University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE5320, Fall 2023 — Midterm — Wednesday, October 11

- Exam ends at 11:45AM; begin as instructed (target 10:15AM)
  Do not open exam until instructed.
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration. All answers here.
- Unless otherwise noted, answers to two significant figures are sufficient.
- Sign Code of Academic Integrity statement (see last page for code).

____

I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity in completing this exam.

Name:

Solution

1	2a	2b	3	4	5	6	7	8	Total
10	5	5	10	10	20	10	10	20	100

Exam ended up being time constrained.

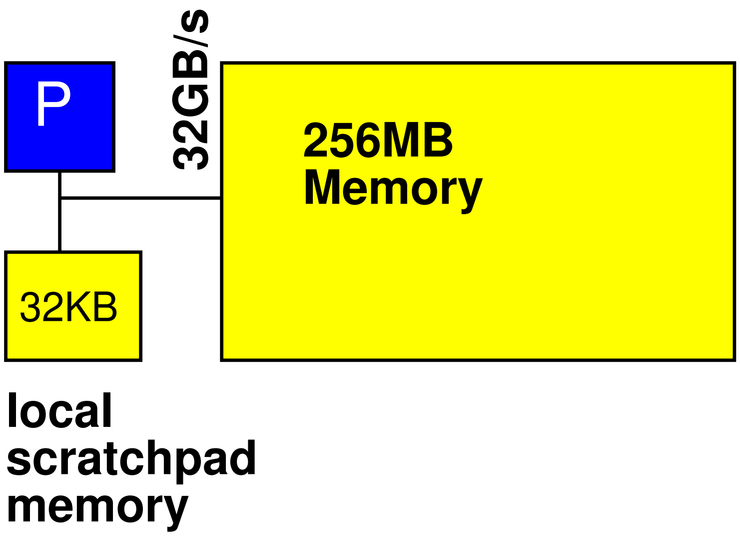
Mean: 57, Std. Dev. 12

Consider the following (very simplified) code to perform Deep Neural Network (DNN) Classification on a stream of matrix inputs.
Boundary conditions omitted for simplicity.)

```
#define DIM1 1024
#define WINDOW 64
#define STEP (WINDOW/2)
#define DIM2 (DIM1/STEP)
#define DIM3 (DIM2*DIM2)
#define STAGES 10
#define NORMALIZE 16
#define THRESH (1<<(NORMALIZE+1))
#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>

uint16_t in[DIM1][DIM1];
uint32_t mout[DIM1][DIM1];
uint16_t s2[DIM1][DIM1];
uint32_t cout[DIM2][DIM2];
uint16_t sinput[STAGES][DIM3];
uint32_t snorm[STAGES][DIM3];

uint16_t cm[DIM1][DIM1];
uint16_t cc[WINDOW][WINDOW];
uint16_t cweights[STAGES][DIM3][DIM3];
// static assignment to weights not shown for bevity

// assume these stream data in and out at the streaming rate
//  as data is available
void read_input(uint16_t input[DIM1][DIM1]);
void write_output(uint16_t sout[STAGES][DIM3], uint16_t s);


void mvmpy(uint16_t a[STAGES][DIM3], uint16_t c[STAGES][DIM3][DIM3],
           uint16_t s, uint32_t o[STAGES][DIM3]) {
  for (int i=0;i<DIM3;i++) { // loop A
      o[s][i]=0;
      for (int x=0;x<DIM3;x++) // loop B
        o[s][i]+=a[s-1][x]*c[s][i][x];
      }
}
```

```
void conv2digest(uint16_t a[DIM1][DIM1], uint16_t w[WINDOW][WINDOW],
                 uint32_t o[DIM2][DIM2]) {
  for (int y=0;y<DIM2;y++) // loop C
    for (int x=0;x<DIM2;x++) // loop D
      {
        o[y][x]=0;
        for (int wy=0;wy<WINDOW;wy++) // loop E
          for (int wx=0;wx<WINDOW;wx++) // loop F
            o[y][x]+=a[y*STEP+wy][x*STEP+wx]*w[wy][wx];
      }
}

void mmmpy(uint16_t a[DIM1][DIM1], uint16_t b[DIM1][DIM1],
           uint32_t o[DIM1][DIM1]) {
  for (int y=0;y<DIM1;y++) // loop G
    for (int x=0;x<DIM1;x++)  { // loop H
        o[y][x]=0;
        for (int k=0;k<DIM1;k++) // loop I
          o[y][x]+=a[y][k]*b[k][x];
      }
}
void nlmap2d (uint32_t i[DIM1][DIM1], uint16_t o[DIM1][DIM1]) {
  for (int y=0;y<DIM1;y++) // loop J
    for (int x=0;x<DIM1;x++) // look K
        if (i[y][x]<THRESH)
          o[y][x]=(i[y][x]>>NORMALIZE);
        else
          o[y][x]=0;
}
void nlmapflat (uint32_t i[DIM2][DIM2], uint16_t o[STAGES][DIM3], uint32_t s) {
  for (int y=0;y<DIM2;y++) // loop L
    for (int x=0;x<DIM2;x++) // loop M
        if (i[y][x]<THRESH)
          o[s][y*DIM2+x]=(i[y][x]>>NORMALIZE);
        else
          o[s][y*DIM2+x]=0;
}
void nlmap (uint32_t i[STAGES][DIM3], uint16_t o[STAGES][DIM3], uint32_t s) {
  for (int x=0;x<DIM3;x++) // loop N
        if (i[s][x]<THRESH)
          o[s][x]=(i[s][x]>>NORMALIZE);
        else
          o[s][x]=0;
}
int main(int argv, char **argc) {
  while (true)     {
      read_input(in);
      mmmpy(in,cm,mout);
      nlmap2d(mout,s2);
      conv2digest(s2,cc,cout);
      nlmapflat(cout,sinput,0);
      for (int s=1;s<STAGES;s++) {
        mvmpy(sinput,cweights,s,snorm);
        nlmap(snorm,sinput,s);
      }
      write_output(sinput,(STAGES-1));
    }
}
```

We start with a baseline, single processor system as shown.



- For simplicity throughout, we will treat non-memory indexing adds (subtracts count as adds), compares, min, max, abs, divides, multiplies, shifts, and logical operations (binary and bitwise) as the only compute operations. We'll assume the other operations take negligible time or can be run in parallel (ILP) with the adds, multiplies, and memory operations. (Some consequences: You may ignore loop and conditional overheads in processor runtime estimates; you may ignore computations in array indecies.)
- Baseline processor can execute one multiply, divide, compare, min, max, shift, abs, or add per cycle and runs at 1 GHz.
- Data can be transfered from the 256 MB main memory at 32 GB/s when streamed in chunks of at least 256B. Assume `for` loops that only copy data can be auto converted into streaming operations.
- Non-streamed access to the main memory takes 10 cycles.
- Baseline processor has a local scratchpad memory that holds 32KB of data. Data can be streamed into the local scratchpad memory at 32 GB/s. Non-streamed accesses to the local scratchpad memory takes 1 cycle.
- By default, all arrays live in the main memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, adds, min, max, divide and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time for compute operations and memory access for each function directly inside the main loop and the total bound for the while loop in main.

loop	Compute	Memory
read_input		$\frac{1024^2\times 2}{32} = 65,536$
mmmpy	$1024^3 \times 2$ $= 2.1 \times 10^9$	$1024^3 \times 3 \times 10$ (4 instead of 3 if re-read o) $= 3.2 \times 10^{10}$
nlmap2d	$1024^2 \times 2$ 2.1×10^6	$1024^2 \times 2 \times 10$ 2 for read i[y][x] once (alternately 3) 2.1×10^7
conv2digest	$32^2 \times 64^2 \times 2$ $= 8.4 \times 10^6$	$32^2 \times 64^2 \times 3 \times 10$ $= 1.26^8$
nlmapflat	$32^2 \times 2$ =2,048	$32^2 \times 2 \times 10$ (alternately 3 as above) =20,480
mvmpy (all STAGES)	$9 \times 1024^2 \times 2$ 1.9×10^7	$9 \times 1024^2 \times 3 \times 10$ 2.8×10^8
nlmap (all STAGES)	$9 \times 1024 \times 2$ =18,432	$9 \times 1024 \times 3 \times 10$ =276,480
write_output		$\frac{10\times 32\times 2}{32} = 320$ or 32×10=320
main while	$\approx 2.1 \times 10^9$	$\approx 3.2 \times 10^{10}$

2. Based on the simple, single processor mapping from Problem 1:

   (a) What function is the bottleneck? Consider both compute and memory. (circle one)

   mmmpy

   nlmap2d

   conv2digest

   mlmapflat

   mvmpy (all STAGES)

   nlmap (all STAGES)

   (b) What is the Amdahl's Law speedup if you only accelerate the identified function? Consider both compute and memory.

   Compute except mmmpy: 29.5×10^6

   Memory except mmmpy: 427×10^6

   compute+memory except mmmpy: 456.5×10^6

   $\frac{(2.1+32)\times 10^9+456.5\times 10^6}{456.5\times 10^6} \approx 76$

3. Parallelism in Loops

   (a) Classify the following loops as data parallel or not? (loop bodies could be executed concurrently)

   (b) Explain why or why not?

Loop	Data Parallel?	Why or why not?
A	Y	
B	Reduce	Could say sequentialized on sum (not data parallel); since associative add, we can compute as a reduce, which we will identify as a separate category for final. For credit, need to callout associative sum or reduce as reason for calling this data parallel.
C	Y	
E	Reduce	associative reduce like B
F	Reduce	associative reduce like B
G	Y	
H	Y	
I	Reduce	associative reduce like B
J	Y	
K	Y	

4. What is the critical path for `mmmpy` function?

   Perform all reads of a[y][x] and b[y][x] in parallel at beginning – 10

   Perform all multiplicies in parallel – 1

   Perform loop I as add reduce – 10 cycles
   (Alternately, perform adds in loop I sequentially for 1024 cycles)

   Perform all writes of o[y][x] in parallel at end – 10

   Total critical path is either 31 or 1045.

(This page intentionally left mostly blank for answers.)

5. Revise the body of `mmmpy` to minimize the memory resource bound by exploiting the scratchpad memory and streaming memory operations.

   (a) Identify the array or arrays whose memory operations account for most of the time in the loop.

   `a`, `b`, `o`

   (b) How would you rewrite `mmmpy` to use the scratchpad memory to reduce the time required to access memory? (show code)
   Hint: an order of magnitude reduction in memory time is possible, but may be tricky. A little under $3\times$ speedup is easier and will receive partial credit.

   - `b` won't fit in the scratchpad
   - Cannot simply stream columns of `b` into a local column array since they will not be stored sequentially in memory.
   - 20 pts for solution that makes it possible to stream `a`, `b`, and `o` while obeying scratchpad size constraints.
   - 12 pts for solution that only streams `a` and `o`; done completely right, this should give about a $3\times$ speedup since only reading `b` 10 cycles at a time.
   - 7 pts for ignoring scratchpad size (and typically putting `b` in scratchpad.
   - 7 pts for trying to stream `b` and treating it as streaming rather than random reads.
   - Points further deducted from these baselines when details within the solution were missing or wrong.

   code solution on next page

```
#define DIM1 1024
#include <stdint.h>

void mmmpy(uint16_t a[DIM1][DIM1], uint16_t b[DIM1][DIM1],
           uint32_t o[DIM1][DIM1]) {
  uint16_t arow[DIM1];
  uint16_t orow[DIM1];
  uint16_t b10rows[10][DIM1];

  for (int y=0;y<DIM1;y++)
    {
      for (int x=0;x<DIM1;x++)  arow[x]=a[y][x];
      for (int x=0;x<DIM1;x++)  orow[x]=0;
      for (int k=0;k<DIM1;k+=10)
        {
          for (int kl=0;kl<10;kl++)
            for (int x=0;x<DIM1;x++)
                b10rows[kl][x]=b[k+kl][x];
          for (int x=0;x<DIM1;x++)  {
            for (int kl=0;kl<10;kl++)
              orow[x]+=arow[k+kl]*b10rows[kl][x];
          }
        }
      for (int x=0;x<DIM1;x++)  o[y][x]=orow[x];
    }
}
```

(c) Account for total memory usage in the local scratchpad (use provided table).

Variable	Size (Bytes)
arow	$2\times1024 = 2048$
orow	$4\times1024 = 4096$
b10rows	$10\times2\times1024 = 20{,}480$

could push to 12 or 13 rows of `b`

(d) Estimate the new memory resource bound for your optimized `mmmpy`.

1024 iterations

stream in arow $\frac{2\times1024}{32}$

clear orow $\frac{4\times1024}{32}$

iterate $1024/10 = 103$ times

stream in 10 brows $\frac{10\times2\times1024}{32}$

local reads and writes of arow, b10row, orow

$10 \times 1024 \times 3$

stream out orow $\frac{4\times1024}{32}$

$= 1024 \times (64 + 128 + 128) + 1024 \times 103 \times (640 + 30720)$

$= 3.3 \times 10^9$

6. Identify concurrency opportunities between loops.

   Which functions can run concurrently, as separate processes, to increase the **throughput** for the while loop in `main`. If they cannot, explain what prevents concurrency. If they can, explain why and what conditions need to be met for the concurrency to work.

	Concurrent?	How? or Why not?
mmmpy + nlmap2d	Y	As soon as each o[y][x] is produced in `mmpy`, `nlmap2d` can compute on it.
nlmap2d + conv2digest	Y	As soon as `nlmap2d` finishes a value, it can be used in first window in `conv2digest`; `conv2digest` just needs to make sure o[y][x] values are ready as used in `conv2digest`; computation will be mostly overlapped.
conv2digest + nlmapflat	Y	As soon as each o[y][x] is produced in `conv2digest`, `nlmapflat` can compute on it.
nlmapflat + mvmpy	Y	As soon as `nlmapflat` produces each value, it can be used in the first dot product (loop B, i=0) in `mvmpy`.
mvmpy + nlmap	Y	As soon as each o[s][i] is produced in `mvmpy`, it can be used in `nlmap`.

(This page intentionally left mostly blank for answers.)

7. Consider building an accelerator for `mmmpy`. Target a throughput of completing one iteration of **loop I** on each cycle.

   - Assume we've pre-loaded the `b` (`cm`) matrix into a memory in the accelerator before the application starts, and this memory is wide and can supply the `b` (`cm`) data needed for one iteration of **loop I** on each cycle.
   - Assume `a` (`in`) data is streamed in from `read_input` at the streaming memory rate.

   (a) What compute operations must be performed in parallel on every cycle to complete loop I? (give number and type as well as computation being performed)

   1024 multiplies and 1024 adds

   (b) What needs to be read from the local `b` memory on every cycle?

   1024 values; a column from `b`.

   (c) How do we need to handle the `a` input stream to support this rate of operation?

   i. Describe why the `a` input streaming rate is adequate to maintain the throughput required by this accelerator.

   Operate on each row of `a` at a time for 1024 cycles (loop H). Can stream in next row of `a` in $\frac{2\times1024}{32} = 64$ cycles. So, have time to bring in each row while computing on previous row.

   ii. How can the input reception be treated to overlap the collection of input data with the computation?

   Shift values of `a` into row-wide shift register as the come. When accelerator is ready for next a row, transfer from the input shift register into the register used to supply the `a` row into the computation.

(d) How can this accelerator be extended to also include the `nlmap2d` computation that follows it while maintaining the same throughput?

Add the threshold computation to the end of the accumulation pipeline since each o[y][x] value thresholded independently.

(e) Assuming this accelerator runs concurrently with the rest of the computation on a processor, what is the new throughput for the while loop in the `main` function? (how many cycles per iteration of the while loop?)

460×10^6 (from 2b)

8. Map the `main` while loop computation to a system composed of:

   - four simple processors (1 GHz as previously outlined),
   - two fast processors (2 GHz, with everything running 2× as fast except data transfer from main memory),
   - one vector processor that can perform 8 16b×16b multiplies or 8 32b adds on each cycle as well as performing 8 vector loads of 16b or 32b data from its scratchpad, and
   - the accelerator from Problem 7.

   Assume each processor has its own scratchpad and has a separate path to the large memory so they can all simultaneously stream at full rate.[1]
   (Hint: can you map the problem to match the throughput provided by the `mmmpy` accelerator?)

   (a) Describe how you would map the computation onto these heterogeneous computing resources. Where is each computation run? What computations share compute units?

[1]Probably not realistic, but we'll use to simplify this problem.

loop	Where Run	Throughput
mmmpy	Problem 7 Accelerator	$\frac{1}{1024^2}$ cycles
nlmap2d	with above	
conv2digest	vector unit	$\frac{1}{\frac{32^2\times64^2\times(2+2)}{8}+32^2\times(2+2)+\frac{1024^2\times2+64^2\times2+1024}{32}}$
nlmapflat	(with above)	
mvmpy	2 Fast + 4 simple split by outputs	$\frac{1}{\frac{9\times1024^2\times(2+2)+9\times1024\times(2+2)}{8}+\frac{1024^2\times2}{32\times4}+\frac{32\times10}{4}+\frac{1024\times2}{32}}$ 1/4 of outputs on each fast processor; cweights also split by output; but all processors need all a/sinputs values
nlmap	(with above)	
write_output	(with above)	

(b) Describe how you would use the scratchpad memories as necessary beyond what you've already answered in Problems 5 and 7 to achieve your target performance. [no further change is a possible answer here.] (Hint: can you make sure the throughput of each `function` is limited by computation or time streaming data from memory?)

Store `cm` in scratchpad for vector unit. Stream in inputs and outputs. Merge `nlmapflat` at end of o[y[x] accumulation so does not require writes to or reads from memory.

`mvmpy`, `nlmap`. Similarly merge `nlmap` at end of o[s][i] accumulation to avoid reads/writes. Keep current and next `a` in scratchpads on processors. Stream in `cweights` one row at a time.

(c) Estimate the throughput your mapping achieves in cycles per `main` while loop iteration.

Limit now is on `mvmpy+nlmap+write_output` computation at: one iteration every 4.7×10^6 cycles

Code of Academic Integrity

Since the University is an academic community, its fundamental purpose is the pursuit of knowledge. Essential to the success of this educational mission is a commitment to the principles of academic integrity. Every member of the University community is responsible for upholding the highest standards of honesty at all times. Students, as members of the community, are also responsible for adhering to the principles and spirit of the following Code of Academic Integrity.*

Academic Dishonesty Definitions

Activities that have the effect or intention of interfering with education, pursuit of knowledge, or fair evaluation of a student's performance are prohibited. Examples of such activities include but are not limited to the following definitions:

A. Cheating Using or attempting to use unauthorized assistance, material, or study aids in examinations or other academic work or preventing, or attempting to prevent, another from using authorized assistance, material, or study aids. Example: using a cheat sheet in a quiz or exam, altering a graded exam and resubmitting it for a better grade, etc.

B. Plagiarism Using the ideas, data, or language of another without specific or proper acknowledgment. Example: copying another person's paper, article, or computer work and submitting it for an assignment, cloning someone else's ideas without attribution, failing to use quotation marks where appropriate, etc.

C. Fabrication Submitting contrived or altered information in any academic exercise. Example: making up data for an experiment, fudging data, citing nonexistent articles, contriving sources, etc.

D. Multiple Submissions Multiple submissions: submitting, without prior permission, any work submitted to fulfill another academic requirement.

E. Misrepresentation of academic records Misrepresentation of academic records: misrepresenting or tampering with or attempting to tamper with any portion of a student's transcripts or academic record, either before or after coming to the University of Pennsylvania. Example: forging a change of grade slip, tampering with computer records, falsifying academic information on one's resume, etc.

F. Facilitating Academic Dishonesty Knowingly helping or attempting to help another violate any provision of the Code. Example: working together on a take-home exam, etc.

G. Unfair Advantage Attempting to gain unauthorized advantage over fellow students in an academic exercise. Example: gaining or providing unauthorized access to examination materials, obstructing or interfering with another student's efforts in an academic exercise, lying about a need for an extension for an exam or paper, continuing to write even when time is up during an exam, destroying or keeping library materials for one's own use., etc.

* If a student is unsure whether his action(s) constitute a violation of the Code of Academic Integrity, then it is that student's responsibility to consult with the instructor to clarify any ambiguities.