CS 259: GPU Microarchitecture LAB 2

Group Member: Kejun Wu, Yujie Cao

TF32 Introduction

The purpose of this lab2 is to add the TF32 (Tensor Float 32) data format support to Vortex RISC-V GPGPU's Tensor Core unit, evaluating the performance of this new numerical format in matrix multiplication tasks through complete SimX simulator implementation (Assignment 8) and RTL hardware implementation (Assignment 9). TF32 is a data format introduced by NVIDIA in the Ampere architecture, which combines the exponent range of FP16 with the precision of FP32. Specifically, TF32 is a 19-bit floating-point format (1 sign bit, 8 exponent bits, 10 mantissa bits) padded to 32 bits, providing precision close to FP32 without code modifications while achieving performance similar to FP16. This characteristic makes TF32 an ideal numerical format choice for scientific computing and AI training. This lab would focus on fused multiply-add (FMA) operations used in MMA. The mathematical operation is:

$$\text{result} = (a \times b) + c$$

The operands a and b are multiplied first. The product is added to c, and there is no intermediate rounding occurs between the multiplication and addition.

TF32 Simx Implementation

`sim/common/tensor_cfg.h`: Here we define the TF32 format. A new "struct" is added for TF32. We update the "fmt_string" function in the same file to handle the new format.

`sim/simx/tensor_unit.cpp`: We add a template specialization for the FMA (Fused Multiply-Add) struct to handle TF32 inputs accumulating into fp32. This will convert TF32 values to standard IEEE fp32 for computation. "rv_xtof_s" is used to convert the custom TF32 format (8 exponent bits, 10 mantissa bits) to fp32. "rv_fmul_s" is IEEE-754 multiply and "rv_fadd_s" is IEEE-754 add. Both honor the requested rounding mode and set IEEE-754 exception flags, returning bit-exact results suitable for RTL/SimX matching. "bit_cast" is used to reinterpret the bits as float.

`sim/simx/tensor_unit.cpp`: FEDP (Fused Element-wise Dot Product) is the core operation in the Tensor Unit for MMA. We also update "select_FEDP(uint32_t IT, uint32_t OT)" function. It enables TF32 input operands with an FP32 accumulator in the SimX model. This integrates TF32 into the Tensor Unit's execution path.

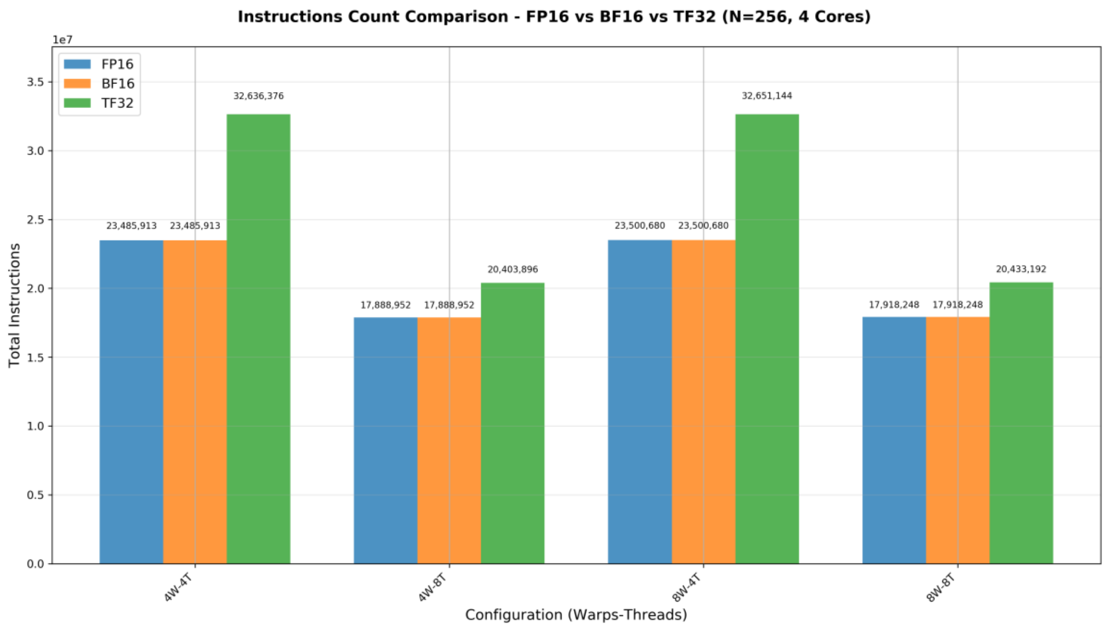
TF32 RTL Implementation

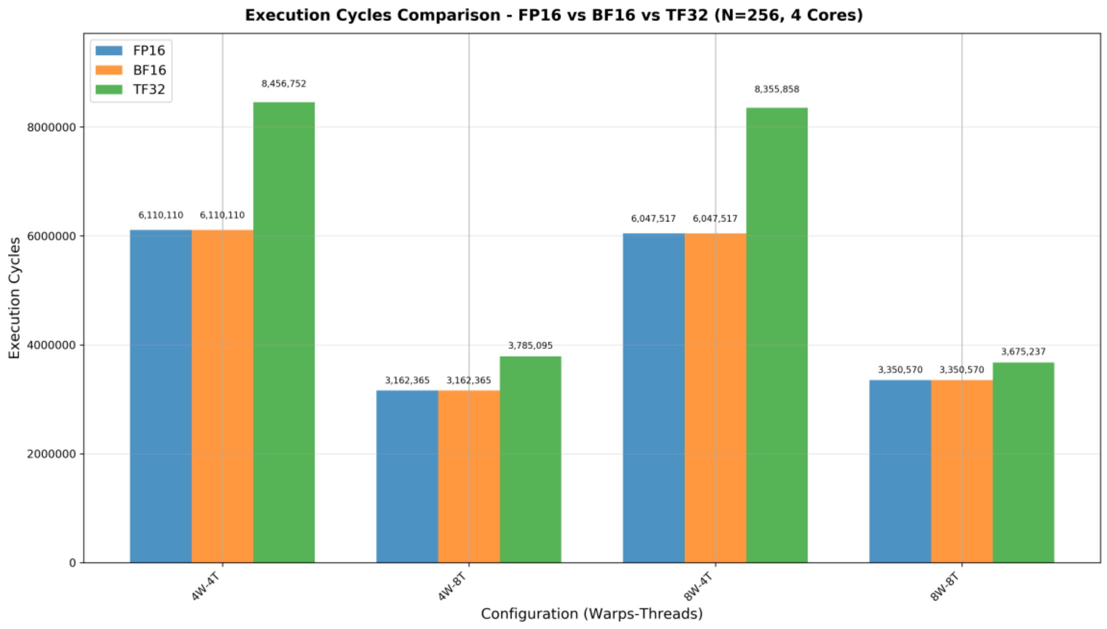
`hw/rtl/tcu/VX_tcu_pkg.sv`: We modify to add the TF32 format ID. The constant is added for TF32. "trace_fmt" is updated for enabling TF32 tracing. This modification defines the ID for TF32 and ensures it can be traced correctly in the hardware.

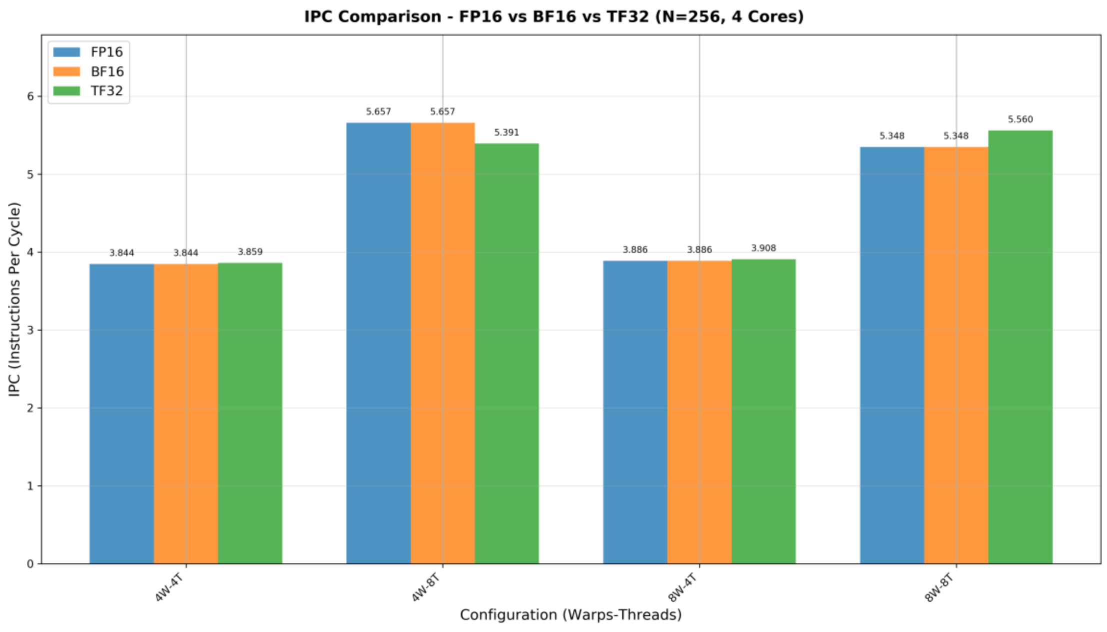
`hw/rtl/tcu/VX_tcu_fedp_bhf.sv`: Here we extend the fused element-wise dot product (FEDP) module to support TF32 multiplication using the Berkeley HardFloat library. We add add a new TF32 multiplication instance. A full 32-bit register to store one element, since TF32 has 19 bits.

To share the same accumulator with fp16/fp16, the TF32 multiplier interleave its 4 outputs with zeros to generate 8 inputs for the next stage.

Results plots and analysis of SimX







Workload

- Matrix a: 256x256; Matrix b: 256x256; Matrix c: 256x256
- Input Type: TF32 vs. FP16/BF16
- Output Type: FP32
- Test Configurations: 4-core GPU
  - 4 warps × 4 threads
  - 4 warps × 8 threads
  - 8 warps × 4 threads
  - 8 warps × 8 threads

Detailed Analysis

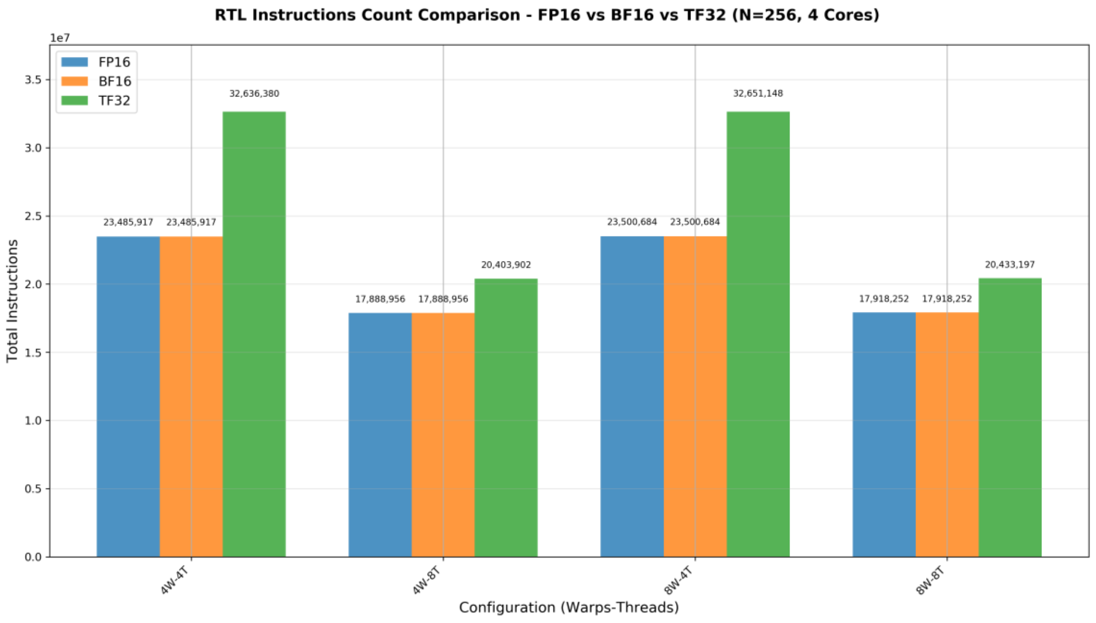
As for the instruction counts, for the same warp/thread configuration, TF32 always incurs more instructions than FP16 and BF16, with overheads roughly in the 15–40% range depending on the configuration. This is expected because TF32 has a 19-bit effective precision (1 sign, 8 exponents, 10 mantissa bits) packed into 32 bits, and in our SimX implementation it requires conversion from TF32 to full FP32 before performing the FMA operation. This conversion, together with the wider data path, increases both the arithmetic complexity and the dynamic instruction count, which directly translates into longer run times. FP16 and BF16 traces practically overlap, indicating that the BF16 path reuses the same infrastructure as FP16. The main difference between FP16 and BF16 is the numerical rather difference than architectural difference.

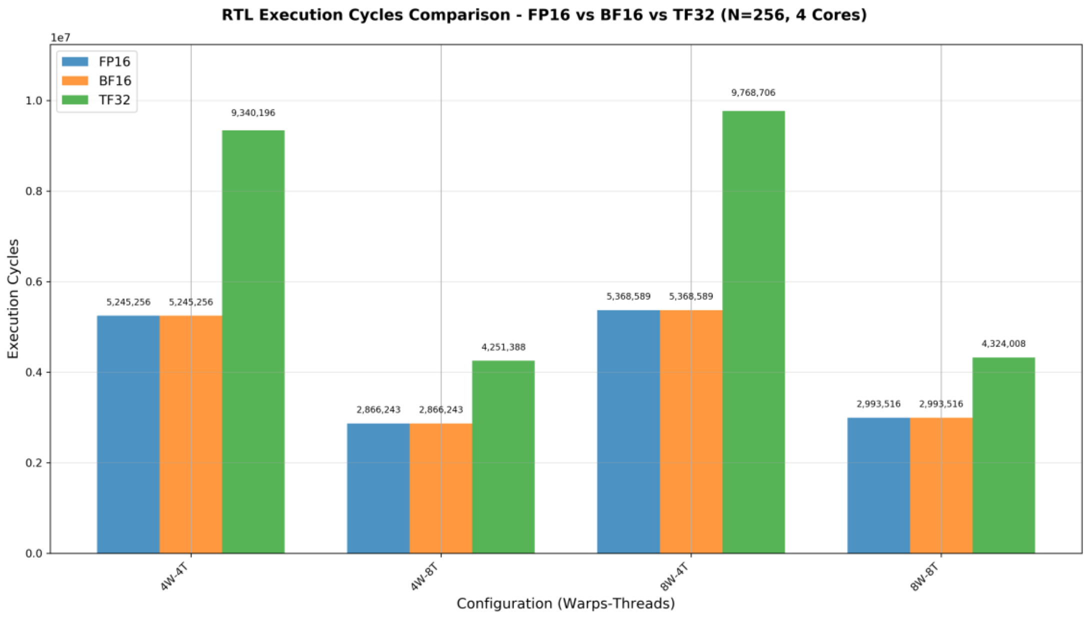
In the aspect of execution cycles, TF32 consistently shows higher execution cycles than FP16 and BF16. This is expected because TF32 has a 19-bit effective precision (1 sign, 8 exponents, 10 mantissa bits) packed into 32 bits, and in our SimX implementation it requires conversion from TF32 to full FP32 before performing the FMA operation. This conversion, together with the wider data path, increases both the arithmetic complexity and the dynamic instruction count, which directly translates into longer run times. In contrast, FP16 and BF16 share a very similar hardware execution path, such as multiplication operations and accumulating trees, so their cycle counts are almost identical across all configurations.

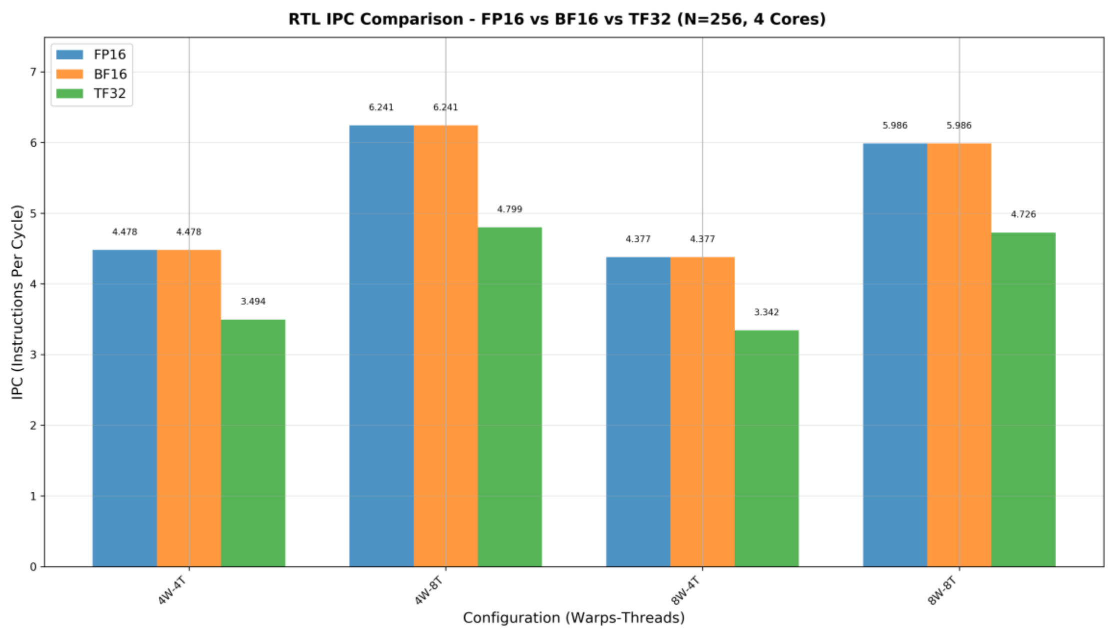
As for the IPC results, in terms of SIMX simulation, TF32 achieves an IPC very close to, and sometimes slightly higher than, FP16 and BF16. FP16 and BF16 show the identical IPC because of the analysis before.

Finally, comparing the four warp/thread configurations shows that increasing the number of threads per warp is generally more beneficial than simply increasing the number of warps. It means that higher thread counts help hide latencies better and improve the overall throughput of the tensor unit. Generally, 4W-8T shows the best performance considering hardware costs. TF32 offers higher numerical precision than FP16/BF16 at a moderate performance cost.

Results plots and analysis of RTL







Workload

- Matrix a: 256x256; Matrix b: 256x256; Matrix c: 256x256
- Input Type: TF32 vs. FP16/BF16
- Output Type: FP32
- Test Configurations: 4-core GPU
    - 4 warps × 4 threads
    - 4 warps × 8 threads
    - 8 warps × 4 threads
    - 8 warps × 8 threads

Detailed Analysis

The first chart reports total instruction counts. FP16 and BF16 curves almost coincide, while TF32 consistently has more instructions for each configuration. FP16 and BF16 share the same hardware execution path except for exponent/mantissa width, so their instruction counts are identical. The main difference between them is precision, not execution cost. Since we are required to use a full 32-bit register to store one element of TF32 instead of 2 likes for fp16, and TF32 multiplier interleaves its 4 outputs with zeros to generate 8 inputs for the next stage to share the same accumulator with fp16/fp16, it results in more dynamic instructions.

For the execution cycles, shown in the second graph, similarly, TF32 has noticeably higher cycle counts than FP16/BF16, because of the interleaving zeros resulting in smaller throughputs.

In the third graph of IPC, FP16 and BF16 achieve the highest IPC, while TF32 has noticeably lower IPC in all four cases. This shows that in RTL the TF32 pipeline not only executes more instructions, but also suffers from reduced per-cycle throughput, which is likely due to longer HardFloat latencies, additional pipeline stages, and the interleaving scheme in "VX_tcu_fedp_bhf.sv" that leaves half of the accumulation slots as zeros. Generally, considering the hardware costs, configuration 4W-8T achieves the best performance.