支援 TVM AI Compiler on RISC-V with P Instructions

Jenq-Kuen Lee

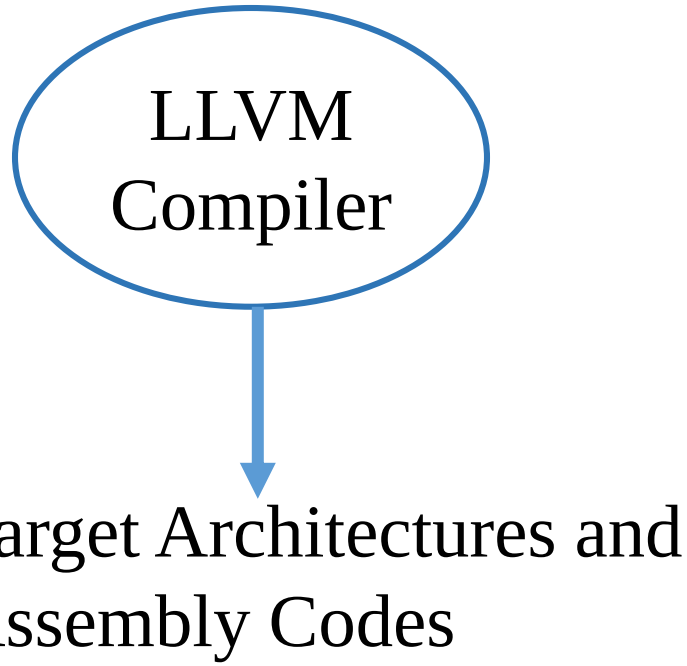
Department of Computer Science
National Tsing Hua University
Hsinchu, Taiwan

Open source compilers have transformed our industry

LLVM

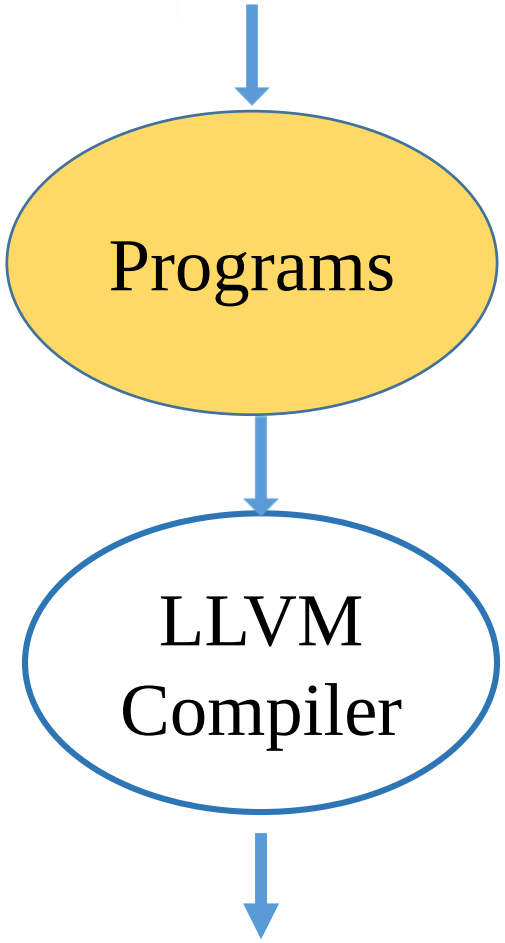
REF: 1st TVM and Deep Learning Compilation Conference. Keynote – Luis Ceze, SAMPL

Compiler Models

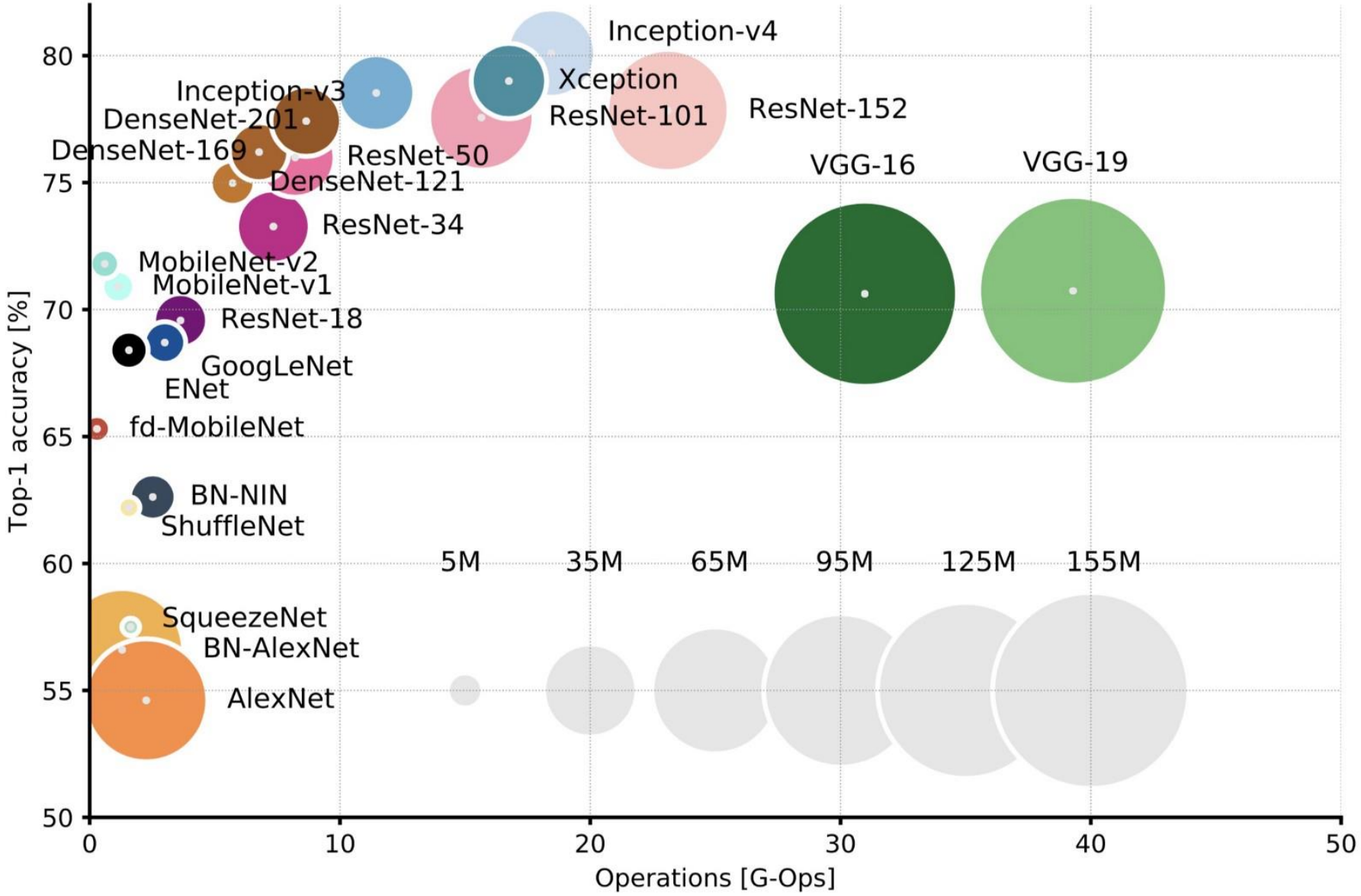


AI Compiler


Microsoft
CNTK
Caffe


Programs
LLVM
Compiler
Target Architectures and
Assembly Codes

AI Models and Performance



From: Eugenio Culurciello

Compiler Optimization Course

- Multi-threading
- Parallellization
- Tiling
- Loop Fusion
- Loop distribution
- Loop Unroll
- Latency Tolerance
- Vectorization
- Subword SIMD Intrinsics
- VLIW Scheduling
- Register Allocation
- Blas and Memory Locality
- Compiler for low-power
- Vector Swizzle
- Double buffering
- Scheduling for heterogeneous computing
- Quantization with Fixed-Point Optimizations

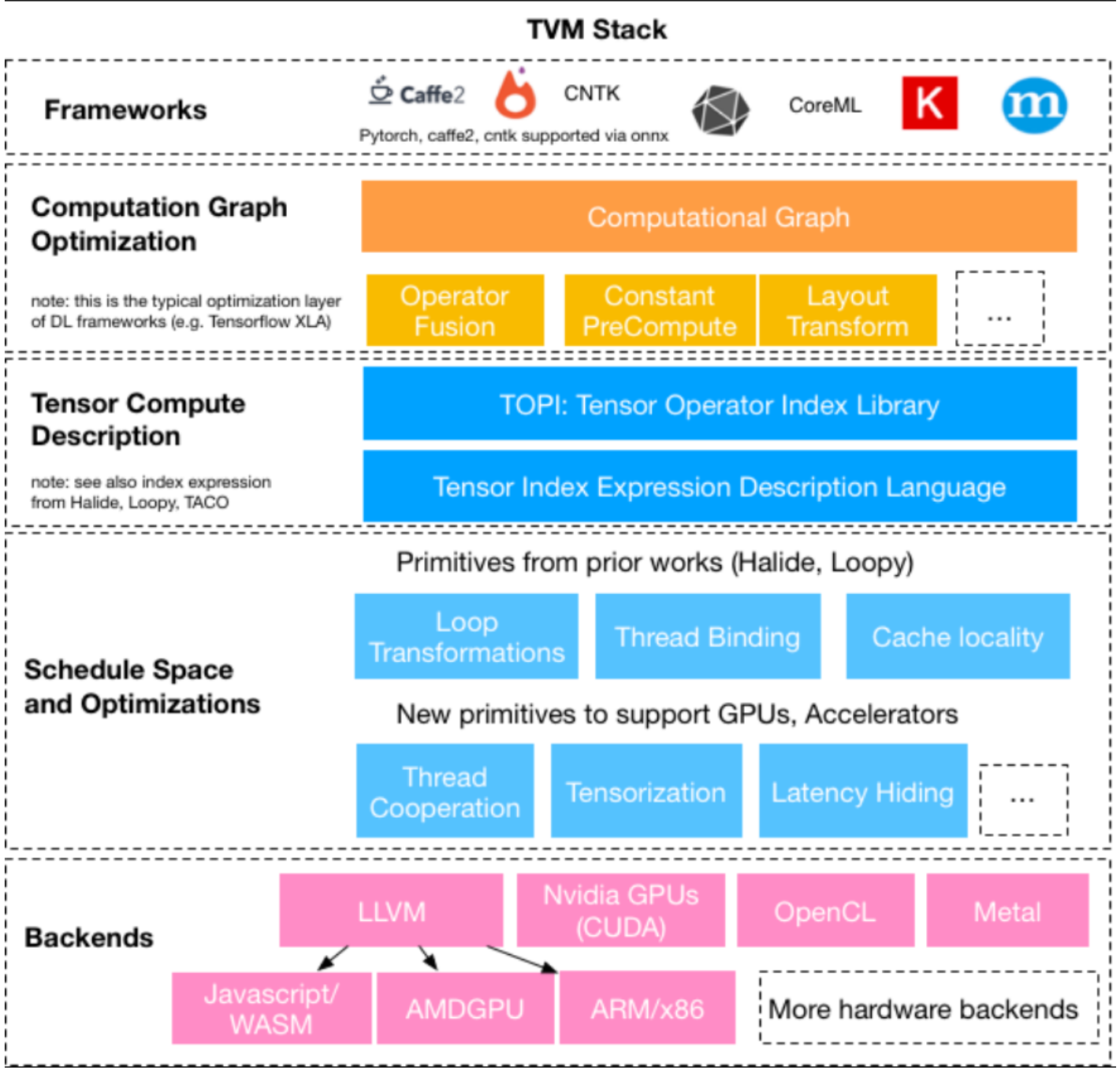
Reference: NTHU CS, Embedded Compiler Course

Halide and Halide IR

```
1  Func blur_3x3(Func input) {
2    Func blur_x, blur_y;
3    Var x, y, xi, yi;
4
5    // The algorithm - no storage or order
6    blur_x(x, y) = (input(x-1, y) + input(x, y) + input(x+1, y))/3;
7    blur_y(x, y) = (blur_x(x, y-1) + blur_x(x, y) + blur_x(x, y+1))/3;
8
9    // The schedule - defines order, locality; implies storage
10   blur_y.tile(x, y, xi, yi, 256, 32)
11         .vectorize(xi, 8).parallel(y);
12   blur_x.compute_at(blur_y, x).vectorize(x, 8);
13
14   return blur_y;
15 }
```

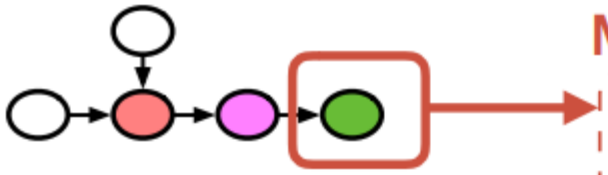
Reference: Halide – a language for fast portable portable computation on images and tensor

TVM
Basic Structure



Reference: TVM, https://tvm.ai/

Existing Approach: Engineer Optimized Tensor Operators



Matmul: Operator Specification

```
C = tvm.compute((m, n),
    lambda y, x: tvm.sum(A[k, y] * B[k, x], axis=k))
```

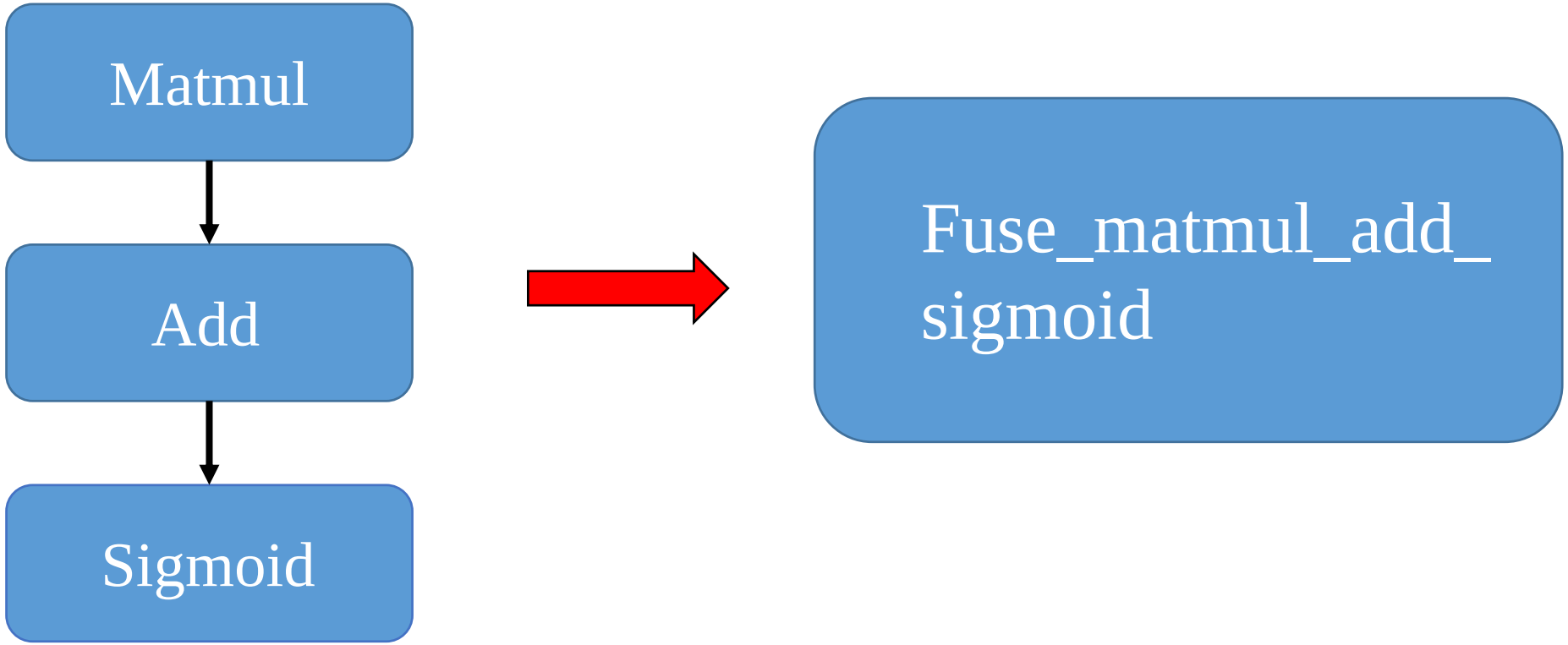
Loop Tiling for Locality

```
for yo in range(128):
  for xo in range(128):
    C[yo*8:yo*8+8][xo*8:xo*8+8] = 0
    for ko in range(128):
      for yi in range(8):
        for xi in range(8):
          for ki in range(8):
            C[yo*8+yi][xo*8+xi] +=
              A[ko*8+ki][yo*8+yi] * B[ko*8+ki][xo*8+xi]
```

Reference: Tianqi Chen, TVM Conference 2018

Optimizing Computational Graphs

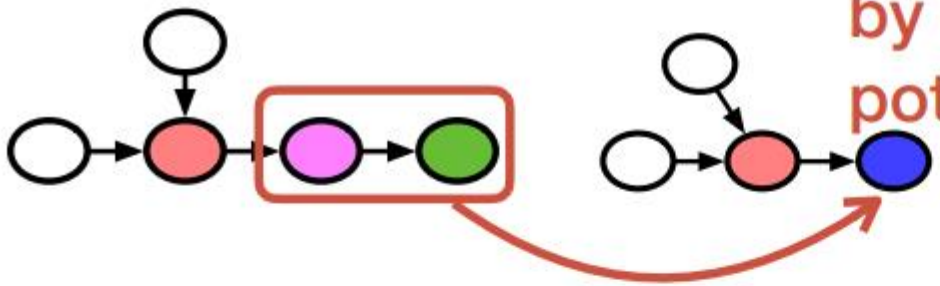
- **Operator Fusion**: Fusion optimization combines multiple operators together into a single kernel.



Related Work: A Function-Composition Approach to Synthesize Fortran 90 Array Operations, Gwan-Hwan Hwang, **Jenq Kuen Lee**, Dz-Ching Ju, Journal of Parallel and Distributed Computing, 54, 1-47, 1998.

Limitations of Existing Approach

New operator introduced
by operator fusion optimization
potentially benefit: 1.5x speedup

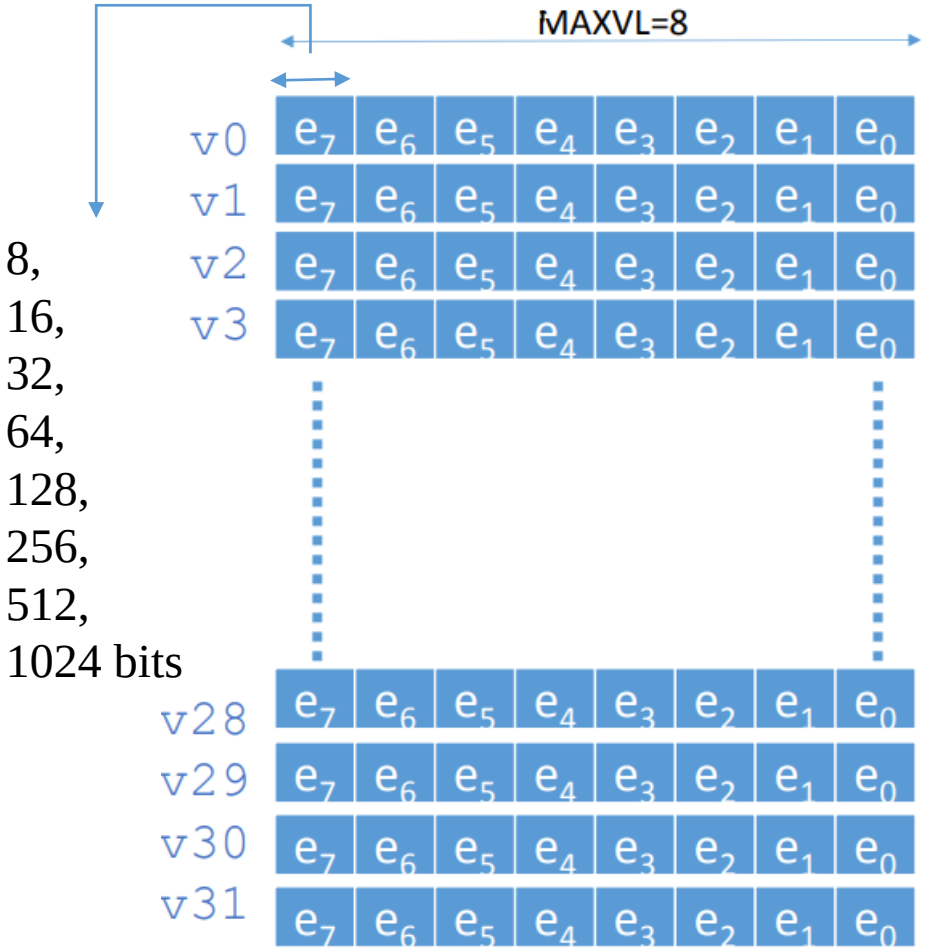


cuDNN

Reference: Tianqi Chen, TVM Conference 2018

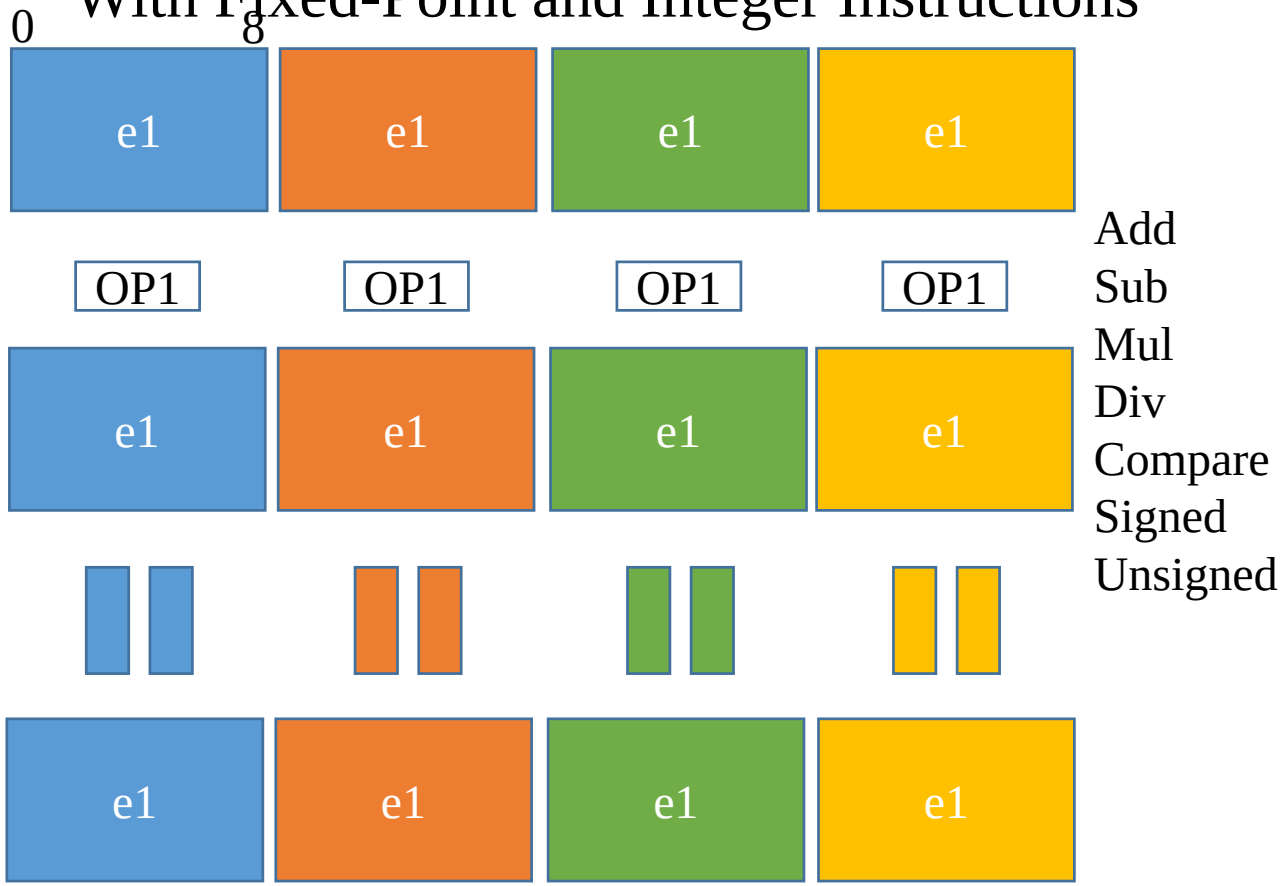
RISC-V with two vector ISAs to support fall-back engine with AI Models

Super Word Vector



Courtesy: Vector ISA, Roger Espasa, Esperanto Technologies

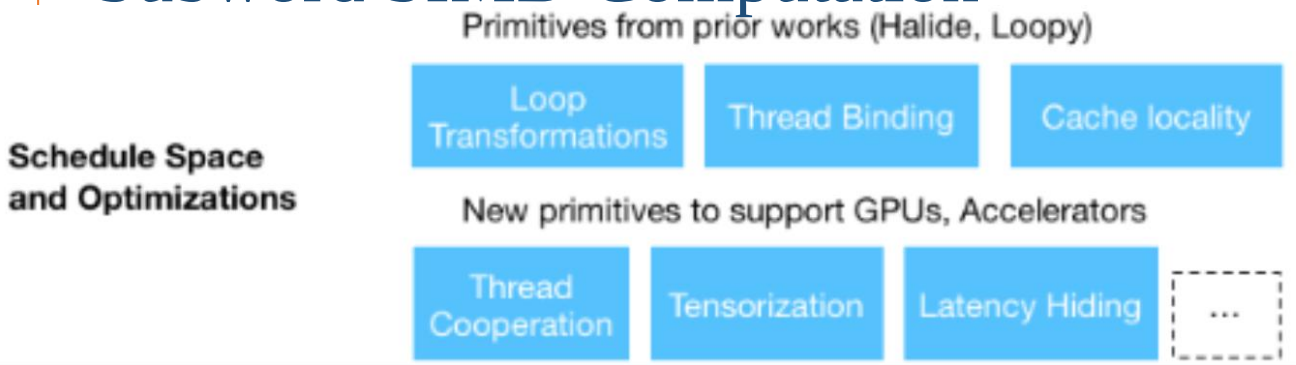
Packed Vector (SubWord SIMD) With Fixed-Point and Integer Instructions



RISC-V DSP (P) Extension Proposal Chuan-Hua Chang, Andes Technology Corporation

Support TVM on RISC-V with Subword SIMD Computation

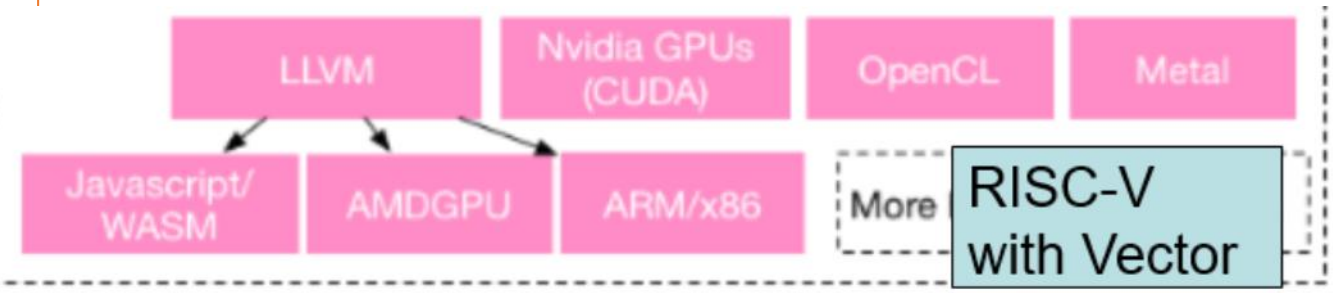
- We add RISC-V target in TVM codegen phase. The TVM RISC-V codegen will lower SIMD computation with Subword SIMD intrinsics.
- The LLVM backend will need to generate the corresponding SIMD instructions.
- Also on-going work to add TVM scheduling to quantize computation into fixed-points, “quantize(width, exponent)”.



New Primitives support RISC-V with vector units

Subword SIMD Quantization

SIMD Rewriting with Intrisnsics



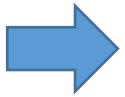

```
Expr CodeGenRISCV::CreateAddIntr(const Add* op,

    ::llvm::Intrinsic::ID vadd_id = ::llvm::Int
    if(bits==8) {
      vadd_id=::llvm::Intrinsic::riscv_simd_sad
    }
    const Expr& e1 = op->a;
    const Expr& e2 = op->b;
    Array<Expr> vcnt_args;
    vcnt_args.push_back(ir::UIntImm::make(UInt(32), vpaddlu_id));
    vcnt_args.push_back(ir::UIntImm::make(UInt(32), 0));
    vcnt_args.push_back(e1);
    vcnt_args.push_back(e2);
    return ir::Call::make(op->type,  "llvm_intrin", vcnt_args, Call::PureIntrinsic);
}
```

tvm/src/codegen/llvm/codegen_riscv.cc

Example – Matrix Multiply

```
produce compute {
  for (i.outer, 0, ((n + 1)/2)) {
    for (j, 0, n) {
      for (i.inner.s, 0, 2) {
        if (likely(((i.outer*2) < (n - i.inner.s)))) {
          compute[((((i.outer*2) + i.inner.s)*n) + j)]
          = (int16)0
        }
      }
      for (k, 0, n) {
        for (i.inner.s, 0, 2) {
          if (likely(((i.outer*2) < (n - i.inner.s)))) {
            compute[((((i.outer*2) + i.inner.s)*n) + j)]
            = (compute[((((i.outer*2) + i.inner.s)*n) + j)]
            + (A[((((i.outer*2) + i.inner.s)*n) + k)]*B[((j*n) + k)]))
          }
        }
      }
    }
  }
}
```

```
%22 = bitcast i8* %21 to i16*
%23 = load i16, i16* %4, align 2, !tbaa !115
%24 = insertelement <2 x i16> undef, i16 %23, i32 0
%25 = shufflevector <2 x i16> %24, <2 x i16> undef,
      <2 x i32> zeroinitializer
%26 = tail call <2 x i16> @llvm.riscv.simd.khm16
      (<2 x i16> %13, <2 x i16> %25)
%27 = tail call <2 x i16> @llvm.riscv.simd.sadd16
      (<2 x i16> zeroinitializer, <2 x i16> %26)
```



```
pkbb16  a6, a7, a6
lhu a7, 0(a2)
li  a5, 7
li  a3, 8
pkbb16  a4, a4, a4
ksll16  a4, a4, a3
ksll16  a6, a6, a5
khm16 t2, a6, a4
pkbb16  a4, a7, a7
pkbb16  a7, t1, t0
lhu t1, 16(a2)
ksll16  a4, a4, a3
ksll16  a7, a7, a5
lhu t3, 4(a1)
lhu t4, 12(a1)
khm16 t0, a7, a4
pkbb16  a4, zero, zero
add16 t0, a4, t0
```

Subword SIMD Intrinsic LLVM IR

In this example, **104 of 229** instructions will be with SIMD computation which process **two element** in one instruction.

Fixed-point

```
with nnvm.compiler.build_config(opt_level=0):
  graph, lib, params = nnvm.compiler.build(sym, target, shape_dict, dtype='fxp16_11', params=params)
```

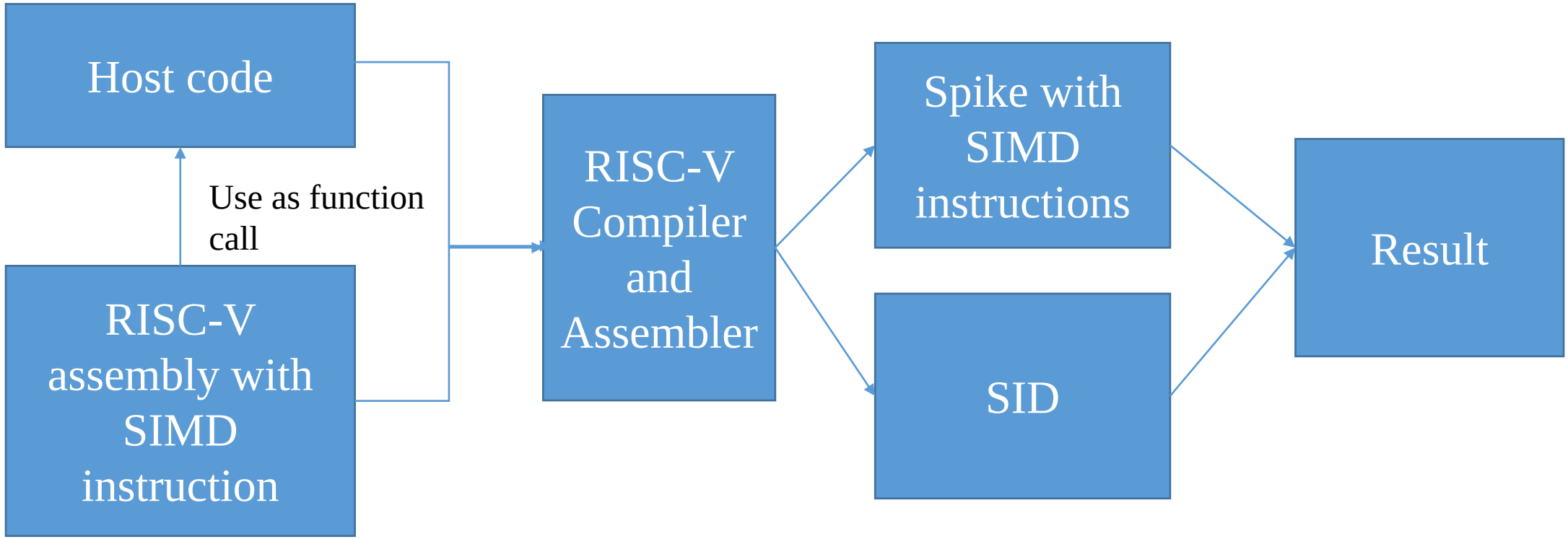
```
%17 = getelementptr inbounds i16, i16* %5, i64 %16
%18 = bitcast i16* %17 to <2 x i16>*
%19 = load <2 x i16>, <2 x i16>* %18, align 8, !tbaa !118
%20 = tail call <2 x i16> @llvm.riscv.simd.khm16.fxp.v2i16(i64 11, <2 x i16> %13, <2 x i16> %19)
%21 = tail call <2 x i16> @llvm.riscv.simd.sadd16.i32(<2 x i16> %9, <2 x i16> %20)
```

```
ksll16  a3, a3, t1
ksll16  t0, t0, t1
khm16 a3, t0, a3
kadd16  a0, a0, a3
```

Saturation Instructions

From RISC-V Target to Result

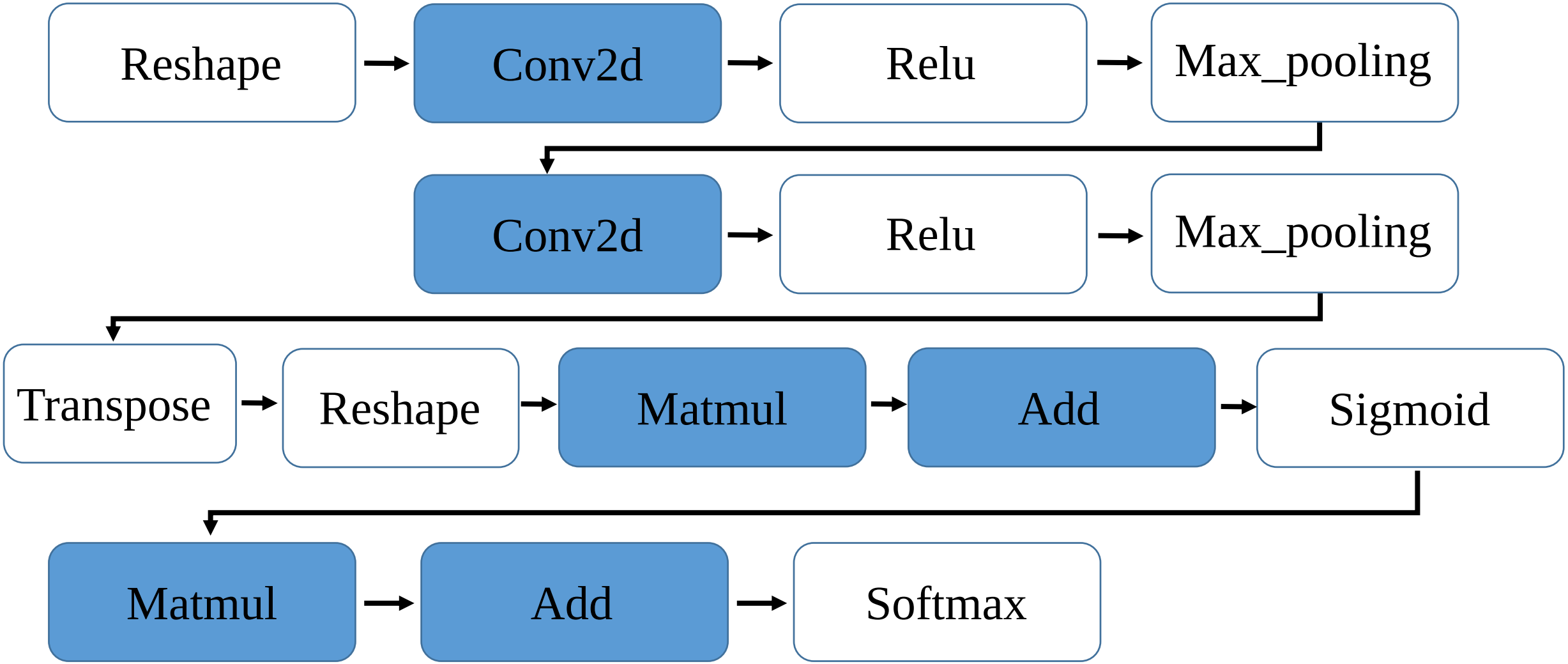
- Run RISC-V on simulator with SIMD instruction support



Experiment Configuration

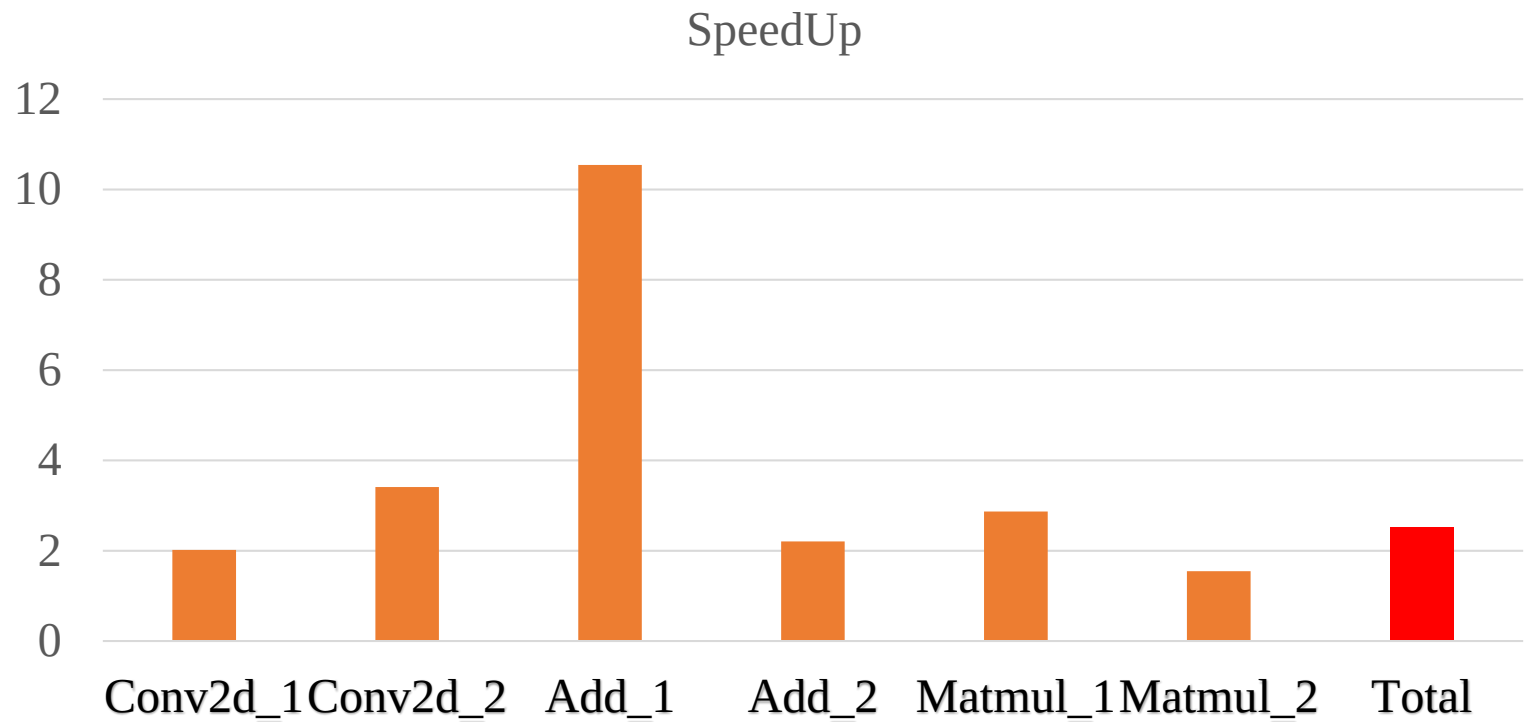
- Simulator: SID
- Chip profile: Andes AE350-AX25 (64-bit)
- SIMD vector type: v2i16 or v4i16
- ISA: RISC-V P Instruction Draft (based on AndeStar V3/V5 DSP ISA)

The CNN model training with MNIST database



The CNN model training with MNIST database Result (in cycles)

	Conv2d_1	Conv2d_2	Add_1	Add_2	Matmul_1	Matmul_2	Total
with SIMD	2691442	80721445	129	29	3607446	18508	133984783
no SIMD	5437062	275322142	1360	64	10351386	28654	338576006
SpeedUp	2.02012973	3.410768254	10.542635	2.20689655	2.8694500	1.54819537	2.52697357



PeakHills Group 成立緣由

- Small compiler startup established 2015 at University incubation center (after three terms of MOEA projects from NTHU CS Pllab).
  - 國家矽導計畫時，本團隊組成了學界科專團隊，總共執行了三期經濟部學界科專計畫
  - 計畫內容涵蓋了針對訊號處理器(DSP)所開發的高效能編譯器及相關系統軟體、異質多核心編譯器最佳化技術、多核心低功耗編譯技術、多核心系統軟體以及多媒體應用框架技術等、多核心繪圖處理器編譯器
  - 相關研發成果也促成了多項技轉及產學合作計畫
- 嵌譯科技
  - 凝聚過往累積的研發能量
  - 專注 LLVM 編譯器及相關嵌入式軟體技術開發
  - AI compiler/framework and 高效能軟體加值硬體

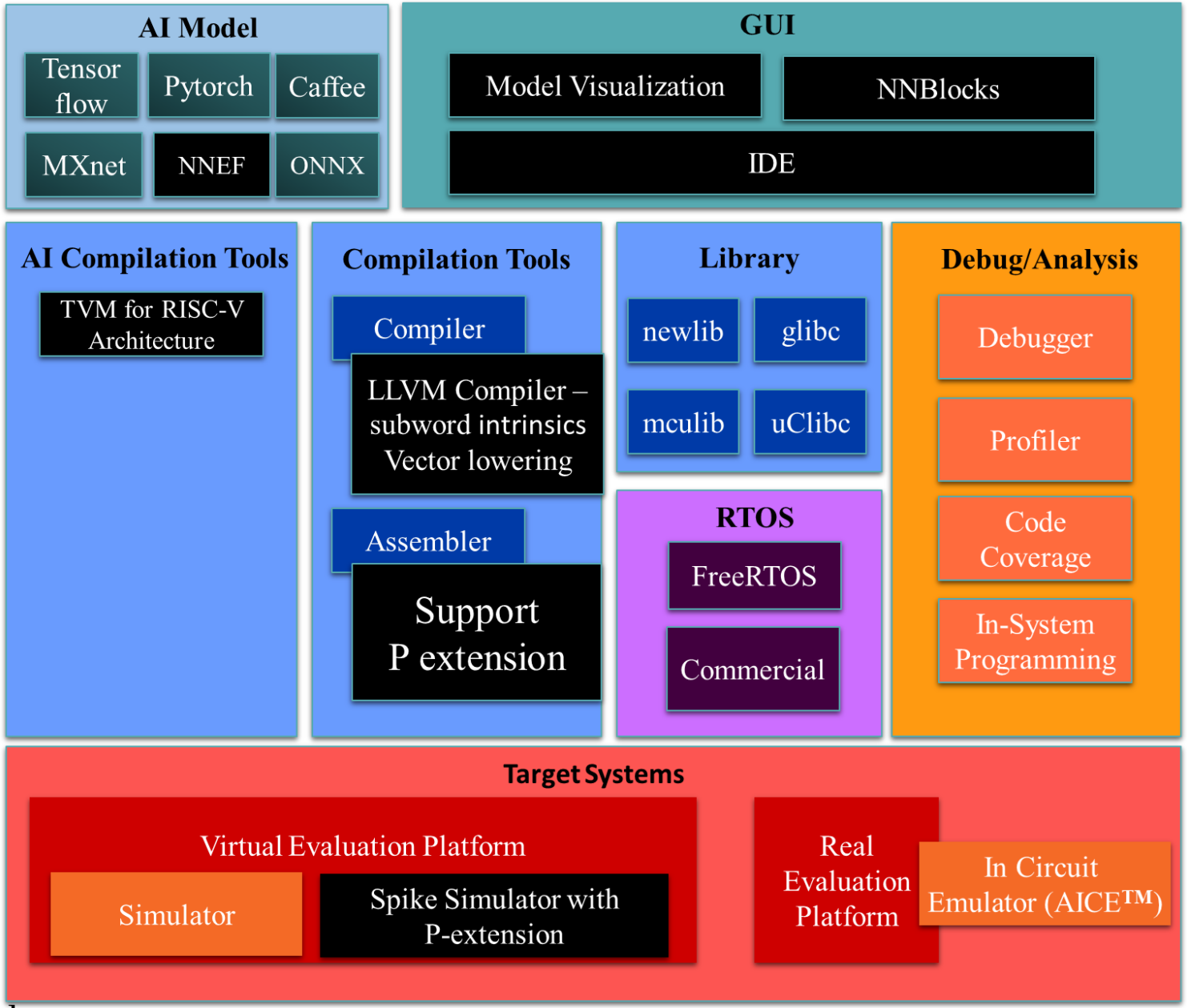
PEAKHILLS GROUP

相關資訊與服務

- Web site: http://www.peakhillsgroup.com
  - Currently a Khronos associate member
- Consulting and Open Source Service
  - TVM Compiler
    - Collaborate with Andes “TVM on RISC-V”
  - Khronos APIs reference design
    - Example: NNEF to Android NNAPI
  - LLVM Compiler

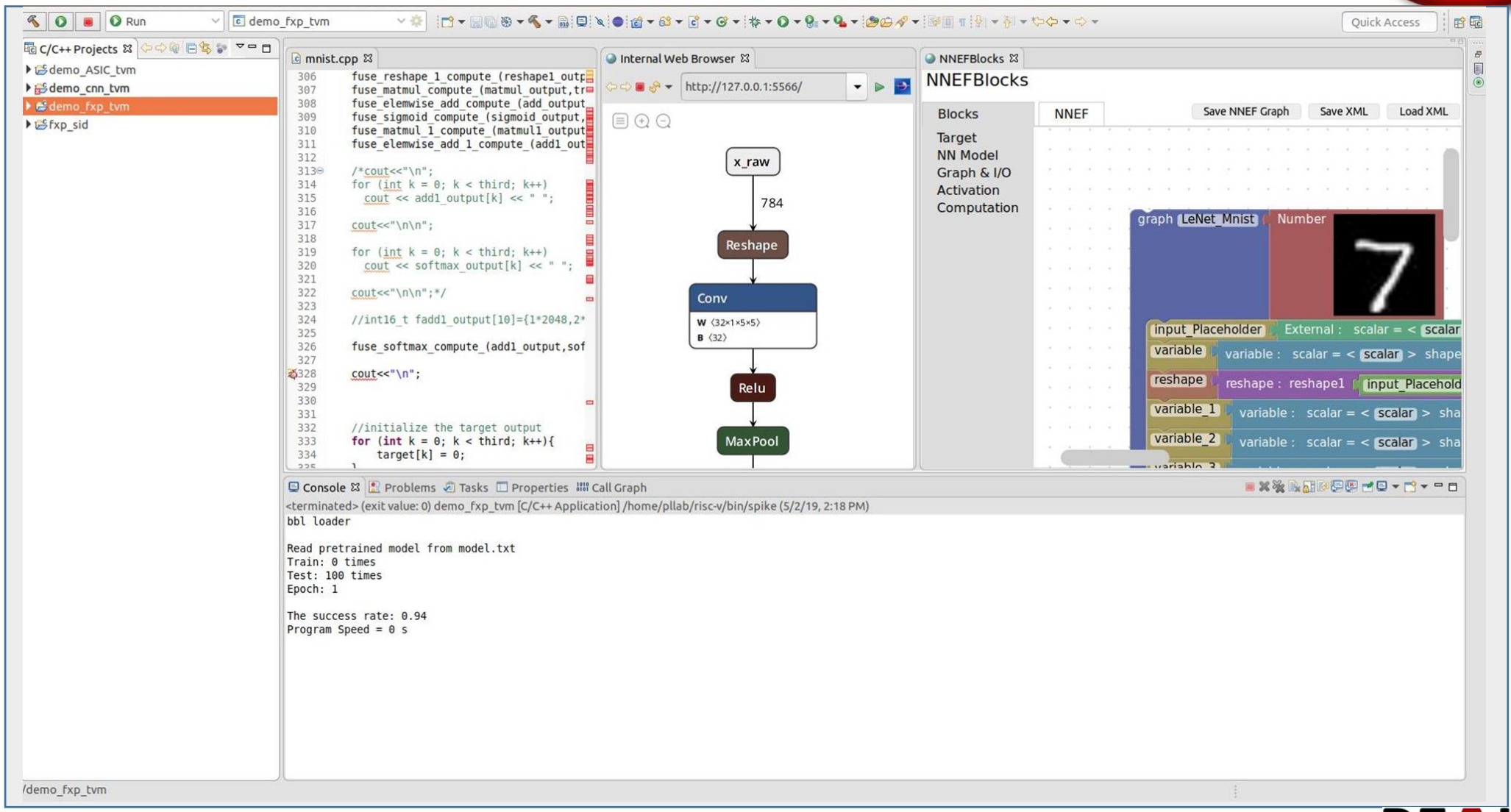
PEAKHILLS GROUP Contact:張元銘總經理

PeakHills Group Revision based on Andes Software Stacks and SID Simulator



Reference: http://www.andestech.com/product-andeSight.php?cls=7&fbclid=IwAR1BuwNHxSu5C3_KHfxK421Pd6GpmvG9xZwKnEOi8KMlwTvyP0b-w5ErpPU

Eclipse with TVM + Blockly



Summary

- We enable TVM to generate the AI inference program with SIMD instructions and run it on RISC-V simulator.
- Also has some discussions with AWS team to add RISC-V back-end for TVM deep learning compiler.
- Look forward to contributing the codes to TVM and Neo source trees.
- Currently the work is with Spike and Sid RISC-V simulator and we look forward to using Gem5 and real chips for performance tuning.
- The result might be better using the enhanced llvm backend modeled with pipeline architectures.
- Please also reference our papers/talks on TVM 2018, Seattle, and RISC-V Workshop, Hsinchu, 2019.