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

ESE5320, Fall 2022 | Midterm | Wednesday, October 5

- 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:

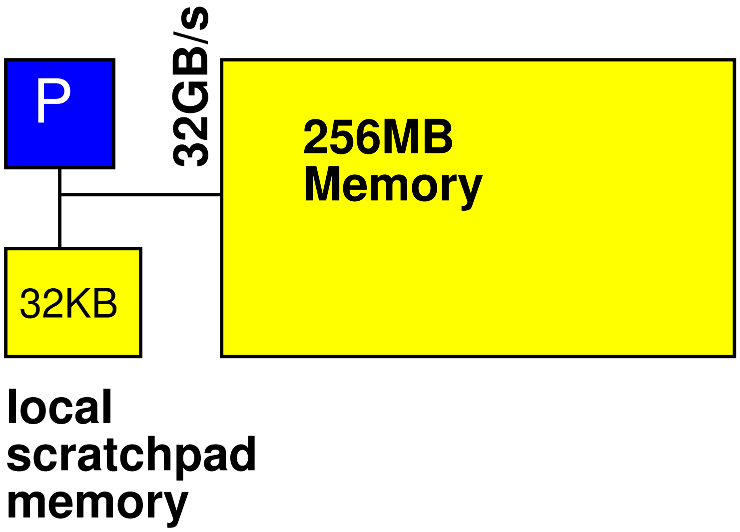
1	2a	2b	3	4	5	6	7	8	Total
10	5	5	10	10	20	10	10	20	100

Consider the following (very simplified) code to find paths in a gird with obstacles.

```
#define MAX_TARGETS 100
#define MAX_TIMESTEPS 100
#define WIDTH 1000
#define HEIGHT 1000
#include <stdint.h>
#include <stdlib.h>
#include <stdbool.h>

#define BLOCKED (1<<15)-1
#define USED (1<<14)
#define SOURCE 0
#define TARGET ((1<<14)|2)
#define NOPATH (MAX_TIMESTEPS+1)
#define FREE NOPATH

#define MASK16 ((1<<16)-1)

typedef struct pair_xy
{
  uint16_t y;
  uint16_t x;
} pair_xy;

uint16_t min(uint16_t a, uint16_t b); // assume single instruction
uint16_t max(uint16_t a, uint16_t b); // assume single instruction
void read_obstacles(uint16_t g[HEIGHT][WIDTH]);
            // marks obstacles BLOCKED in g
void read_sources_and_targets(uint16_t g[HEIGHT][WIDTH],
                              pair_xy source[MAX_TARGETS],
                              pair_xy target[MAX_TARGETS]);
   // loads targets into target, sources into source
   // marks target in g
void share_paths(pair_xy paths[MAX_TARGETS][MAX_TIMESTEPS]);
   // reports out results
void reset_tgrid(uint16_t tgrid[MAX_TIMESTEPS][HEIGHT][WIDTH],
                 pair_xy path[MAX_TARGETS][MAX_TIMESTEPS],
                 pair_xy target[MAX_TARGETS],pair_xy source[MAX_TARGETS],
                 int targ) {
  //   Need to know when path done to stop following
  bool pdone[MAX_TARGETS];
  for (int t=0;t<MAX_TARGETS;t++) { pdone[t]=false; } // loop F

  for (int step=0;t<MAX_TIMESTEPS;t++) { // loop G
    for (int y=0;y<HEIGHT;y++)  // loop H
      for (int x=0;x<WIDTH;x++) // loop I
        { tgrid[step][y][x]=NOPATH; }
    if (t==0)
      for (int s=0;s<MAX_TARGETS;s++) // loop J
        { tgrid[0][source[s].y][source[s].x]=0; }
    for (int t=0;t<targ;t++) // loop K
      if (!pdone[t]) {
          tgrid[step][path[t][step].y][path[t][step].x]=USED;
          if ((path[t][step].x==target[t].x)
              && (path[t][step].y==target[t].y)) // end of path
            { pdone[t]=true; }
      }
}
```

```
uint16_t new_cost(uint16_t grid[HEIGHT][WIDTH],
                     uint16_t tgrid[MAX_TIMESTEPS][HEIGHT][WIDTH],
                  uint16_t t, uint16_t y, uint16_t x){
  uint16_t below=max(grid[y-1][x],tgrid[t][y-1][x]);
  uint16_t above=max(grid[y+1][x],tgrid[t][y+1][x]);
  uint16_t left=max(grid[y][x-1],tgrid[t][y][x-1]);
  uint16_t right=max(grid[y][x+1],tgrid[t][y][x+1]);
  return(min(min(below,above),min(left,right))+1);
}

uint32_t predecessor(uint16_t tgrid[MAX_TIMESTEPS][HEIGHT][WIDTH],
                     uint16_t t, uint16_t y, uint16_t x) {
  if (tgrid[t-1][y-1][x]==t-1) { return((y-1)<<16 | x); }
  if (tgrid[t-1][y+1][x]==t-1) { return((y+1)<<16 | x); }
  if (tgrid[t-1][y][x-1]==t-1) { return(y<<16 | (x-1)); }
  if (tgrid[t-1][y][x+1]==t-1) { return(y<<16 | (x+1)); }
  abort(); // inconsistency: predecessor not find match
}

void find_paths () {
  uint16_t grid[HEIGHT][WIDTH];
  uint16_t tgrid[MAX_TIMESTEPS][HEIGHT][WIDTH];
  pair_xy target[MAX_TARGETS];
  pair_xy source[MAX_TARGETS];
  pair_xy path[MAX_TARGETS][MAX_TIMESTEPS];
  bool found;
  int found_time;

  read_obstacles(grid);
  read_sources_and_targets(grid,source,target);
  reset_tgrid(tgrid,path,target,source,0);

  for(int targ=0;targ<MAX_TARGETS;targ++) { // loop A
    found=false;
    for (int t=0;((!found) || (t<MAX_TIMESTEPS));t++) // loop B
      for (int y=0;y<HEIGHT;y++) // loop C
        for (int x=0;x<WIDTH;x++) { // loop D
          uint16_t cost=new_cost(grid,tgrid,t,y,x);
          if (grid[y][x]==FREE) { tgrid[t+1][y][x]=cost; }
          bool found_now=((target[targ].x==x)&&(target[targ].y==y)
                          &&(cost<NOPATH));
          found|=found_now;
          if (found_now) { found_time=t; }
        }
    path[targ][found_time].x=target[targ].x;
    path[targ][found_time].y=target[targ].y;
    for(int it=found_time;it>0;it--) { // loop E
      uint32_t pxy=predecessor(tgrid,it,path[targ][it].y,path[targ][it].x);
      path[targ][it-1].y=pxy>>16;
      path[targ][it-1].x=(pxy&MASK16);
    }
    reset_tgrid(tgrid,path,target,source,targ); // loop F-K inside
  }
  share_paths(path);
}
```

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, 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 20 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.
- Arrays `source`, `target`, and `pdone` live in local scratchpad memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time for compute operations and memory access for each loop directly inside loop A and the total bound for loop A.

loop	Compute	Memory
B		
E		
F		
G		
A		

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

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

   B

   E

   F

   G

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

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		
B		
C		
D		
E		
F		
G		
H		
I		
K		

4. What is the critical path for the body of loop A?

(This page intentionally left mostly blank for answers.)

5. Revise the body of `loop B` 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.

   (b) How would you use the scratchpad memory to reduce the time required to access memory? (You don't need to give code, but you need to describe clearly how the code would change. You may show code if that is the most efficient way to communicate your changes.)

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

Variable	Size (Bytes)
source	
target	
pdone	

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

(This page intentionally left mostly blank for answers.)

6. Assume you have a vector processor that can provide 16 vector lanes for 16b (including uint16_t) operations. The vector processor can read or write 256b from its local scratchpad memory in one cycle using a vector read or vector write operation. Build on your memory optimizations in the previous question. If necessary, describe any additional memory optimizations you may do beyond the previous question for this vector case. Assuming perfect vectorization, what is the impact on the compute and memory resource bounds for `loop B`? (state new bounds; show work.)

7. Identify concurrency opportunities between loops.

   Which loops can run concurrently, as separate processes, to increase the **throughput** for loop A? 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?
B + E		
E + F		
F + B	Y	

   Hint: we're giving you that there is concurrency between F and B. Note that they both iterate over timesteps. Identify the constraints required for them to run concurrently.

8. Map the `loop A` computation to a system composed of two simple processors (1 GHz as previously outlined), two fast processors (2 GHz, with everything running 2× as fast except data transfer from main memory), and four vector processors (Problem 6). 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]

   (a) Describe how you would map the computation onto these heterogeneous computing resources.

   (b) As necessary, describe how you would use the scratchpad memories as necessary beyond what you've already answered in Problems 5 and 6. [no further change is a possible answer here.]

   (c) Estimate the performance your mapping achieves in cycles per `loop A` iteration.

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

(This page intentionally left mostly blank for answers.)

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.