Romou: Rapidly Generate High-Performance Tensor Kernels for Mobile GPUs

MobiCom' 22

Rendong Liang, Ting Cao, Jicheng Wen, Manni Wang, Yang Wang, Jianhua Zou and Yunxin Liu

University of California, Microsoft Research, Microsoft STCA, Xi'an Jiao Tong University and Tsinghua University

Introduction

More and more AI application on mobile devices.

Google Assistant

starryai - Create Art with AI

FaceApp: Face Editor

Introduction

Inference on local compared with on cloud:

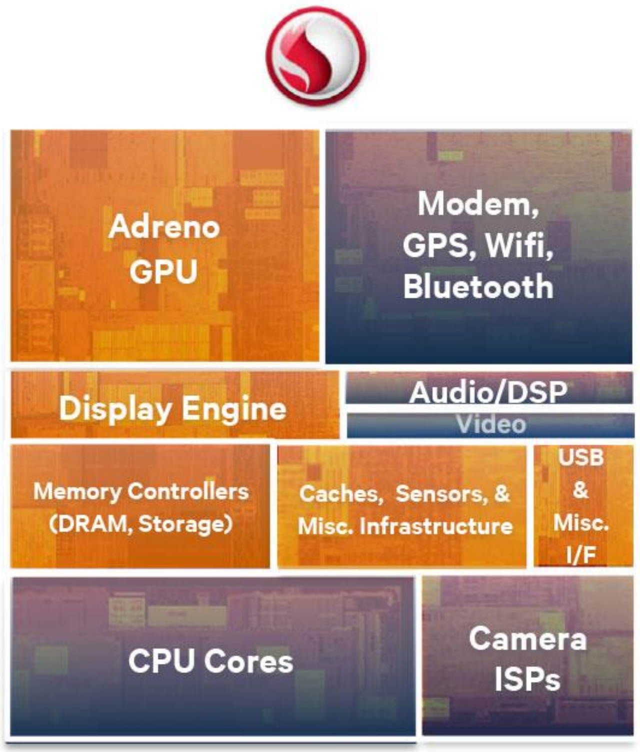
More privacy guarantee.

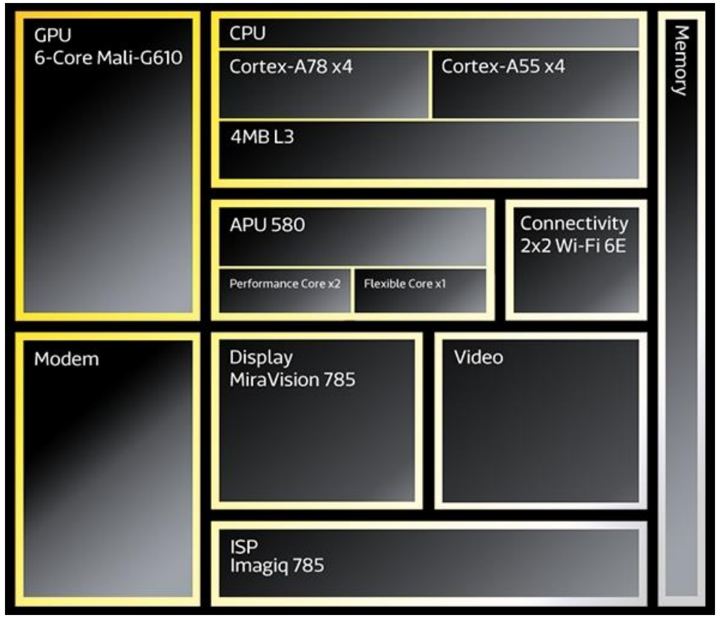
More reliable network resilience.

More quick responsibility.

Introduction

Mobile GPUs are powerful, ubiquitous, and accessible accelerators.





Introduction

To utilize mobile GPUs, we always need to take kernels of operators.

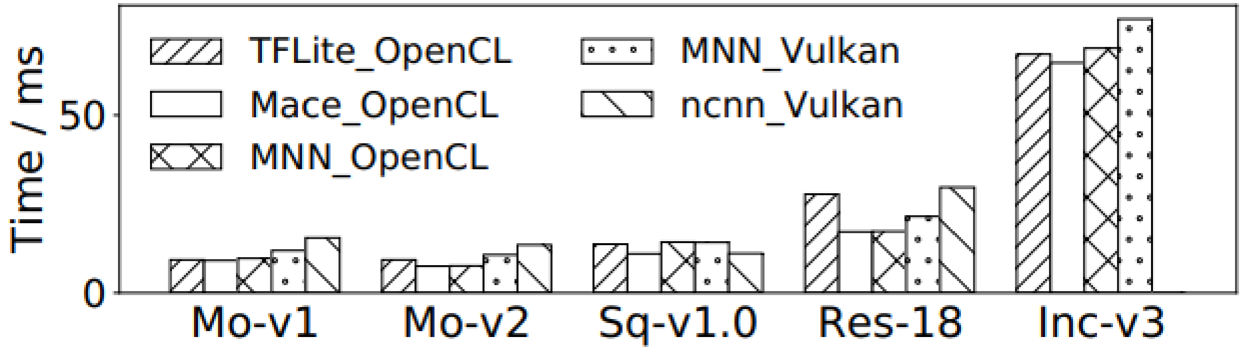
28 lines (25 sloc) | 1.02 KB

```
#include <common.h>
// Supported data types: half/float
__kernel void bias_add(OUT_OF_RANGE_PARAMS
                       GLOBAL_WORK_GROUP_SIZE_DIM3
                       __private const int input_height,
                       __read_only image2d_t input,
                       __read_only image2d_t bias,
                       __write_only image2d_t output) {
  const int ch_blk = get_global_id(0);
  const int width_idx = get_global_id(1);
  const int hb_idx = get_global_id(2);

#ifndef NON_UNIFORM_WORK_GROUP
  if (ch_blk >= global_size_dim0 || width_idx >= global_size_dim1
      || hb_idx >= global_size_dim2) {
    return;
  }
#endif
  const int width = global_size_dim1;

  const int pos = mad24(ch_blk, width, width_idx);
  DATA_TYPE4 in = READ_IMAGET(input, SAMPLER, (int2)(pos, hb_idx));
  const int b_idx = select(0, hb_idx / input_height, input_height > 0);
  DATA_TYPE4 bias_value = READ_IMAGET(bias, SAMPLER, (int2)(ch_blk, b_idx));
  DATA_TYPE4 out = in + bias_value;

  WRITE_IMAGET(output, (int2)(pos, hb_idx), out);
}
```

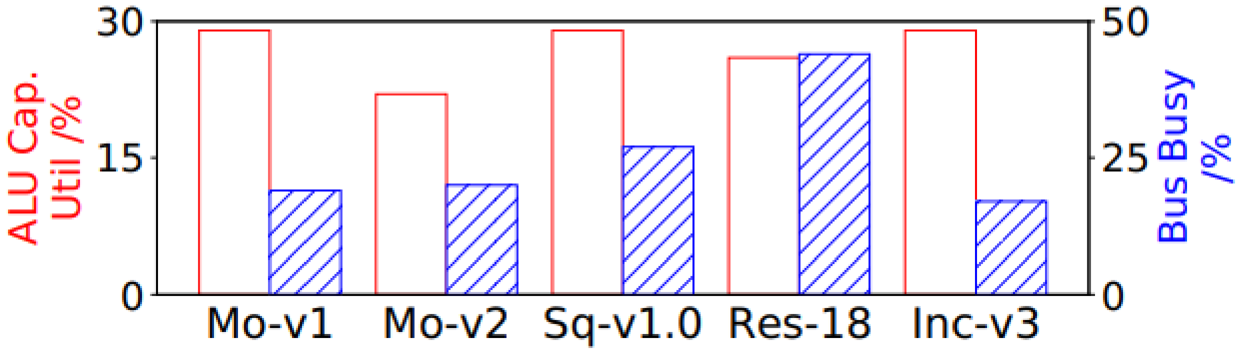
bias_add kernel implemented by OpenCL in MACE

Introduction

Manual implemented kernels are always suboptimal for the deployed hardware.



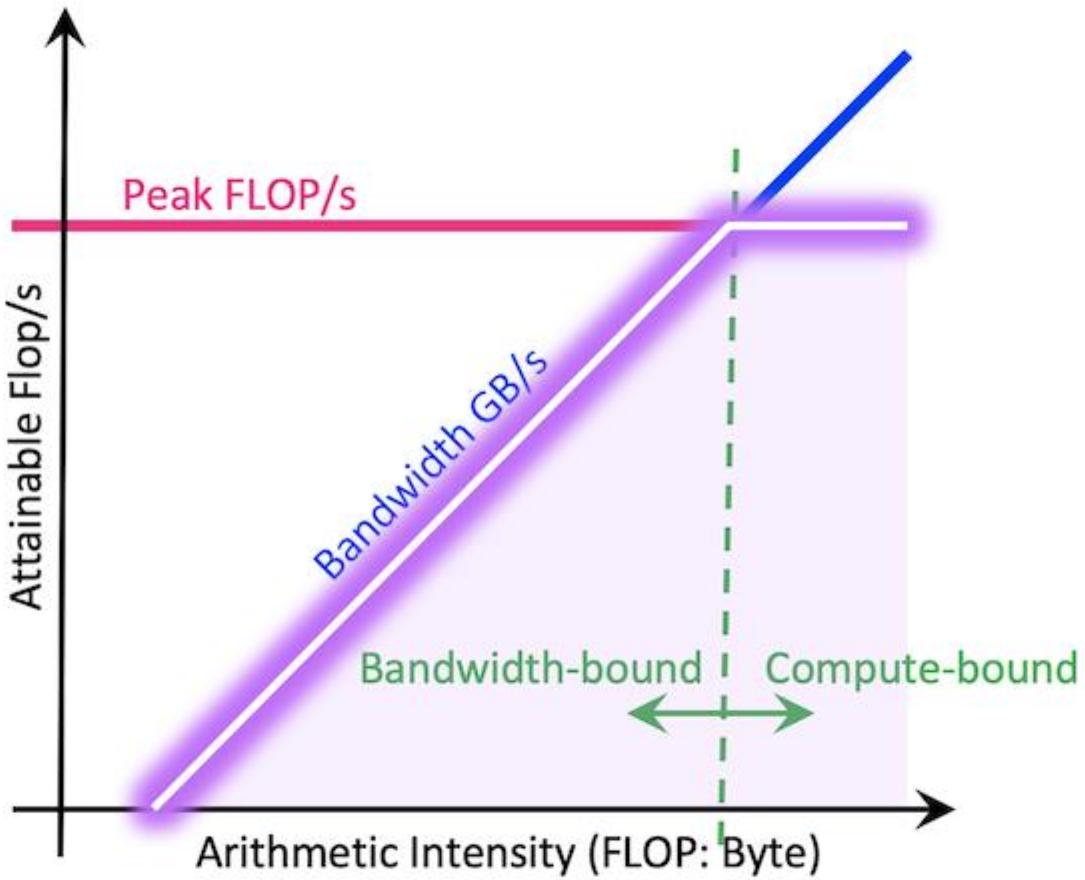
(a)



(b)

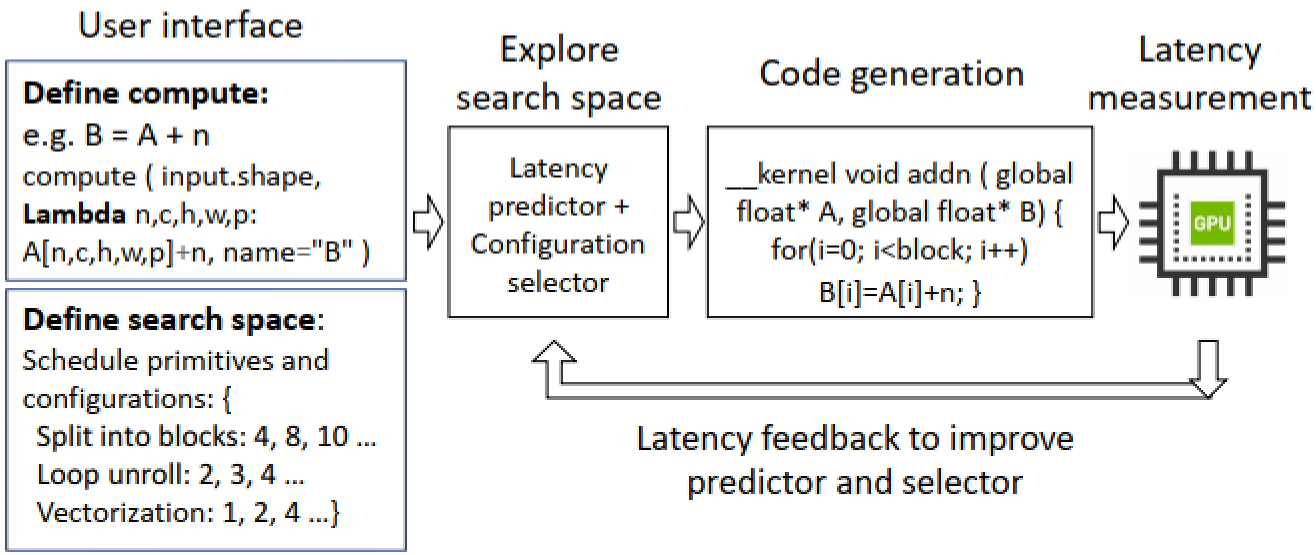
Introduction

Roofline model.



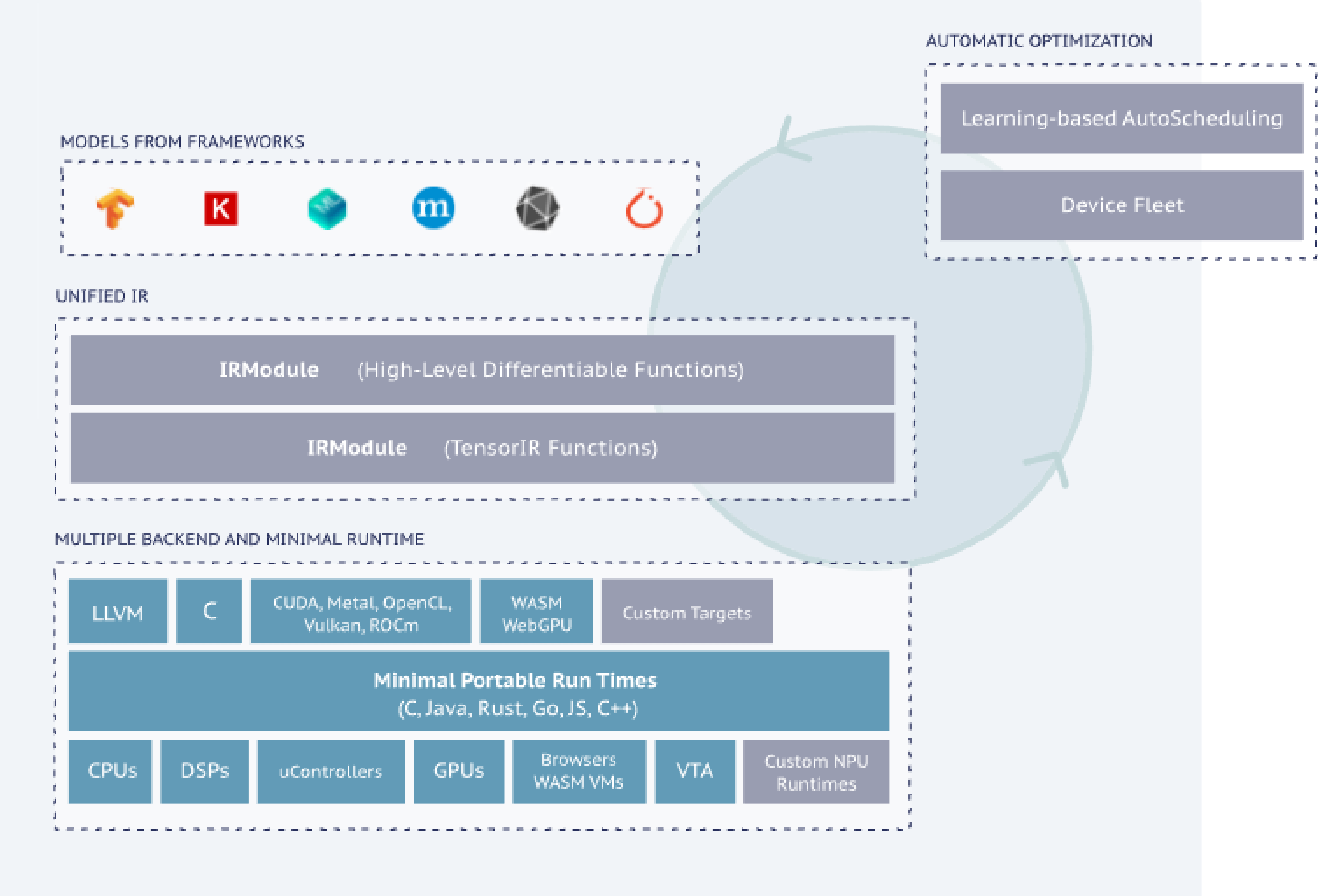
Introduction

Manual implemented kernels are always suboptimal for the deployed hardware.
So that model compiler has been introduced.



Introduction

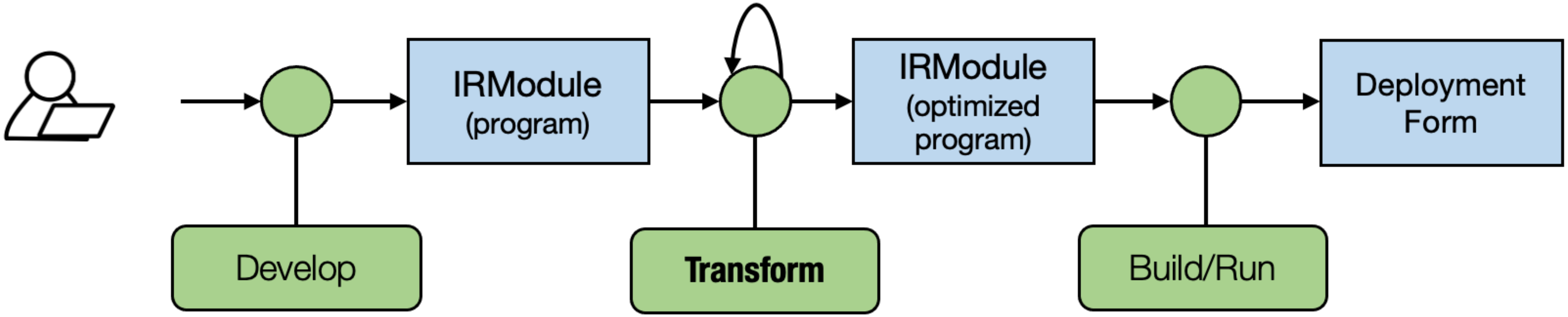
Manual implemented kernels are always suboptimal for the deployed hardware.
So that model compiler has been introduced.



TVM [OSDI'18]

Introduction

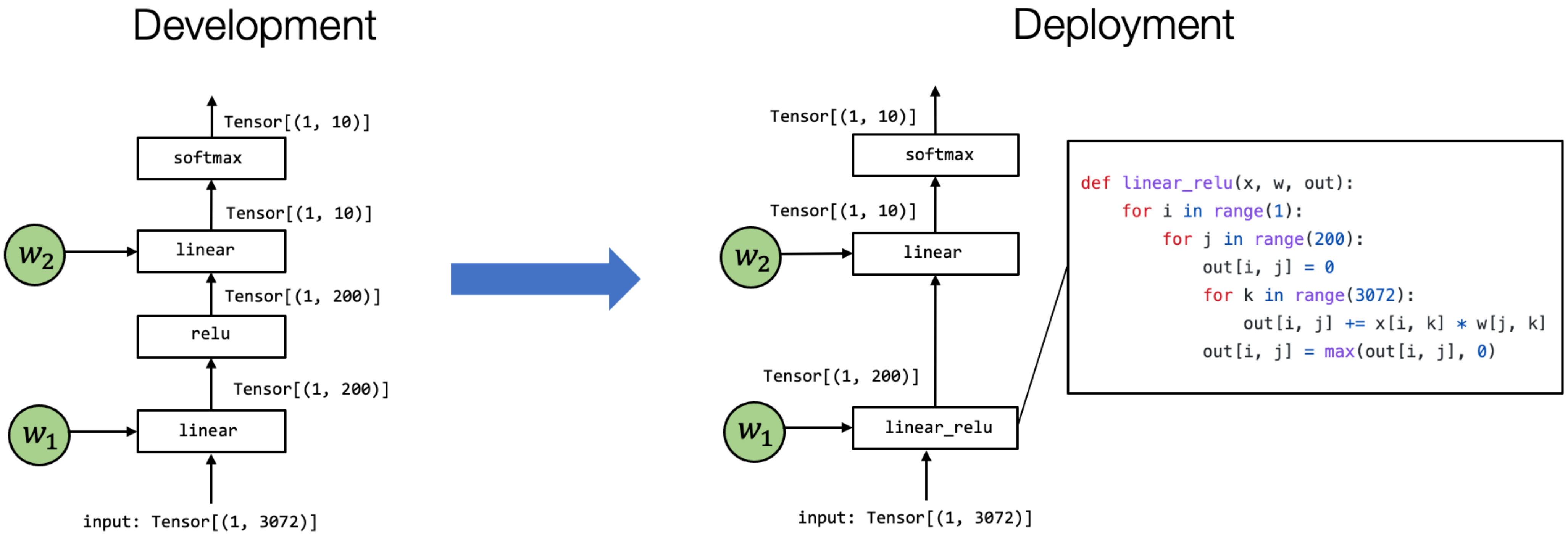
Manual implemented kernels are always suboptimal for the deployed hardware.
So that model compiler has been introduced.



TVM [OSDI'18]

Introduction

Manual implemented kernels are always suboptimal for the deployed hardware.
So that model compiler has been introduced.

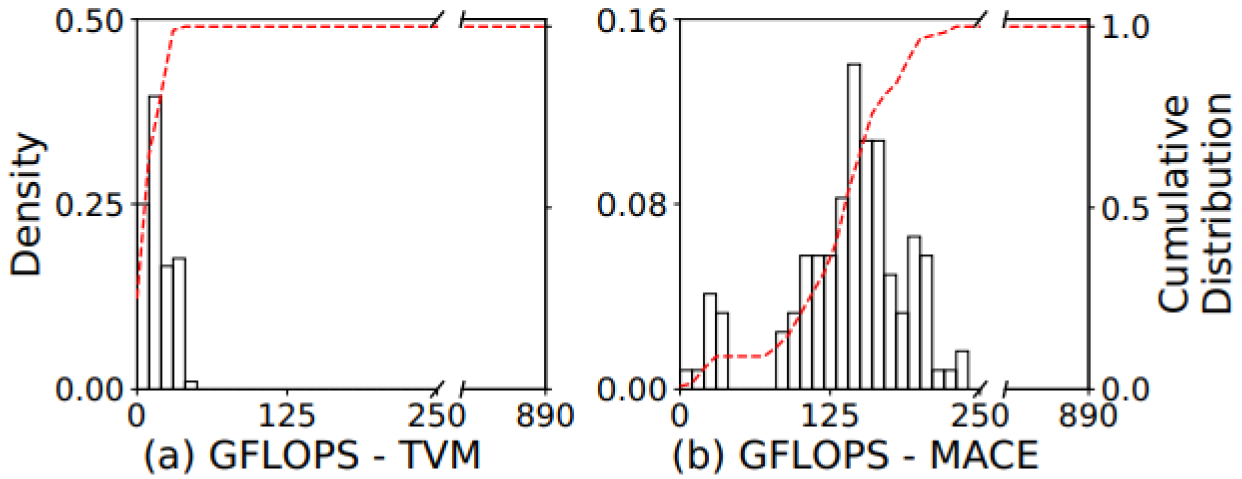


Most MLC process can be viewed as transformation among tensor functions (that can be represented with different abstractions).

TVM [OSDI'18]

Introduction

However, TVM cannot perform competitive performance with manual library.



Introduction

Why TVM performs so bad?

1. No mobile feature supported.
2. The search space is too large to arrive the optimal.

Introduction

Why TVM performs so bad?

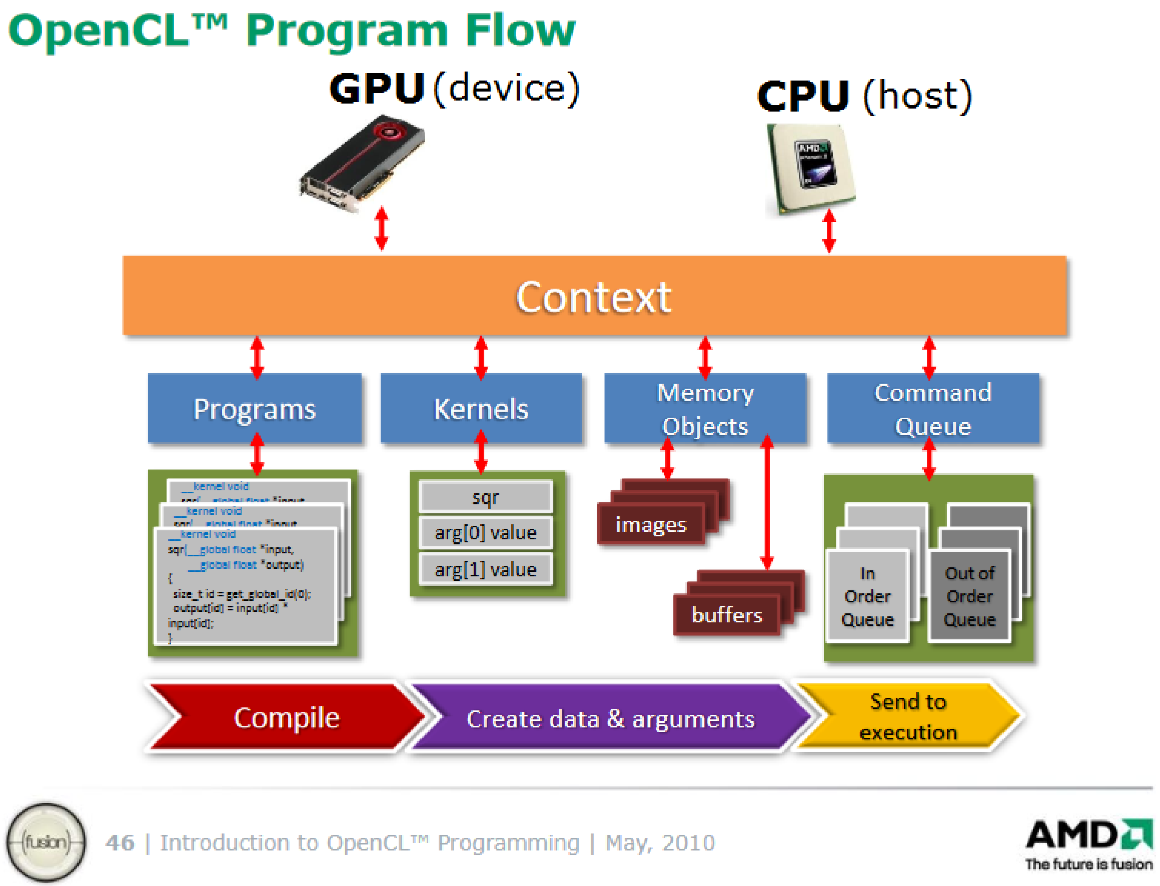
1. No mobile feature supported.
   a. Texture cache
   b. Scalar-vector computing
2. The search space is too large to arrive the optimal.
   a. Prune search space
   b. Eliminating redundant calculation

New challenges:

C1: Black-box hardware information
C2: Traditional server-centered compiler

Introduction

Mobile GPU programming



Key1 - ArchProbe

Goal

To disclose and quantify performance-vital hardware features.

Challenges

C1: predictable hardware behavior of micro-benchmark kernels
C2: high-resolution timing

Solutions

S1: avoid compiler optimizations
S2: Using the law of large numbers

Features

- The number of registers
- Memory hierarchy: cache size, cacheline and bandwidth
- Warp size
- The number of ALUs

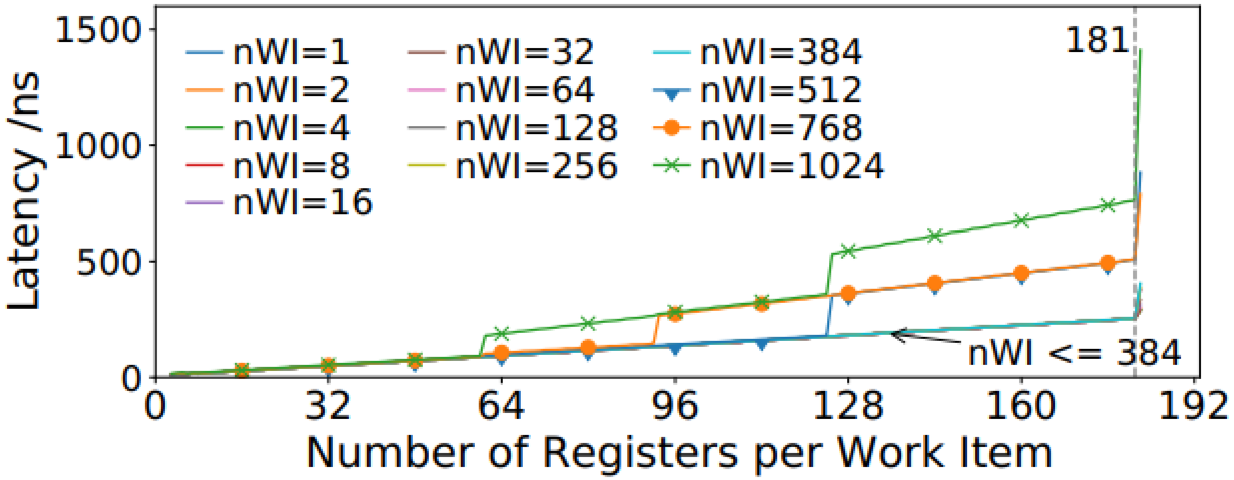
Key1 - ArchProbe

Detect the number of registers

```
for (nWorkItem = 1; nWorkItem<maxLogicalThread; nWorkItem+=step)
  for (nReg = 0; nReg < threshold; nReg++)
    runKernel(reg_count, (nWorkItems,1,1)/*work group size*/,
              1/*total work groups*/, nReg, clEventTimer);

/* Generate kernel codes for different nReg */
for (int i = 0; i < nReg; ++i){
  reg_declare += format("float reg_data", i, " = ", i, ";\n");
  reg_comp += format("reg_data", i, " *= reg_data",
                     i==0? nReg-1: i-1, ";\n");
  save_to_mem += format("out_buf[", i, " * i] = reg_data",
                        i, ";\n");}

auto src = format(R"(
__kernel void reg_count(__global float* out_buf) {
  )", reg_declare, R"(
  int i = 0;
  for (; i < N; ++i) { /*run N times to reduce timing error*/
  )", reg_comp, R"( }
  i = i >> 31; /* make output buffer index a variable */
  /*save results to memory in case of dead code elimination*/
  )", save_to_mem, R"( } )");
```

Key1 - ArchProbe

Detect the number of registers



384x181 register file size
Shared within work items

Key1 - ArchProbe

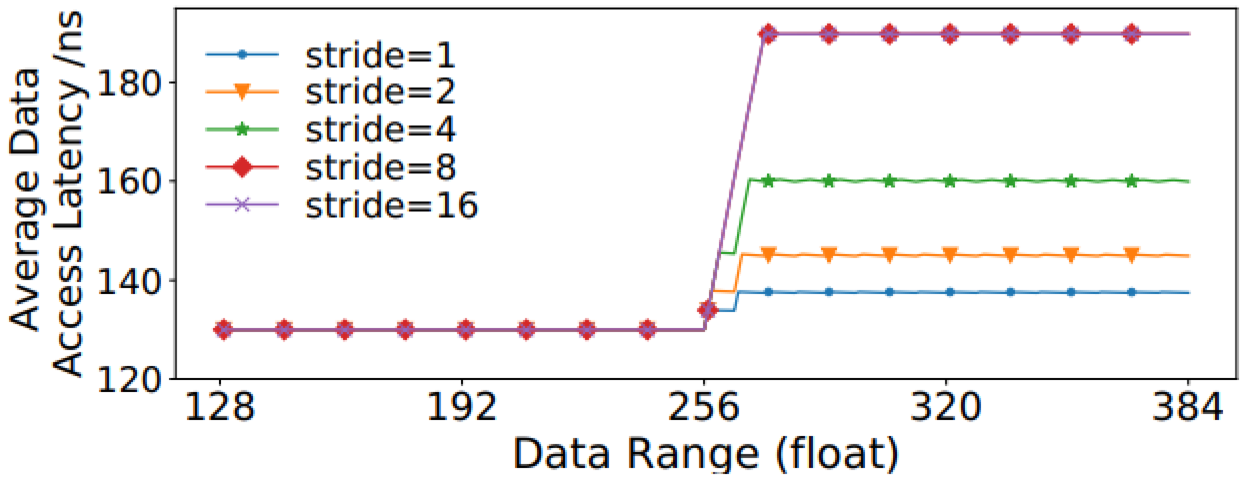
Detect memory hierarchy

Pointer-chase method.

```
/*Array initialization. Buffer type is needed in the CPU side.*/
int* idx_buf = mapImageToBuffer(src_image);
for (size_t i = 0; i < dataRange; i++)
  idx_buf[i] = (i + stride) % dataRange;
src_img = unmapImage(idx_buf);

/* Work group size (1, 1, 1), one work group */
__kernel void image_cache(__read_only image1d_t src,__global int* dst)
{ int idx = 0;
  for (int i = 0; i < N; ++i)
    idx = read_imagei(src, SAMPLER, idx).x;
  *dst = idx; }
```

Key1 - ArchProbe

Detect memory hierarchy

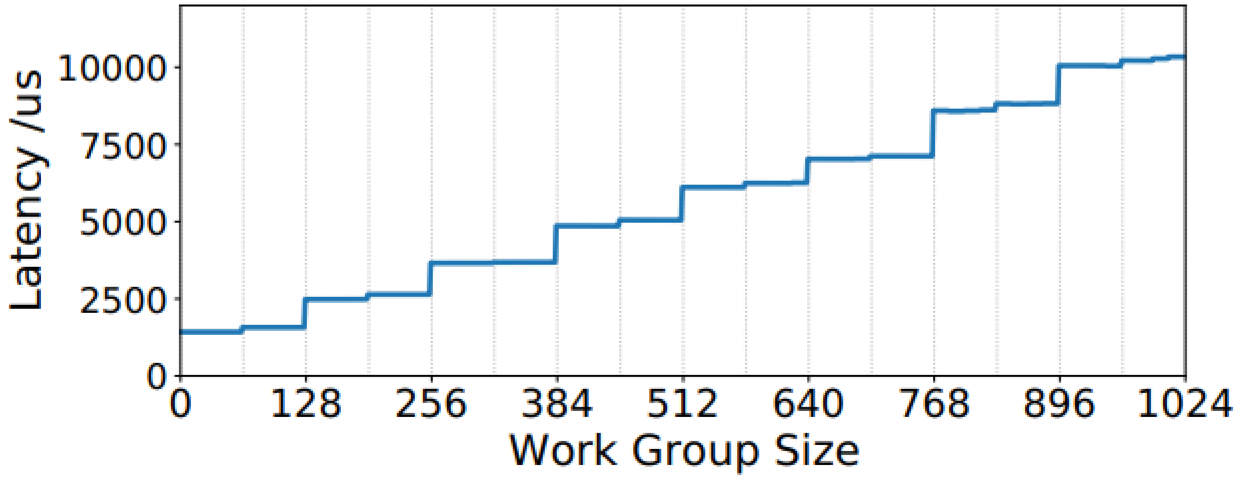


256x4B L1 texture cache, 32B cacheline size.
Bandwidth can be calculated.

Key1 - ArchProbe

Detect warp size

Run enough work groups to potentially saturate all the ALUs, and then gradually increase the work items in each work group.



The warp size is 64 or 128.

Key1 - ArchProbe

Detect the number of ALUs

```
/*Pick group size (64,6,2) as an example, one work group*/
__kernel void warp_size (__global int* output) {
  __local int local_counter;
  local_counter = 0;
  barrier(CLK_LOCAL_MEM_FENCE); /*sync all work items*/
  int i = atomic_inc(&local_counter);
  output[globalID0 + globalID1*globalSize0 +
         globalID2*globalSize0*globalSize1] = i;}
```

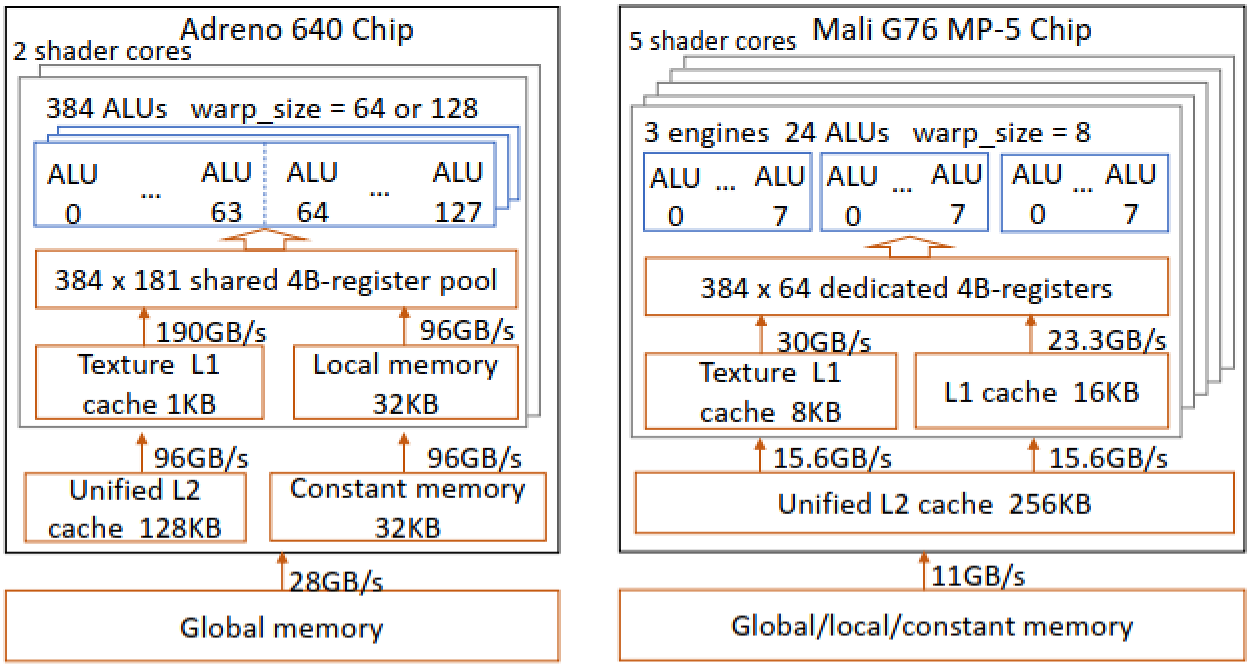
Key1 - ArchProbe

Detect the number of ALUs

Warp0	ItemID	(0,0,0)	(1,0,0)	...	(63,0,0)	(0,1,0)	...	(63,1,0)
	Output	0	4	...	192	195	...	388
Warp1	ItemID	(0,2,0)	(1,2,0)	...	(63,2,0)	(0,3,0)	...	(63,3,0)
	Output	1	3	...	188	191	...	378
Warp2	ItemID	(0,4,0)	(1,4,0)	...	(63,4,0)	(0,5,0)	...	(63,5,0)
	Output	2	5	...	190	193	...	380

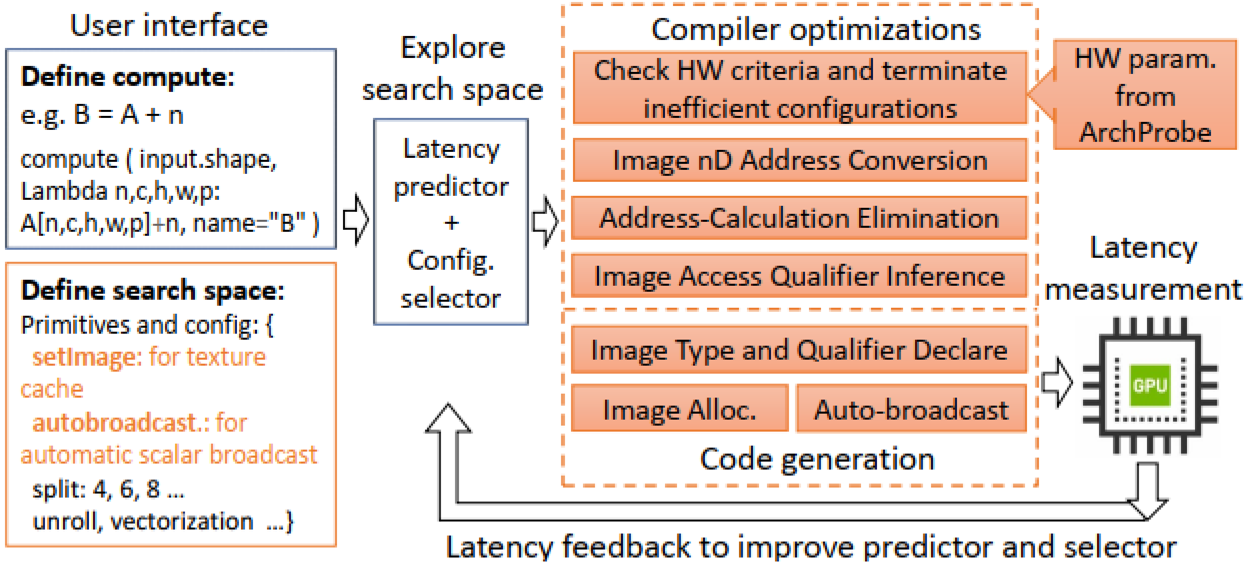
The number of ALUs is 384.

Key1 - ArchProbe



Key2 - Romou

New primitives for supporting mobile features



Key2 - Romou

End-to-end kernel generation

```
# Operator developer interface
def addConstant(cfg, input:Tensor, n:int)->Tensor:
  B = compute(input.shape, lambda n,c,h,w,p:A[n,c,h,w,p]+n, name="B")
  ↪  # p is packing, p=4 for image type
  cfg.define_knob('set_input_image',[True,False])
  cfg.define_knob('autobroadcast',[True,False])
  ... # Define other configurations
  return B

def scheduleAddConstant(cfg, B:Tensor):
  input, = B.op.input_tensors
  s = create_scheule(B.op)
  if cfg['set_input_image'].val == True: s[B].setImage(input)
  if cfg['autobroadcast'].val == True: s[B].autobroadcast(B)
  ... # Define Other primitives
  return s

# Generated IR
# For each element in B
B[@ir._2d_coord(linearIndex,@ir.imageWidth(B),dtype=int32)]
  = ((imgwfloat32*)A[@ir._2d_coord(linearIndex,
  @ir.imageWidth(A),dtype=int32)]+n)}}

# Generated OpenCL kernel
cl_image_format fmt={CL_RGBA, CL_HALF_FLOAT};
# w*c/p is image width, h is image height
cl_image_desc desc={CL_MEM_OBJECT_IMAGE2D, w*c/p, h};
cl_mem A=clCreateImage(context, CL_MEM_READ, &fmt, &desc);
cl_mem B=clCreateImage(context, CL_MEM_WRITE, &fmt, &desc);
__constant sampler_t sampler = TEXTURE_CONFIG;

__kernel void addConstant(__read_only image2d_t A,
                          __write_only image2d_t B) {
  write_imagef(B,(int2)(linearIndex%(p*get_image_width(B))/p,
    linearIndex/(p*get_image_width(B))), read_imagef(A,
    sampler, (int2)(linearIndex%(p*get_image_width(A))/p,
    linearIndex/(p*get_image_width(A)))) + n);}
```

Key2 - Romou

Address calculation elimination

Algorithm 1 Common address calculation elimination

```
Input: AST (Abstract Syntax Tree) of a kernel
Output: AST with common address calculation eliminated
function REWRITECOMMSUBEXPR(node)
    reversely add subExpr in node.addrExpr to exprList
    for subExpr ∈ exprList do
        if exprVarMap[subExpr] then
            replace(node.addrExpr,exprVarMap[subExpr])
            return
        end if
    end for
    exprVarMap.insert(node.addrExpr, newVar)
    replace(node.addrExpr,newVar)
end function
function TRAVERSEAST(node)
    if node.type==forNode then
        enterForNode ← enterForNode+1
        TRAVERSEAST(node)
        enterForNode ← enterForNode-1
        for subExpr ∈ exprVarMap do
            ▷ exprVarMap is the <expression,variable> map.
            add the declaration node for exprVarMap[subExpr]
            exprVarMap.delete(subExpr)
        end for
    else if enterForNode and (node.type==loadNode or storeNode) then
        REWRITECOMMSUBEXPR(node)
    end if
    TRAVERSEAST(node.next)
end function
```

```
for (int i; i < coarsening_size - 1; i++) {
  write_imagef(B, int2((linearIndex+i*2*p)%
    (p*get_image_width(B))/p, (linearIndex+i*2*p)/
    (p*get_image_width(B))), B_local+i*2);
  write_imagef(B, int2((linearIndex+(i*2+1)*p)%
    (p*get_image_width(B))/p, (linearIndex+(i*2+1)*p)/
    (p*get_image_width(B))), B_local+(i*2+1)); };
```

```
const int comm1=linearIndex/(p*get_image_width(B));
const int comm2=linearIndex%(p*get_image_width(B))/p;
for (int i; i < coarsening_size - 1; i++) {
  write_imagef(B, int2(comm2+i*2, comm1),B_local+i*2);
  write_imagef(B, int2(comm2+i*2+1, comm1),B_local+i*2+1);}
```

Key2 - Romou

Hardware-aware search space pruning

HW feature	kernel exclusion criteria
L1 cache	Access more data in one loop iteration than L1 cache
register	Overuse available registers for the work group size
buffer	Use buffer type when L1 cache is for texture only
local memory	Use local memory when work group size $> \alpha \cdot$ ALUs
warp	Work group size $<$ warp size
data access width	Use inefficient data access width

Experiments

Setup

Mobile GPU	Adreno 630	Adreno 640	Mali G76
Phone	Google Pixel 3XL	Google Pixel 4XL	Vivo X30
SoC	Snapdragon 845	Snapdragon 855	Exynos 980
the number of cores	2	2	5
the number of total ALUs	512	768	120
Frequency	710 MHz	585 MHz	800 MHz
Computation bandwidth	720 GFLOPS	890 GFLOPS	190 GFLOPS
DRAM bandwidth	26 GB/s	28 GB/s	11 GB/s

Experiments

Speedup over generated kernels and hand-optimized kernels

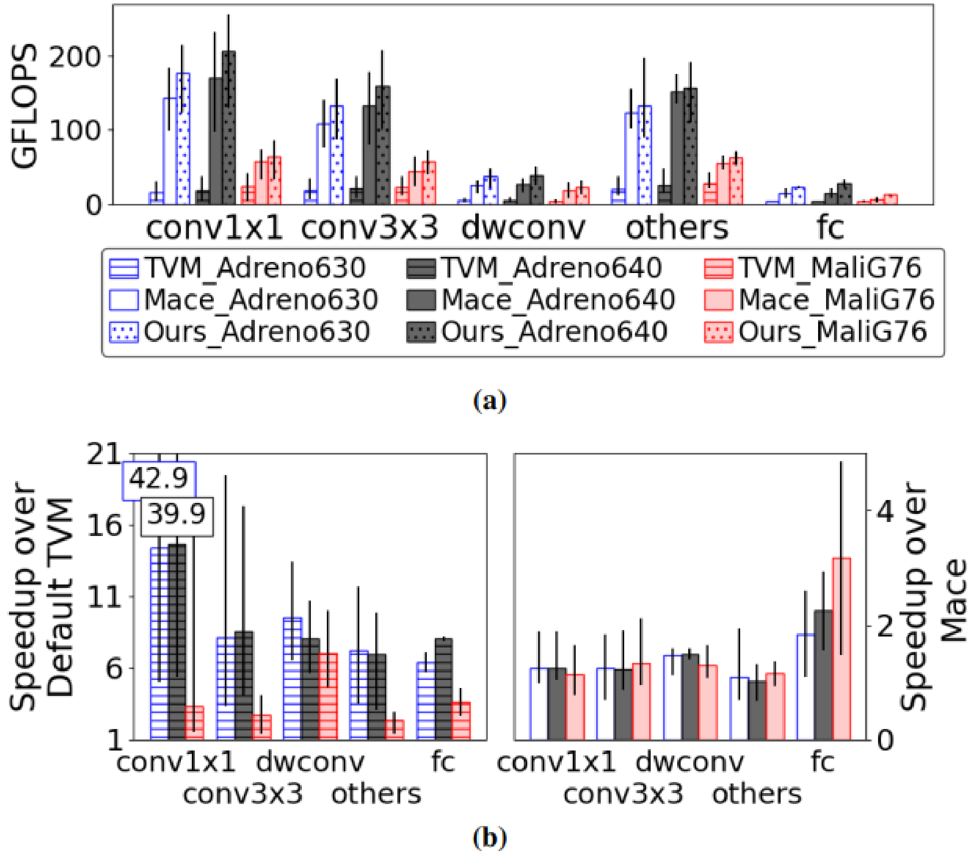


Figure 13: The average, max, and min (a) performance and (b) corresponding speedup for all the evaluated operators by Romou compared to TVM and Mace.

Experiments

Speedup for DNN models

Figure 14: Operator performance comparison in the order of operator execution (x axis) for (a) MobileNetV1, (b) SqueezeNetV1.0, (c) ResNet-18, and (d) InceptionV3.

Experiments

Speedup for DNN models

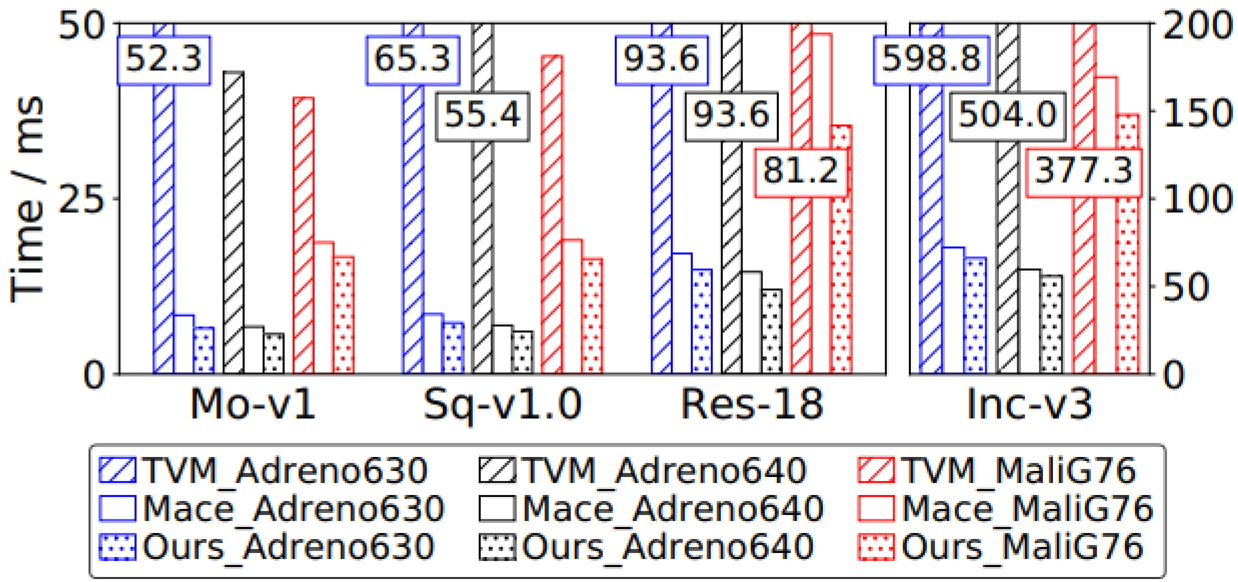


Figure 15: The sum of operator latencies for each model. Romou achieves up-to 9× speedup and 37% improvement compared to TVM and Mace respectively (text box marks TVM time).

Experiments

Speedup for DNN models

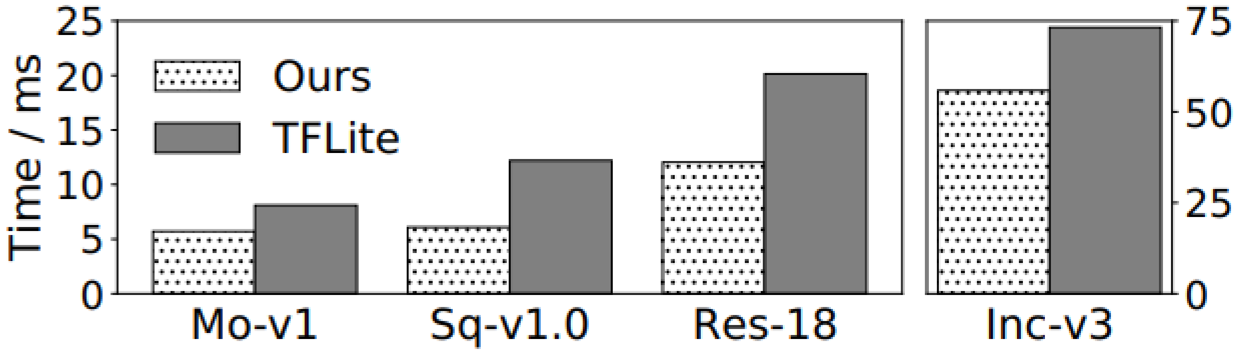


Figure 16: Latency sum of all the operators for each model on Adreno640. Romou achieves up-to 2× speedup compared to TFLite mobile GPU backend.

Experiments

Speedup breakdown

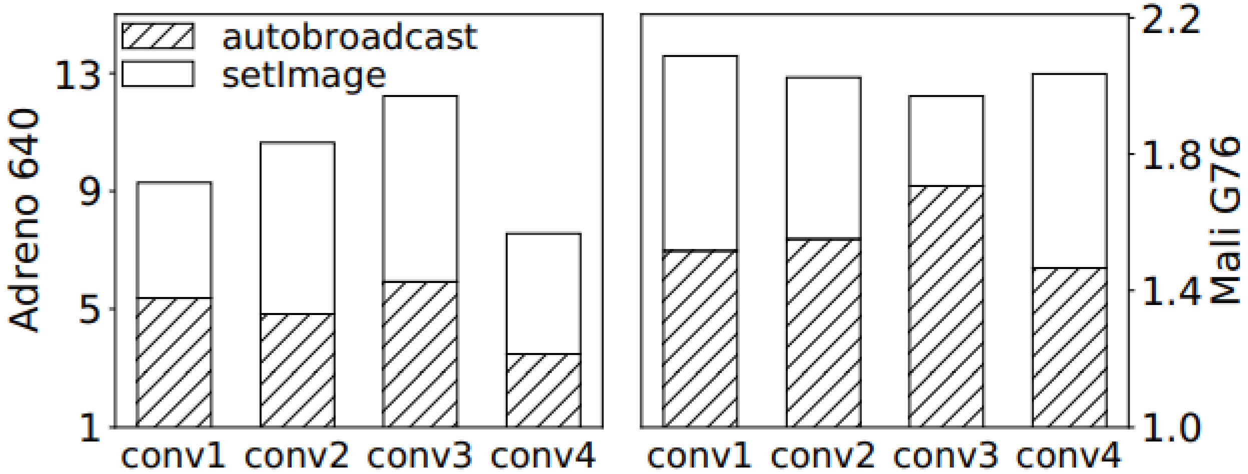


Figure 17: Speedup breakdown for each primitive of Romou compared to TVM for four example 1×1 convolution. The size [H,W,Cin,Cout] for each convolution: [64,64,512,256], [18,18,128,768], [56,56,128,128], and [28,28,256,256].

Experiments

Searching cost

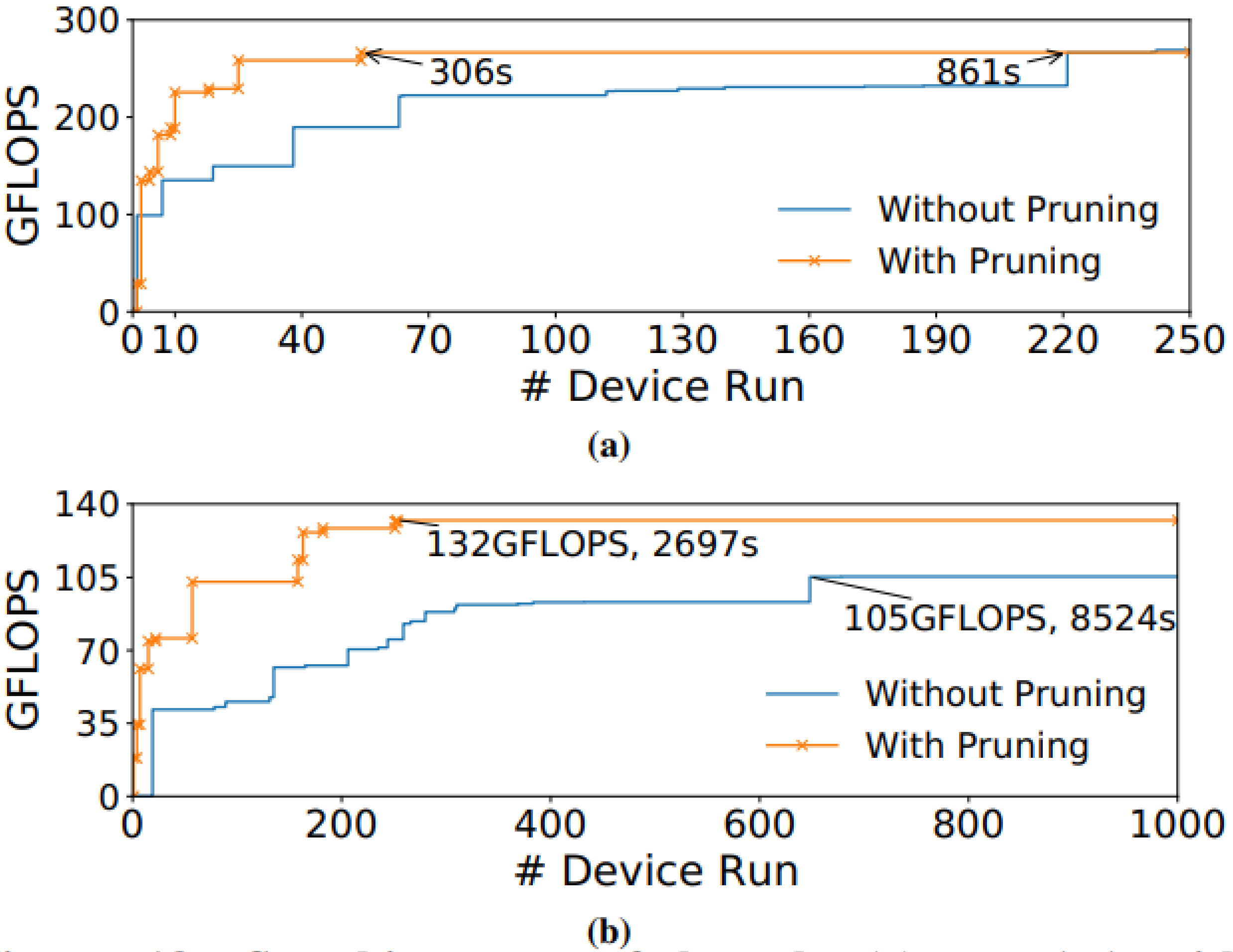


Figure 18: Searching cost of kernels (a) conv1x1 with [H,W,Cout,Cin] = [28,28,256,256]; and (b) conv3x3 with stride=2 [H,W,Cout,Cin] = [36,36,384,288].

Conclusion

Advantages:

- Fine-grained demystify of mobile GPU and probing technology: ArchProbe.
- Fully utilize the hardware information to optimize the model compiling and introduce new feature into TVM to improve its performance on mobile devices.

Disadvantages:

- What will happen if increase the number of iterations of scheduling in TVM?
- Dependent of realistic profiling in model compiling.
- Long time searching will arouse the thermal throttling on mobile devices and made profiling become inaccurate.

Thank You !

Dec 18, 2022

Presented by Mengyang Liu