Sparse Matrix Compression Primitives With OpenCL Framework to Support Halide

Chao-Lin Lee[1], Chen-Ting Chao[1], Jenq-Kuen Lee[1],
Chung-Wen Huang[2], and Ming-Yu Hung[2]

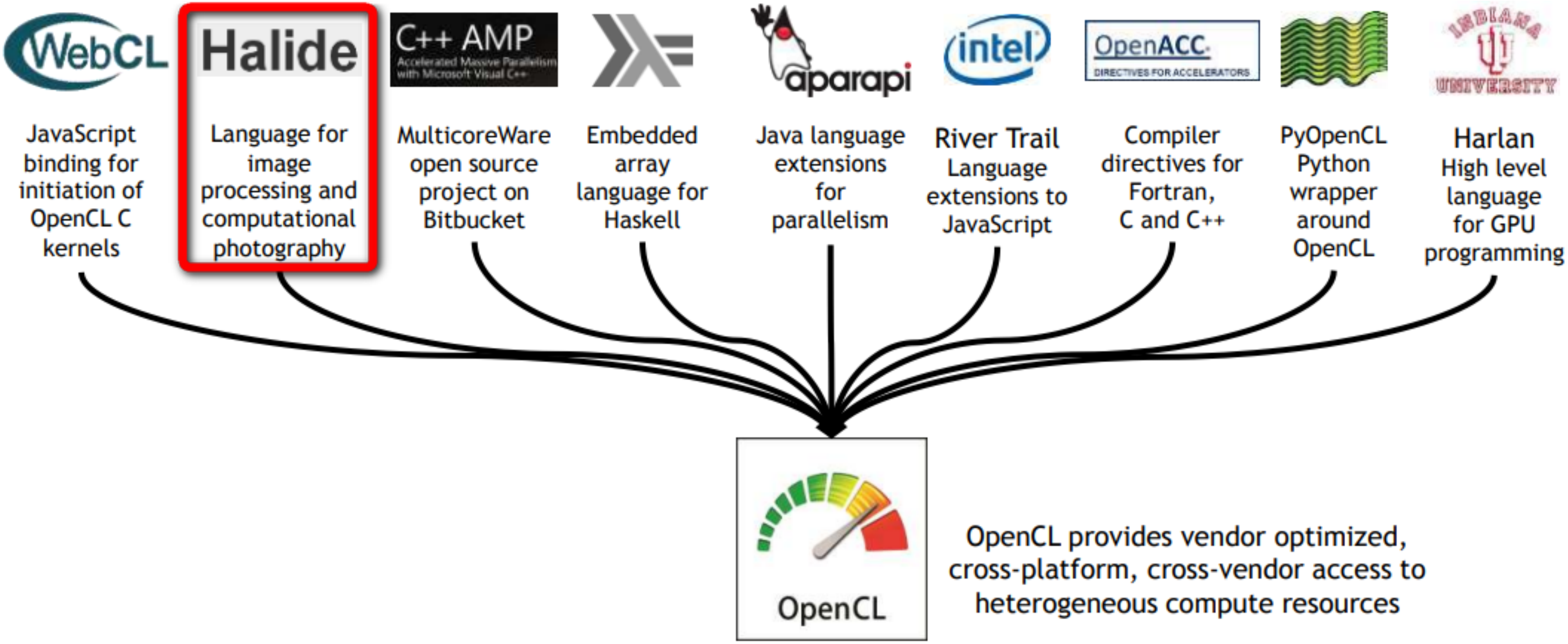
National Tsing-Hua University,
Hsinchu, Taiwan[1]

MediaTek Inc.[2]

Outline

- Background
- Halide overview
- Our Design in OpenCL with Halide
- Conclusion

Background


WebCL
Halide
C++ AMP
Accelerated Massive Parallelism with Microsoft Visual C++
aparapi
intel
OpenACC
DIRECTIVES FOR ACCELERATORS
INDIANA UNIVERSITY
OpenCL

JavaScript binding for initiation of OpenCL C kernels

Language for image processing and computational photography

MulticoreWare open source project on Bitbucket

Embedded array language for Haskell

Java language extensions for parallelism

River Trail Language extensions to JavaScript

Compiler directives for Fortran, C and C++

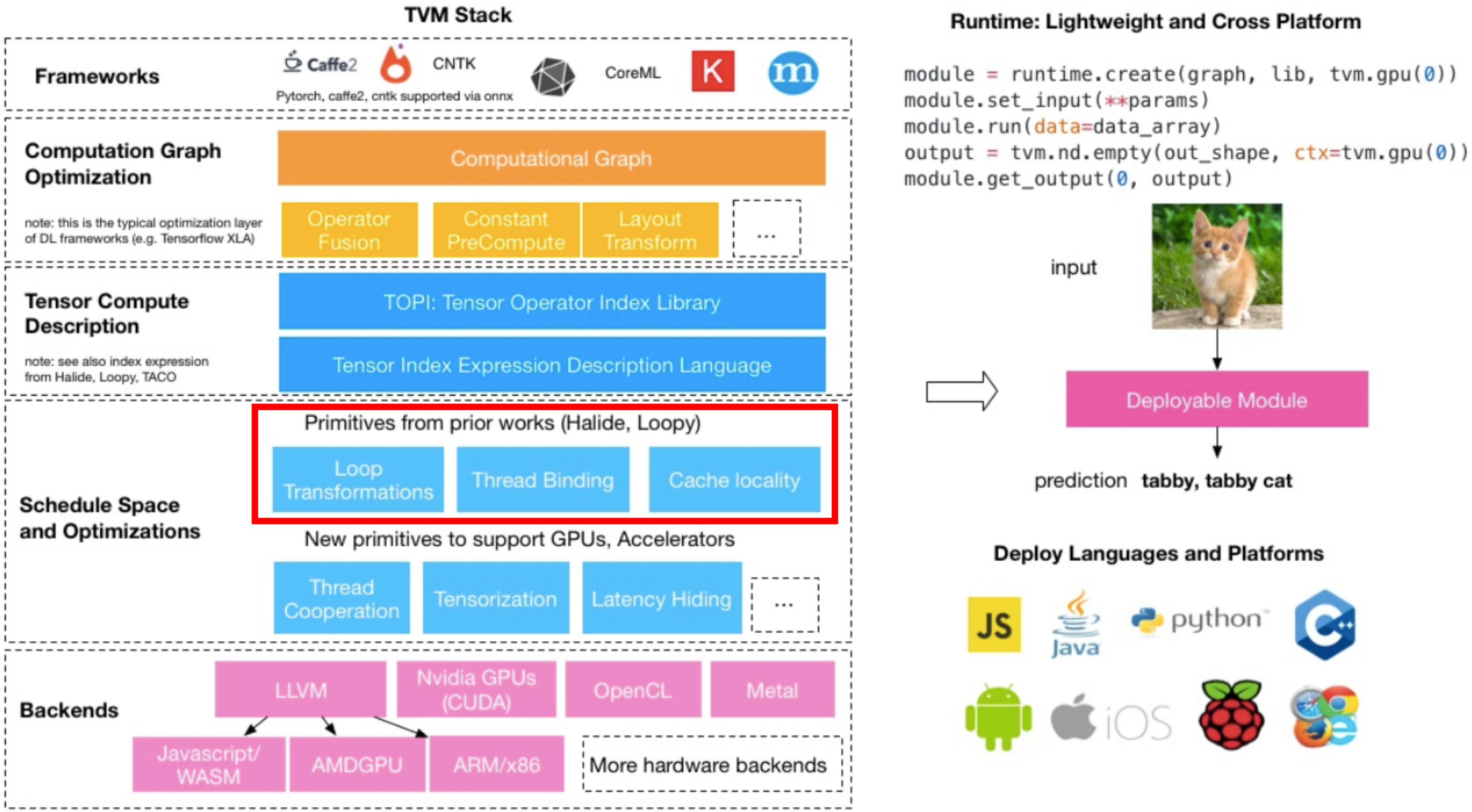
PyOpenCL Python wrapper around OpenCL

Harlan High level language for GPU programming

OpenCL provides vendor optimized, cross-platform, cross-vendor access to heterogeneous compute resources

Courtesy of Khronos, SIGGRAPH, Vancouver 2014

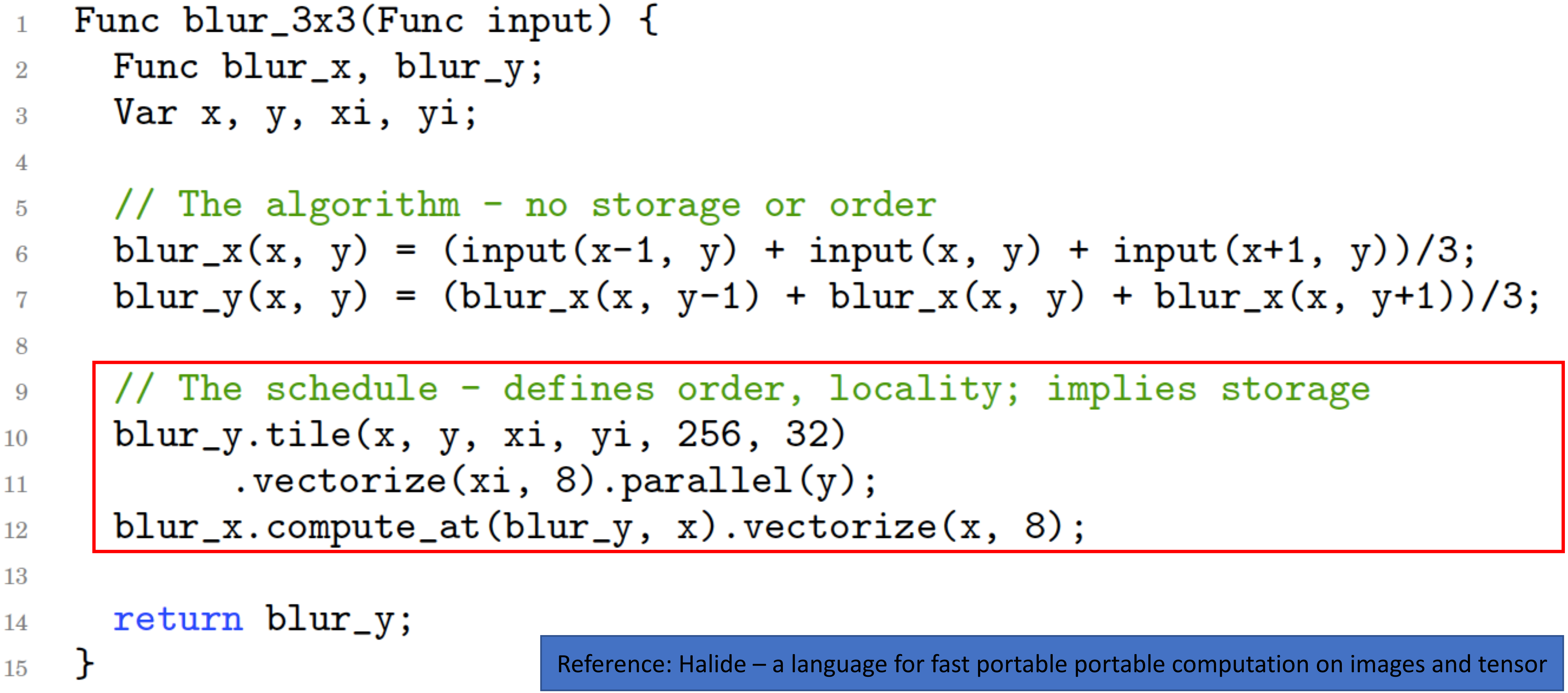
Halide in TVM


TVM Stack
Frameworks
Caffe2
CNTK
CoreML
K
m
Pytorch, caffe2, cntk supported via onnx
Computation Graph Optimization
note: this is the typical optimization layer of DL frameworks (e.g. Tensorflow XLA)
Computational Graph
Operator Fusion
Constant PreCompute
Layout Transform
...
Tensor Compute Description
note: see also index expression from Halide, Loopy, TACO
TOPI: Tensor Operator Index Library
Tensor Index Expression Description Language
Schedule Space and Optimizations
Primitives from prior works (Halide, Loopy)
Loop Transformations
Thread Binding
Cache locality
New primitives to support GPUs, Accelerators
Thread Cooperation
Tensorization
Latency Hiding
...
Backends
LLVM
Nvidia GPUs (CUDA)
OpenCL
Metal
Javascript/ WASM
AMDGPU
ARM/x86
More hardware backends
Runtime: Lightweight and Cross Platform
input
Deployable Module
prediction tabby, tabby cat
Deploy Languages and Platforms
JS
Java
python
iOS

```
module = runtime.create(graph, lib, tvm.gpu(0))
module.set_input(**params)
module.run(data=data_array)
output = tvm.nd.empty(out_shape, ctx=tvm.gpu(0))
module.get_output(0, output)
```

Source: https://tvm.ai/

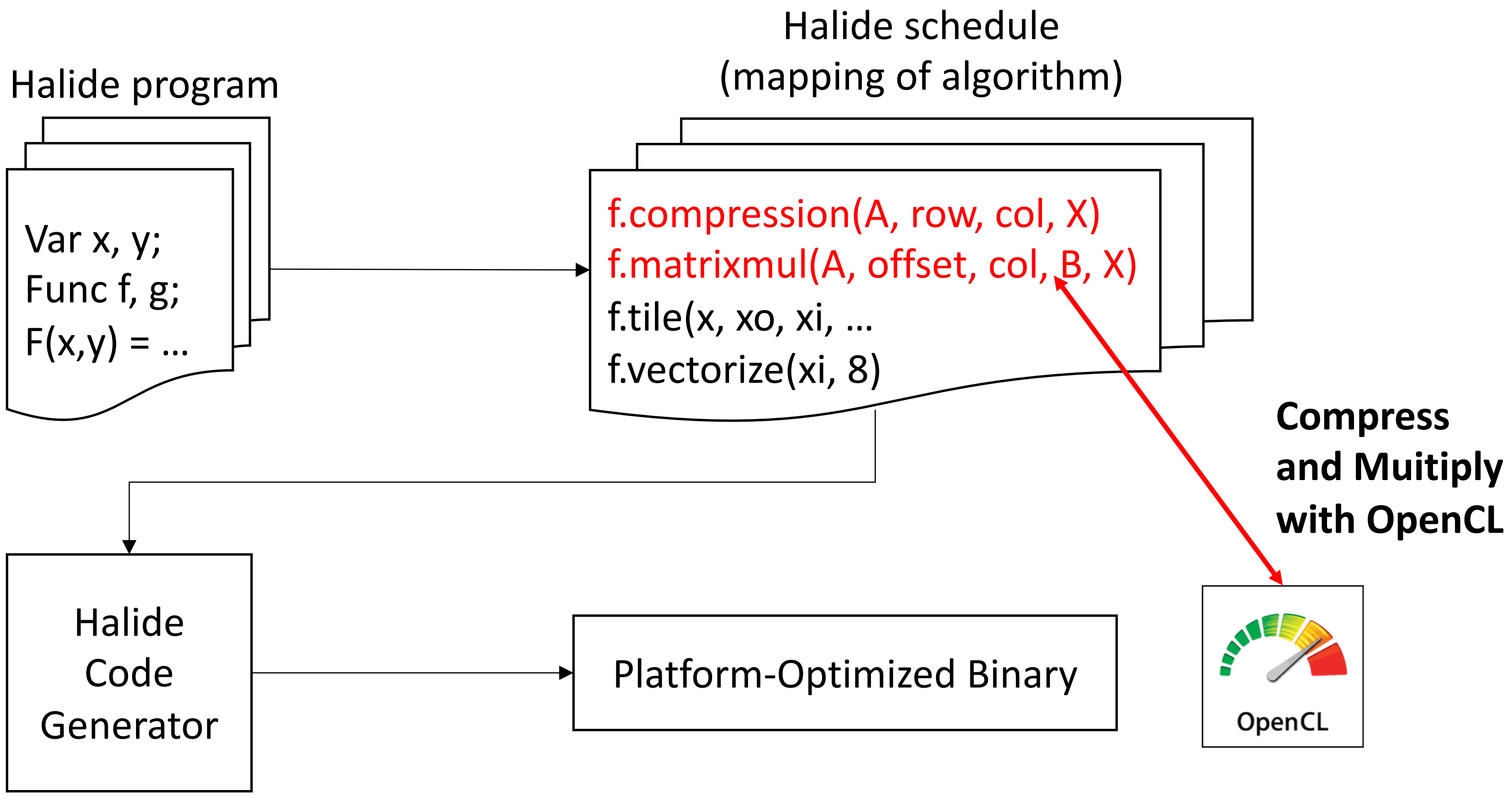
Halide and Halide IR



```
Func blur_3x3(Func input) {
  Func blur_x, blur_y;
  Var x, y, xi, yi;

  // The algorithm - no storage or order
  blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
  blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;

  // The schedule - defines order, locality; implies storage
  blur_y.tile(x, y, xi, yi, 256, 32)
        .vectorize(xi, 8).parallel(y);
  blur_x.compute_at(blur_y, x).vectorize(x, 8);

  return blur_y;
}
```

Reference: Halide – a language for fast portable portable computation on images and tensor

The Flow for SPMM Enabled in Halide with OpenCL



Halide program

Var x, y;
Func f, g;
F(x,y) = …

Halide schedule
(mapping of algorithm)

f.compression(A, row, col, X)
f.matrixmul(A, offset, col, B, X)
f.tile(x, xo, xi, …
f.vectorize(xi, 8)

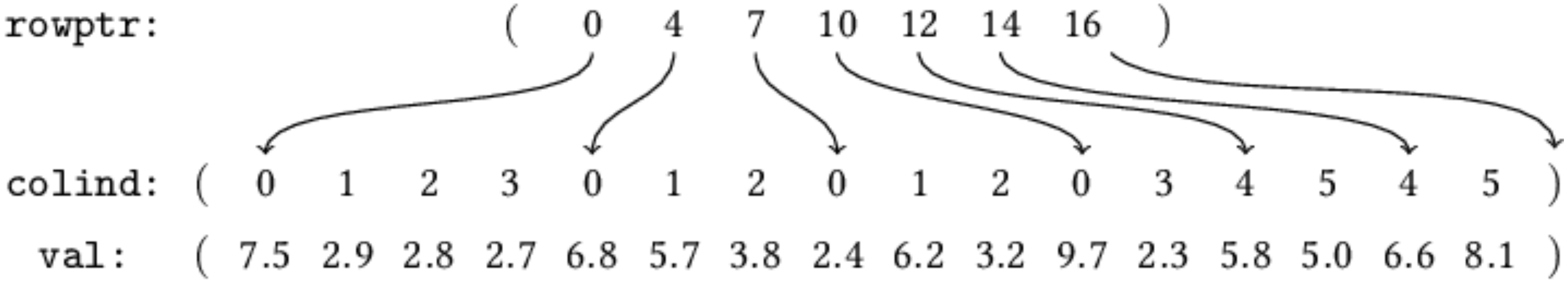
Compress and Muitiply with OpenCL

Halide Code Generator

Platform-Optimized Binary

CSR

$$A = \begin{pmatrix} 7.5 & 2.9 & 2.8 & 2.7 & 0 & 0 \\ 6.8 & 5.7 & 3.8 & 0 & 0 & 0 \\ 2.4 & 6.2 & 3.2 & 0 & 0 & 0 \\ 9.7 & 0 & 0 & 2.3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5.8 & 5.0 \\ 0 & 0 & 0 & 0 & 6.6 & 8.1 \end{pmatrix}$$

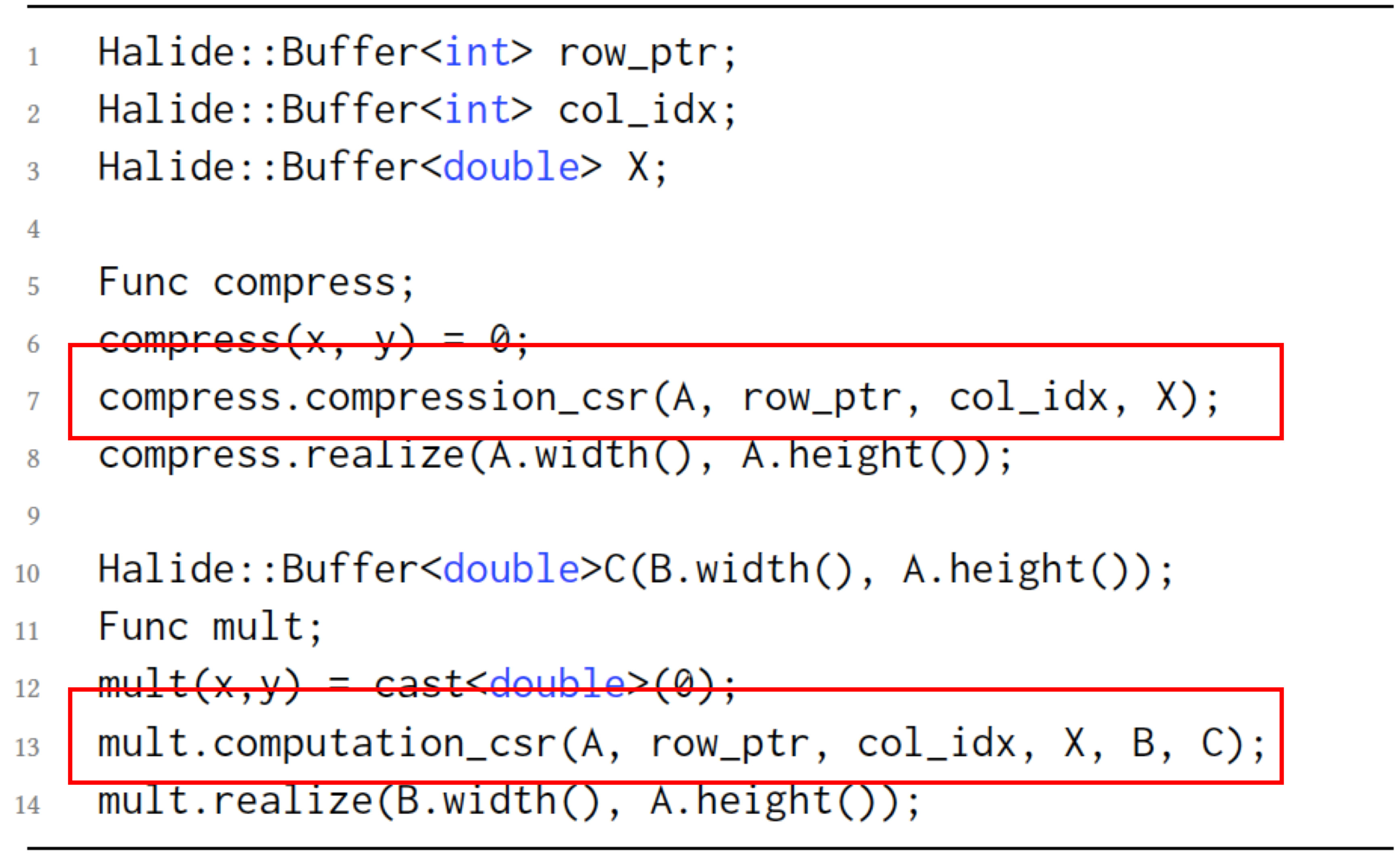


rowptr: (0 4 7 10 12 14 16)

colind: (0 1 2 3 0 1 2 0 1 2 0 3 4 5 4 5)

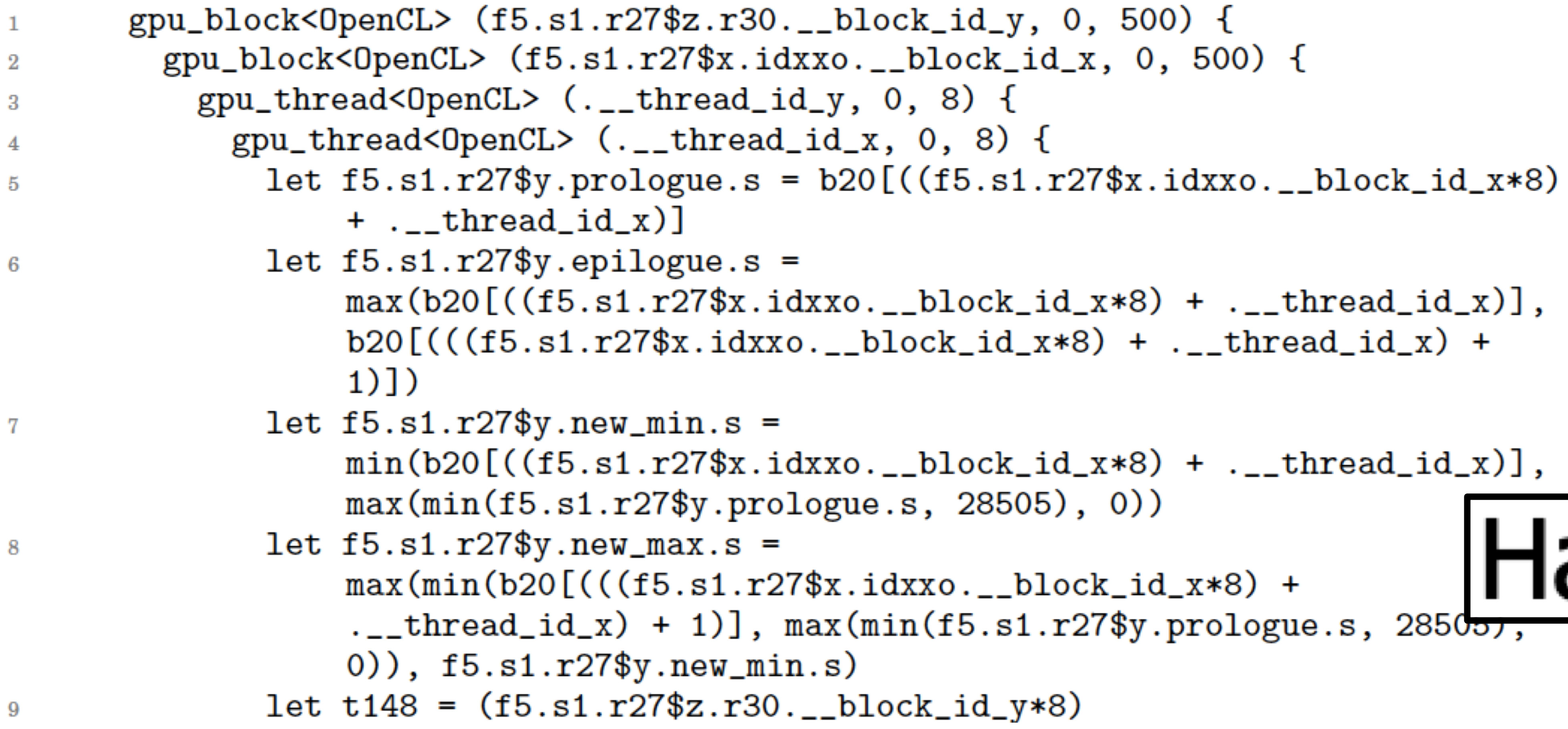
val: (7.5 2.9 2.8 2.7 6.8 5.7 3.8 2.4 6.2 3.2 9.7 2.3 5.8 5.0 6.6 8.1)

Our Sparse GEMM in Halide



```
Halide::Buffer<int> row_ptr;
Halide::Buffer<int> col_idx;
Halide::Buffer<double> X;

Func compress;
compress(x, y) = 0;
compress.compression_csr(A, row_ptr, col_idx, X);
compress.realize(A.width(), A.height());

Halide::Buffer<double>C(B.width(), A.height());
Func mult;
mult(x,y) = cast<double>(0);
mult.computation_csr(A, row_ptr, col_idx, X, B, C);
mult.realize(B.width(), A.height());
```

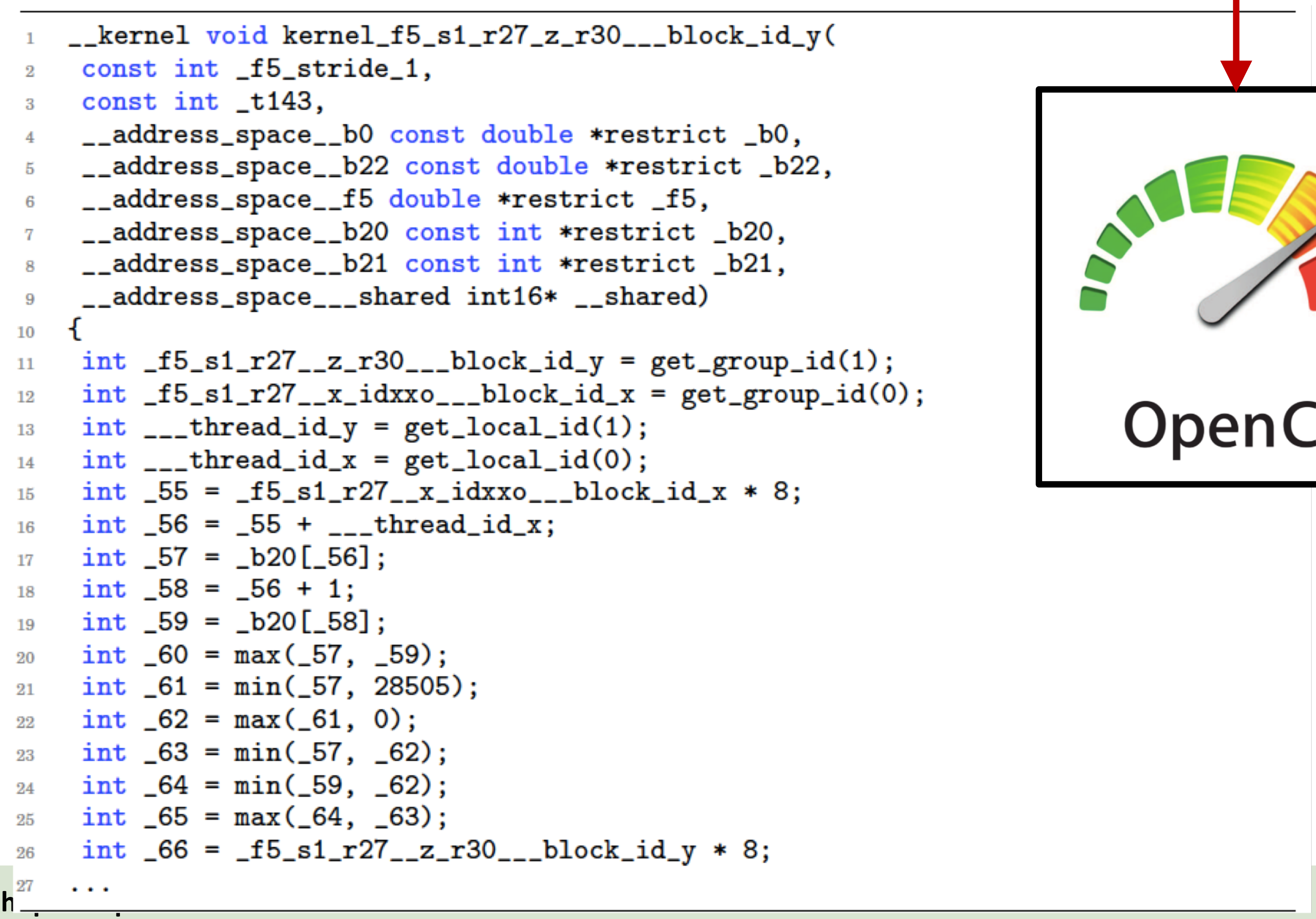
SPMM Parallelization in Halide with OpenCL Kernel



```
gpu_block<OpenCL> (f5.s1.r27$z.r30.__block_id_y, 0, 500) {
  gpu_block<OpenCL> (f5.s1.r27$x.idxxo.__block_id_x, 0, 500) {
    gpu_thread<OpenCL> (.__thread_id_y, 0, 8) {
      gpu_thread<OpenCL> (.__thread_id_x, 0, 8) {
        let f5.s1.r27$y.prologue.s = b20[((f5.s1.r27$x.idxxo.__block_id_x*8)
            + .__thread_id_x)]
        let f5.s1.r27$y.epilogue.s =
            max(b20[((f5.s1.r27$x.idxxo.__block_id_x*8) + .__thread_id_x)],
            b20[(((f5.s1.r27$x.idxxo.__block_id_x*8) + .__thread_id_x) +
            1)])
        let f5.s1.r27$y.new_min.s =
            min(b20[((f5.s1.r27$x.idxxo.__block_id_x*8) + .__thread_id_x)],
            max(min(f5.s1.r27$y.prologue.s, 28505), 0))
        let f5.s1.r27$y.new_max.s =
            max(min(b20[(((f5.s1.r27$x.idxxo.__block_id_x*8) +
            .__thread_id_x) + 1)], max(min(f5.s1.r27$y.prologue.s, 28505),
            0)), f5.s1.r27$y.new_min.s)
        let t148 = (f5.s1.r27$z.r30.__block_id_y*8)
```

Halide

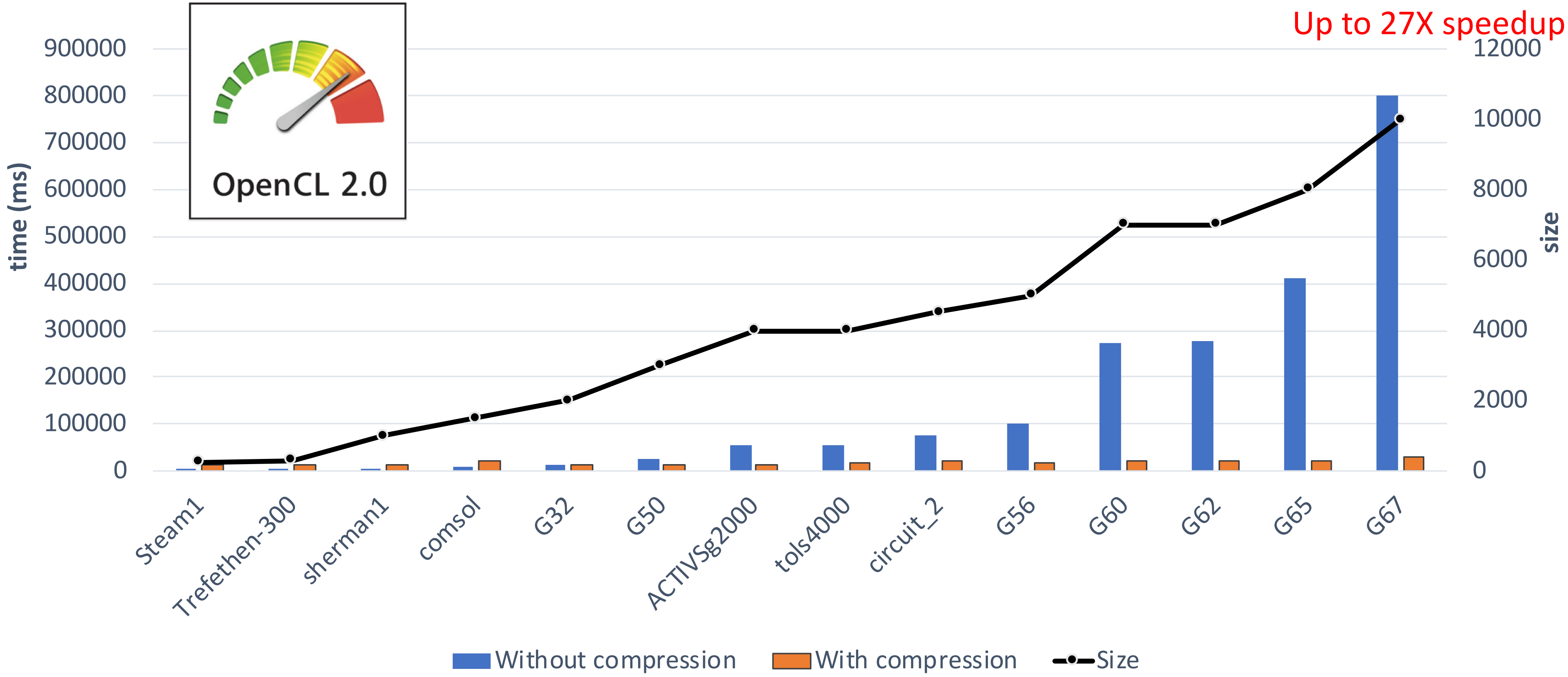
SPMM Parallelization in Halide with OpenCL Kernel



```
__kernel void kernel_f5_s1_r27_z_r30___block_id_y(
 const int _f5_stride_1,
 const int _t143,
 __address_space__b0 const double *restrict _b0,
 __address_space__b22 const double *restrict _b22,
 __address_space__f5 double *restrict _f5,
 __address_space__b20 const int *restrict _b20,
 __address_space__b21 const int *restrict _b21,
 __address_space___shared int16* __shared)
{
 int _f5_s1_r27__z_r30___block_id_y = get_group_id(1);
 int _f5_s1_r27__x_idxxo___block_id_x = get_group_id(0);
 int ___thread_id_y = get_local_id(1);
 int ___thread_id_x = get_local_id(0);
 int _55 = _f5_s1_r27__x_idxxo___block_id_x * 8;
 int _56 = _55 + ___thread_id_x;
 int _57 = _b20[_56];
 int _58 = _56 + 1;
 int _59 = _b20[_58];
 int _60 = max(_57, _59);
 int _61 = min(_57, 28505);
 int _62 = max(_61, 0);
 int _63 = min(_57, _62);
 int _64 = min(_59, _62);
 int _65 = max(_64, _63);
 int _66 = _f5_s1_r27__z_r30___block_id_y * 8;
 ...
```

OpenCL

Sparse Matrix Compress and Multiplication with Halide / OpenCL



Conclusion

- Matrix compression CSR in Halide with OpenCL

- Exploit the opportunities of optimizing GEMM convolution layers with sparse matrix compression

Thank you