BLIS Retreat 2022

Automatic Generation of a Family of Matrix Multiplication Routines with Apache TVM

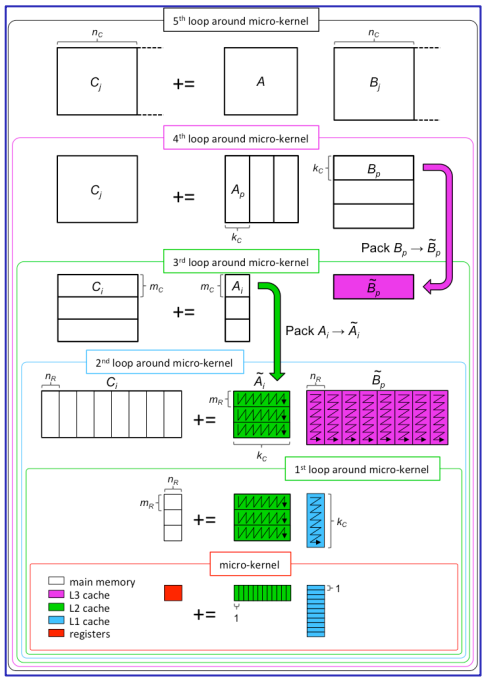
Guillermo Alaejos, Adrián Castelló, Pedro Alonso-Jordá,

Francisco D. Igual, Enrique S. Quintana-Ortí

September 22nd and 23rd, 2022

Previously on BLIS Retreat 2021…

- Motivation
  - Could we implement BLIS GEMM with Apache TVM?
- Conclusion
  - YES!! Proof of concept of Apache TVM for BLIS




```
for (jc:  0, 2){
  for (pc:  0, 2)
    for (bn:  0, 64) "parallel"
      for (bz:  0, KC)
        Br[ramp(((bn*2688) + (bz*8)), 1, 8)] =
                          B_1[ramp(((((pc*344064)
                          + (bz*1024)) + (jc*NC)) + (bn*8)),
1, 8)]
    for (ic:  0, 2)
      for (an:  0, 224) "parallel"
        for (az:  0, KC)
          Ar[ramp(((an*2688) + (az*8)), 1, 8)] =
                A_1[ramp(((((ic*1204224)
                + (an*5376)) + (pc*KC)) + az), 672, 8)]

      for (jr:  0, 64) "parallel"
        for (ir:  0, 224)
          for (k:  0, KC)
            MICROKERNEL
```

```
.LBB1_5:
        add     x9, x22, x8, lsl #5
        ldr     q16, [x21, x8, lsl #4]
        ldp     q18, q17, [x9]
        add     x8, x8, #1
        cmp     x8, #336
        fmla    v7.4s, v16.4s, v18.s[0]
        fmla    v6.4s, v16.4s, v18.s[1]
        fmla    v5.4s, v16.4s, v18.s[2]
        fmla    v4.4s, v16.4s, v18.s[3]
        fmla    v3.4s, v16.4s, v17.s[0]
        fmla    v2.4s, v16.4s, v17.s[1]
        fmla    v1.4s, v16.4s, v17.s[2]
        fmla    v0.4s, v16.4s, v17.s[3]
        b.ne    .LBB1_5
```

Motivation

- Explore all the Apache TVM possibilities:
  - to generate GEMM algorithms
  - to implement BLIS microkernels
  - to improved portability
  - to reduced development-efforts
  - to perform close to manual code

MAKE IT REAL!!

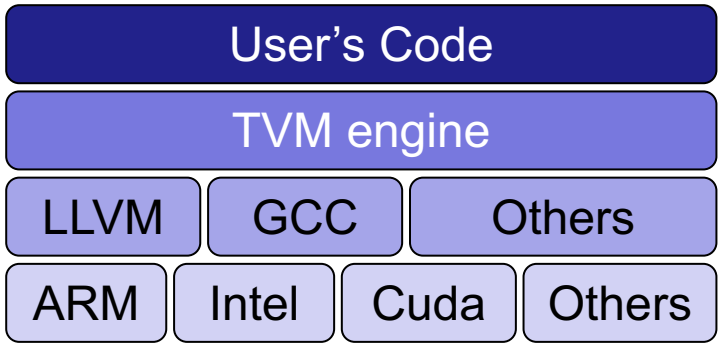
Motivation

- Which is the app target for this work?
  - Convolutional neural networks (CNNs)
- Why?
  - Convolution operator take up to 90-95% of the execution time
  - Four major approaches for the convolution:
    - Lowering: im2col/im2row + large **GEMM** (matrix multiplication)
    - Direct convolution: blocked as a collection of small **GEMMs**
    - Winograd: 1/3 of operations correspond to **GEMM**
    - FFT

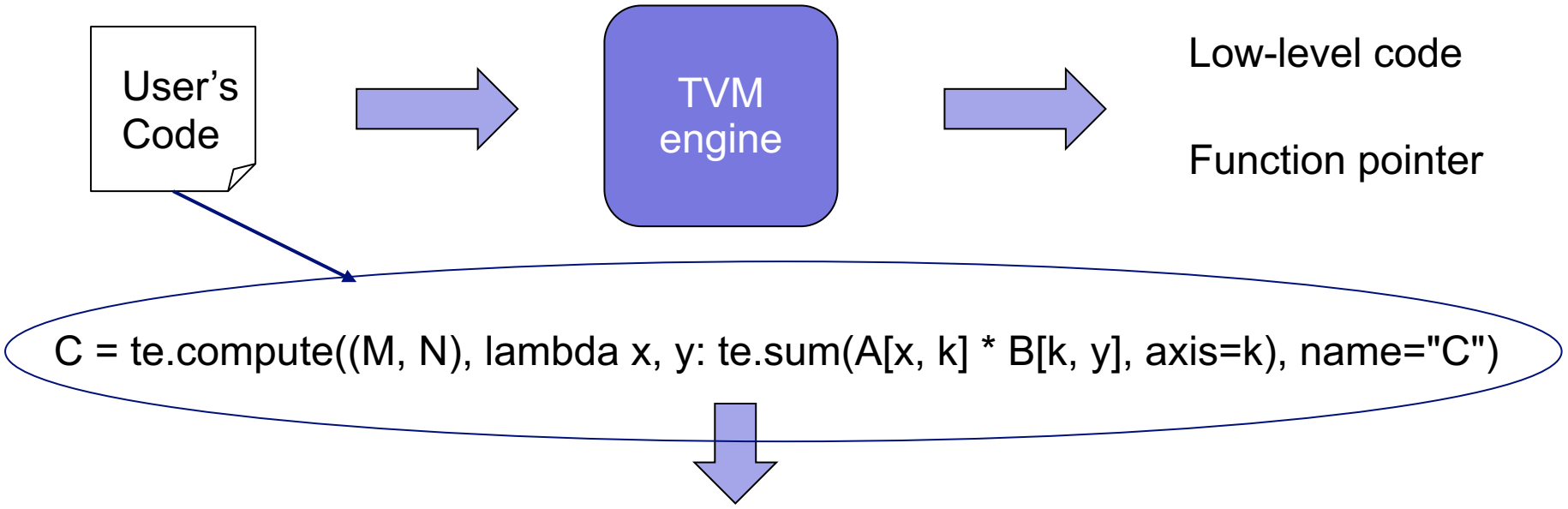
GEMM dependent apps!

Apache TVM

- Open-source machine learning compiler framework
- Generates code for multiple backends
  - CPUs, GPUs, ML accelerators
- Python API interface
- User-guided, low-level optimizations

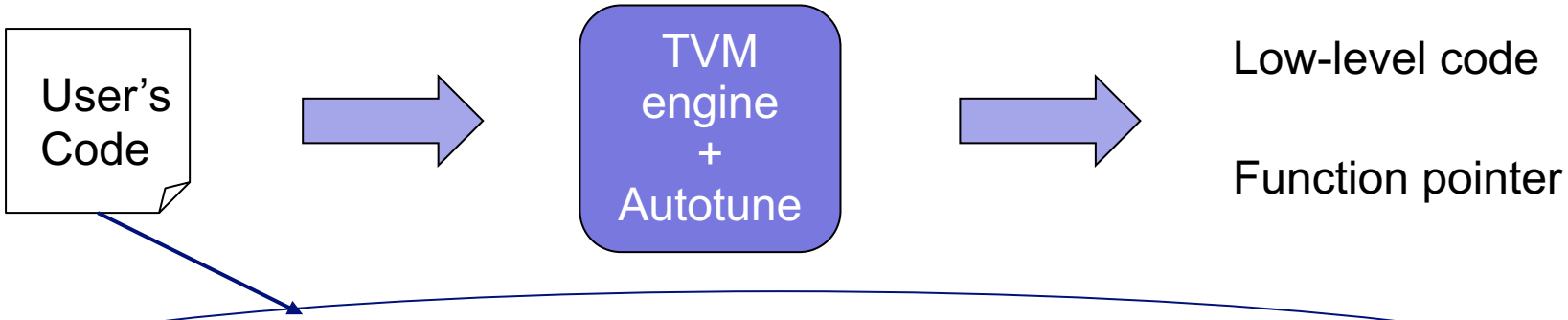


Code generation with Apache TVM




```
primfn(A: handle, B: handle, C: handle) -> ()
  attr = {"global_symbol": "main", "tir.noalias": True}
  buffers = {: Buffer(C_1: Pointer(float32), float32, [1024, 1024], []),
             : Buffer(B_1: Pointer(float32), float32, [1024, 1024], []),
             : Buffer(A_1: Pointer(float32), float32, [1024, 1024], [])}
  buffer_map = {A: , B: , C: } {
  attr [C_2: Pointer(float32)] "storage_scope" = "global";
  allocate(C_2, float32, [1048576]);
  for (x: int32, 0, 1024) {
    for (y: int32, 0, 1024) {
      C_2[((x*1024) + y)] = 0f32
      for (k: int32, 0, 1024) {
        C_2[((x*1024) + y)] = ((float32*)C_2[((x*1024) + y)] + ((float32*)A_1[((x*1024) + k)]*(float32*)B_1[((k*1024) + y)]))
      }
    }
  }
}
```

Code generation with Apache TVM



C = te.compute((M, N), lambda x, y: te.sum(A[x, k] * B[k, y], axis=k), name="C")

```
allocate(C.local: Pointer(local float32), float32, [65536]), storage_scope = local;
for (y.outer: int32, 0, 16) {
  for (x.c.outer.outer.inner: int32, 0, 4) {
    for (x.c.outer.inner.init: int32, 0, 256) {
      C.local[ramp(((x.c.outer.outer.inner*16384) + (x.c.outer.inner.init*64)), 1, 64)] = broadcast(0f32, 64)
    }
    for (k.outer: int32, 0, 32) {
      for (x.c.outer.inner: int32, 0, 256) {
        C.local[ramp(((x.c.outer.outer.inner*16384) + (x.c.outer.inner*64)), 1, 64)] +=
          (broadcast((float32*)A_1[(((x.c.outer.outer.inner*262144) + (x.c.outer.inner*1024)) + (k.outer*32))], 64)
          * B_1[ramp(((k.outer*32768) + (y.outer*64)), 1, 64)]))
        ... #unroll x 32
        C.local[ramp(((x.c.outer.outer.inner*16384) + (x.c.outer.inner*64)), 1, 64)] +=
          (broadcast((float32*)A_1[(((x.c.outer.outer.inner*262144) + (x.c.outer.inner*1024)) + (k.outer*32)) + 31)], 64)
          * B_1[ramp((((k.outer*32768) + (y.outer*64)) + 31744), 1, 64)]))
      }
    }
  }
  for (x.inner: int32, 0, 1024) {
    for (y.inner: int32, 0, 64) {
      C_1[(((x.inner*1024) + (y.outer*64)) + y.inner)] = (float32*)C.local[((x.inner*64) + y.inner)]
    }
  }
}
```

Code generation with Apache TVM

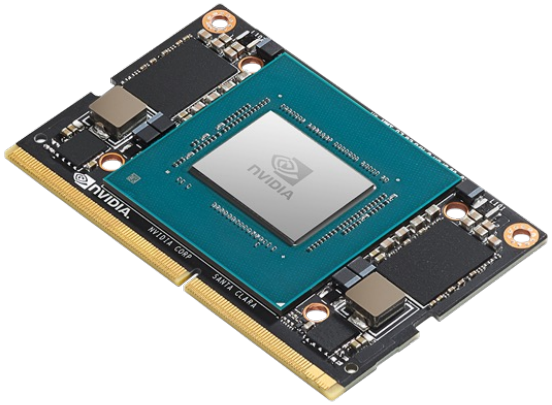
- Why not use Auto TVM?
  - Usually 1,000 configurations (20 to 60 minutes/config)
  - Non-deterministic search algorithm
    - Distinct number of splits with distinct factors for GEMMs

- Instead
  - Automatic generation (TVM) + **Experience**

K. Goto and R. A. van de Geijn. 2008. Anatomy of high-performance matrix multiplication. ACM Trans. Math. Softw.

Experimental results

- Hardware:
  - NVIDIA Jetson-Xavier (8 ARM Carmel cores)
  - Intel Ice Lake (Intel Xeon Platinum 8358)
- Software:
  - Apache TVM 0.8
  - BLIS 0.8.1
  - OpenBLAS 0.3.19
  - ARMPL 21.1
  - Intel MKL 2021.4.0
- Application
  - Square matrices
  - ResNet50 v1.5 with ImageNet (BS=128)

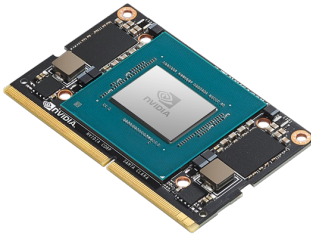


ResNet50 v1.5 + ImageNet

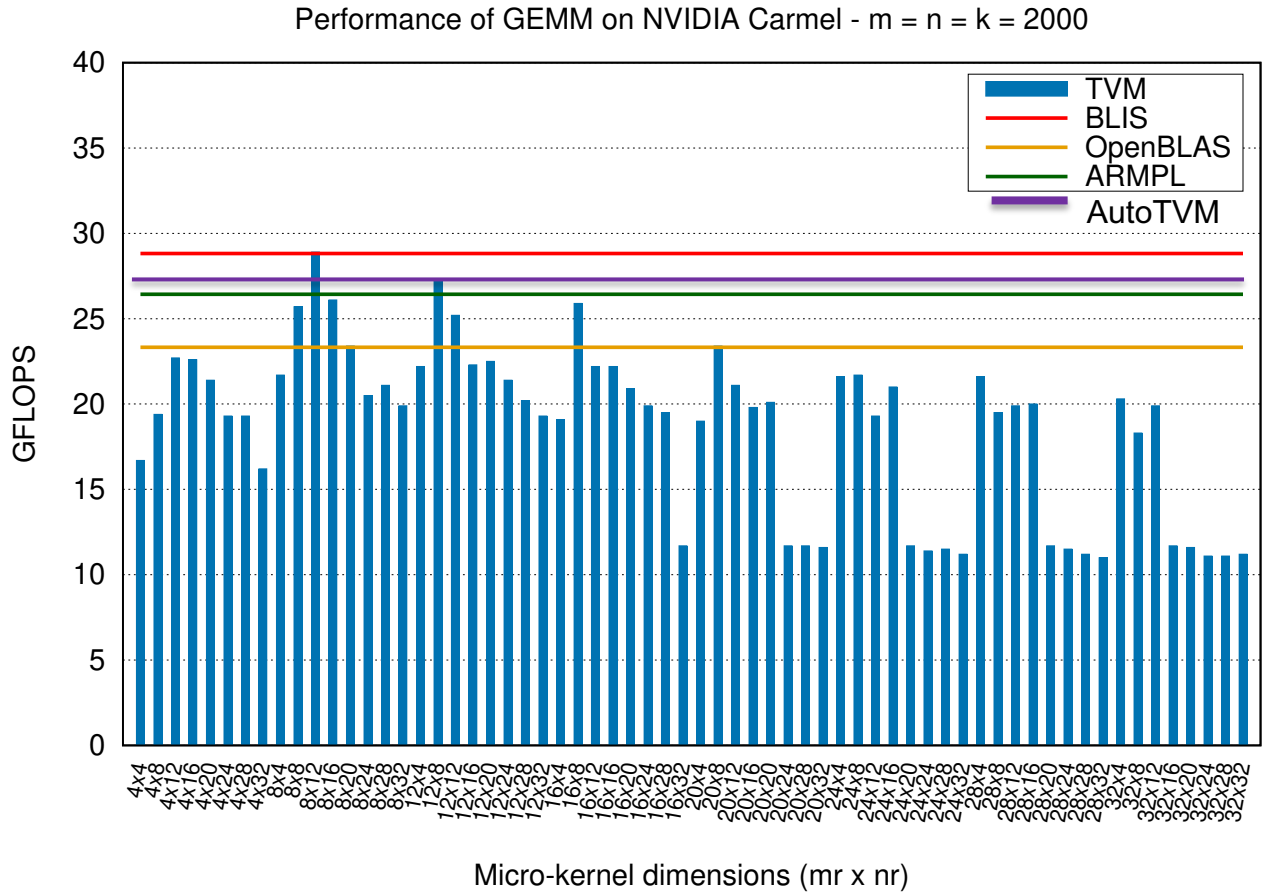
ID	Layers	m	n	k
1	001	1605632	64	147
2	006	401408	64	64
3	009/021/031	401408	64	576
4	012/014/024/034	401408	256	64
5	018/028	401408	64	256
6	038	401408	128	256
7	041/053/063/073	100352	128	1152
8	044/056/066/076	100352	512	128
9	046	100352	512	256
10	050/060/070	100352	128	512
11	080	100352	256	512
12	083/095/105/115/125/135	25088	256	2304
13	086/098/108/118/128/138	25088	1024	256
14	088	25088	1024	512
15	092/102/112/122/132	25088	256	1024
16	142	25088	512	1024
17	145/157/167	6272	512	4608
18	148/160/170	6272	2048	512
19	150	6272	2048	1024
20	154/164	6272	512	2048

Experimental results

- Square matrices (m = n = k = 2000)



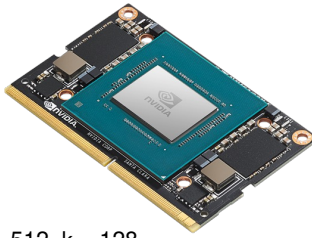
target="llvm –mcpu=arm_cpu"

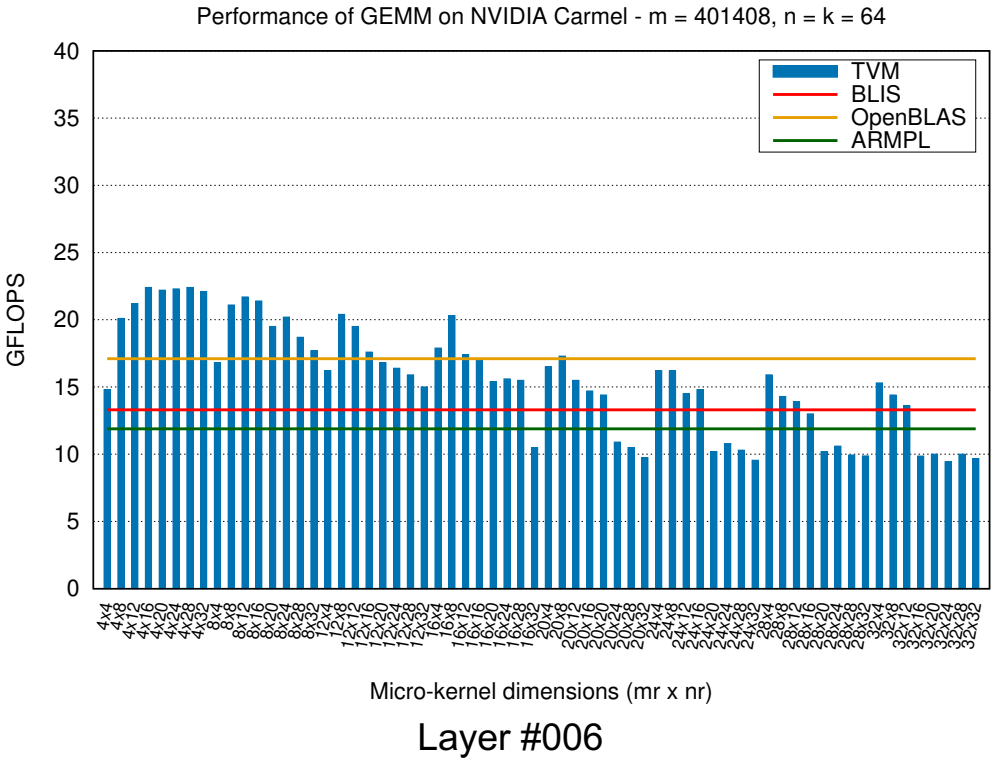


Best TVM ukernel 8x12 (as BLIS)

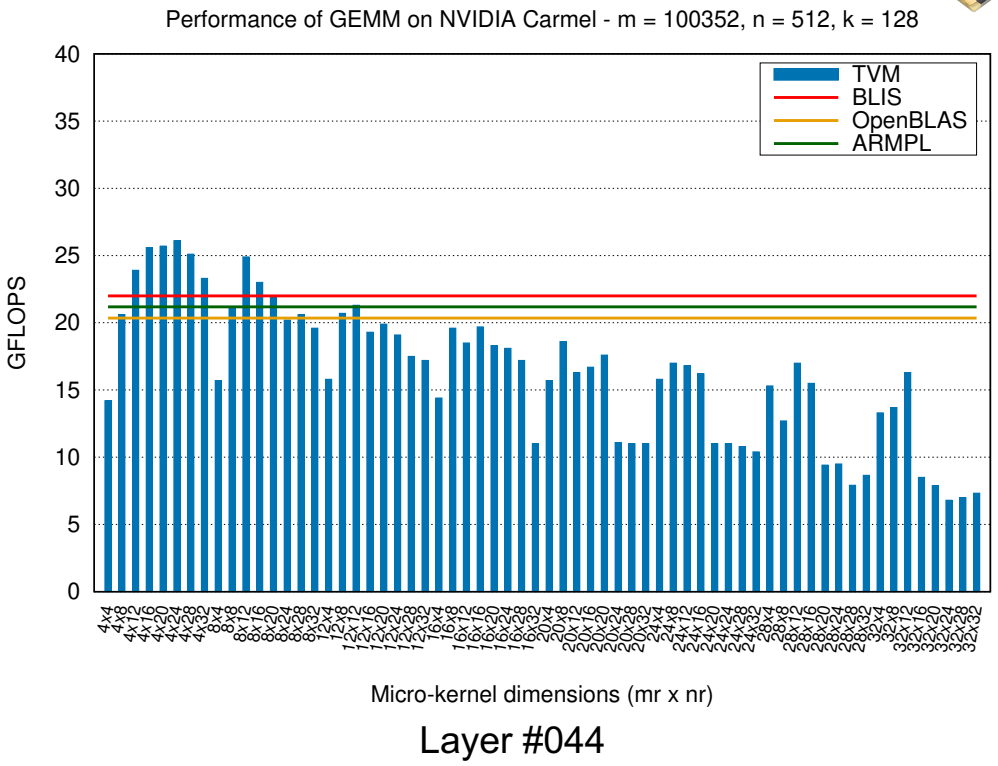
Experimental results

- 2 Convolutional layers





Layer #006



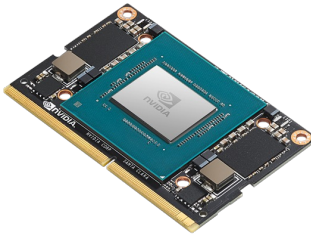
Layer #044

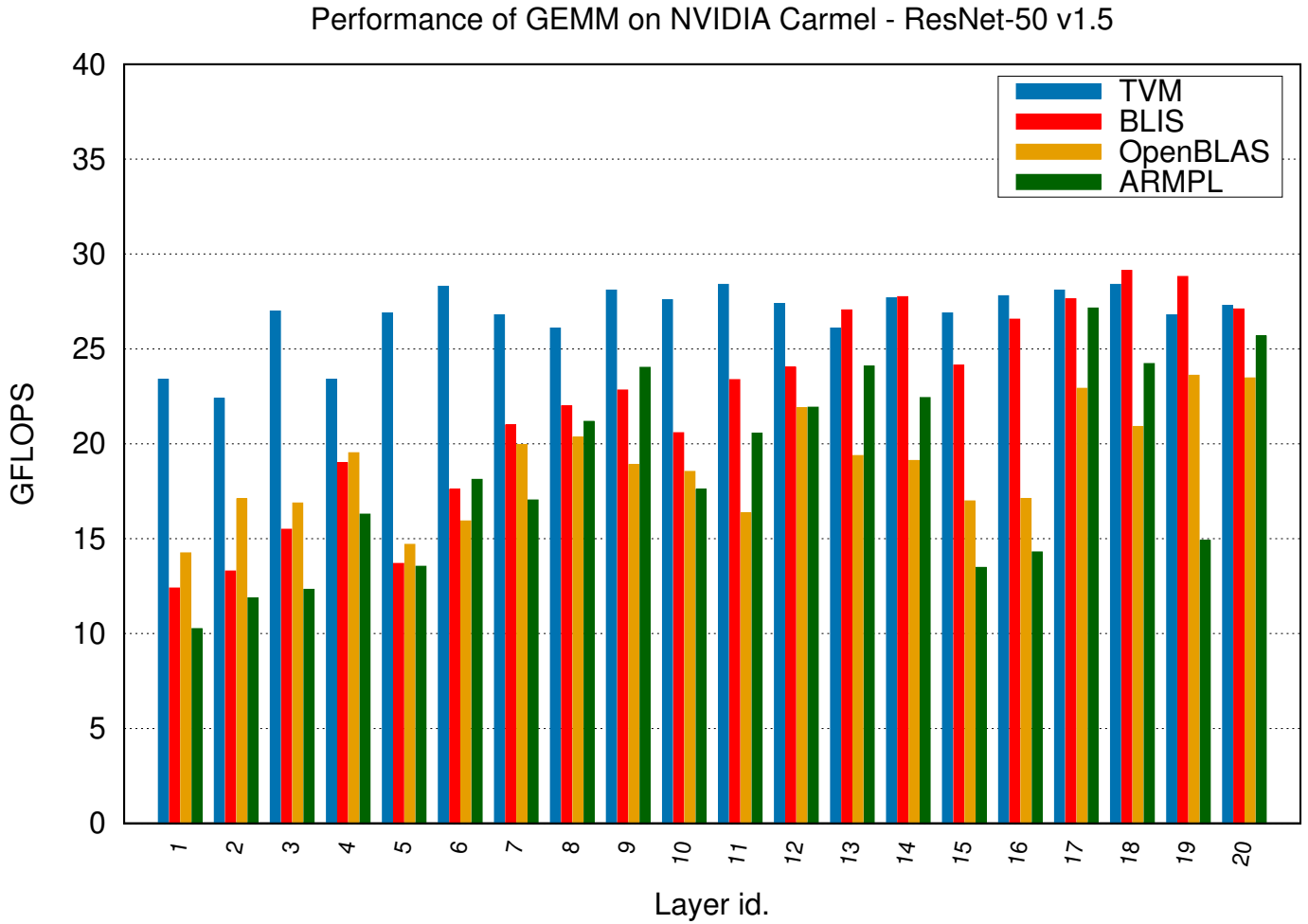
Best TVM ukernel 4x16 (BLIS 8x12)

Best TVM ukernel 4x24 (BLIS 8x12)

Experimental results

- 20 different convolutional layers of Resnet50 v1.5

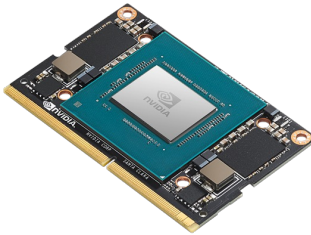


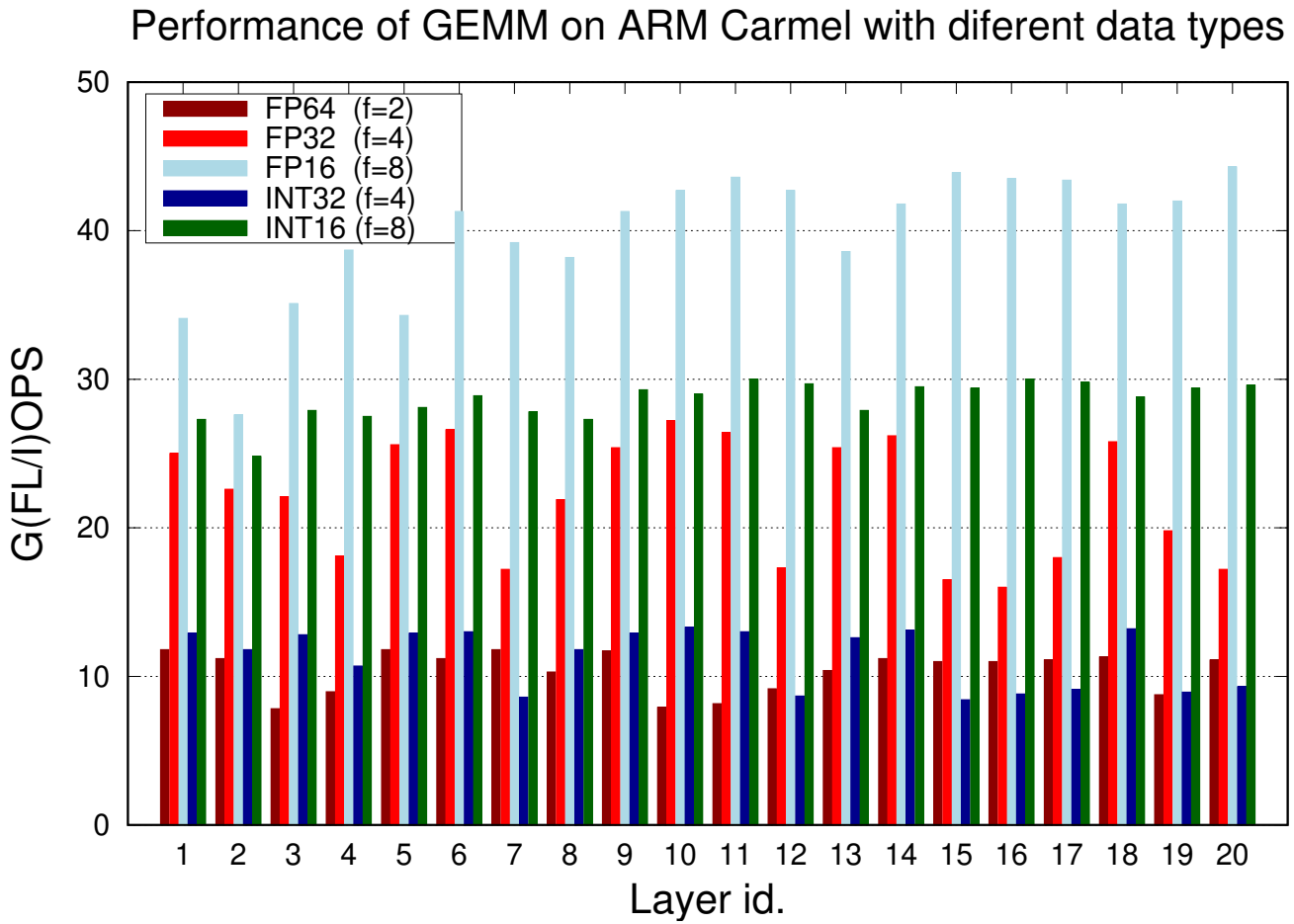


Best TVM ukernel for each layer!

Experimental results

- 20 different convolutional layers of Resnet50 v1.5

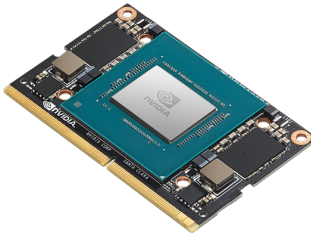


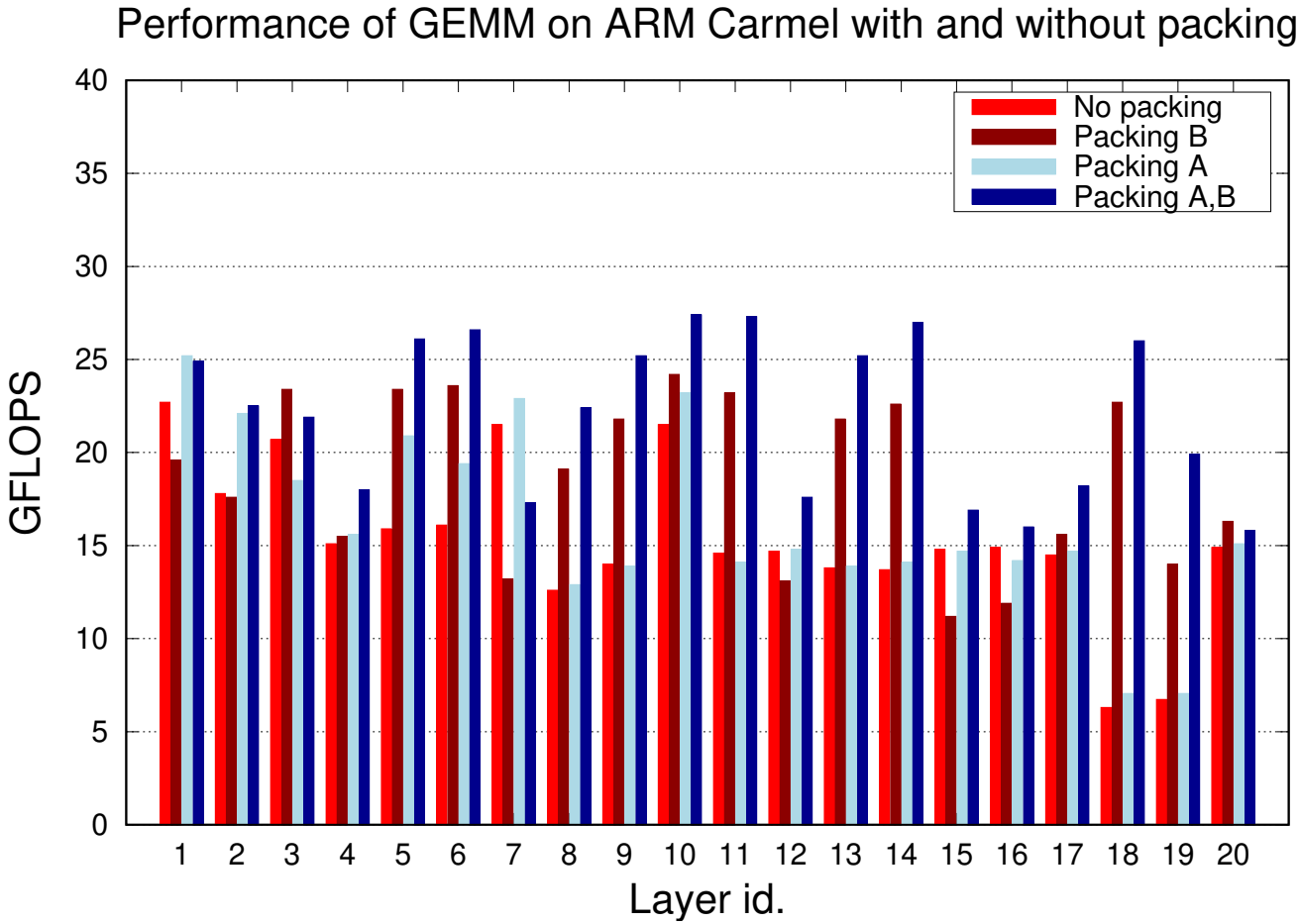


dtype = datatype
xo , xi = sched[xc].split(factor = f)

Experimental results

- 20 different convolutional layers of Resnet50 v1.5






```
sched[Xc] = compute_at(sched[C], LX)
```

GEMM code generation (algorithms)

- BLIS base algorithm (B3A2C0)
  - B → L3, A → L2, B → L1, C → registers

- Variants
  - A3B2C0 (swap A and B roles)
  - B3C2A0
  - C3B2A0
  - A3C2B0
  - C3A2B0

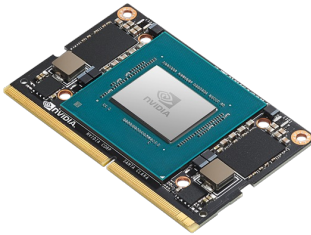
A in registers

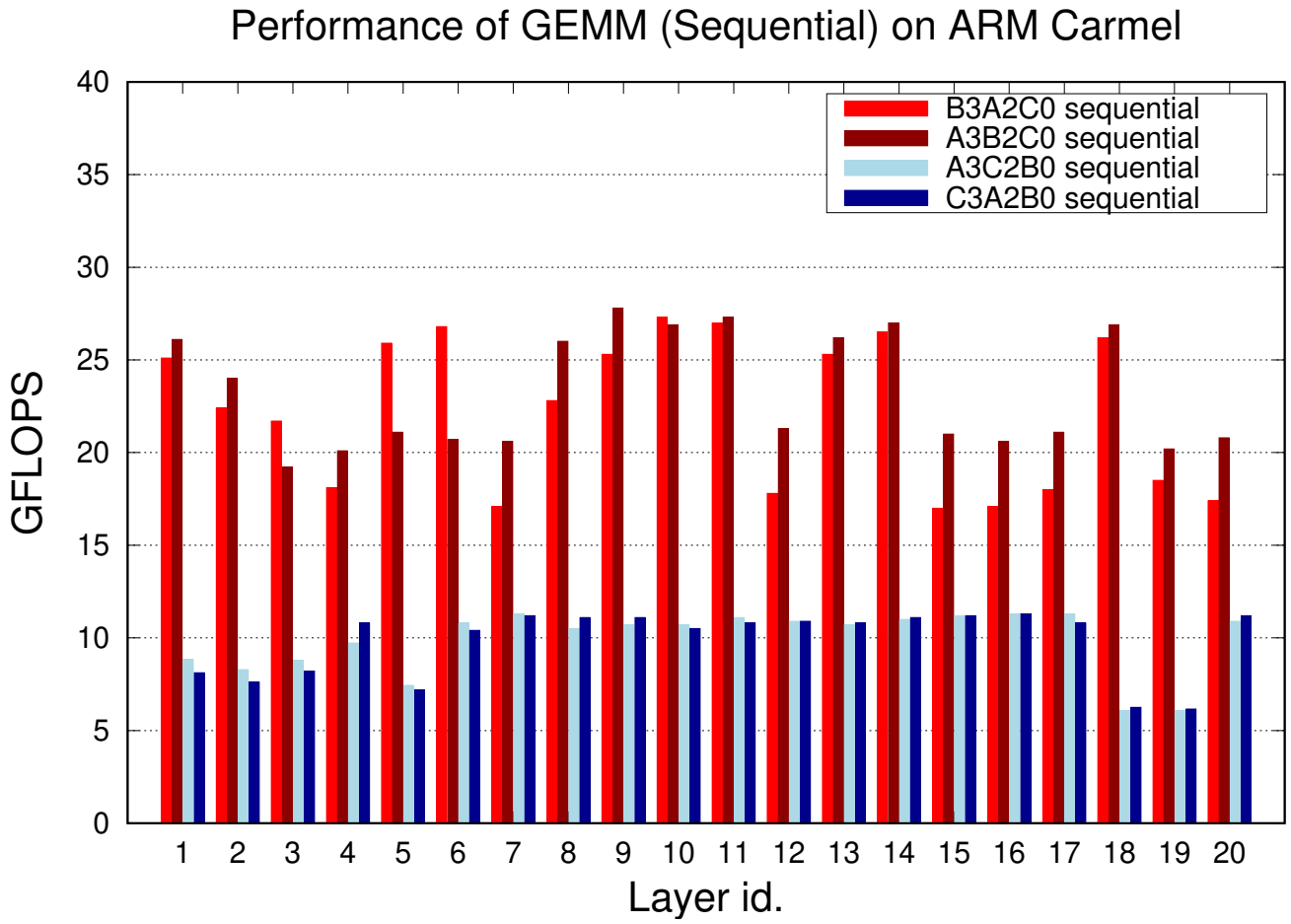
B in registers

New microkernel with A or B resident

Experimental results

- 20 different convolutional layers of Resnet50 v1.5

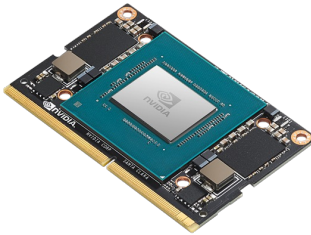


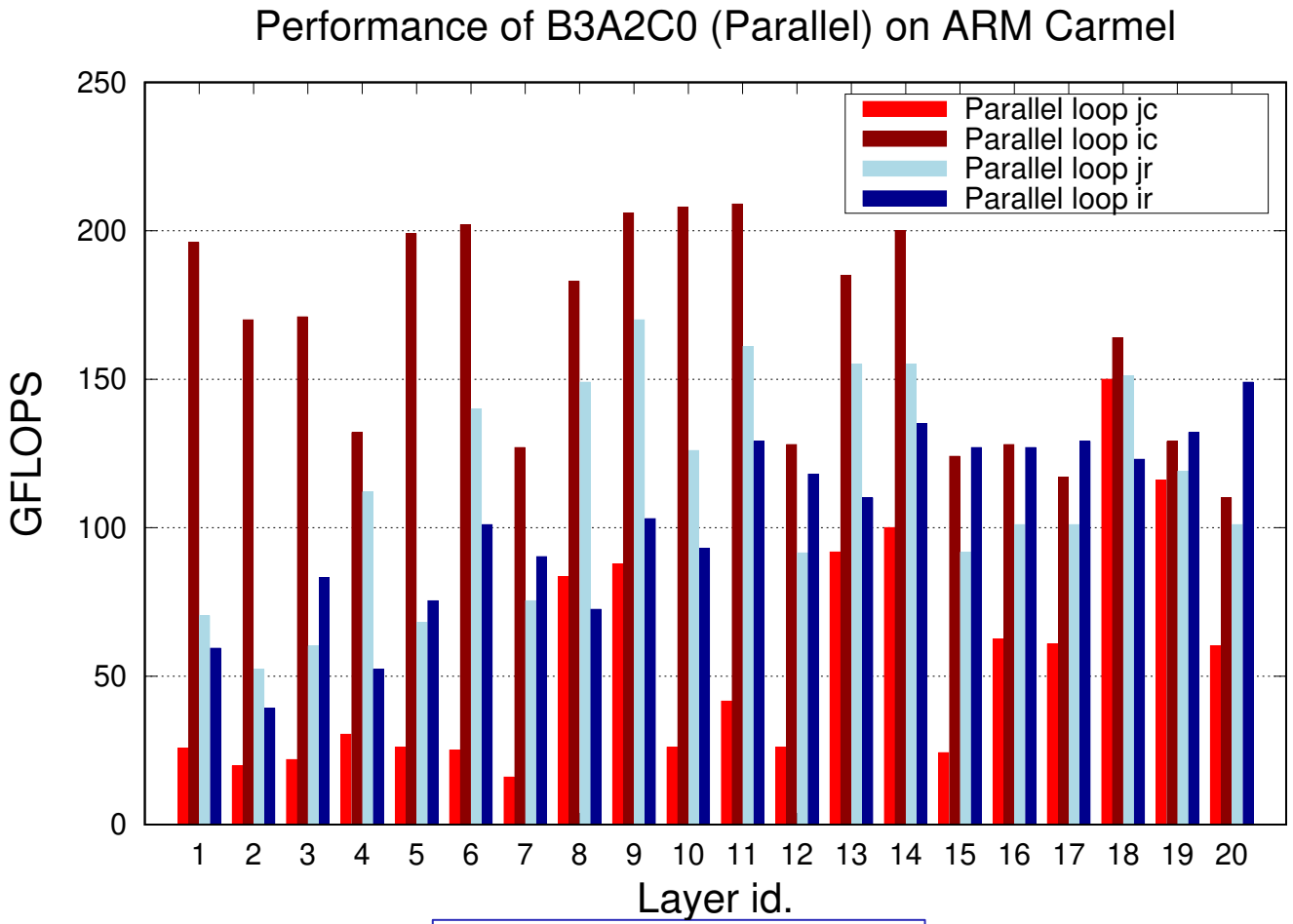


sched[C].reorder (L1, L2, L3, L4, L5, L6, L7)

Experimental results

- 20 different convolutional layers of Resnet50 v1.5

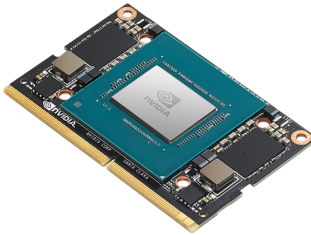


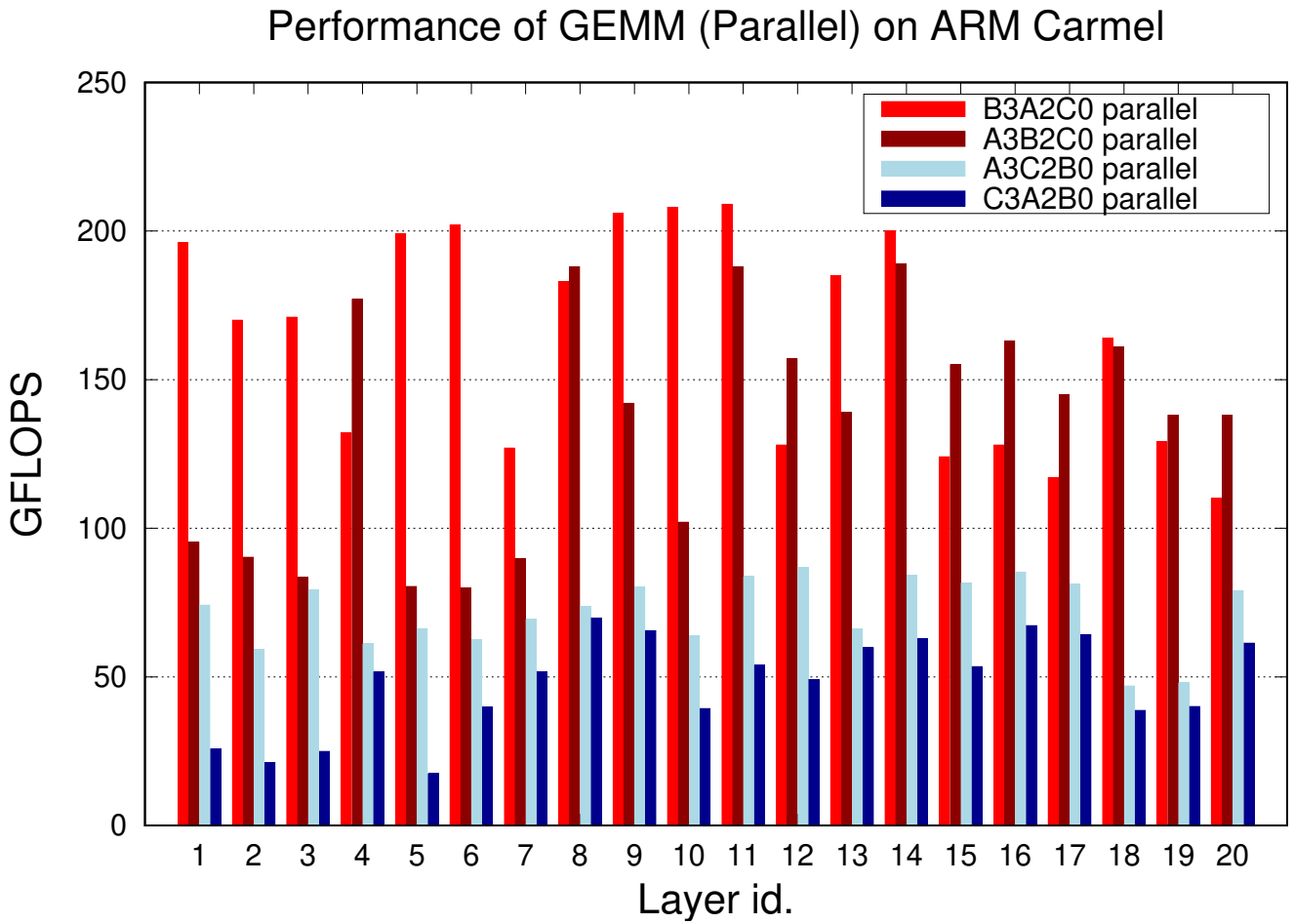


sched[C].parallel (LX)

Experimental results

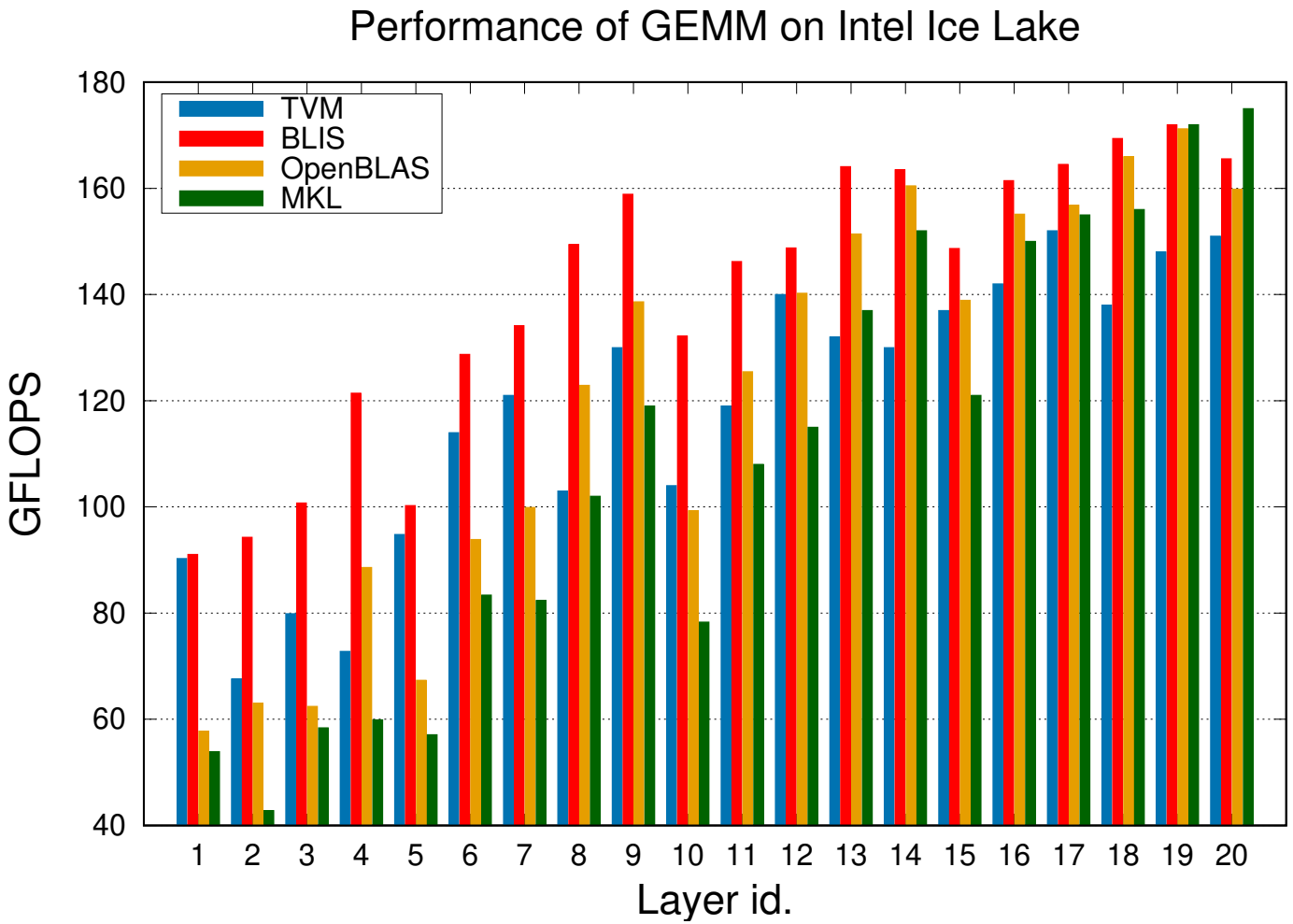
- 20 different convolutional layers of Resnet50 v1.5





Portability

- 20 different convolutional layers of Resnet50 v1.5

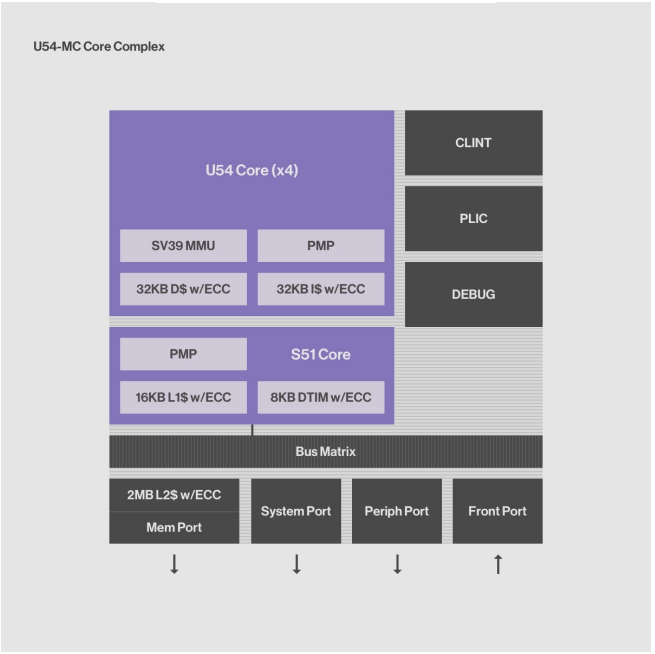


12x32 ukernel for TVM

target="llvm –icelake-avx512"

Portability

target="llvm –mcpu=sifive-u54"



Missing mature hardware (floating point SIMD units)!

Conclusions

- Implemented BLIS GEMM with Apache TVM
- Evaluated ukernels and algorithms
- Exposed the flexibility of TVM
- Compared against other libraries

IT IS REAL!!

Thank you!