LSU EE 7722 **Homework 6** **Due: 3 May 2025**

Solution *Formatted 11:10, 9 May 2025*

This assignment describes some important accelerators, but papers on only two of them need to be read, the Google TPUv1 and the AttAcc PIM system.

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. This week we will look at some accelerators designed specifically for machine learning workloads that are dominated by matrix/matrix multiplication. These include Google's TPU chips, Cerebras wafer-scale systems, and Tesla's DOJO. Google's TPU chips have been developed over several generations so one can assume that they are a good approach for accelerating ML workloads. Cerebras is a newer product, so one must wait a few years to see if its approaches are truly useful.

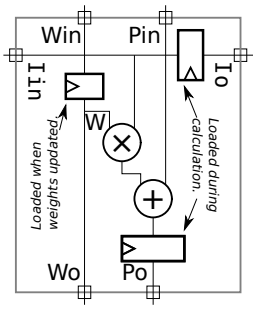
The NVIDIA GPUs, the Google TPUs, Cerebras' systems, and DOJO each take a different approach to accelerating these workloads.

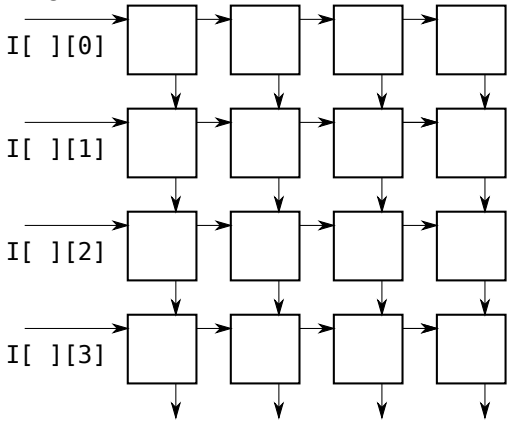
One way to look at them is by the path an operand takes to a functional unit. In the NVIDIA GPUs and operand's starting point can be DRAM, the L2 cache, the L1 cache (or shared memory), or a register. First consider a register. Each thread can read up to 255 registers. So the hardware must provide a path for each of those 255 registers to the operand ports of an FP32 unit (plus other functional units). The amount of energy it takes to move an operand from a register to the functional unit increases with the number of registers. So, if a thread could only access 16 registers the energy needed to move an operand from a register to a functional unit would be lower. Energy is often a limiter, for the obvious reason that electricity isn't free, but also because it determines an upper limit on clock frequency beyond which the chip can't be kept from overheating.

As we've noted in class, with more registers one can hide more latency, which may be one reason NVIDIA GPUs have so many compared to CPU ISAs. (To make a fair comparison in a GPU one must consider the maximum number of registers accessible by all warps, since the register file must provide a register for all of them. For a CPU there are two other factors. First, Intel-64 implementations have two contexts. So, that doubles the number of registers. Another factor is that many Intel cores are dynamically scheduled, so when comparing the number of registers one must look at the size of the physical register file. To keep things simple we won't consider CPU ISAs here.)

The GPU L1 cache is larger (and varies by device). Energy per access is increased by the size itself and the fact that one must do a tag-store lookup to see if the data is cached. Even more energy is needed if the data is brought from the L2, and more from DRAM.

What sets the different accelerators apart then is the path taken by an operand. The path taken by an operand is simplest in a systolic array. (For a detailed description of systolic arrays used in DNN accelerators see [12].) A systolic array consists of simple *processing elements* connected into a 2D mesh:





A systolic array can be used to compute many different things, the one illustrated above can be used for matrix/matrix multiplication. Notice that the value provided to each functional unit (multiplier or adder) is read from one register. That takes much less energy than reading one of many values from a register

file. Clearly one needs more than just a systolic array to make an accelerator. But the approach used by Google's TPUs is to have most of the computation done by large systolic arrays, realizing significant energy savings. Google's original TPU had one 256×256 systolic array capable of operating on 8b integer values, accumulating to 32-bit values [5]. Later designs had two or four 128×128 systolic arrays, augmented by 128-element vector units [4]. *Note from the future: TPU v6e once again uses* 256×256 *systolic arrays.*

Intermediate between the conventional (relatively) memory hierarchy of the NVIDIA GPUs and the minimalist systolic array, is an old idea: a mesh (2D array) connection of simple processing elements, but not as simple as those in a systolic array.

Two examples are the Cerebras wafer-scale systems [7] and Tesla's DOJO [10]. These will be discussed in class next week (the week of 29 April 2024).

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 [5], 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. That is, you need to understand that paper fairly well. 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 [4, 8], concentrate on [4]. Google has developed a TPUv4, those who are curious can read [3] and [2], but reading those papers is not required. The systolic array has been increased to 256×256 in TPUv5 and TPUv6, and TPUv7 was announced for 2025.

Several things set Cerebras systems apart from other accelerators. First, they span an entire wafer, not just a chip. (Normally chips are fabricated on wafers, which are then cut up into chips. In a wafer-scale system the wafer is not cut up, the entire wafer is used.) The Cerebras machines will not be covered in details this semester (2025), but for those who are curious see [7].

CPUs and GPUs do well on code with higher computational intensity (ratio of number of FP operations to number of operands). Consider code computing `a[i] = b[i] + c[i]`. Its computational intensity is $\frac{1}{3}$, that of a GPU is over 100 for SP, so the FP units will be idle most of the time. It would not be possible using current technology to increase data bandwidth of CPUs and GPUs to the point where they could sustain execution of code at $\frac{1}{3}$.

One idea for handling computations with low computational intensity that has been considered for decades is performing the computation near the memory, perhaps even on the DRAM chips, a technique called *processing in memory (PIM)*. Consider the computation `a[i] = a[i] * s`. To perform the computation `a` is read from memory to the GPU (or CPU) and the updated `a` is written back. The time bound is twice the array size divided by the data bandwidth. DRAM chips contain a hierarchy of structures, at or near the lowest level are *banks*, where the data is stored. A single DRAM chip can contain many banks (say 128). When reading from the chip only one of these may provide the data, which sort of wastes the remaining bandwidth. To compute `a[i]=a[i]*s` one might place compute units near each bank and send a command to each bank to start the computation. Each bank would read its part of the `a` array, and so the amount of data movement would be 128× what is possible for a device off the DRAM chip.

That huge potential is the promise. There are several problems. One is that some kind of a compute device would need to be put near each bank (or at a place higher in the hierarchy). They would probably be idle most of the time. Another issue is that digital logic takes up more space on a DRAM chip than it does on CPU and GPU chips. For that reason these *processing elements* must be kept very simple. For one, they usually only can get data from the DRAM bank they are near or from commands sent from a CPU or GPU. That's okay for computing `a[i] = a[i] * s` because the `a` are local to the bank and only one external piece of data is needed, `s`. A PIM would have much more trouble computing `a[i] = b[i] + c[i]` because `b` and `c` would be in different banks, and the result might need to be written to a third. In principle it is possible to interleave `a`, `b`, and `c` so that, say `a[7], b[7], c[7]` are all in the same bank, but that might make things more difficult for the prior step in the computation that wrote `b` and `c` and the next step that reads `a`.

This is why PIM systems are research topics rather than commercial products. (There have been brave

companies trying to find a market, see [6, 1].) One important computation that is driving recent PIM activity is the attention computation in transformer models [11]. The computationally tricky part is a matrix/vector multiply. That happens during generation (of new tokens) when a *query*, the vector, needs to be multiplied by a matrix of *keys*, one key for each token in the context. The size of the key matrix can be huge can can't practically be reused. (A key will be used for each subsequent generated token, but the amount of data passing through the system between the first use of a key and its next use is huge so a key can't be sitting around in a cache, much less a register, waiting for the next use.)

One interesting paper describing using a PIM for the attention computation is Park, *et al* [9]. The paper contrasts the reuse possible in some parts of the a transformer computation (the fully-connected [FC] networks used to create the keys, and also queries and values, and FC networks for other purposes) with computing attention.

Read Park 24 [9], focus on the introduction and try to understand what the GEMV units do and where they might be placed. This will be discussed further in class.

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 as described by Norman Jouppi and his many collaborators. Several of the questions below are based on the TPUv1 paper [5]. (These questions were also asked in 2024.)

In the matrix multiply code used in class (in `mm.cu`) we worked with two sizes.

(*a*) Describe the problem with multiplying matrix A, 96×32, by matrix B, 32×29700 on the TPUv1. Matrix elements are 8 bit integers, so data type is not the problem.

There is a minor problem and a major problem. The minor problem is that matrix B is much too large to fit on the 256×256 systolic array. But A can, so instead of computing AB, compute $(B^{\mathrm{T}}A^{\mathrm{T}})^{\mathrm{T}}$. So long as it is easy to get the transposed arrays, computing the product of the transposes is not a problem.

The major problem is that A is too small to use the entire systolic array. The simplest solution is to put one copy of A^{T} on the systolic array. But that would occupy $\frac{32\times96}{256\times256} = \frac{3}{64} \approx 4.69\%$ of the array. We could do better by putting two copies of A^{T}, one starting at row 0, column 0, the other at row 32, column 96, which would double occupation to 9.38%. Not great but twice as good as before. In the parts below utilization is based on the entire computation, accounting for the time to start streaming the results in and draining the results out. Because B has lots of columns (and so B^{T} has lots of rows) utilization will be close to occupation.

☑ Which matrix should be the weights? (Note that either A or B could be considered the weights.)

As described above, A^{T}, the transpose of the A matrix.

☑ How many cycles will it take to complete the multiplication?

If just one copy of A were used $29700 + 32 + 96 = 29828$ cycles. If two copies were placed in the systolic array then two rows of B^{T} would enter each cycle, and so the time would be $29700/2 + 32 + 2 \times 96 = 15074$ cycles.

☑ What fraction of the systolic array will be utilized?

Based on the cycles given above for when one copy of A is on the array, the utilization in $\frac{29700\times32\times96}{(29700+32+96)256\times256} \approx 4.67\%$. When there are two copies $\frac{29700\times32\times96}{(29700/2+32+2\times96)256\times256} \approx 9.24\%$.

(*b*) Describe the problem with multiplying matrix A, 1536×512, by matrix B, 512×2970 on the TPUv1. Matrix elements are 8 bit integers, so data type is not the problem.

In this case both matrices are larger. The A matrix dimensions are integer multiples of the systolic array dimensions, so again put A^{T} in the systolic array. But to do this we need to do the computation in $\frac{1536}{256} \times \frac{512}{256} = 6 \times 2 = 12$ steps.

☑ Which matrix should be the weights? (Note that either A or B could be considered the weights.)

The A matrix again, transposed again.

☑ How many cycles will it take to complete the multiplication?

Multiplying a single 2970×256 tile of B by a 256×256 tile of A would take $2970 + 256 + 256$ cycles. If the tiles are not overlapped, meaning the second tile does not start until the first tile completely finishes the time to multiply all $6 \times 2 = 12$ tiles of B would take $(2970 + 256 + 256) \times 6 \times 2 = 41784$ cycles. This computation also ignores the time to load the weights.

☑ What fraction of the systolic array will be utilized?

If it took 41784 cycles to multiply the arrays then utilization would be $\frac{2970 \times 512 \times 1536}{41784 \times 256 \times 256} = 85.3\%$. If the tiles could be overlapped then the utilization would be much closer to 100%.

Problem 2: Do your best to answer the questions below about AttAcc described in Park 24 [9], if necessary make guesses of numbers, such as a clock frequency.

(*a*) The paper describes three configurations, buffer, BG, and bank. For each, compute the maximum computation rate of the GEMV units in FP operations per second. Base this on the descriptions of the GEMV units, including how many there are, what they can do, and the clock frequency.

The AttAcc system analyzed consists for 40 HBMs.

Each HBM consists of 8 dice, and each die has $8 \times 4 = 32$ bank groups (BG) and four banks per BG or $8 \times 4 \times 4 = 128$ banks. So each HBM has $8 \times 32 = 256$ bank groups and $8 \times 128 = 1024$ banks. Finally, an AttAcc has $40 \times 256 = 10240$ bank groups and $40 \times 1024 = 40960$ banks.

Each GEMV has 16 multiply units and 16 add units and is clocked at 666 MHz for a rate of $16 \times 666\,\text{MHz} = 10.6\,\text{GFLOP/s}$, counting a multiply and add as one operation as we've done in class.

In Att_{BG} there is one GEMV per BG and so the aggregate FP rate is $10240 \times 10.6\,\text{GFLOP/s} = 109.117\,\text{TFLOP/s}$ assuming all GEMVs can sustain simultaneous operation without overheating. In Att_{bank} there is one GEMV per bank and so the aggregate FP rate is $4 \times 10240 \times 10.6 = 436.470\,\text{TFLOP/s}$ again assuming all GEMVs can sustain simultaneous operation without overheating. The end of Section 4.1 describes the number of GEMVs operating in parallel as 40960, though elsewhere they mention power constraints that would limit the number of simultaneously operating GEMVs. For this problem the higher number was used.

(*b*) To compute `score[i] = dot( query, key[i] )` one needs to send `query` to each GEMV unit (assume column-wise partitioning). Suppose there are m GEMV units. If the xPU had to send `query` to each GEMV unit individually the amount of data sent would be md, where d is the number of elements in query. Given the description in the paper, is that much data sent? Note that if the length of the context were L then to compute the scores in an xPU the amount of data moved would be Ld elements, so if $L \approx d$ there would not be much savings.

No, not that much. The xPU communicates with an AttAcc controller, which itself communicates with multiple HBMs. Assuming one controller manages all 40 HBMs, the xPU would send the d-element query just once.

An HBM controller sends commands to the HBM dice. Whether the query would be sent once or once per GEMV is ambiguous. The end of Section 5.1 describes commands that an HBM controller can send to its dice. Command `PIM_WR_GB` writes data into a GEMV buffer. The paper isn't clear whether it writes the same data into the same buffer address on every GEMV in the HBM or just one of them. Other commands, such as `PIM_ACT_AB` operate on all GEMV units in an HBM. Given how important broadcasting data is to performance it's likely that `PIM_WR_GB` can write every GEMV's buffer.

References:

[1] Devaux, F. The true processing in memory accelerator. In *2019 IEEE Hot Chips 31 Symposium (HCS)* (2019), pp. 1–24. http://dx.doi.org/10.1109/HOTCHIPS.2019.8875680.

[2] Jouppi, N., Kurian, G., Li, S., Ma, P., Nagarajan, R., Nai, L., Patil, N., Subramanian, S., Swing, A., Towles, B., Young, C., Zhou, X., Zhou, Z., and Patterson, D. A. TPU v4: An optically reconfigurable supercomputer for machine learning with hardware support for embeddings. In *Proceedings of the 50th Annual International Symposium on Computer Architecture* (New York, NY, USA, 2023), ISCA '23, Association for Computing Machinery. https://doi.org/10.1145/3579371.3589350.

[3] 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.

[4] 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.

[5] 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.

[6] Kim, B., Cha, S., Park, S., Lee, J., Lee, S., Kang, S.-h., So, J., Kim, K., Jung, J., Lee, J.-G., Lee, S., Paik, Y., Kim, H., Kim, J.-S., Lee, W.-J., Ro, Y., Cho, Y., Kim, J. H., Song, J., Yu, J., Lee, S., Cho, J., and Sohn, K. The Breakthrough Memory Solutions for Improved Performance on LLM Inference . *IEEE Micro 44*, 03 (May 2024), 40–48. https://doi.ieeecomputersociety.org/10.1109/MM.2024.3375352.

[7] Lie, S. Cerebras architecture deep dive: First look inside the hardware/software co-design for deep learning. *IEEE Micro 43*, 3 (2023), 18–30. http://dx.doi.org/10.1109/MM.2023.3256384.

[8] 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.

[9] Park, J., Choi, J., Kyung, K., Kim, M. J., Kwon, Y., Kim, N. S., and Ahn, J. H. AttAcc! unleashing the power of PIM for batched transformer-based generative model inference. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2* (New York, NY, USA, 2024), ASPLOS '24, Association for Computing Machinery, p. 103119. https://doi.org/10.1145/3620665.3640422.

[10] Talpes, E., Sarma, D. D., Williams, D., Arora, S., Kunjan, T., Floering, B., Jalote, A., Hsiong, C., Poorna, C., Samant, V., Sicilia, J., Nivarti, A. K., Ramachandran, R., Fischer, T., Herzberg, B., McGee, B., Venkataramanan, G., and Banon, P. The microarchitecture of DOJO, Tesla's exa-scale computer. *IEEE Micro 43*, 3 (2023), 31–39. http://dx.doi.org/10.1109/MM.2023.3258906.

[11] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems* (2017), I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30, Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[12] Xu, R., Ma, S., Guo, Y., and Li, D. A survey of design and optimization for systolic array-based DNN accelerators. *ACM Comput. Surv. 56*, 1 (Aug 2023). https://doi.org/10.1145/3604802.