Deep Learning Inference: Manual or Automatic Generation of Optimized Kernels for RISC-V ”V”?

Guillermo Alaejos
Adrián Castelló
Pedro Alonso
Enrique S. Quintana-Ortí
Francisco D. Igual

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSIDAD COMPLUTENSE MADRID

Motivation

Convolutional neural networks (CNNs) are key for computer vision & signal processing

Convolution operator take up to 90-95% of the execution time

Four major approaches for the convolution:

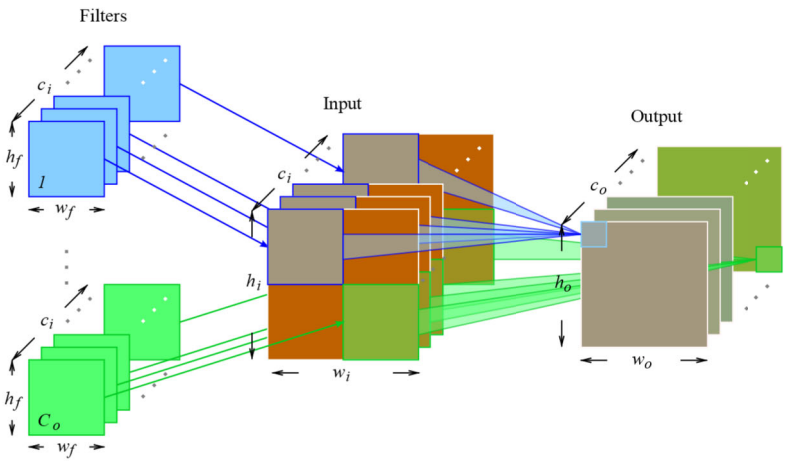
- Lowering: im2col/im2row + large GEMM (matrix multiplication)
- Direct convolution: blocked as a collection of small GEMMs
- Winograd: 1/3 of operations correspond to GEMM
- FFT

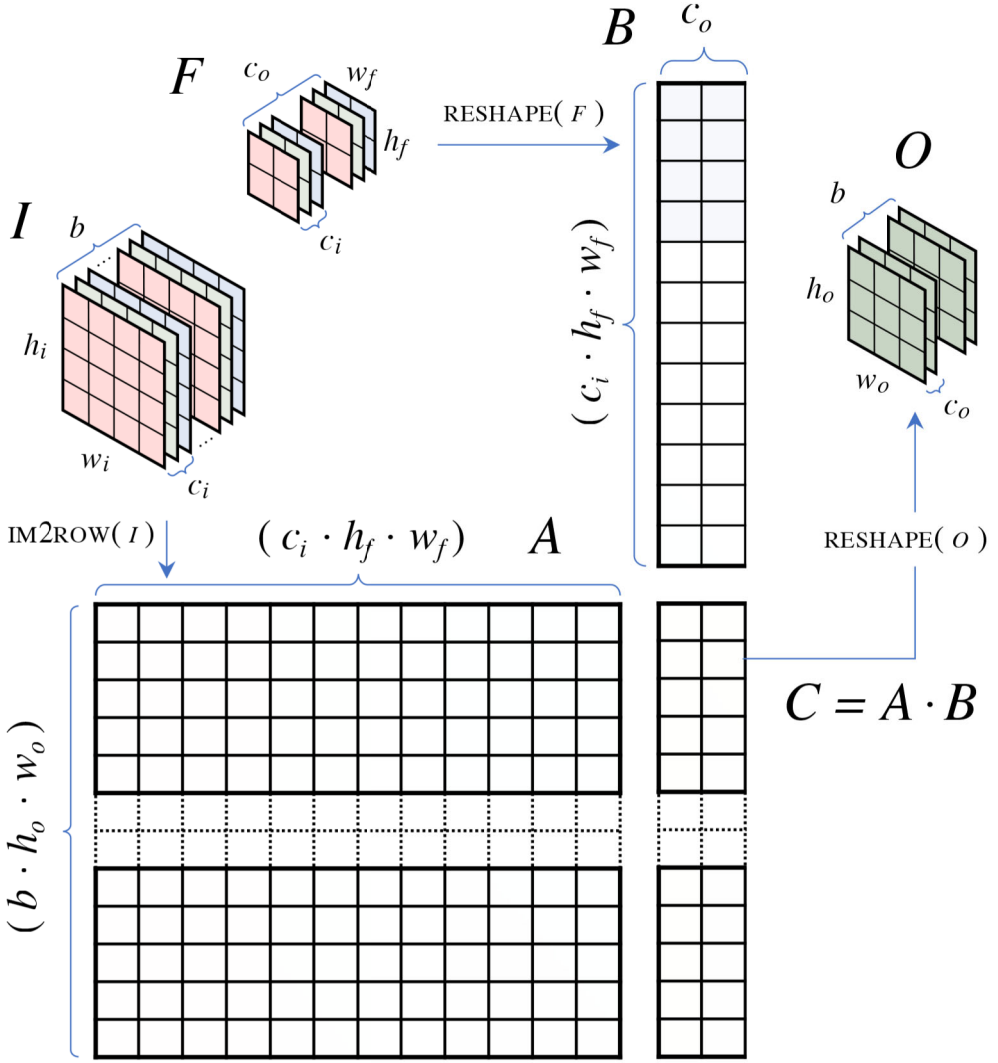
Motivation

Lowering is by far the most flexible technique

Also highly parallel (thread-level parallelism)

$$O = \text{CONV}(F, I)$$





Motivation

Libraries for GEMM → Convolution?

- GotoBLAS2, OpenBLAS, BLIS, oneAPI, AMD AOCL, Intel MKL, ARMPL,…

- Large memory footprint (for IoT)
- Missing functionality: INT8/INT16/INT32, FP16/BF16/TF32
- Suboptimal performance (later)
- No support for operator fusion (deep learning)

Motivation

Libraries for GEMM → Convolution?

- Intel/AMD MMX, SSE, AVX; Intel AVX-512
- ARM NEON, SVE

Hardware-specific solution in a heterogenous world!

- RISC-V “V”
- Microcontrollers...

Motivation

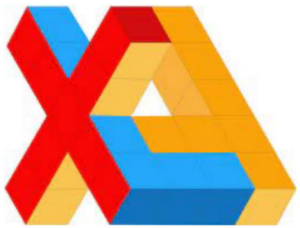
Libraries for GEMM → Convolution?

- Intel/AMD MMX, SSE, AVX; Intel AVX-512
- ARM NEON, SVE

Hardware-specific solution in a heterogenous world!

- RISC-V “V”
- Microcontrollers...

→ Automatic generation… but guided by experience!



Motivation

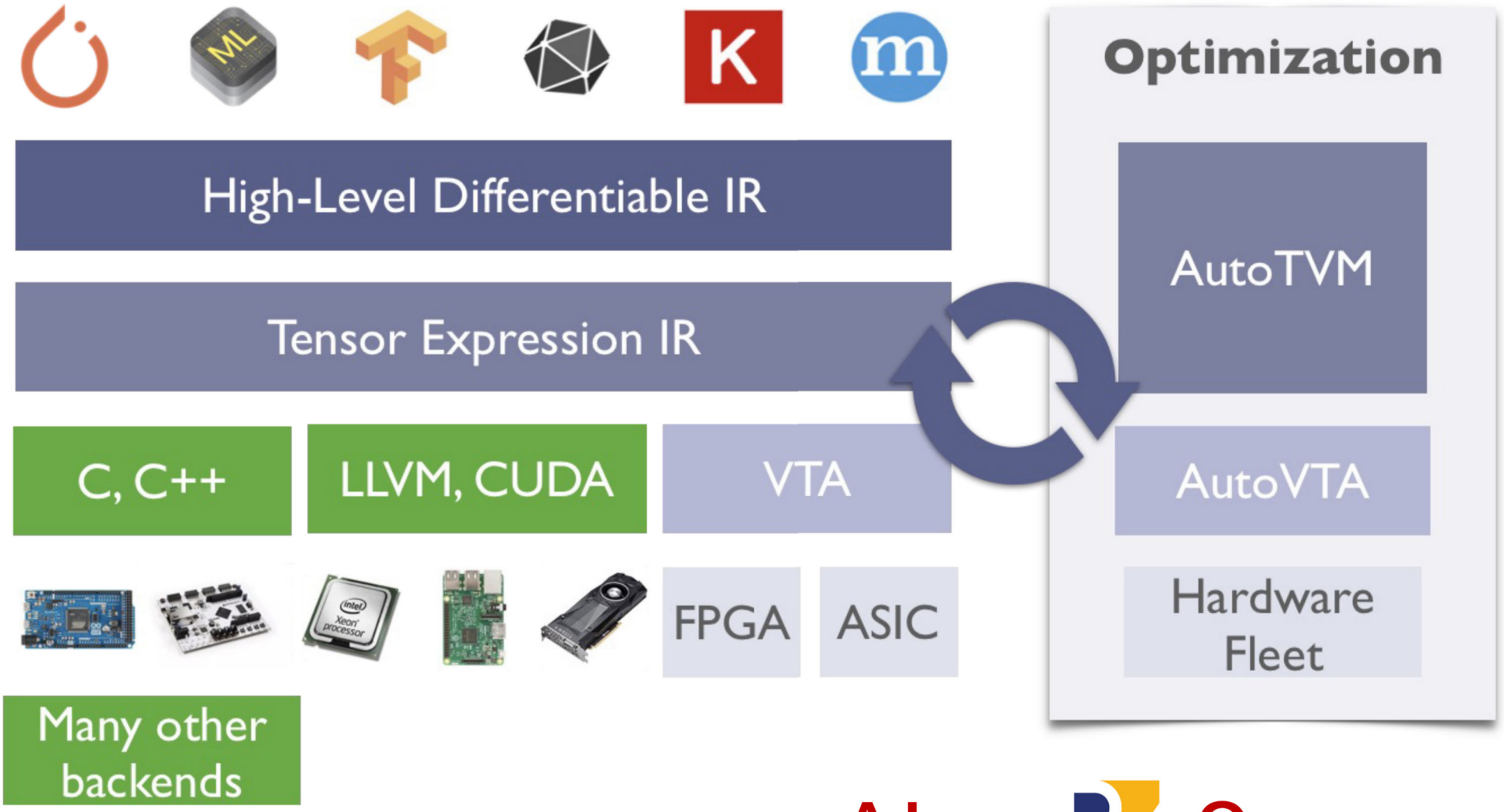
Deep learning is critical for the project application workflows:

- Reduced order models for digital twins
- Cyclone detection
- Urgent computing (earthquakes, tsunamis)

Outline

- Apache TVM
- Guided high performance GEMM with TVM
- Porting to RISC-V “V”
- Experimental Analysis (ARM NEON)
- Conclusions

Apache TVM


Optimization
High-Level Differentiable IR
Tensor Expression IR
AutoTVM
C, C++
LLVM, CUDA
VTA
AutoVTA
FPGA
ASIC
Hardware Fleet
Many other backends
Also RISC-V ?

Apache TVM

$$C = A \cdot B$$

```
for ( i=0; i<m; i++ )
 for ( j=0; j<n; j++ )
  for ( p=0; p<k; p++ )
   C[i][j] += A[i][p]
            *  B[p][j];
```


```
1  def basic_GEMM(m, n, k):
2    # B1) Define operation
3    A = te.placeholder((m, k), name="A")
4    B = te.placeholder((k, n), name="B")
5    p = te.reduce_axis((0, k), "p")
6    C = te.compute((m, n), lambda i, j:
7                   te.sum(A[i, p] * B[p, j],
8                   axis=p), name="C")
9
10   # B2) Prepare schedule
11   sched  = te.create_schedule(C.op)
12   (i, j) = C.op.axis
13   (p,)   = C.op.reduce_axis
14
15   # B3) Loop schedule i->j->p
16   sched[C].reorder(i, j, p)
17
18   # B4) Generate code with LLVM backend
19   return tvm.build(sched, [A, B, C],
20                    target="llvm")
```

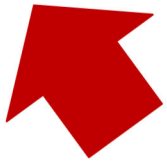
Even with AutoTVM, the search space for optimization is huge!

High Performance (HP) GEMM

GotoBLAS

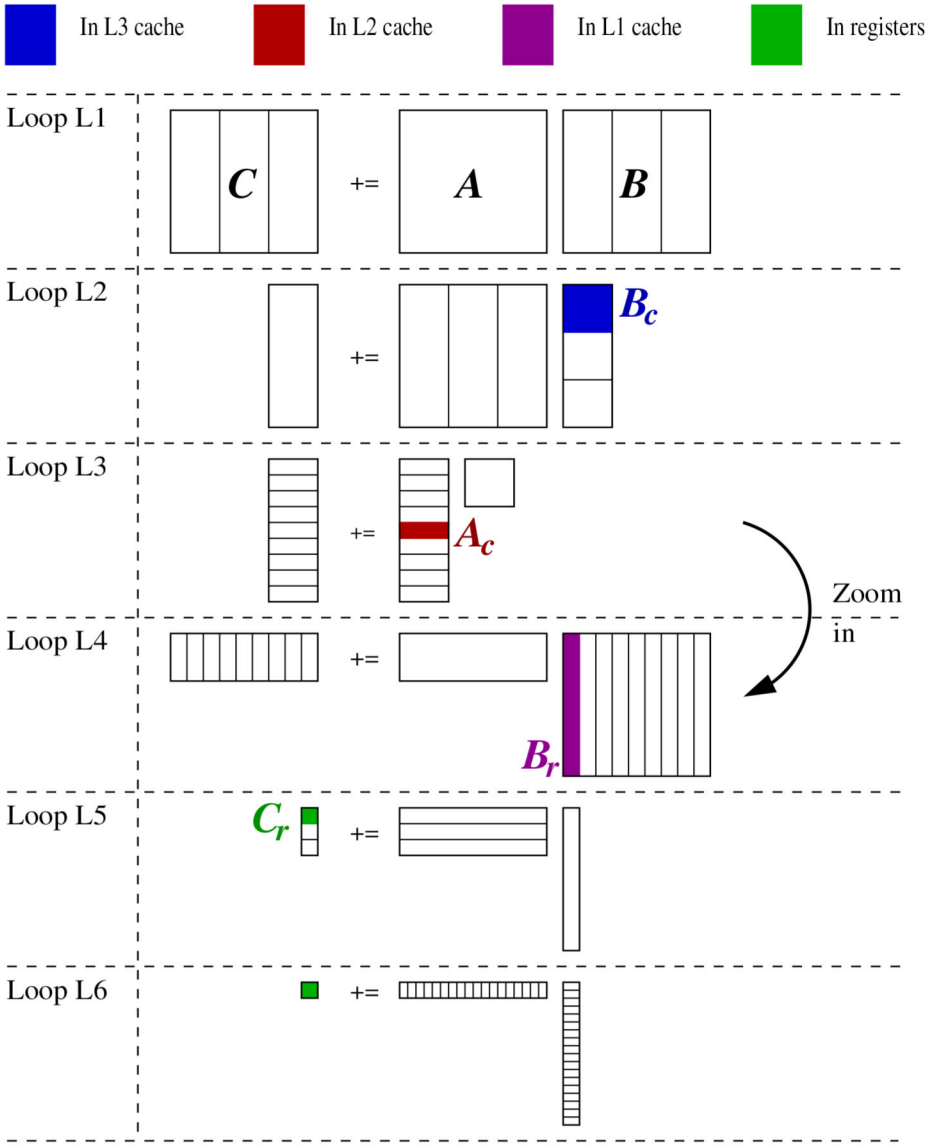
K. Goto and R. A. van de Geijn. 2008. Anatomy of high-performance matrix multiplication. ACM Trans. Math. Softw.

- GotoBLAS2
- BLIS
- OpenBLAS
- ARMPL
- AMD AOCL
- Intel MKL, oneAPI?



Automatic generation (TVM) + **Experience**

High Performance (HP) GEMM




```
for ( jc = 0; jc < N; jc += Nc )                    // L1
  for ( pc = 0; pc < K; pc += Kc ) {                // L2
    Bc := B[pc : pc + Kc − 1][jc : jc + Nc − 1];
    for ( ic = 0; ic < M; ic += Mc ) {              // L3
      Ac := A[ic : ic + Mc − 1][pc : pc + Kc − 1];
      for ( jr = 0; jr < Nc; jr += Nr )             // L4
        for ( ir = 0; ir < Mc; ir += Mr )           // L5
          // Micro-kernel
          for ( pr = 0; pr < Kc; pr++ )             // L6
            Cc[ir : ir + Mr − 1][jr : jr + Nr − 1]
                  += Ac[ir : ir + Mr − 1][pr]
                  ·   Bc[pr][jr : jr + Nr − 1];
  } }
```

- 5 + 1 loops
- 2 packing routines
- A micro-kernel

- Reduce cache misses
- Access with unit stride from micro-kernel
- Accommodate (SIMD) vectorization & expose loop parallelism

Guided HP GEMM with TVM

1) Tiling:

Why?

- Reduce cache misses

How?

- Partition $m, n, k \rightarrow mc, nc, kc$
- Loop order: jc, pc, ic, jr, ir, pr

```
for ( jc = 0; jc < N; jc += Nc )                    // L1
  for ( pc = 0; pc < K; pc += Kc ) {                // L2
    Bc := B[pc : pc + Kc - 1][jc : jc + Nc - 1];
    for ( ic = 0; ic < M; ic += Mc ) {              // L3
      Ac := A[ic : ic + Mc - 1][pc : pc + Kc - 1];
      for ( jr = 0; jr < Nc; jr += Nr )             // L4
        for ( ir = 0; ir < Mc; ir += Mr )           // L5
          // Micro-kernel
          for ( pr = 0; pr < Kc; pr++ )             // L6
            Cc[ir : ir + Mr - 1][jr : jr + Nr - 1]
                  += Ac[ir : ir + Mr - 1][pr]
                  .  Bc[pr][jr : jr + Nr - 1];
  } }
```

Guided HP GEMM with TVM

1) Tiling:

```
for ( jc=0; jc<n; jc+=nc )
 for ( pc=0; pc<k; pc+=kc )
  for ( ic=0; ic<m; ic+=mc )
   for ( jr=0; jr<nc; jr++ )
    for ( ir=0; ir<mc; ir++ )
     for ( pr=0; pr<kc; pr++ )
      C[ic+ir][jc+jr]
        += A[ic+ir][pc+pr]
        * B[pc+pc][jc+jr];
```

```
1  def block_GEMM(m, n, k, mc, nc, kc):
2    # B1) as in basic_GEMM
3    # Omitted for brevity
4
5    # B2) Prepare schedule
6    sched = te.create_schedule(C.op)
7    ic, jc, \
8    ir, jr = sched[C].tile(C.op.axis[0],
9                           C.op.axis[1], mc, nc)
10   pc, pr = sched[C].split(p, factor=kc)
11
12
13
14
15
16
17
```

Partition:

$$m, n, k \rightarrow mc, nc, kc$$

Cache configuration parameters, chosen using analytical model

T. M. Low, F. D. Igual, T. M. Smith, E. S. Quintana-Ortí. 2017. Analytical modeling is enough for high performance BLIS. ACM Trans. Math. Softw.

Guided HP GEMM with TVM

1) Tiling:

```
for ( jc=0; jc<n; jc+=nc )
 for ( pc=0; pc<k; pc+=kc )
  for ( ic=0; ic<m; ic+=mc )
   for ( jr=0; jr<nc; jr++ )
    for ( ir=0; ir<mc; ir++ )
     for ( pr=0; pr<kc; pr++ )
      C[ic+ir][jc+jr]
        += A[ic+ir][pc+pr]
        * B[pc+pc][jc+jr];
```

Schedule:

$$jc, pc, ic, jr, ir, pr$$

```
1  def block_GEMM(m, n, k, mc, nc, kc):
2    # B1) as in basic_GEMM
3    # Omitted for brevity
4
5    # B2) Prepare schedule
6    sched = te.create_schedule(C.op)
7    ic, jc, \
8    ir, jr = sched[C].tile(C.op.axis[0],
9                           C.op.axis[1], mc, nc)
10   pc, pr = sched[C].split(p, factor=kc)
11
12   # B3) Loop schedule as in B3A2C0
13   sched[C].reorder(jc, pc, ic, jr, ir, pr)
14
15   # B4) Generate code with LLVM backend
16   return tvm.build(sched, [A, B, C],
17                    target="llvm")
```

Guided HP GEMM with TVM

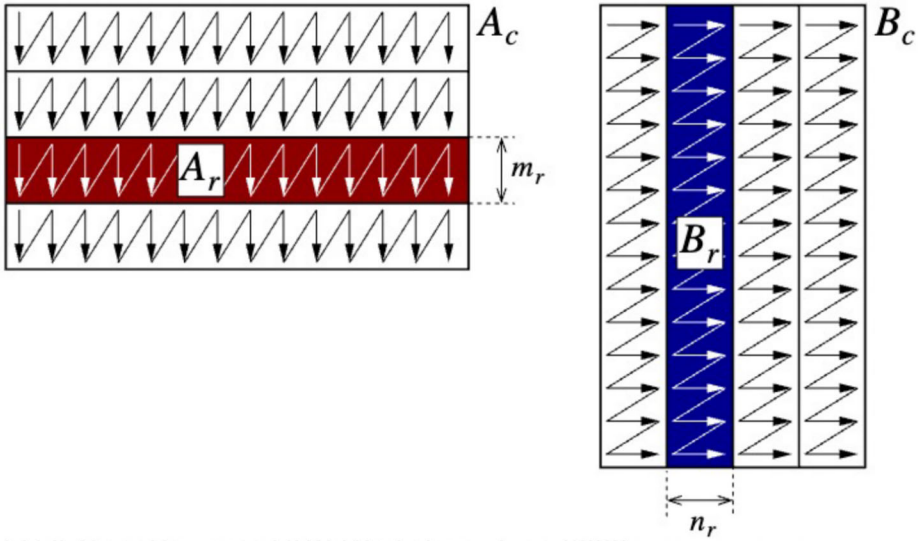
2) Packing:

Why?

- Reduce cache misses
- Access with unit stride

How?

- Buffer $Ac \rightarrow mc\ x\ kc$
- Buffer $Bc \rightarrow kc\ x\ nc$




```
for ( jc = 0; jc < N; jc += Nc )                    // L1
  for ( pc = 0; pc < K; pc += Kc ) {                // L2
    Bc := B[pc : pc + Kc − 1][jc : jc + Nc − 1];
    for ( ic = 0; ic < M; ic += Mc ) {              // L3
      Ac := A[ic : ic + Mc − 1][pc : pc + Kc − 1];
      for ( jr = 0; jr < Nc; jr += Nr )             // L4
        for ( ir = 0; ir < Mc; ir += Mr )           // L5
          // Micro-kernel
          for ( pr = 0; pr < Kc; pr++ )             // L6
            Cc[ir : ir + Mr − 1][jr : jr + Nr − 1]
                  += Ac[ir : ir + Mr − 1][pr]
                  ·  Bc[pr][jr : jr + Nr − 1];
  } }
```

Guided HP GEMM with TVM

2) Packing:

- Buffer $Ac \rightarrow mc\ x\ kc$
- Buffer $Bc \rightarrow kc\ x\ nc$

```
def packed_GEMM(m, n, k, mc, nc, kc, mr, nr):
  # P1) Define operation
  A = te.placeholder((m, k), name="A")
  B = te.placeholder((k, n), name="B")

  # 4D view into A and B to induce creation
  # of buffer by TVM
```

```

  p = te.reduce_axis((0, k), "p")
  C = te.compute((m, n), lambda i, j:
        te.sum(Ac[p//kc, i//mr,
                  tvm.tir.indexmod(p, kc),
                  tvm.tir.indexmod(i, mr)] *
               Bc[p//kc, j//nr,
                  tvm.tir.indexmod(p, kc),
                  tvm.tir.indexmod(j, nr)],
               axis=p), name="C")

  # P2) Prepare schedule
  sched = te.create_schedule(C.op)
  ic, jc, \
  ir, jr  = sched[C].tile(C.op.axis[0],
                          C.op.axis[1],
                          mc, nc)
  pc, pr = sched[C].split(p, factor=kc)

  # P3) Place Ac, Bc in the desired loops


  # P4) Loop schedule as in B3A2C0
  sched[C].reorder(jc, pc, ic, jr, ir, pr)

  # P5) Generate code with LLVM backend
  return tvm.build(sched, [A, B, C],
                   target="llvm")
```

Guided HP GEMM with TVM

3) Micro-kernel:

Why?

- SIMD vectorization

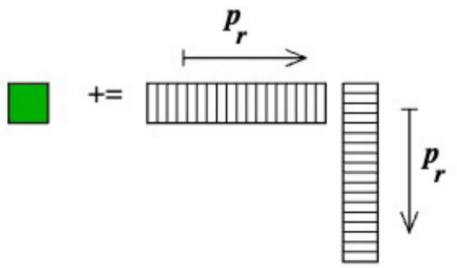
How?

- $C \rightarrow mr\ x\ nr$ in registers
- Ac, Bc streamed from caches

```
for ( jc = 0; jc < N; jc += Nc )                    // L1
  for ( pc = 0; pc < K; pc += Kc ) {                // L2
    Bc := B[pc : pc + Kc - 1][jc : jc + Nc - 1];
    for ( ic = 0; ic < M; ic += Mc ) {              // L3
      Ac := A[ic : ic + Mc - 1][pc : pc + Kc - 1];
      for ( jr = 0; jr < Nc; jr += Nr )             // L4
        for ( ir = 0; ir < Mc; ir += Mr )           // L5
          // Micro-kernel
          for ( pr = 0; pr < Kc; pr++ )             // L6
            Cc[ir : ir + Mr - 1][jr : jr + Nr - 1]
                  += Ac[ir : ir + Mr - 1][pr]
                  .  Bc[pr][jr : jr + Nr - 1];
  } }
```

```
for ( pr=0; pr<kc; pr++ )
  C(ic+ir:ic+ir+mr-1,jc+jr:jc+jr+nr-1)
    += Ac(ir:ir+mr-1,pr)
    *  Bc(pr,jr:jr+nr-1);
```

L6



Guided HP GEMM with TVM

3) Micro-kernel:

- $C \rightarrow mr\ x\ nr$ in registers
- Ac, Bc streamed from caches

Architecture-specific piece of code

Also RISC-V ?

```
1  def opt_GEMM(m, n, k, mc, nc, kc, mr, nr):
2    # P1), P2), P3) as in pack_GEMM
3    # Omitted for brevity
4
5    # P4) Expose loops inside micro-kernel
6
7
8
9    # P5) Loop schedule as in B3A2C0
10   sched[C].reorder(jc, pc, ic,
11                    jr, ir, pr, it, jt)
12
13   # P6) Unroll+vectorize micro-kernel loops
14
15
16
17
18
19
20
21   # P7) Generate code with LLVM backend
22   return tvm.build(sched, [A, B, C],
23                    target="llvm")
```

Guided HP GEMM with TVM

Recap:

- We can mimic all optimization techniques in modern instances of GEMM: tiling, packing, vectorization, parallelization

→ Why didn't we use BLIS (or any other HP library) GEMM then?

- Large memory footprint
- Missing functionality: INT8/INT16/INT32, FP16/BF16/TF32?
- Suboptimal performance
- Hardware-specific solution in a heterogenous world

Porting to RISC-V

Will these ideas/techniques apply?

Nothing fundamentally different in RISC-V "V" that impedes it

ARM NEON vector intrinsics

```
for ( k=0; k<kc-1; k++ ) {

      // Prefect for next iteration
      baseA = baseA+Amr;
      baseB = baseB+Bnr;

      A0n = vld1q_f32(&Aptr[baseA]);
      B0n = vld1q_f32(&Bptr[baseB]);

      C0 = vfmaq_laneq_f32(C0, A0, B0, 0);
      C1 = vfmaq_laneq_f32(C1, A0, B0, 1);
      C2 = vfmaq_laneq_f32(C2, A0, B0, 2);
      C3 = vfmaq_laneq_f32(C3, A0, B0, 3);

      A0 = A0n;
      B0 = B0n;
    }
    // Last iteration
    …
```

RISC-V vector intrinsics (1.0)

```
for ( k=0; k<kc-1; k++ ) {
      // Prefect for next iteration
      baseA = baseA+Amr;
      baseB = baseB+Bnr;

      A0n = vle32_v_f32m1(&Aptr[baseA], vl);
      B0n = vle32_v_f32m1(&Bptr[baseB], vl);

      C0 = vfmacc_vf_f32m1(C0, b00, A0, vl);
      C1 = vfmacc_vf_f32m1(C1, b01, A0, vl);
      C2 = vfmacc_vf_f32m1(C2, b02, A0, vl);
      C3 = vfmacc_vf_f32m1(C3, b03, A0, vl);

      A0 = A0n;
      vsse32_v_f32m1(&b00,0, B0n, 1);
      vsse32_v_f32m1(&b01,0, B0n, 2);
      vsse32_v_f32m1(&b02,0, B0n, 3);
      vsse32_v_f32m1(&b03,0, B0n, 4);
    }
    // Last iteration
    …
```

Porting to RISC-V

Will these ideas/techniques apply?

Nothing fundamentally different in RISC-V “V” that impedes it

ARM ISA

```
1  fmov    v16.4s, 5.0e-1
2  ldr     q2, [sp, 176]
3  ldr     q1, [sp, 192]
4  fsub    v2.4s, v2.4s, v1.4s
5  ldr     q1, [sp, 208]
6  fadd    v2.4s, v2.4s, v1.4s
7  fmul    v0.4s, v2.4s, v16.4s
```

RISC-V “V” ISA

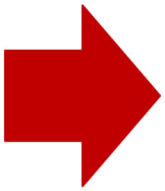
```
1  vsetvli     a3,a6,e32,m1
2  vle32.v     v4,0(s5)
3  vle32.v     v1,0(a3)
4  vfmv.v.f    v5,fs0
5  addi        a5,sp,784
6  vfsub.vv    v1,v4,v1
7  vle32.v     v3,0(a3)
8  vfadd.vv    v1,v1,v3
9  vfmul.vv    v1,v1,v5
10 vs1r.v      v5,0(a5)
```

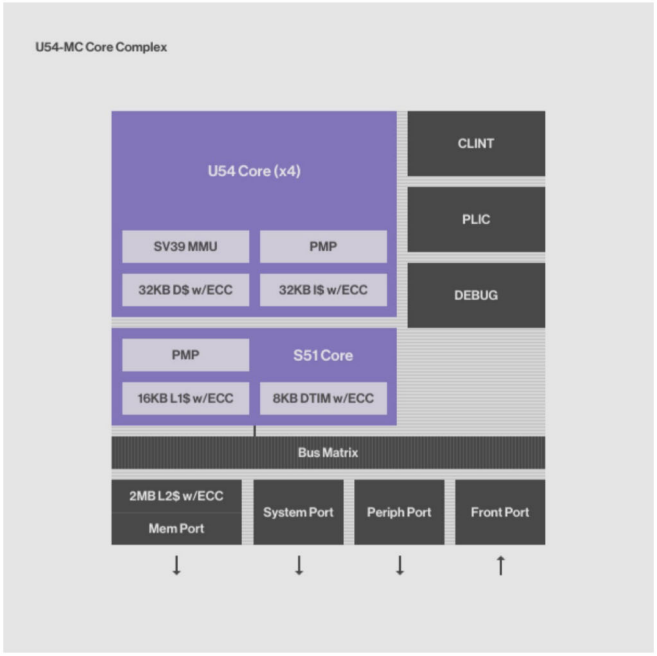
Porting to RISC-V

Will these ideas/techniques apply?

- Support from TVM

```
21 # P7) Generate code with LLVM backend
22 return tvm.build(sched, [A, B, C],
23     target="llvm -mcpu=sifive-u54")
```



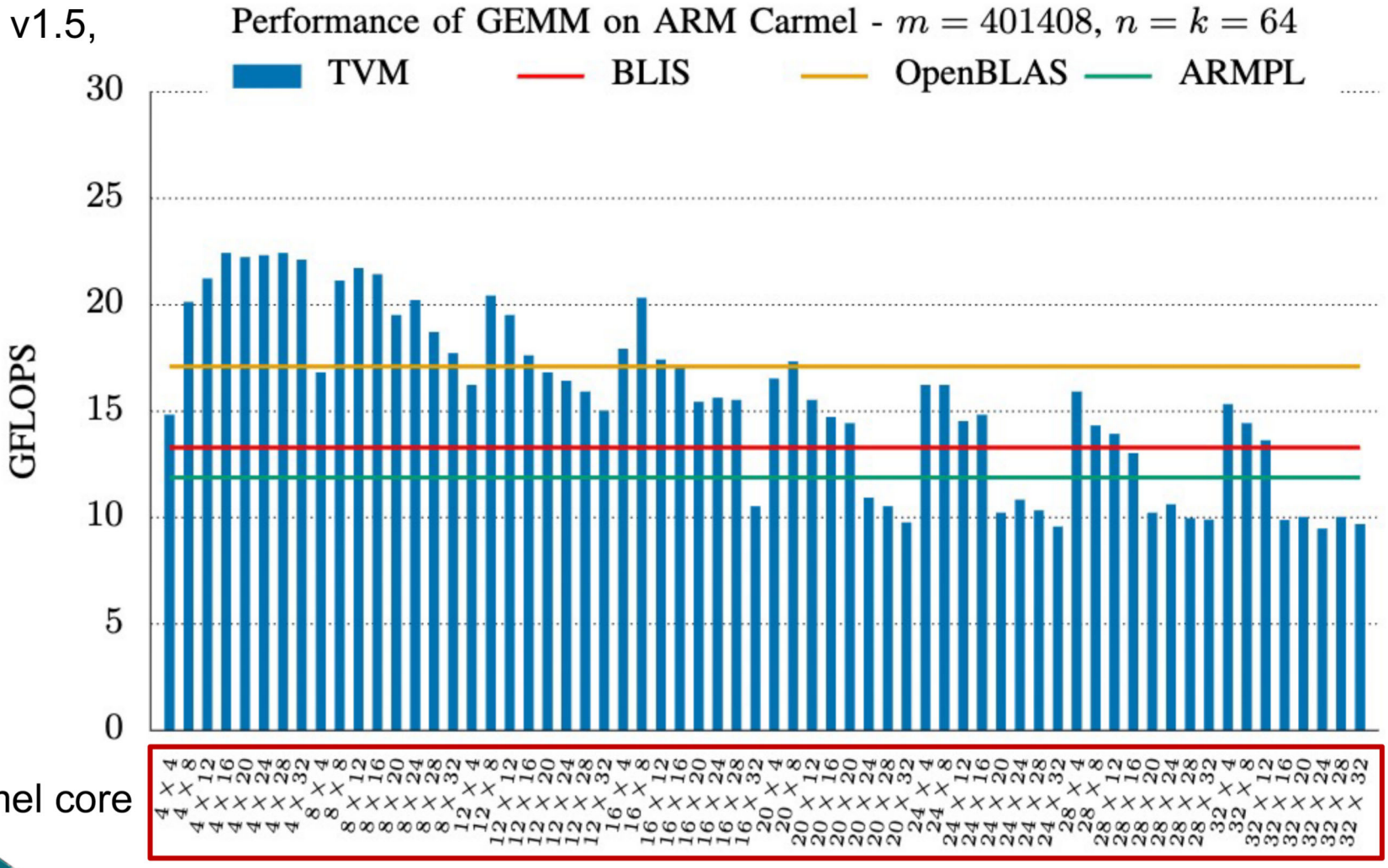


Missing mature hardware:

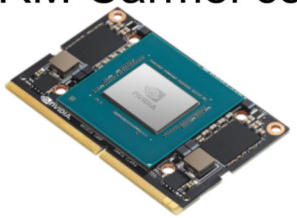
- Floating point SIMD units
- Mixed precision with extended accumulators

Experimental Analysis (ARM NEON)

ResNet-50 v1.5, layer #44

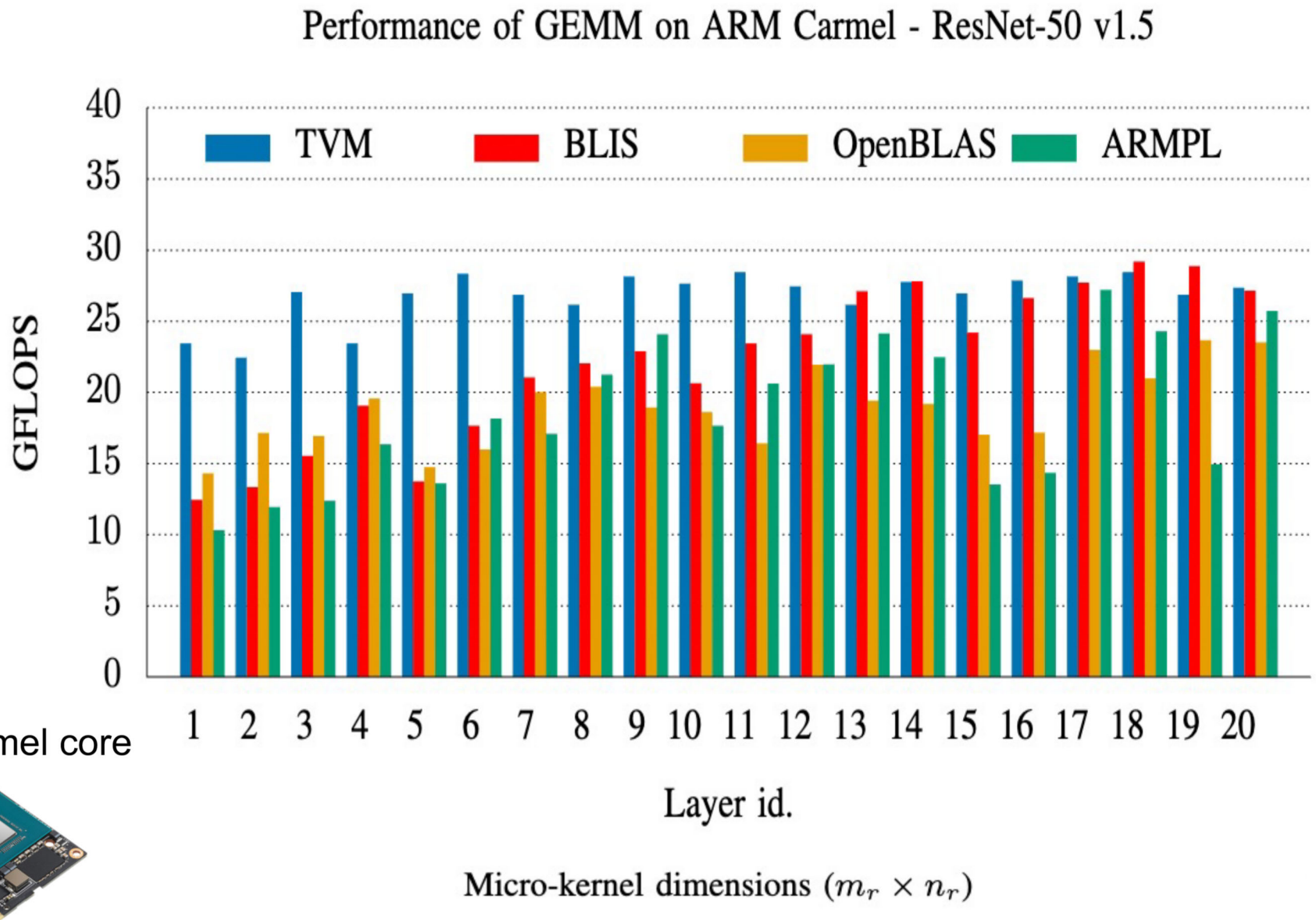


ARM Carmel core



Search space for TVM

Experimental Analysis (ARM NEON)



ARM Carmel core

Conclusions

Libraries provide a solution that is too rigid in a heterogenous world!

but

Automatic generation yields an explosion on the optimization search space

For convolution & GEMM, automatic generation guided by experience

- Search space limited to mr, nr. Only integer multiples of SIMD length

- Nothing fundamentally different in RISC-V “V” that impedes the application of the same techniques

Thanks!

eFlows4HPC

Enabling dynamic and Intelligent workflows in the future EuroHPC ecosystem

www.eFlows4HPC.eu

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955558. The JU receives support from the European Union's Horizon 2020 research and innovation programme and Spain, Germany, France, Italy, Poland, Switzerland, Norway.