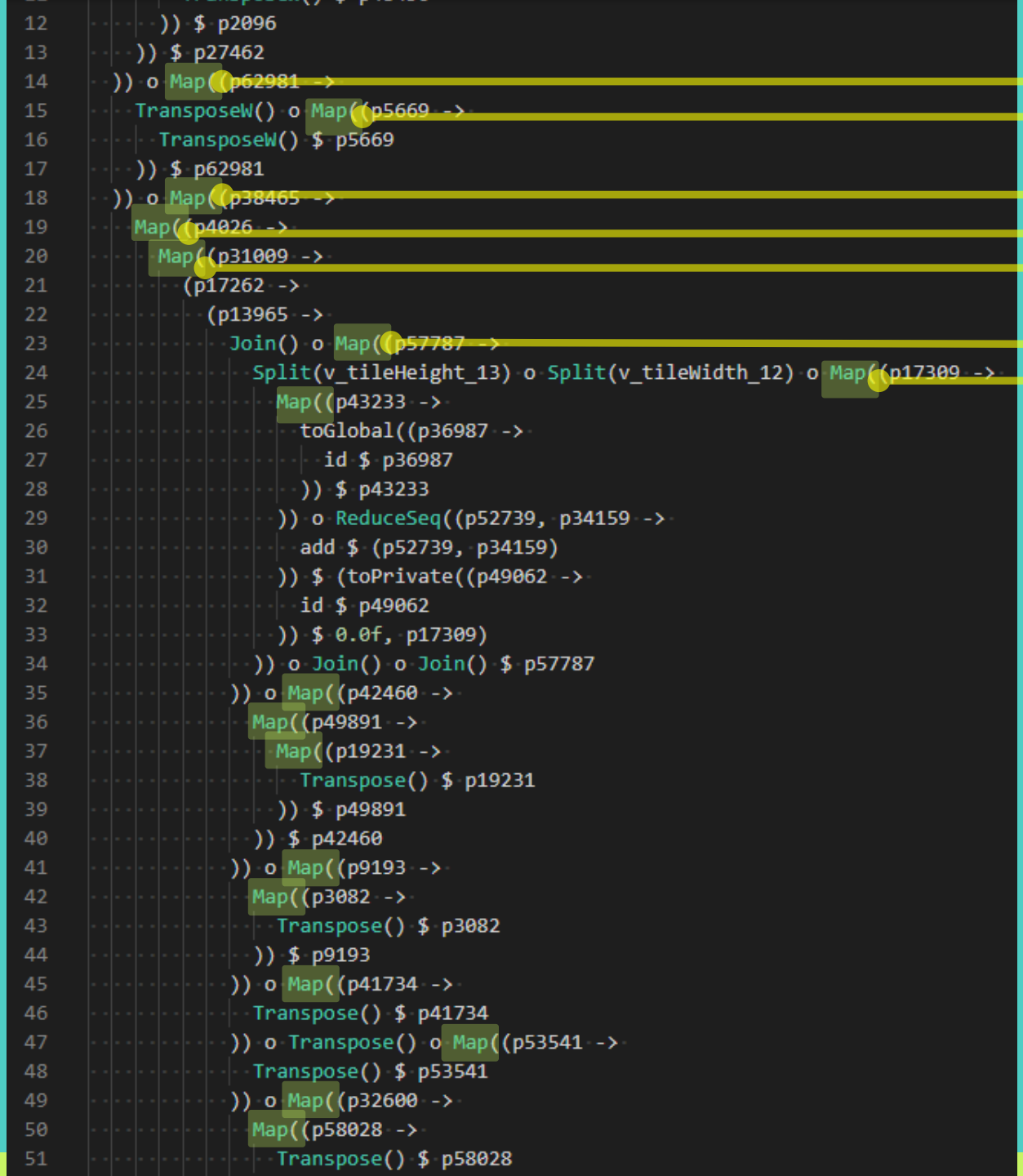
Sequential Map Fusion Heuristic

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
```




```
12          )) $ p2096
13        )) $ p27462
14      )) o Map((p62981 ->
15        TransposeW() o Map((p5669 ->
16          TransposeW() $ p5669
17        )) $ p62981
18      )) o Map((p38465 ->
19        Map((p4026 ->
20          Map((p31009 ->
21            (p17262 ->
22              (p13965 ->
23                Join() o Map((p57787 ->
24                  Split(v_tileHeight_13) o Split(v_tileWidth_12) o Map((p17309 ->
25                    Map((p43233 ->
26                      toGlobal((p36987 ->
27                        id $ p36987
28                      )) $ p43233
29                    )) o ReduceSeq((p52739, p34159 ->
30                      add $ (p52739, p34159)
31                    )) $ (toPrivate((p49062 ->
32                      id $ p49062
33                    )) $ 0.0f, p17309)
34                  )) o Join() o Join() $ p57787
35                )) o Map((p42460 ->
36                  Map((p49891 ->
37                    Map((p19231 ->
38                      Transpose() $ p19231
39                    )) $ p49891
40                  )) $ p42460
41                )) o Map((p9193 ->
42                  Map((p3082 ->
43                    Transpose() $ p3082
44                  )) $ p9193
45                )) o Map((p41734 ->
46                  Transpose() $ p41734
47                )) o Transpose() o Map((p53541 ->
48                  Transpose() $ p53541
49                )) o Map((p32600 ->
50                  Map((p58028 ->
51                    Transpose() $ p58028
```


50

```
12        )) $ p2096
13      )) $ p27462
14    )) o Map((p62981 ->
15      TransposeW() o Map((p5669 ->
16        TransposeW() $ p5669
17      )) $ p62981
18    )) o MapWrg(2,(p38465 ->
19      MapWrg(0,(p4026 ->
20        MapSeq((p31009 ->
21          (p17262 ->
22            (p13965 ->
23              Join() o MapLcl(2,(p57787 ->
24                Split(v_tileHeight_13) o Split(v_tileWidth_12) o MapLcl(0,(p17309
25                  MapSeq((p43233 ->
26                    toGlobal((p36987 ->
27                      id $ p36987
28                    )) $ p43233
29                  )) o ReduceSeq((p52739, p34159 ->
30                    add $ (p52739, p34159)
31                  )) $ (toPrivate((p49062 ->
32                    id $ p49062
33                  )) $ 0.0f, p17309)
34                )) o Join() o Join() $ p57787
35              )) o Map((p42460 ->
36                Map((p49891 ->
37                  Map((p19231 ->
38                    Transpose() $ p19231
39                  )) $ p49891
40                )) $ p42460
41              )) o Map((p9193 ->
42                Map((p3082 ->
43                  Transpose() $ p3082
44                )) $ p9193
45              )) o Map((p41734 ->
46                Transpose() $ p41734
47              )) o Transpose() o Map((p53541 ->
48                Transpose() $ p53541
49              )) o Map((p32600 ->
50                Map((p58028 ->
51                  Transpose() $ p58028
```

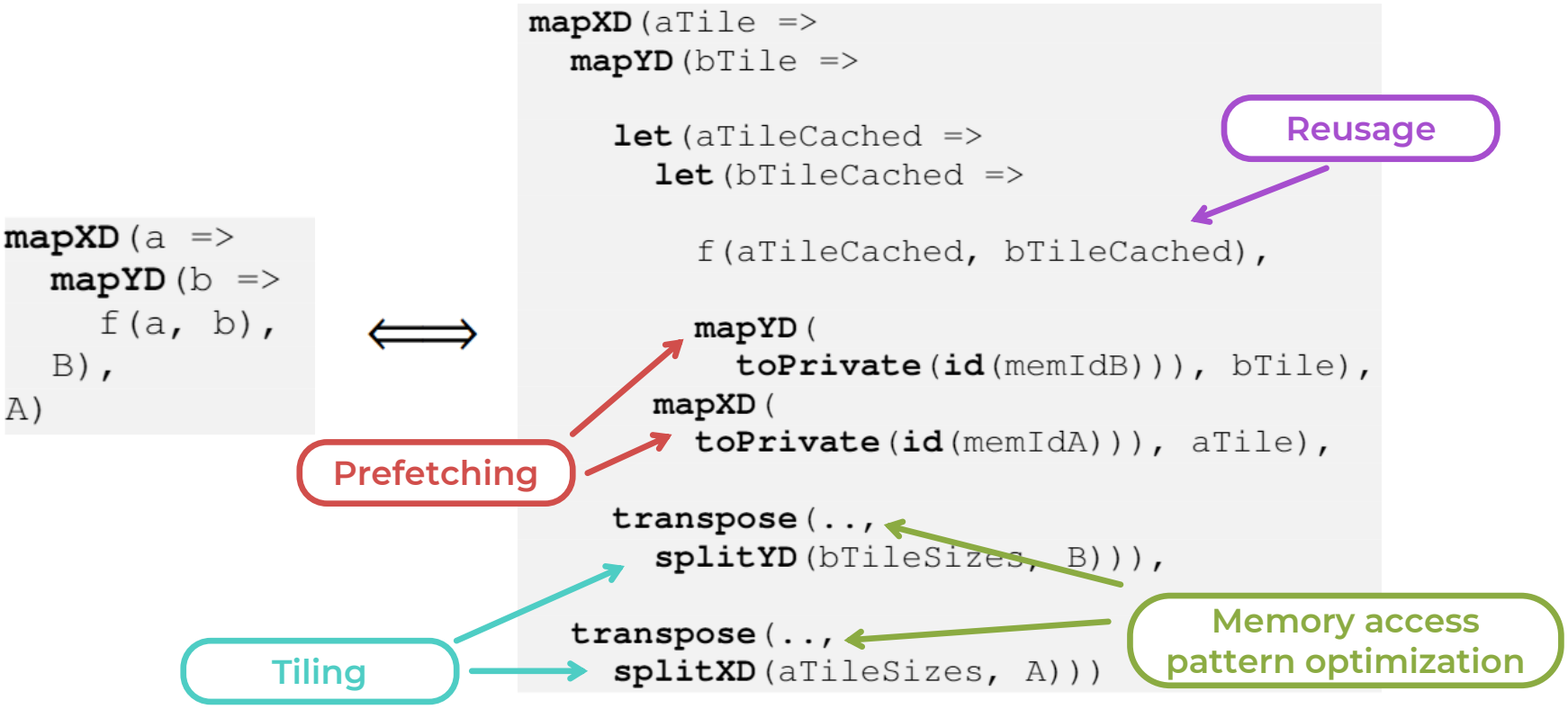

51

Guided Rewriting


Rewrite points expanded
x3

Guided Rewriting: Tiling

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



Guided Rewriting: Tiling

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