Tetris Hardware Evaluation Report

Background

Please refer to the TETRIS paper on NIPS'18

Y.Ji and et al "TilE-matching the TRemendous Irregular Sparsity." Part of: Advances in Neural Information Processing Systems 31 (NIPS 2018) pre-proceedings

The abstract of the paper is described as follows: "Compressing neural networks by pruning weights with small magnitudes can significantly reduce the computation and storage cost. Although pruning makes the model smaller, it is difficult to get practical speedup in modern computing platforms such as CPU and GPU due to the irregularity. Recent efforts on hardware-friendly pruning involve structured sparsity with different granularity and dimensionality. Simply increasing the sparsity granularity can lead to better hardware utilization, but it will compromise the sparsity for maintaining accuracy. In this work, we propose a novel method, TETRIS, to achieve both better hardware utilization and higher sparsity. Just like a tile-matching game, we cluster the irregularly distributed weights with small value into structured blocks by reordering the input/output dimension and structurally prune them. Results show that it can achieve comparable sparsity with the irregular element-wise pruning and demonstrate negligible accuracy loss. Ideal speedup, proportional to the sparsity, is experimentally demonstrated. Our proposed method provides a new solution toward algorithm and architecture co-optimization for accuracy-efficiency trade-off."

Architecture Description

Pros and Cons of the Reorder-based Block Sparsity. The benefit is obvious. Fewer computation since not all the weight block needs considering. In addition, fewer memory traffic to load weight and also load feature map. The only overhead is the "reordered" weight. As a result, for each partition of the weight (a weight block), it refers to a "un-continues" set of feature maps data. Functionally, the address of such "un-continues" data is calculated by a dedicated hardware according to the reordered weight block index. In terms of performance, the "un-continues" reading of the feature map changes the memory access pattern from sequential to random. This change will result longer latency and underutilized bandwidth (only part of the data in a cacheline is useful) for multi-bank/channel DRAM. However, if using on-chip multi-bank SRAM,

which is our case, it suffers from the risk of underutilized bandwidth due to potential bank conflict.

Overview of the Architecture. As shown in Figure-1, there are five components in the accelerator design, and they are the feature map buffer (Fmap-RAM), the address generator (AdrGen), the accumulation buffer (Accumulate RAM), the interconnection (NoC), and the processing unit arrays (Tiles, a.k.a. Pes). The connections of these components are through the NoC, which allows any components (including every single Tile or every single bank of the accumulator buffer when there are many) to transfer data among each other. One exception is the Fmap-RAM, which is connected to the AdrGen and then to the NoC. To transfer data in/out of the accelerator, there are two interfaces. One is the interface to the read-only DRAM through the NoC, and the other one is the interface the PCIe which is connected to the Famp-RAM.

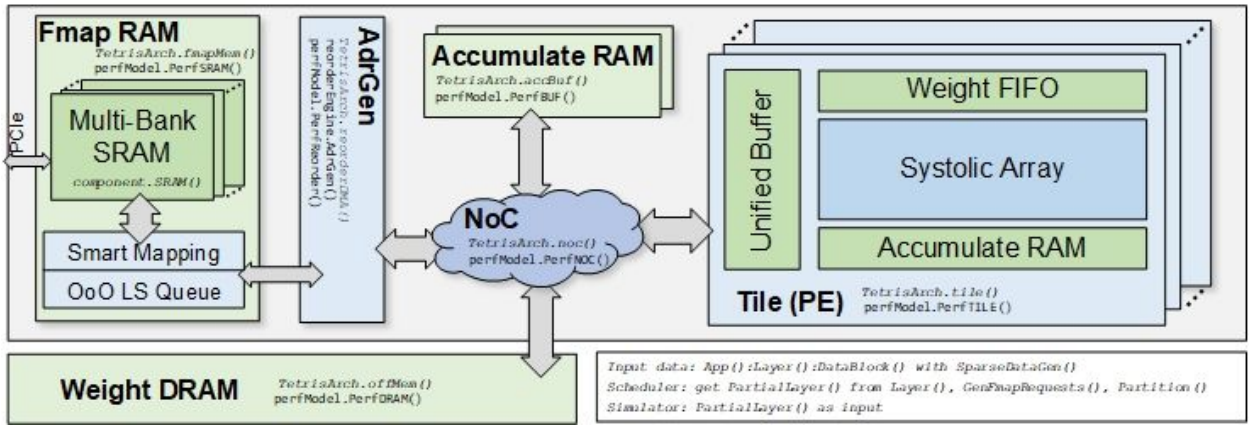


Figure 1. The overview of the hardware architecture.

Datapath and Workflow. The accelerator run a DNN inference task layer by layer. It loads the input data (image) from the PCIe to the Fmap-RAM once, before running the task. During the layer-by-layer computing, *all intermedium data (feature maps) are stored on-chip in Fmap-RAM*. The accelerator writes the results back after the task is done via the PCIe interface. When computing for one layer, there are 5 steps. Step-1, load weight data from the read-only DRAM to the Tiles. Each tile gets one “block” of weight. Step-2, each tile sends read requests of the input feature map data according to the weight block to the address generator. Step-3, the address generator calculates the physical address of the feature map, according to the index of the reordered weight block. Step-4, data is read from the multi-bank Fmap-RAM and sent to the Tiles who demands the data. Step-5, the Tile computes the matrix-matrix (vector) multiplication of the weight block and the requested feature map data, and then accumulate the results with data from either its last compute or reading from the Accumulate-RAM via NoC.

Step-1 to Step-5 runs iteratively until the all computation related to the loaded weight block is done. Then, the output feature map data are written back to the Fmap-RAM with the help of the AdrGen.

Tile (PE). One tile is set to compute matrix-matrix (vector) multiplication with one piece of a preloaded weight block and its related input feature maps. The hardware structure is adopted from TPUv1 design with the united buffer, the weight FIFO, the systolic array, and the accumulate RAM.

Accumulate RAM. It buffers the data to be accumulated. It is a single/multi-bank SRAM.

Fmap RAM. It buffers all the feature maps between layers. Note that the logic address of the feature maps is continues, however, unlike other accelerators, it is read randomly instead of sequentially. To maximize the bandwidth of the multi-bank SRAM to avoid bank conflict, two techniques are introduced. One is the smart mapping. It remaps the continues logic feature map address into a set of devices address (e.g., though a hash function). With the observation of the randomness of the access pattern, the devices address minimizes the bank conflict possibility. The other technique is the out-of-order reading. With a load queue, it priorities the requests that can uses the idle banks, in order to maximize the bandwidth of the Fmap-RAM. Note that we leave these two optimizations as future work.

AdrGen. It calculates the “random” feature map data address according to the weight block after reorder.

NoC. The topology of the NoC is flexible. It could be set as mesh, bufferfly, and any dataflow as it is discussed in Eyeriss-v2.

Task Mapping Guidelines

First, we run the DNN layer-by-layer. Second, within each layer, we may well need multiple passes. When the number of weight blocks in one layer (N_{wb}) is larger than the number of Tiles (N_{tile}) in the hardware, we run the first pass with the first N_{tile} weight blocks, then the next N_{tile} weight blocks as the second pass. We maximize the accumulation locality with partition the weight block into passes by priorities clustering weight blocks with the same output channels into the same pass. Third, for each pass, each of the Tile is set to run all related computing of a whole weight block. When a weight block (e.g., 64-by-64) is larger than the size of the systolic array in the Tile (e.g., 16-by-16), the tile runs multiply iterations to get the job done, instead of using multiple tiles to compute for one weight block. Note that if the weight block size is small than the systolic array size, we cannot map multiple weight block on one Tile, and then the hardware is underutilized.

Simulator Description

We adopt a mixed approach for the simulator: it is overall a roofline-based analytic model while with the trace-driven simulation for the random access on Fmap-RAM. Specifically, taken the workload from the mapping of an input DNN, the simulator calculates the throughput (note, not latency) and energy consumption of all the components with the performance, power, and area (PPA) parameters from circuit evaluation. Note that for NoC, following the approach in MESTRO and eyerissv2, we model it with a user set average throughput with zero latency. The modeling of the Fmap-RAM is an exception. Instead of the simple analytic mode, it reads the real address traces and figure out the latency (effective bandwidth) with the cycle-by-cycle consideration of bank conflicts. Finally, the accelerators' overall effective performance is modeled by a roofline model of all the components performance.

For the circuit evaluation, we get the PPA of the computing units from DC systhesis while the SRAM from CACTI.

Please refer to the hints in Figure-1 and the comments in the code for the detail.

Evaluation Results

Experiment Setup

We configure the hardware as:

- 700MHz clock, 28nm technology
- Tiles: 512, 4x4 systolic array with INT8 arithmetic
- Fmap-RAM: 16 banks, 512KB per bank, 8B width ([future-work] fine tune the circuit parameter)
- Acc-RAM: 8 banks, 256KB per bank, 16B width
- NoC: 2GB/s average bandwidth
- DRAM: DDR4-2666, single channel, 19.2B/s

We run the following workloads:

- VGG8, VGG16
- [future-work] Resnet, MLP

We run different sparsity setup as:

- Dense: in order to show how sparsity benefits
- Sparse: block sparsity with 4x4 block size and sparsity setting layer-by-layer in the TETRIS paper, to study the relationship between sparsity, block size, and the performance.
- [future-work] Elementwise: to show the benefit of the structured sparsity

We evaluate the following design options:

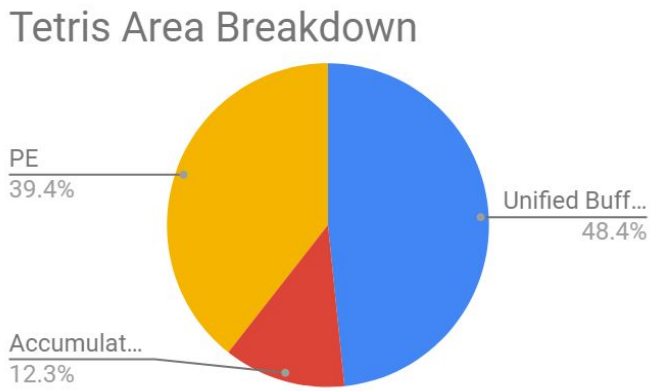
- vanilla: storing feature map in Fmap-RAM with continues address without remap the logic address to the device address and also without OoO load queue
- [future-work] w/o ooo: only with remap but no OoO load queue
- [future-work] Proposed: with remap and OoO load queue for Fmap-RAM to migrate random access overhead
- [future-work] Ideal: without any back conflict for Fmap-RAM, to see the overhead of the reorder

[future-work] We exam the scalability/sensitivity by explore the following configurations:

- Tile size
- Tile number

Area evaluation.

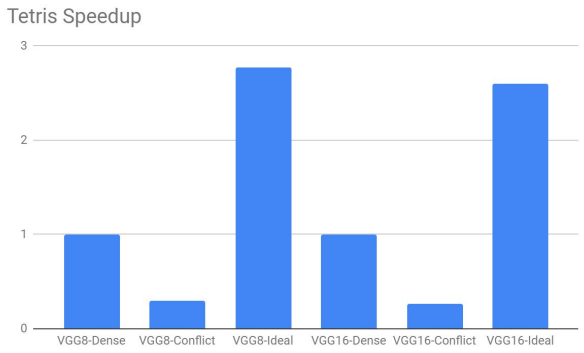
The total area footprint in 28nm technology is 9.987mm^2, the breakdown is shown below.

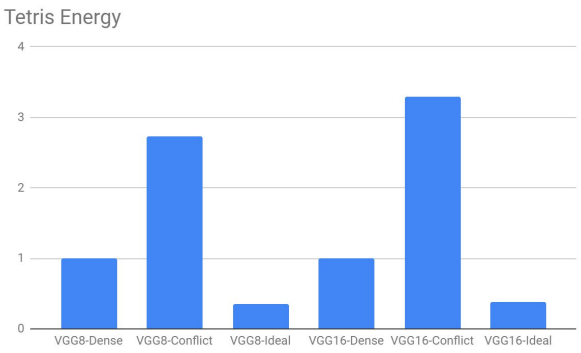


Benefits of TETRIS.

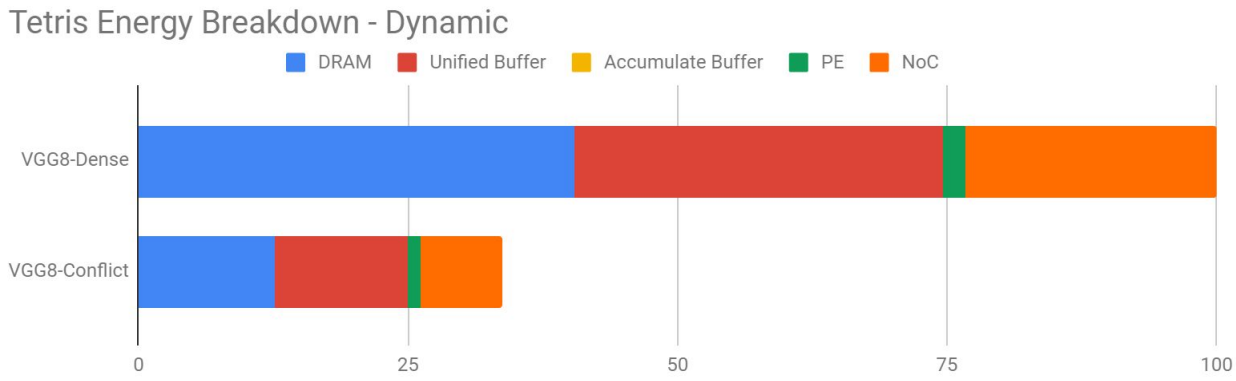
We show the speed up and energy consumption of the proposed hardware. We show three designs (1) VGG-dense is the baseline design which does not explore any sparsity; (2) VGG-conflict is the vanilla design which leverages the block-based sparsity but does not adopt any techniques to deal with the reordering overhead from the bank-conflict. This is the lower bound of the hardware design. (3) VGG-idea is the speed-of-the-light design which embraces the block sparsity benefit with zero overhead, by assuming no bank conflict. This is the upper bound of the hardware design. Our proposed design should seat between the (2) and (3) with the techniques described in the architecture section, but we leave the quantitative evaluation as the future work.

The conclusion from the current results are: (1) the proposed TETRIS method can achieve up to 2.7x speed up and 65% energy saving on accelerators; (2) however, without proper method to mitigate the reorder overhead, it total offset the benefit from the sparsity (0.3x slowdown and 229% more energy), urging future architecture solutions to solve the bank conflict overhead.





We also provide dynamic energy breakdown data to zoom in and figure out where the benefit of TETRIS comes from. The dynamic energy saving comes from both the memory access and the computation saving, while the former dominates. The saving on on-chip SRAM contributes the majority part of 37% out of the 65% energy saving, while the saving from the off-chip DRAM contributes another 27%.



Appendix-1: Future Work

This is a preliminary evaluation for the hardware support of TETRIS. Future work includes (1) techniques to mitigate the bank conflict and hence reduce the reordering overhead; (2) studies of architecture design chose to make most benefit from the TETRIS; (3) insights of block-size and sparity selection given a set up of hardware by extensive experiments.

Appendix-2: To-do-list for Better Evaluations

Benefits of TETRIS.

[Todo] fixed configuration of (1) hardware, (2) design option of “proposed”

x-axis: benchmark, i.e., VGG, MLP, etc

y-axis: (fig-1) speedup, (fig-2) energy [with stack, energy breakdown of SRAM, DRAM, and PE], (fig-3) hardware unitization, (fig-4) SRAM/DRAM effective bandwidth

legends: (1) dense (2) element w/ xx% sparsity (3) xx-block w/ same sparsity

Reorder overhead.

[Todo] fixed configuration of (1) hardware, (2) block size and sparsity

x-axis: benchmark, i.e., VGG, MLP, etc

y-axis: (fig-1) speedup, (fig-2) energy

legends: design options, i.e., vanilla, w/o OOO, proposed, and ideal

Impact of the sparsity.

[Todo] fixed configuration of (1) hardware, (2) design options, i.e., proposed, (3) block size,

x-axis: (in-group) VGG, MLP, (between group) sparsity

y-axis: (fig-1) speedup, (fig-2) energy

legends: (1) blocked (2) elementwise

Impact of the block size.

[Todo] fixed configuration of (1) hardware, (2) design options, i.e., proposed, (3) sparsity

x-axis: (in-group) VGG, MLP, (between group) blocksize

y-axis: (fig-1) speedup, (fig-2) energy

legends: (1) blocked (2) elementwise

Scalability.

[Todo] fixed configuration of (1) design options, i.e., proposed, (2) sparsity and blocksize

x-axis: (in-group) systolic array size, (between group) number of tile

y-axis: (fig-1) speedup, (fig-2) energy

legends: benchmark