LSU EE 7722 **Homework 3** Solution **Due: 1 May 2023**

Note: late submissions accepted without penalty until 5 May.

In class we spent much of the semester showing how recent generation GPUs can be used for matrix/matrix multiplication, though understanding that they were designed for a larger class of computations. In this assignment we will look at some accelerators designed specifically for machine learning workloads that are matrix/matrix multiplication. These include Google's TPU chips (which are used in production) and an idea proposed in the research literature, *Scale-Out Systolic Arrays (SOSA)*. Both designs use systolic arrays, of course. As mentioned in class, the computation of matrix/matrix multiplies using systolic arrays requires far less register reads and writes than the same computation performed on a device that reads operands for each FMADD instruction from a register file.

For this assignment, and to prepare for the final exam read the following papers. First, read the papers describing Google's TPU accelerators, all of which use large systolic arrays. The first paper, Jouppi 17 [3], describes Google's first TPU, now called TPUv1. Though TPUv1 is primitive compared to later designs, the paper does provide more detail, and so will be the basis of some questions in this assignment. The TPUv1 systolic array was limited to integer arithmetic, limiting its use to inference. Google's next GPUs TPUv2 and TPUv3 were designed to handle both inference and training. To support training these chips use systolic arrays that operate on 16-bit floating-point data, the BF16 (brain-float) format. These are described in several papers, see [2, 4]. TPUv4i, designed just for inference, is described in a fourth paper [1].

Also read the SOSA paper, [5], in which a design using smaller systolic arrays in proposed. The paper shows how the lower energy efficiency of the smaller systolic arrays can be overcome by better utilization when the systolic arrays are appropriately interconnected and a more meaningful metric, according to the paper, is used.

You should be able to get copies of all of these papers for free on campus. Off campus you might be asked to pay. Please E-mail me if you have any problems getting a free copy.

Problem 1: Answer the following questions about Google's TPUs and described by Norman Jouppi and his many collaborators. Several of the questions below are based on the TPUv1 paper [3]. This paper shows data bandwidth in units of GiB/s. Be aware that $1\,\text{GiB} = 2^{30}\,\text{B}$. Some of the questions below ask about the number of clock cycles or to reason about clock cycles with other information provided in GiB/s. To answer such questions one needs to use the clock frequency, which for TPUv1 is $\phi = 0.7\,\text{GHz}$.

(*a*) In TPUv1 how many clock cycles would it take to load one set of weights (one tile) from the off-chip memory (DDR3 DRAM Chips) into the Weight FIFO? Helpful information is in Figure 1 in [3].

Each weight is 8 bits—one byte—and there are 2^{8+8} of them for a total $64\,\text{kiB}$. The bandwidth between the off-chip weight DRAM and the TPU chip is $30\,\text{GiB/s}$, and so is the bandwidth between the off-chip interface and the Weight FIFO. The time then to move $64\,\text{kiB}$ of data is

$$\frac{64\,\text{kiB}}{30\,\text{GiB/s}} = 21.33 \times 2^{-20}\,\text{s} = 21.33 \times 2^{-20}\,\text{s}\,\frac{0.7 \times 10^9\,\text{cyc}}{\text{s}} \approx 1424\,\text{cyc}$$

.

(*b*) The paper explains on page 3 that it takes 256 cycles to load the weight matrix. (That's not the answer to the subproblem above.) Suppose the array were 128×512. How long would it take to load the weight matrix from the FIFO?

Weights are loaded one row per clock cycle. The top row of cells reads weights from the weight FIFO, in other rows a cell reads a weight from the cell above (a cell in row r, column c, reads its weight from the cell in row $r-1$, column c). A cell will place the weight arriving from the cell above into one of its (presumably two) weight buffers. Proceeding in this way weights move down at a rate of one row per clock cycle. The process stop when weights arrive at the bottom row.

A 128×512 array has 128 rows and 512 columns and so it would take 128 cycles to load the weights. That's assuming the weight FIFO can provide 512 weights per cycle.

(*c*) Figure 1 shows a bandwidth of 167 GiB/s between the Unified Buffer and the MXU, but off-chip bandwidth for activations is only 10 GiB/s. Why the imbalance? That is, how is it possible that enough data can be provided to the MXU from off chip to keep it fed at 256 bytes per cycle?

Call the data arriving to the TPU chip *network inputs*. A network input is used to compute a *hidden layer*, and the hidden layer may be used to compute subsequent hidden layers. Call the output of the last layer the *network output*. The hidden layers and network outputs would all be computed by the systolic array. So in effect, data arriving at the TPU would pass through the systolic array multiple times.

In the time for one input (one row of an input matrix) to arrive $\frac{128}{10\,\mathrm{GiB/s}} \approx 12\,\mathrm{ns}$ plus the time for one network output to be sent back to the host, another $12\,\mathrm{ns}$, the systolic array can shift in

$$2\frac{128}{10\,\mathrm{GiB/s}}\phi = 2\frac{128}{10\,\mathrm{GiB/s}}\frac{0.7\times10^9\,\mathrm{cyc}}{\mathrm{s}} \approx 16.7$$

rows. So 16 hidden layers (plus the output layer) would balance the bandwidths under these assumptions. Other factors can increase the number of times the systolic array is used, such as computing multiple hidden layer *channels*.

(*d*) To compute the large layer on Homework 1 and 2 we need to multiply a 1536×512 matrix by a 512×2970 matrix. Roughly how long would that take on TPUv3 (not TPUv1 this time)? Explain any assumptions.

In our homework assignments the $1536\times512 = d_{\mathrm{qkv}}\times d_{\mathrm{model}}$ matrix, w_{qkv}, was for the weights and the $512\times2970 = d_{\mathrm{model}}\times n_{\mathrm{umm}}$ matrix, h_{in}, was the inputs. The problem in the assignment was described as computing $h_{\mathrm{qkv}} = w_{\mathrm{qkv}}\times h_{\mathrm{in}}$, and so the weights were first. In the TPU systolic arrays the second matrix of the product $A\times B$, is stored in the array (and that array, B here, is called the weights). So we would need to compute $h_{\mathrm{in}}^T \times w_{\mathrm{qkv}}^T$ (the transpose of the input matrices).

With the weight matrix being 1536×512 and the TPUv2 systolic array being 128×128 the weights need to be broken into $\frac{1536}{128}\times\frac{512}{128} = 12\times4 = 48$ tiles. After a weight tile is loaded, 2970 partial columns of h_{in} (or 2970 partial rows of h_{in}^T) are streamed through the systolic array. That is done for each tile, so a total of $48\times2970 = 142560$ partial columns are streamed through.

The size of w_{qkv} is $1536\times512 = 786432$ elements. Though in our assignments the elements were floats, 32 bits, lets assume here that they are 16 bit types, say BF16. Then the total size is $1536\,\mathrm{kiB}$ which can very comfortably fit in the $16\,\mathrm{MiB}$ of vector memory in a TPU core. The size of h_{in} and h_{qkv} are a tad above $2.9\,\mathrm{MiB}$ and $8.7\,\mathrm{MiB}$, respectively. So the inputs and outputs can also be comfortably buffered in the vector memory of one core. Since the problem was for TPUv3 we'll use both cores. For simplicity we'll place the first 1485 columns of h_{in} into one core, and the remaining columns in the other.

The papers are not clear on how one core uses two MXUs. It would be trivial if there are two separate simultaneously active threads, one for each MXU, but such a core would be more like two cores if each thread could read and write every register. Assume instead that sublane 0 is for one MXU and sublane 1 is for the other and that those 4-D DMA operations can shuffle the two input streams into a contiguous batch for the vector loads to grab (remembering that a single vector load instruction is executed by the 128×8 lanes and sublanes).

First, lets also assume for simplicity that weights for one computation can be loaded while another is being computed. (This capability was explicitly claimed for TPUv1, but not the others.) With all of these observations and assumptions we can compute the time it takes to shift in $48\frac{1}{4}2970$ partial columns of h_{in} (which the TPU will see as partial rows of h_{in}^T). At a clock frequency of $940\,\mathrm{MHz}$ that will take

$$48\frac{1}{4}2970 = 35640\,\mathrm{cyc} = 35640\,\mathrm{cyc}\times\frac{\mathrm{s}}{940\times10^6\,\mathrm{cyc}} = 37.9\,\mu\mathrm{s}$$

If each tile is loaded once and weights cannot be loaded during a computation, then the time for loading the tiles is $\frac{48}{4}128 = 1536\,\mathrm{cyc}$, which would only add a few percent to the execution time.

Problem 2: Yüzügüler *et al* describe an accelerator in which *pods* each containing a 32×32 systolic array are interconnected by carefully chosen networks.

(*a*) The paper notes that the utilization of the systolic arrays depends on both the bisection bandwidth and the latency of the network connecting the pods to storage for weights, activations, and partial sums. Suppose that each SM in an Nvidia H100 had a pod with a 32×32 systolic array, along with the skewing and de-skewing hardware. The systolic arrays would read and write data from the H100 L2 cache, and

perhaps make use of the L1. What would the utilization of the systolic arrays in this setup be based on the H100 latency and throughputs shown below collected using the microbenchmarks described in a previous homework. Be sure to state assumptions made in computing your answer.

A SoSA systolic array reads 32 bytes per cycle. To sustain operation of a systolic array would require also 32 bytes of weights per cycle in (to load the next tile) and 32×2 bytes out, the 16-bit sums. The total amount of data, counting both directions, is 128 bytes per cycle. Based on the microbenchmarks an H100 can sustain about 48 bytes per cycle per SM from the L2 cache, which would not be enough to sustain 100% utilization. (The benchmark shows 48.6 bytes per cycle but 48 may be the actual number.) Relying only on L2 the utilization would be $\frac{48.6}{128} = 37.5\%$ based on throughput.

Since outputs are 16 bits, it would make sense to consider an output-stationary arrangement where partial sums are kept on an SM, perhaps in shared memory. Then the throughput needed would be 64 bytes per cycle and utilization would be $\frac{48.6}{64} = 75\%$ based on throughput.

```
GPU 0: NVIDIA H100 PCIe @ 1.75 GHz WITH 81089 MiB GLOBAL MEM
GPU 0: L2: 51200 kiB   MEM<->L2: 2039.0 GB/s
GPU 0: CC: 9.0  SM: 114  SP-FP32/SM: 128  DP-FP64/SM: 64  TH/BL: 1024
GPU 0: SHARED: 49152 B/BL  233472 B/SM  CONST: 65536 B  # REGS: 65536
GPU 0: PEAK: 25609 SP GFLOPS  12804 DP GFLOPS  COMP/COMM:  50.2 SP  50.2 DP

                     --Insn-- ------L1 Per SM------- -L2 GB/s-- DRAM
Lv Blk V SPL      s /add  %  SW TW  BXW B/cyc  GB/s   /SM /GPU GB/s
ME 114 1  s0     280  2.9  5   4  1  0.0  10.9  17.8  17.8 2025 2023
L2 114 1  s0     503  2.8 18   4  1  0.0  48.6  66.9  66.9 7629   78
L1 114 2  s0     464  1.9 46   8  2  0.0 133.1 225.9   0.1    8    8
L1 114 1  s0     467  2.5 61   4  1  0.0 131.9 224.7   0.1    8    8
L2 114 1  s0     507  2.8 18   4  1  0.0  48.2  66.5  66.5 7581   77
L2  57 1  s0     369  2.8 13   4  1  0.0  54.5  91.3  91.3 5206   99
L2   2 1  s0     561  2.8  0   4  1  0.0  34.7  60.1  60.1  120   60
L2   1 1  s0     513  2.8  0   4  1  0.0  37.6  65.7  65.7   66   66

Kernel mb_g:
             ---Data Touched--- --------------------Latency----------------------
 nbl    iter    Block     Total   ns  cyc !<--------------500 ns-------------->!
 512   20480  512 kiB   256 MiB  394  692 ****************************
 512   10240  256 kiB   128 MiB  396  695 ****************************
 256   10240  256 kiB    64 MiB  391  686 ****************************
 128   10240  256 kiB    32 MiB  147  257 ***********
  64   10240  256 kiB    16 MiB  139  244 **********
  32   10240  256 kiB     8 MiB  139  244 **********
  16   10240  256 kiB     4 MiB  144  252 **********
   8   10240  256 kiB     2 MiB  144  252 **********
   4   10240  256 kiB     1 MiB  143  251 **********
   2   10240  256 kiB   512 kiB  150  263 ***********
   1   10240  256 kiB   256 kiB  150  262 ***********
   1   10000  128 kiB   128 kiB   37   65 **
   1   10000   64 kiB    64 kiB   28   50 **
   1   10000   32 kiB    32 kiB   24   42 *
   1   10000   16 kiB    16 kiB   22   38 *
   1   10000    8 kiB     8 kiB   21   36 *
```

References:

[1] Jouppi, N. P., Hyun Yoon, D., Ashcraft, M., Gottscho, M., Jablin, T. B., Kurian, G., Laudon, J., Li, S., Ma, P., Ma, X., Norrie, T., Patil, N., Prasad, S., Young, C., Zhou, Z., and Patterson, D. Ten lessons

from three generations shaped googles TPUv4i : Industrial product. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)* (2021), pp. 1–14. http://dx.doi.org/10.1109/ISCA52012.2021.00010.

[2] Jouppi, N. P., Yoon, D. H., Kurian, G., Li, S., Patil, N., Laudon, J., Young, C., and Patterson, D. A domain-specific supercomputer for training deep neural networks. *Commun. ACM 63*, 7 (June 2020), 6778. https://doi.org/10.1145/3360307.

[3] Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., Ghaemmaghami, T. V., Gottipati, R., Gulland, W., Hagmann, R., Ho, C. R., Hogberg, D., Hu, J., Hundt, R., Hurt, D., Ibarz, J., Jaffey, A., Jaworski, A., Kaplan, A., Khaitan, H., Killebrew, D., Koch, A., Kumar, N., Lacy, S., Laudon, J., Law, J., Le, D., Leary, C., Liu, Z., Lucke, K., Lundin, A., MacKean, G., Maggiore, A., Mahony, M., Miller, K., Nagarajan, R., Narayanaswami, R., Ni, R., Nix, K., Norrie, T., Omernick, M., Penukonda, N., Phelps, A., Ross, J., Ross, M., Salek, A., Samadiani, E., Severn, C., Sizikov, G., Snelham, M., Souter, J., Steinberg, D., Swing, A., Tan, M., Thorson, G., Tian, B., Toma, H., Tuttle, E., Vasudevan, V., Walter, R., Wang, W., Wilcox, E., and Yoon, D. H. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2017), ISCA '17, ACM, pp. 1–12. http://doi.acm.org/10.1145/3079856.3080246.

[4] Norrie, T., Patil, N., Yoon, D. H., Kurian, G., Li, S., Laudon, J., Young, C., Jouppi, N., and Patterson, D. The design process for google's training chips: TPUv2 and TPUv3. *IEEE Micro 41*, 2 (2021), 56–63. http://dx.doi.org/10.1109/MM.2021.3058217.

[5] Yüzügüler, A. C., Sönmez, C., Drumond, M., Oh, Y., Falsafi, B., and Frossard, P. Scale-out systolic arrays. *ACM Trans. Archit. Code Optim. 20*, 2 (mar 2023). https://doi.org/10.1145/3572917.