Name _____

GPU Microarchitecture
EE 7722
Solve-Home Final Examination

Tuesday, 9 May 2023 to Friday, 12 May 2023 16:00 CDT

Work on this exam alone. Regular class resources, such as notes, papers, documentation, and code, can be used to find solutions. Do not discuss this exam with classmates or anyone else, except questions or concerns about problems should be directed to Dr. Koppelman.

Problem 1	_____	(33 pts)
Problem 2	_____	(34 pts)
Problem 3	_____	(33 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: [33 pts] Appearing below (on the following pages) are statistics on the unmodified base kernel and a solved w2 kernel from Homework 1 running on an RTX 4090, both computing layer 1. Recall that the `N-Rd` column shows the measured amount of data read by all SMs, normalized to the size of the two input arrays. The table show that the kernels read at least 1000× more than the ideal amount. This normalized amount does not change much with block size running the w2 kernel, but grows to almost 3500 for the base kernel.

(*a*) Write an expression that computes this normalized read amount for the `w2` kernel.

(*b*) The normalized read amount for the base kernel gets larger with larger blocks. This effect is larger on the 4090, with about 102400 B L1 of cache per SM than it is on the H100, with about 233472 B of L1 cache per SM.

Explain why L1 cache size is effecting base more than w2. Try to estimate the amount of cache space needed by each kernel and show that the H100 can more comfortably fit this size.

```
GPU 0: NVIDIA GeForce RTX 4090 @ 2.52 GHz WITH 24214 MiB GLOBAL MEM
GPU 0: L2: 73728 kiB   MEM<->L2: 1008.1 GB/s
GPU 0: CC: 8.9  SM: 128  SP-FP32/SM: 128  DP-FP64/SM:  2  TH/BL: 1024
GPU 0: SHARED: 49152 B/BL  102400 B/SM  CONST: 65536 B  # REGS: 65536
GPU 0: PEAK: 41288 SP GFLOPS  645 DP GFLOPS  COMP/COMM:  163.8 SP  5.1 DP
Using GPU 0
Call of cuCtxGetCurrent, cbid 304, serial 1
Layer shape 0: smpls=300.  wds=99.  heads=4.  d_q=d_k=d_v=8.
Layer shape 0: d_qkv=96.   d_model=32.  d_ws=29700
Layer shape 0: w_qkv: 12 kiB   h_in: 3712 kiB
Layer shape 1: smpls=30.  wds=99.  heads=8.  d_q=d_k=d_v=64.
Layer shape 1: d_qkv=1536.   d_model=512.  d_ws=2970
Layer shape 1: w_qkv: 3072 kiB   h_in: 5940 kiB
Cache preference set to cudaFuncCachePreferL1.

Kernel (qkv_base<ls_d_model(layer_specs[1])>):
   --Insn-- --L1--- -- L1<->L2 ---
wp /itr  %  SW  BXW N-Rd N-Wr GB/s   t/s === Util: FP++  Insn-- Data**  =======
 1  3.7  2  16 13.3 1044  1.0  583  16554 -
 2  3.7  3  16 13.3 1028  1.0  997   9540 --
 3  3.7  4  16 13.3 1023  1.0 1219   7758 --
 4  3.7  4  16 13.3 1020  1.0 1230   7668 ---
 8  3.7  4  16 14.2 1016  1.0 1236   7606 ---
16  3.7  4  16 14.8 1232  1.0 1467   7767 --
32  3.7  2  16 11.5 3482  1.0 2379  13513 -

Kernel (qkv_base_w2<ls_d_model(layer_specs[1])>):
   --Insn-- --L1--- -- L1<->L2 ---
wp /itr  %  SW  BXW N-Rd N-Wr GB/s   t/s === Util: FP++  Insn-- Data**  =======
 1  4.6  1   3  0.0 1044  1.0  442  21852 -
 2  4.6  3   3  0.0 1028  1.0  865  10991 --
 3  4.6  4   3  0.0 1023  1.0 1126   8400 --
 4  4.6  5   3  0.0 1020  1.0 1565   6030 ---
 8  4.6 10   3  0.0 1016  1.0 2986   3147 +-----
16  4.6 15   3  0.0 1014  1.0 4615   2032 *---------
32  4.6 15   3  0.0 1014  1.0 4756   1971 *---------
```

```
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
Using GPU 0
Call of cuCtxGetCurrent, cbid 304, serial 1

Layer shape 1: smpls=30.  wds=99.  heads=8.  d_q=d_k=d_v=64.
Layer shape 1: d_qkv=1536.   d_model=512.  d_ws=2970
Layer shape 1: w_qkv: 3072 kiB   h_in: 5940 kiB
Cache preference set to cudaFuncCachePreferL1.
Launching with 114 blocks of up to 32 warps.

Kernel (qkv_base<ls_d_model(layer_specs[1])>):
   --Insn-- --L1--- -- L1<->L2 ---
wp /itr  %  SW  BXW N-Rd N-Wr GB/s   t/s === Util: FP++  Insn-- Data**  =======
 1  4.7  1  16 13.5 1044  1.0  303  31872 -
 2  4.7  2  16 13.5 1028  1.0  515  18471 --
 3  4.7  3  16 13.5 1023  1.0  716  13216 --
 4  4.7  3  16 13.5 1020  1.0  755  12492 --
 8  4.7  4  16 13.5 1016  1.0  774  12146 ---
16  4.7  3  16 14.1 1025  1.0  757  12521 --
32  4.7  3  16 14.8 1348  1.0  972  12823 --

Kernel (qkv_base_w2<ls_d_model(layer_specs[1])>):
   --Insn-- --L1--- -- L1<->L2 ---
wp /itr  %  SW  BXW N-Rd N-Wr GB/s   t/s === Util: FP++  Insn-- Data**  =======
 1  4.9  1   3  0.0 1044  1.0  162  59450 -
 2  4.9  2   3  0.0 1028  1.0  320  29685 -
 3  4.9  2   3  0.0 1023  1.0  479  19763 --
 4  4.9  3   3  0.0 1020  1.0  630  14969 --
 8  4.9  6   3  0.0 1016  1.0 1168   8045 ----
16  4.9 11   3  0.0 1014  1.0 2261   4149 +-------
32  4.9 20   3  0.0 1014  1.0 4229   2217 ++------------
```

Problem 2: [34 pts] Answer the following questions about the Homework 2 solution. The code appears on the following pages, and has been checked into the repo.

(*a*) Appearing below are some data collected from a solution to Homework 2. As one can see from the template arguments, in the first run a thread computes 1 output element per iteration, in the second run each thread computes a tile of 4 rows by 4 columns (16 outputs), and in the third 8 rows by 8 columns.

```
Kernel (qkv_hw2_a<ls_d_model(layer_specs[1]),1,1,1>), 40 regs, 0 local.
   ---Insn---- --L1--- -- L1<->L2 ---
wp /itr %  TAc SW  BXW N-Rd N-Wr GB/s Imb   t/s === Util: FP++  Insn-- ========
 4  4.6  4 100  3  0.0 1014  1.0 1269  1t   7390 --
 8  4.6  8 100  3  0.0 1013  1.0 2412  1t   3884 ----
16  4.6 13 100  3  0.0 1013  1.0 3969  1t   2360 +------
32  4.6 15 100  3  0.0 1013  1.0 4520  1t   2072 +-------

Kernel (qkv_hw2_a<ls_d_model(layer_specs[1]),4,4,1>), 48 regs, 0 local.
   ---Insn---- --L1--- -- L1<->L2 ---
wp /itr %  TAc SW  BXW N-Rd N-Wr GB/s Imb   t/s === Util: FP++  Insn-- ========
 4  1.7  6 100  3  0.0  255  1.0 1534  1t   1544 +--
 8  1.7 11 100  3  0.0  254  1.0 2794  1t    846 ++----
16  1.7 18 100  3  0.0  198  1.0 3436  1t    537 +++-------
32  1.7 27 100  3  0.0  113  1.0 2887  1t    368 +++++---------

Kernel (qkv_hw2_a<ls_d_model(layer_specs[1]),8,8,1>), 128 regs, 0 local.
   ---Insn---- --L1--- -- L1<->L2 ---
wp /itr %  TAc SW  BXW N-Rd N-Wr GB/s Imb   t/s === Util: FP++  Insn-- ========
 4  1.5 13 100  3  0.0  128  1.0 1834  1i    652 +++--
 8  1.5 21 100  3  0.0  100  1.0 2339  1i    401 ++++-----
16  1.5 29 100  3  0.0   57  1.0 1893  1i    288 ++++++------
```

For the first run the improvement when going from 4 to 16 warps is $7390/2360 = 3.14$. For the second and third the improvements are 2.88 and 2.26, respectively.

☐ Why might the relative benefit of additional warps be smaller when a thread computes more output elements?

(*b*) Continuing with the data from the previous subpart, when a thread computes more elements, the total data read from the L2 cache drops.

☐ Could it be that lower demand on L2 to L1 bandwidth is a factor in the improved performance? ☐ Explain.

☐ Could it be that L2 load latency is a factor in the improved performance? ☐ Explain.

(*c*) Consider the following code in the main loop:

```
if ( no_work )
  { /* This part intentionally left blank. */ }
else if ( we_check == 0 )
  ii_iter(i_word_0,i_qkv_0,false,false);
else
  ii_iter(i_word_0,i_qkv_0,true,d_qkv!=ch_qkv*dbs_qkv);
```

The first case is for a thread that is assigned a row or column that is outside the array. The middle case is when all threads in a warp that have work can operate on a full `m_wd` columns and `m_ht` rows, the third case is when a thread has work and at least one thread does not operate on a full set of rows and columns.

Let t_o be the time needed for a thread to compute `ii_iter(i_word_0,i_qkv_0,false,false)` and let t_p be the time needed for a thread to execute `ii_iter(i_word_0,i_qkv_0,true,d_qkv!=ch_qkv*dbs_qkv)`.

Provide the range of times needed for a warp to compute its output for the cases described below. They are all correct, but in some cases code is calling the slower version unnecessarily.

Take branch divergence into account. For background on branch divergence see the Volta whitepaper [3] section titled "Independent Thread Scheduling," which starts on page 26. (Volta is CC 7.0, so it is two generations behind the CC 8.9 Ada implemented by RTX 4090 and the CC 9.0 Hopper implemented by the H100s.) The whitepaper is linked to `https://www.ece.lsu.edu/gp/gpu-descriptions.html` page. To answer this problem you can assume either the old divergence handling or the new one introduced with Volta. In both cases assume that reconvergence occurs right after the `if/else`.

☐ Best, average, and worst case time for a warp for the no-no-work special case:

```
if ( !need_check )
  ii_iter(i_word_0,i_qkv_0,false,false);
else
  ii_iter(i_word_0,i_qkv_0,true,d_qkv!=ch_qkv*dbs_qkv);
```

☐ Best, average, and worst case time for a warp for the no-no-work special case:

```
if ( !no_work && we_check == 0 )
  ii_iter(i_word_0,i_qkv_0,false,false);
else
  ii_iter(i_word_0,i_qkv_0,true,d_qkv!=ch_qkv*dbs_qkv);
```

```
template< int t_d_model = 0, int m_wd = 1, int m_ht = 1, int m_dp = 1>
__global__ void qkv_hw2_a()
{
  const Layer_POD& ld = layer_dev;
  constexpr int wp_sz = 32;
  const int tid = blockIdx.x * blockDim.x + threadIdx.x;
  const int wp = tid / wp_sz;
  const int num_blocks = gridDim.x;

  const int d_model = t_d_model ?: ld.d_model;
  const int d_qkv = 3 * d_model;
  assert( d_qkv == ld.d_qkv );
  const int n_words = ld.n_samples * ld.n_words;

  const int n_col_groups = ( n_words + m_wd - 1 ) / m_wd;
  const int cg_per_block = ( n_col_groups + num_blocks - 1 ) / num_blocks;
  const int block_cg_start = blockIdx.x * cg_per_block;
  const int block_cg_stop = block_cg_start + cg_per_block;
  const int block_word_stop = min( n_words, block_cg_stop * m_wd );

  constexpr int sector_len = m_ht > 1 ? 32 : 1;
  constexpr int ch_qkv = m_ht * sector_len;
  constexpr int dbs_qkv = ( d_qkv + ch_qkv - 1 ) / ch_qkv;

  timing_start(wp);

  auto ii_iter = [&](int i_word_0, int i_qkv_0, bool word_check, bool qkv_check)
  {
    acc_t q[m_wd][m_ht]{};

    for ( int i_model = 0; i_model < d_model; i_model++ )
      {
        acc_t h_elts[m_wd];
        for ( int i_wd = 0;  i_wd < m_wd;  i_wd++ )
          {
            const int i_word = i_word_0 + i_wd;
            const bool i_word_invalid = word_check && i_word >= block_word_stop;
            h_elts[i_wd] =
              i_word_invalid ? 0 : ld.h_in_dev[ i_word*d_model + i_model ];
          }

        for ( int i_ht = 0; i_ht < m_ht;  i_ht++ )
          {
            const int i_qkv = i_qkv_0 + i_ht * sector_len;
            const bool i_qkv_invalid = qkv_check && i_qkv >= d_qkv;
            acc_t w_elt =
              i_qkv_invalid ? 0 : ld.w_qkv2_dev[ i_qkv + i_model * d_qkv ];

            for ( int i_wd = 0;  i_wd < m_wd;  i_wd++ )
              q[i_wd][i_ht] += w_elt * h_elts[i_wd];
          }
      }
```

```
    for ( int i_ht = 0; i_ht < m_ht;  i_ht++ )
      for ( int i_wd = 0; i_wd < m_wd; i_wd++ )
        {
          const int i_word = i_word_0 + i_wd;
          const int i_qkv = i_qkv_0 + i_ht * sector_len;
          if ( ( !word_check || i_word < block_word_stop )
               && ( !qkv_check || i_qkv < d_qkv ) )
            ld.h_qkv_dev[ i_word * d_qkv + i_qkv ] = q[i_wd][i_ht];
        }
  };

  for ( int i = threadIdx.x; true; i += blockDim.x )
    {
      int ie = i;
      const int i_qkv_lo = ie % sector_len;
      ie /= sector_len;
      const int i_qkv_hi = ie % dbs_qkv;
      ie /= dbs_qkv;
      const int i_qkv_0 = i_qkv_hi * ch_qkv + i_qkv_lo;

      const int i_cg_0 = block_cg_start + ie;
      const int i_word_0 = i_cg_0 * m_wd;

      const bool no_work = i_word_0 >= block_word_stop || i_qkv_0 >= d_qkv;
      const bool need_check = !no_work
        && ( i_word_0 + m_wd > block_word_stop || i_qkv_0 + ch_qkv > d_qkv );
      const uint32_t wp_mask = 0xffffffff;

      const bool d_check = m_wd > 1 || m_ht > 1;
      if ( !d_check && i_word_0 >= block_word_stop ) break;

      const uint32_t we_check =
        d_check ? __ballot_sync(wp_mask, need_check) : need_check;

      if ( no_work )
        { /* This part intentionally left blank. */ }
      else if ( we_check == 0 )
        ii_iter(i_word_0,i_qkv_0,false,false);
      else
        ii_iter(i_word_0,i_qkv_0,true,d_qkv!=ch_qkv*dbs_qkv);

      if ( !d_check ) continue;

      const bool im_done = i_word_0 >= block_word_stop;
      if ( __all_sync(wp_mask,im_done) ) break;
    }

  timing_end(wp);
}
```

Problem 3: [33 pts] Answer the following questions on the Yüzügüler scale-out systolic arrays paper, [4].

(*a*) In the SOSA paper [4] performance is described in units of effective throughput per Watt. The Jouppi 17 TPUv1 paper [2] does not use the unit effective throughput per Watt, but there is enough data there to compute it.

☐ Compute effective throughput per Watt for the TPUv1 based on the data provided in [2].

☐ Indicate where you are getting each piece of data used in your calculation by referring to a table number, section number, etc.

☐ Use data that describes a workload of multiple benchmarks, not just one.

(*b*) In SoSA [4] there is no significant storage associated with a pod. Call the three matrix multiply input operands the *activations*, *weights*, and *input partial sums*. (In $T = AW + S$, activations are A, weights are W, the input partial sums are S, and the result partial sums are T.) In SoSA the activations, weights, and partial sums each are placed in their own memory units. Each memory unit is divided into N banks. An $N \times N$ network connects a memory to the N pods. The activations' memory unit would need a second interconnection network to receive data from the pods.

Consider an alternative design, which will be called a *stationary organization here*, in which one of the three operands was placed near a pod. For example, one might place bank i of the activation memory unit next to pod i, for $0 \geq i < N$. Doing so would eliminate the need for one or two interconnection networks. (Two for partial sums.) There should be no harm in fixing what was bank i to pod i because the other one or two operands can be routed to reach pod i.

Following terminology for other matrix and convolution accelerators [1], call a SoSA version with the activations placed near the pods *input stationary*, one with the weights near the pods *weight stationary*, and one with the partial sums near the pods *output stationary*.

☐ Based on data in the paper, perhaps estimated, compute the power benefit of a stationary organization (pick one).

☐ Would a stationary organization prevent a calculation from being done that could be done with the interconnect described in the SOSA paper? *Hint: The answer is yes, see Figure 8.* How bad would the impact be? ☐ Can the problem be avoided for at least one of the stationary organizations perhaps by adjusting tiling?

References:

[1] Chen, Y.-H., Emer, J., and Sze, V. Eyeriss: A spatial architecture for energy-efficient dataflow for convolutional neural networks. In *Proceedings of the 43rd International Symposium on Computer Architecture* (Piscataway, NJ, USA, 2016), ISCA '16, IEEE Press, pp. 367–379. https://doi.org/10.1109/ISCA.2016.40.

[2] 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.

[3] NVIDIA Corporation. NVIDIA Tesla V100 GPU architecture. Tech. rep., NVIDIA Corporation, August 2017.

[4] 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.