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

ESE532, Spring 2017 Midterm Wednesday, March 1

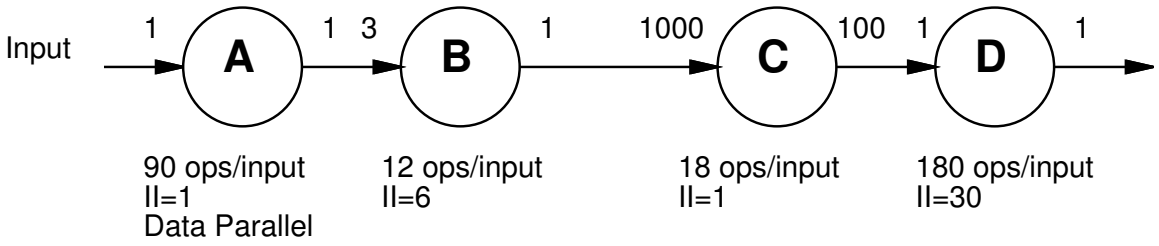
- Exam ends at 4:20PM; begin as instructed (target 3:00PM)
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration.
- Sign Code of Academic Integrity statement (see last page for code).

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

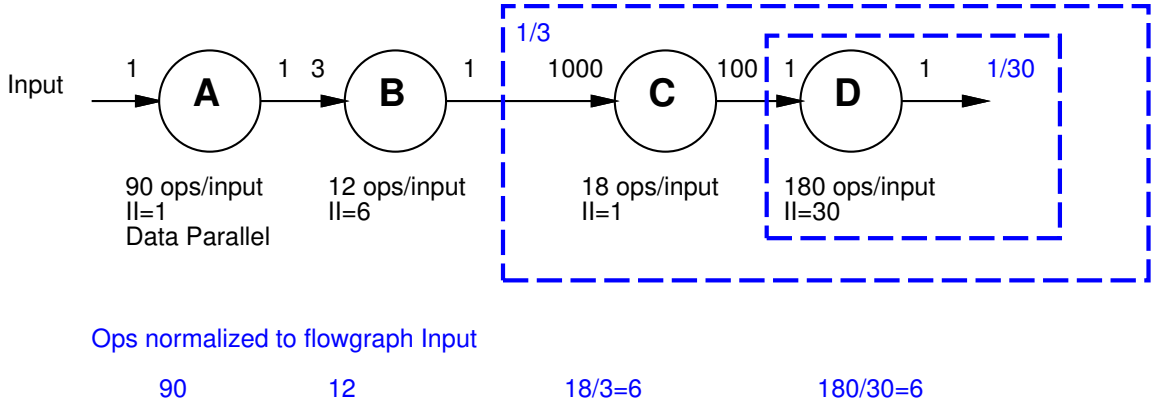
Name: Solution

Problem 1			Problem 2							
a	b	c	a	b	c	d	e	f	g	Total
10	10	10	5	10	10	10	15	15	5	100

1. Consider the Data Flow Graph



[Reminder: the consumer of an operator that consumes n inputs and produces m outputs will see only $\frac{m}{n}$ of the tokens seens by its predecessor. ops/input annotation is relative to the inputs to the operator. An operator shown consuming n inputs and executing c ops/input, will take $n \cdot c$ ops to process those n inputs.]



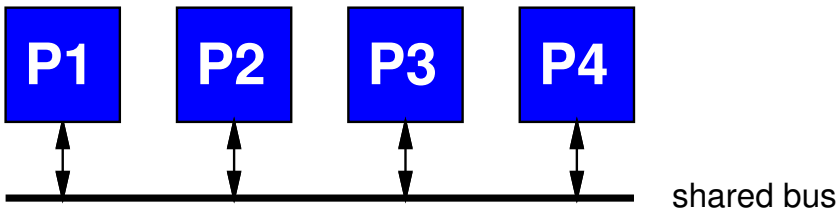
(a) Running on a single 100 MHz processor (assuming it successfully completes one of the "ops" required by a flowgraph node per cycle)

i. What is the throughput achieved in terms of inputs processed per second?

$1/(90+12+6+6) = \frac{100\text{MHz}}{114} = 0.88\text{M/s}$

ii. What is the Amdahl's Law maximum speedup possible if one could accelerate one operator (identify operator)? $\frac{114}{24} = 4.75$ (A)

(b) Running on 4 identical 100 MHz processors:



Assume processor-to-processor communication of one token takes one cycle on the shared bus.

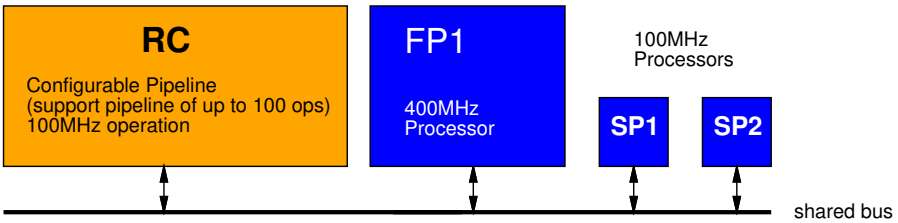
i. How would you map operators to processors?

P1	P2	P3	P4
A1	A2	A3	B, C, D

A1, A2, A3 are the result of dividing A into 3 equal pieces; this is viable since A is data parallel.

ii. Throughput achieved? $\frac{100\text{MHz}}{30}$=3.3M/s

(c) Running on a heterogeneous platform with a configurable accelerator pipeline, a fast processor, and two slower processors as shown below (one cycle transfer per token over bus):



i. How would you map operators to processors?

RC	FP1	SP1	SP2
Configurable Pipeline 100 datapath operations/cycle @ 100MHz	Processor 400 MHz	Processor 100 MHz	Processor 100 MHz
A	B	C	D

A is our bottleneck. With an II of 1 and fewer "ops" in the computation than the configurable pipeline supplies, we can build a complete pipeline and achieve a throughput of one result per cycle.
With an II of 6, the configurable pipeline will not be very useful for B. It would still take six 10 ns cycles to complete each input. We can run it on the faster processor to get it down to a throughput of one input every 30 ns.
B and C remain one input per 60 ns, and hence the rate limiting bottlenecks.

ii. Throughput achieved? $\frac{100\text{MHz}}{6}$=16.6M/s

2. Consider the following code:

```
#define WSIZE 5
#define FSIZE 1024

uint16_t window[WSIZE][WSIZE]; // uint16_t = 16b unsigned int
uint16_t frame_in[FSIZE][FSIZE];
uint16_t frame_out[FSIZE][FSIZE];

for(int y=0;y<(FSIZE-WSIZE+1);y++)
  for(int x=0;x<(FSIZE-WSIZE+1);x++) {
     frame_out[y][x]=0;
     for (int xoff=0;xoff<WSIZE;xoff++)
         for (int yoff=0;yoff<WSIZE;yoff++)
             frame_out[y][x]+=window[yoff][xoff]*frame_in[y+yoff][x+xoff];
     }
```

We run this on a system with a frame memory that is 32MB. It allows one 16b read or write at a time with a 10 ns latency and no pipelining. The single processor performs multiply-accumulate (Y=A+B*C) operations in 5 ns (unpipelined). For simplicity, assume other operations (e.g., loop management, index calculations) take no time (are dominated by the memory and multiply-accumulate operations).

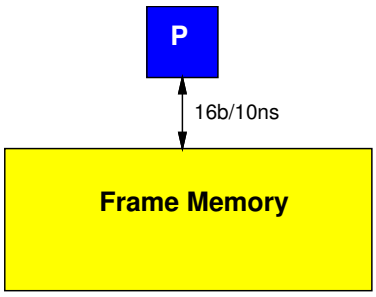
(a) How many multiply-accumulate operations need to be performed per frame?
$5 \times 5 \times 1020 \times 1020 \approx 26\text{M}$

(b) As written running on a single processor as shown.

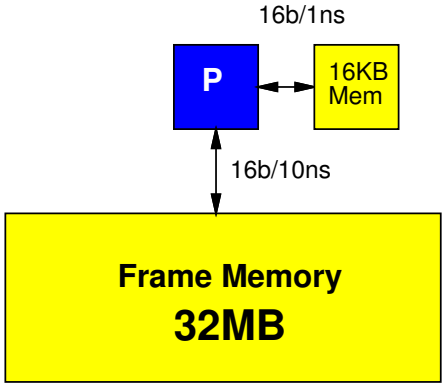
i. Assuming `window`, `frame_in` and `frame_out` live in the 32MB frame memory, how many memory operations are performed per frame?
$4 \times 5 \times 5 \times 1020 \times 1020 \approx 104\text{M}$
(1) read from window, (2) read from frame_in, (3) read from frame_out, (4) write to frame_out.
Also accepted 2 or 3.

ii. Where is the bottleneck on this single processor platform? Memory

iii. Assuming the code snippet above is called repeatedly, what is the throughput achieved in frames per second? $\frac{1}{26\text{M} \times 5\text{ns} + 104\text{M} \times 10\text{ns}} = 0.85$



(c) We add a local scratchpad memory that allows 16b access (read or written) in a 1 ns and holds 16KB of memory. We rewrite the code as shown on facing page. Variables with the `local_` are allocated to the scratchpad memory.



i. How much data must be read from the frame memory?

$$\boxed{2\text{B} \times (1024 \times 1024 + 5 \times 5) \approx 2\text{MB}}$$

ii. How many data operations are performed on the local memory?

$$\boxed{5 \times 5 + (2 \times 5 \times 5 + 1) \times 1020 \times 1020 + 1020 \times 1020 \approx 52\text{M}}$$

iii. Where is the bottleneck now?

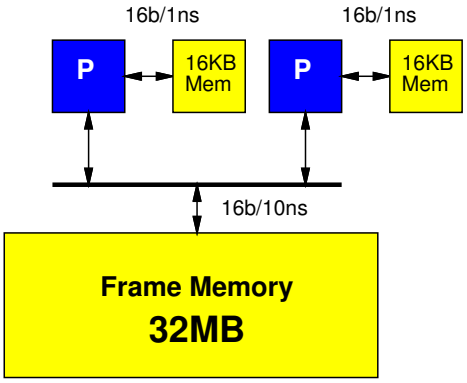
$$\boxed{\text{Computation (Multiply Accumulate Operations)}}$$

iv. Throughput achieved in frames per second?

$$\boxed{\frac{1}{26\text{M} \times 5\text{ns} + 2 \times 1\text{M} \times 10\text{ns} + 52\text{M} \times 1\text{ns}} \approx 5.0}$$

```
#define WSIZE 5
#define FSIZE 1024
uint16_t window[WSIZE][WSIZE]; // uint16_t = 16b unsigned int
uint16_t local_window[WSIZE][WSIZE]; // uint16_t = 16b unsigned int
uint16_t frame_in[FSIZE][FSIZE];
uint16_t frame_out[FSIZE][FSIZE];
uint16_t local_0[FSIZE];
uint16_t local_1[FSIZE];
uint16_t local_2[FSIZE];
uint16_t local_3[FSIZE];
uint16_t local_4[FSIZE];
uint16_t *local_line[WSIZE];
for (int xoff=0;xoff<WSIZE;xoff++)
  for (int yoff=0;yoff<WSIZE;yoff++)
      local_window[yoff][xoff]=window[yoff][xoff];
for(int x=0;x<(FSIZE-WSIZE+1);x++) { // ERROR -- should range to FSIZE not FSIZ
  local_0[x]=frame_in[0][x];
  local_1[x]=frame_in[1][x];
  local_2[x]=frame_in[2][x];
  local_3[x]=frame_in[3][x];
  local_4[x]=frame_in[4][x];
   }
local_line[0]=local_0;
local_line[1]=local_1;
local_line[2]=local_2;
local_line[3]=local_3;
local_line[4]=local_4;
for(int y=0;y<(FSIZE-WSIZE+1);y++) {
  for(int x=0;x<(FSIZE-WSIZE+1);x++) {
     int tmp=0;
     for (int xoff=0;xoff<WSIZE;xoff++)
         for (int yoff=0;yoff<WSIZE;yoff++)
             tmp+=local_window[yoff][xoff]*((local_line[yoff])[x+xoff]);
     frame_out[y][x]=tmp;
   }
  uint16_t *tmp_line=local_line[0];
  local_line[0]=local_line[1];
  local_line[1]=local_line[2];
  local_line[2]=local_line[3];
  local_line[3]=local_line[4];
  local_line[4]=tmp_line;
  for(int x=0;x<(FSIZE-WSIZE+1);x++) {// ERROR -- should range to FSIZE not FSI
    local_line[4]=frame_in[y+4][x]; // here (and above) hardcoded for WSIZE=5
    }
 }
```

(d) Given two processors as shown, how would you use both processors to accelerate the task?



i. Describe how you would divide the work between the two processors; show snippets of code that changes from part (c) as appropriate.

Give the computation of `frame_out` for y=0 to 509 to one processor and y=510 to 1019 to the second.

First processor, only change is:

```
for(int y=0;y<(FSIZE-WSIZE+1);y++) {
```

to:

```
for(int y=0;y<(FSIZE-WSIZE+1)/2;y++) {
```

Second processor, change to:

```
for(int y=(FSIZE-WSIZE+2)/2;y<(FSIZE-WSIZE+1);y++) {
```

Also need to change setup of `local_line`:

```
for(int x=0;x<(FSIZE-WSIZE+1);x++) {
  local_0[x]=frame_in[(FSIZE-WSIZE+2)/2+0][x];
  local_1[x]=frame_in[(FSIZE-WSIZE+2)/2+1][x];
  local_2[x]=frame_in[(FSIZE-WSIZE+2)/2+2][x];
  local_3[x]=frame_in[(FSIZE-WSIZE+2)/2+3][x];
  local_4[x]=frame_in[(FSIZE-WSIZE+2)/2+4][x];
   }
```

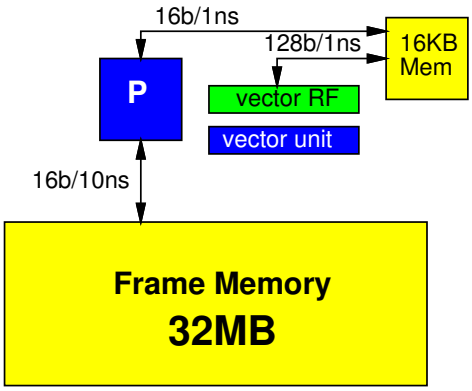
ii. Throughput achieved in frames per second?

$$\frac{1}{\frac{26}{2}\text{M}\times 5\text{ns}+2\times 1\text{M}\times 10\text{ns}+\frac{52}{2}\text{M}\times 1\text{ns}} \approx 9$$

This cuts the multiply-accumulates (bottleneck) and the local memory reads roughly in half, but doesn't change the frame memory reads. Nonetheless, accepted 2× solution in (c).

Left blank for pagination.

(e) Given a processor with a vector unit capable of processing eight 16b values in a cycle. Assume the local memory is widenned to allow transfers up to 128b of data in 1 ns (for simplicity of the problem, we will assume the memory system can support accesses that are not 128b aligned). For simplicity, we provide the following vector instruction set. All vector operations (including multiply) complete in 1 ns. The vector register file holds 16 vector registers.



- VADD sz, V1, V2, Vdst – add corresponding elements from V1 to V2 and store in Vdst (sz=16 says treat the 128b as 8 16b words)
- VMUL sz, V1, V2, Vdst – multiply corresponding elements from V1 to V2 and store in Vdst; perform operation on sz data items (sz=16 says treat the 128b as 8 16b words)
- VLD Rsrc, Vdst – load 128b at local memory addressed by Rsrc into Vdst
- VADDREDUCE sz, len, V1, Rdst – Perform a sum reduce add on the first len values in V1 and store into Rdst; sz is the values being reduced (sz=16 says treat the 128b as 8 16b words)

i. Show (on facing page) how you replace the inner two loops of the computation in (c) with vector operations. (Inner loop may not show how you setup some vector and scalar (normal processor) registers; summarize your use of registers to make the code clear.)

ii. Where is the bottleneck now?
Reading/Writing Frame Memory

iii. Throughput achieved in frames per second? $\frac{1}{15\times 1\text{M}\times 1\text{ns}+2\times 1\text{M}\times 10\text{ns}} \approx 29$

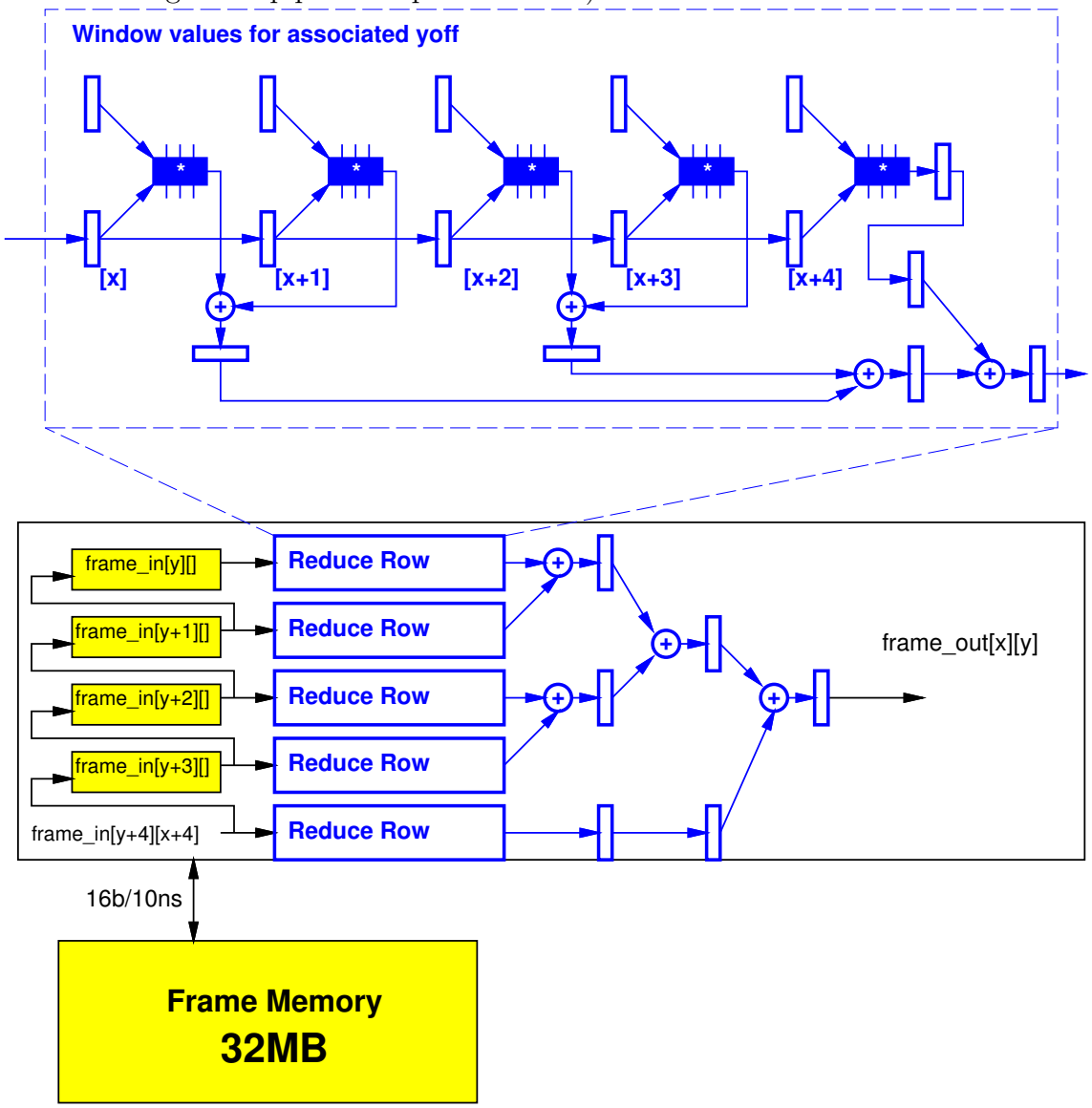
Code for Problem 2(e).i

Put `window[0][*]` in V0, `window[1][*]` in V1, ...`window[4][*]` in V4, and leave them there throughout the computation.
Put current frame pixels in V5 through V10.
Frame output value in `r3`, `frame_out` pointer in `r4`, pointers into `local_line` in `r5` through `r9`.

```
vld r5, v5
vld r6, v6
vld r7, v7
vld r8, v8
vld r9, v9
vmul 16, v0,v5,v5
vmul 16, v1,v6,v6
vmul 16, v2,v7,v7
vmul 16, v3,v8,v8
vmul 16, v4,v9,v9
vadd 16, v5,v6,v5
vadd 16, v5,v7,v5
vadd 16, v5,v8,v5
vadd 16, v5,v9,v5
vaddreduce 16, 5, v5, r3
st r3,r4 // store into frame_out
addi r4,2,r4 // advance frame_out pointer
addi r5,2,r5 // advance line pointers
addi r6,2,r6
addi r7,2,r7
addi r8,2,r8
addi r9,2,r9
```

(f) Design a hardware accelerator using building blocks with primitive 4 cycle pipelined multiply units, 1 cycle adder units, and registers operating on a 1 GHz clock.

i. Show a pipeline datapath for the inner two-loops with a pipeline II=1. Assume (as shown) there are memories configured to deliver pixels on lines y, y+1, y+2, y+3, y+4 into the pipeline. (Hierarchical schematics allowed. Make sure logic and pipeline depth are clear.)



ii. Throughput achieved in frames per second? $\boxed{50}$

iii. Latency through your pipeline (from arrival of the last pixel for a window to the output back to frame memory)? $\boxed{10\text{ns}}$

(g) Assuming we can widen the frame memory while maintaining the 10 ns cycle time, how wide does the memory need to be so that memory operations on the large memory is not the bottleneck using the hardware accelerator you designed in part (f)?

$\boxed{> 20 \times 16\text{b} = 320\text{b}}$

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 students 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 persons paper, article, or computer work and submitting it for an assignment, cloning someone elses 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 students 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 ones 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 students 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 ones 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 students responsibility to consult with the instructor to clarify any ambiguities.

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

ESE532, Fall 2017 — Midterm — Monday, October 23

- Exam ends at 4:20PM; begin as instructed (target 3:00PM)
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration.
- 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.

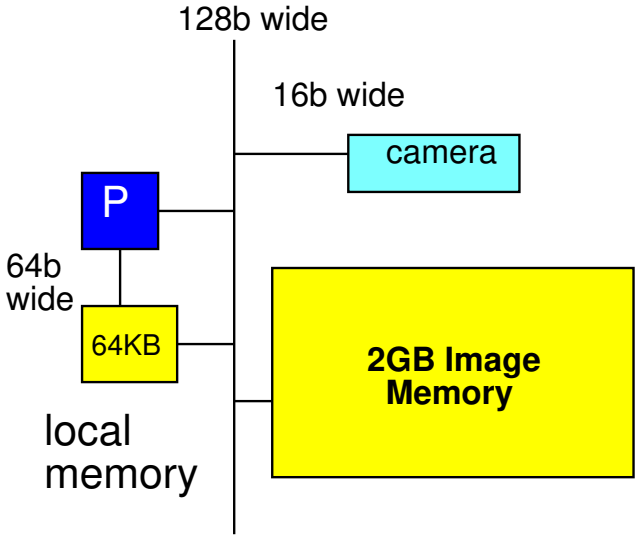
Name: Solution

1	2	3	4	5	6	7	8	9	10	Total
10	5	5	10	10	10	5	15	15	15	100

Consider the following code.

```
uint16_t SI[32][32];
uint64_t RI[16384][16384]; // in 2GB image memory
uint64_t cost, best_cost=MAX_COST;
int xguess=8192; int yguess=8192;
int oldx=8192; int oldy=8192;
int newx=8192; int newy=8192;
int x, y, xoff, yoff, dx, dy;
while (true) {
// A
  for (y=0;y<32;y++)
    for (x=0;x<32;x++)
      SI[y][x]=getPixel(); // reads from camera
// B
  for (yoff=-15; yoff<16;yoff++)
    for (xoff=-15; xoff<16;xoff++)  {
      cost=0;
      for (y=0;y<32;y++)
        for (x=0;x<32;x++)
           cost+=DIST(SI[y][x],RI[yguess+y+yoff][xguess+x+xoff]);
      if (cost<best_cost)
        {
          newx=xguess+xoff;
          newy=yguess+yoff;
          best_cost=cost;
        }
      }
// C
  for (y=0;y<32;y++)
    for (x=0;x<32;x++)
      RI[newy+y][newx+x]=UPDATE(SI[y][x],RI[newy+y][newx+x]);
// D
  dy=newy-oldy;
  dx=newx-oldx;
  yguess=newy+dy;
  xguess=newx+dx;
  oldy=newy;
  oldx=newx;
  newy=yguess;
  newx=xguess;
  best_cost=MAX_COST;
  }
```

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



- Base processor can execute one instruction per cycle and runs at 1 GHz.
- Base processor has a local memory that holds 64KB with single cycle access for 64b data.
- 2GB image memory can perform one operation (read or write) every 20 cycles that transfers 2048b of data.
  - Reading or writing less than 2048b still costs 20 cycles.
  - For the processor, assume you have a macro READ2048 that will initiate a 2048b read from the 2GB Image Memory into the processor local memory and a macro WRITE2048 that will initiate a 2048b write to the image memory from the processor local memory.
  - For the pipelined accelerator (on later questions) assume you have a data mover that can similarly initiate 2048b block transfers from image memory into an associated FIFO and another data mover than can initiate a 2048b transfer from an associated FIFO memory to the image memory.
- Function DIST is a macro that contains 10 primitive operations (instructions)
  - critical path is 4 primitive operations
  - value returned from DIST is a 16b value
- Function UPDATE is a macro that contains 100 primitive operations (instructions)
  - critical path is 15 primitive operations
  - value returned from update is a 64b value
- getPixel() can be called once every 3 cycles and delivers a single, 16b pixel value.
- Assume you store SI in local memory.
- RI only fits in the 2GB image memory.
- Assume scalar (non-array) variables can live in registers.
- You may ignore loop and conditional overheads in processor runtime estimates.

1. Estimate time to perform one iteration of the outer while loop body on a single processor for the code as shown, taking each reference to RI as a separate read or write to the memory.
(note: for this and all estimates, two significant figures is sufficient.)

A	$32 \times 32 \times (3+1)$	4096
B	$32 \times 32 \times 31 \times 31 \times (20+1+10+1+2+2)$ (potentially a few more 1's for update on new best)	35M
C	$32 \times 32 \times (20+1+100+20+2)$	143K
D	9	9
		35M

Processing Time Estimate	35ms

2. What is the lower bound for the processing time to perform one loop body of the outer while loop just considering the 2GB memory and taking each reference to RI as a separate read or write to the memory.

A	none	0
B	$32 \times 32 \times 31 \times 31 \times 20$	20M
C	$32 \times 32 \times (2 \times 20)$	40K
D	none	0
		20M

Memory Lower Bound Estimate	20ms

3. What is the lower bound for the processing time of the outer while loop body just considering the computational operations (computational resource bound)?

A	none	0
B	$32 \times 32 \times 31 \times 31 \times (10+1+2+2)$ (potentially a few more 1's for update on new best)	15M
C	$32 \times 32 \times (100+2)$	102K
D	9	9
		15M

Computational Lower Bound Estimate	14ms

4. What is the latency (critical path) lower bound for the computations in loop B? Assume all the data is available. (This question is about the computation, so the answer will not include any time for reading data out of memory. Previous and subsequent questions ask you about limits reading data from memory.)

Compute all 1M DIST in parallel	4
Compute cost sum as associative reduce	$\log(32 \times 32) = 10$
Compute min over all xoff, yoff as an associative reduce (alternately, could consider this a subtract followed by a mux select)	$\log(32 \times 32) = 10$ (20)
Total	24

Latency (critical path) Lower Bound Estimate	24 ns

If we consider the selection of the best cost as a sequential operation in order to guarantee that we pick the smallest (yoff,xoff) of a given cost, then instead of doing a min reduce, we might then have a sequential selection of best. That would be 14 for the DIST and cost sum followed by 32×32 for the best update, for a total of 1024+14=1038. But, we could still select the smallest (yoff,xoff) without sequentializing the whole computation; that will lead to a critical path of 24 or 34 cycles.

5. What is the lower bound on the number of reads and writes necessary from the 2GB memory for one iteration of the loop body of the outer while loop? Exploit the full width of the memory and assume you store and reuse values from the processor's local memory. Assuming you can achieve this, what is the lower bound for the processing time to perform one loop body of the outer while loop just considering the 2GB memory operations (i.e., revise your answer to question 2).

Reads	128
Writes	32
Memory Lower Bound Estimate	3200 ns

The entire region we need to read in RI for a single outer-loop-body is $(32+32)(32+32)$=4K 64b values (32KB). That is 64 rows, where each row is a contiguous set of $64 \times 8 = 512$B. Each read (or write) is 2048b or 256B. So, we need 2 reads per row, or $2 \times 64 = 128$ reads. When we go to write, it's 32 rows of 32 values. We only need one write per row, for a total fo 32 writes.

6. Describe how you would use the processor's local memory block to achieve the lower bound above. With this change, estimate the time to perform one iteration of the outer while loop body on a single processor.

Add a loop before B that reads in all the values needed for the B loop into a variable RI_local. This takes up half of the 64KB local memory. This is done in 2 reads per row, starting with &RI[yguess-15][xguess-15]. B now works only on RI_local. C also works on RI_local. After C add a post loop to write the updated 32 RI_local rows back to RI in the 2GB image memory.

A	$32 \times 32 \times (3+1)$	4096
Bpre	128×20	2560
B	$32 \times 32 \times 31 \times 31 \times (1+1+10+1+2+2)$ (potentially a few more 1's for update on new best)	17M
C	$32 \times 32 \times (1+1+100+1+2)$	105K
Cpost	32×20	640
D	9	9
		17M

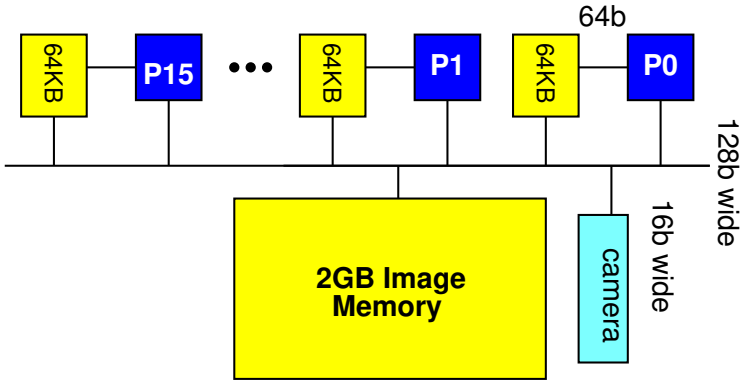
Processing Time	16ms

7. Working from this memory-optimized, sequential version, if you only speed up one of the labeled code segments (A, B, C, D), which one should you speedup and what is the Amdahl's Law limit on the speedup you can achieve for the outer-loop body?

$$Speedup = \frac{A + Bpre + B + C + Cpost + D}{A + Bpre + C + Cpost + D} = \frac{16M + 114K}{114K} = 147 \qquad (1)$$

Speedup Which (circle)	A [B] C D
Upper Bound Speedup	150

8. Building on your memory solution and assuming you have 16 identical processors, describe how you would assign tasks to processors to accelerate this computation. Estimate the throughput achievable in outer-loop-bodies per second on the 16 processor task mapping. Assume it is possible to broadcast to all 16 processors or specify for a read-response from the Image Memory to go to all 16 processors. Assume you have a facility to synchronize on a rendevous point amoung processors (e.g., barrier) that will allow the task set to continue on the cycle after the last processor arrives at the synchronization point. As part of your answer, identify what operations can be run concurrently and what operations must be sequentialized.



A: Read in image from P0 and broadcast values to all processors.
Bpre: Perform the Bpre on P0, broadcasting values to all processors.
B: Split among all 16 processor by outer loop yoff. Processor p performs loop instances yoff=$2p-15$ and $2p+1-15$.
Barrier synchronization on completion of B loops. Must complete B loops before can perform C.
P0 gathers up the best cost and associated newx, newy from the 16 processors and computes the overall best cost and newx, newy.
C: split among 16 processors by outer loop y. Processor p performs loop instances y=$2p$ and $2p+1$.
Barrier synchronization on completion of C loops.
Cpost: Have each processor, in sequence, write its RI updates.
D: Perform calculations on P0.

Throughput (outer-loop-bodies/s)	940

(This page intentionally left mostly blank for pagination and answer space.)

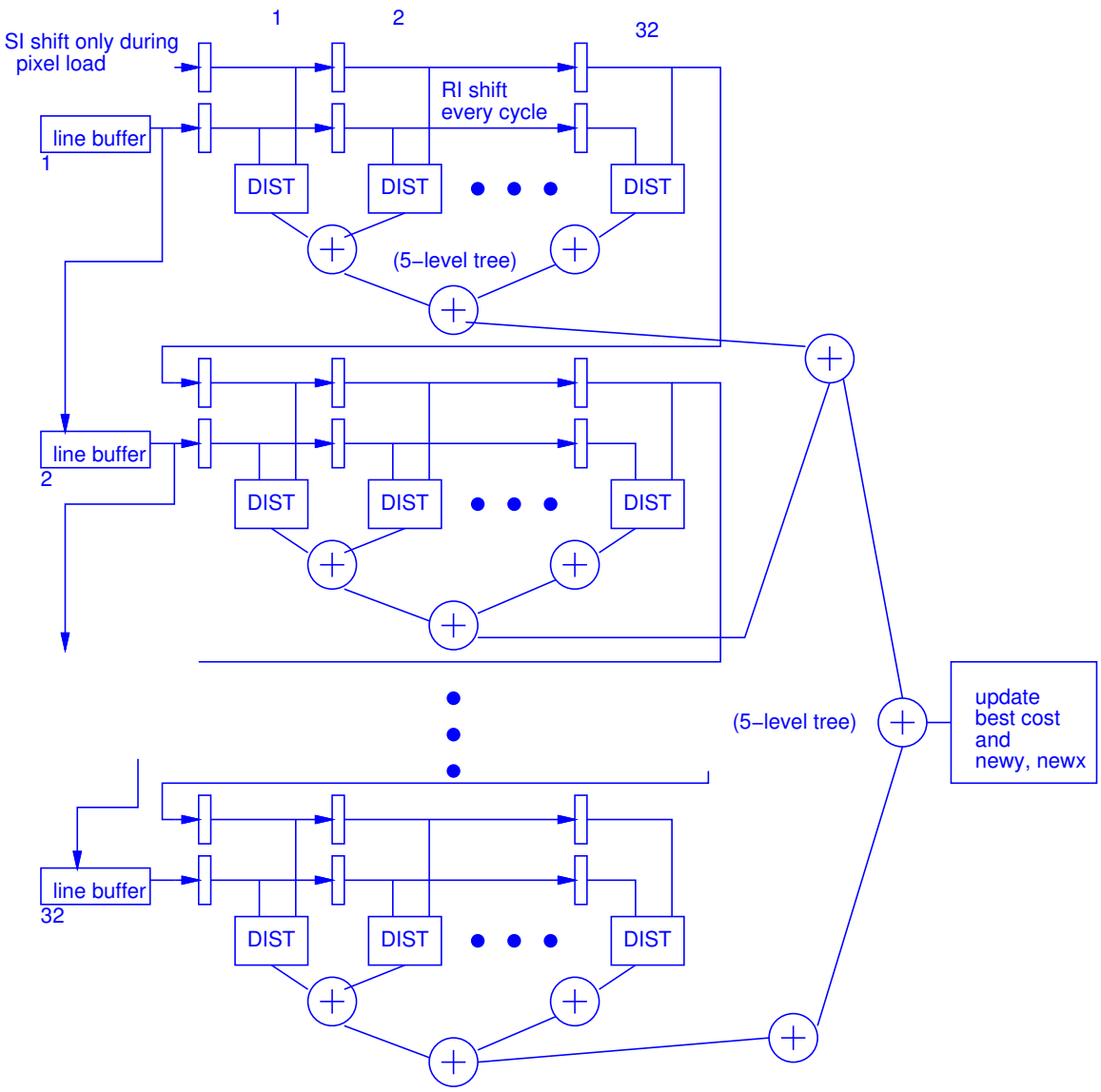
A	$32 \times 32 \times (3+1)$	4096
Bpre	128×20	2560
B	$2 \times 32 \times 31 \times 31 \times (1+1+10+1+2+2)$ (potentially a few more 1's for update on new best)	1M
Bmin	16×10 crude approximation for calculate best	160
C	$2 \times 32 \times (1+1+100+1+2)$	6720
Cpost	32×20	640
D	9	9
		1.06M

9. Describe how to build a pipeline for the B loop that allows the computation in this loop to execute in a little over 1024 cycles (1024 cycles plus the time to drain the pipelines). You may include customized local memories for data storage. Draw pipelined structure (but you won't be able to show every element). Counting each primitive operation as 1 unit and each KB of memory used as 1 unit, estimate the area required for this pipelining. (To simplify accounting for this problem, we will assume register cost is negligible.)

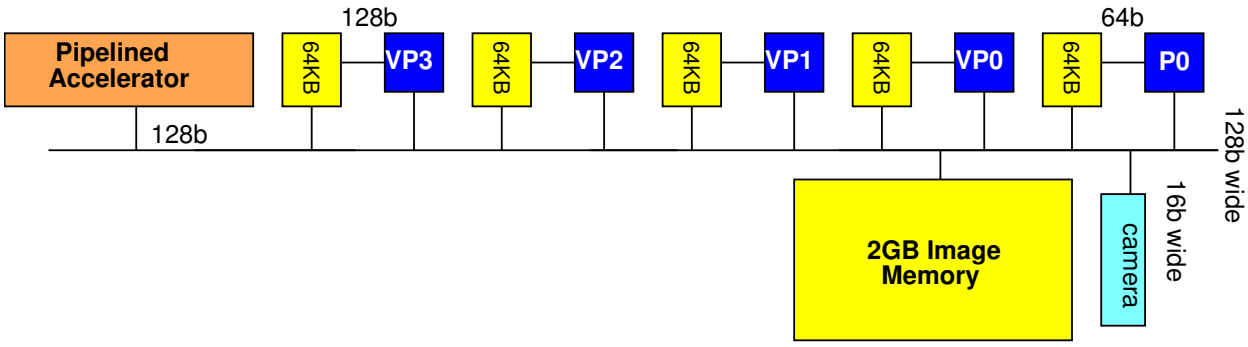
   Fully unroll and pipeline the inner two loops (y and x) so that the datapath can start the computation of the cost for a unique (xoff, yoff) pair every cycle. This requires fully partitioning SI, so there is a register for each element of the SI array. It requires a shift register configuration for the active portions of the RI row, and line buffers for the portion for the portion of the row that won't be used again until the next yoff. The additional cycles beyond the 1024 cycles to initiate each (xoff,yoff) position are the ones to drain the pipeline—4 for DIST, 10 for the sum reduce for the cost add, plus 1 for the best cost update.

   Compute hardware is 10 operations for DIST plus 1 for the add for each of the 32×32 unrolled inner loop bodies, or 11K. The best cost update is less than 10 primitive operators. We need at most 32 line buffers holding 64×8 Bytes or 16KB plus another 32 lines worth of memory totaling another 16KB. With care, we can probably use only 32×8 line buffers with the other 32 values in the shift registers and loading the other 32 lines just as needed. Nonetheless, even using all 32KB, this only adds another 32 units of area, which remains a second order area term.

Area	11K


1
2
32
SI shift only during pixel load
RI shift every cycle
line buffer 1
line buffer 2
line buffer 32
DIST
(5−level tree)
(5−level tree)
update best cost and newy, newx

10. Describe the entire solution using the B pipeline above along with 4 vector and one non-vector processors. What can execute concurrently and what operations must be serialized? Estimate the throughput achievable in outer-loop-bodies per second. Assume the vector processors have a 128b-wide vector processing unit that can process 8 16b primitive operations per cycle and you can perform a perfect vector mapping of the UPDATE routine. Further assume you can transfer 128b in a cycle between the local memory and the vector processing unit.



P0: Runs A reading pixels and loading into SI shift register for B. (4096 cycles)
Stream read first half of RI window into B (and into local memories for VP0 through VP3). (1280 cycles)
Execute B, prefetching next row concurrently while operating at one row position (yoff); also read rows into VP0 through VP3 local memory. (1038+10 cycles)
After B completes, run C on VP0 through VP3.
Give 8 rows to each vector processor. $(8 \times 32 \times (100/8) = 3200)$
Have each VPi, in sequence, write its RI updates. (640)
Perform D on P0. (concurrent with RI writeback).
Total 10,264 cycles.

Throughput (outer-loop-bodies/s)	97K

With additional care to double-buffer the SI values (e.g. a separate shift-register from the storage register to use during the DIST computations), we can run the reading of the pixels for the next frame concurrent with all the rest of the computation. That would reduce the time to max(4096,6168). With a bit more care we can overlap parts of the RI writeback with the UPDATE computation.

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 students 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 persons paper, article, or computer work and submitting it for an assignment, cloning someone elses 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 students 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 ones 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 students 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 ones 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 students responsibility to consult with the instructor to clarify any ambiguities.

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

ESE532, Fall 2018 — Midterm — Monday, October 22

- Exam ends at 11:50AM; begin as instructed (target 10:30AM)
- Problems weighted as shown.
- Calculators allowed.
- Closed book = No text or notes allowed.
- Show work for partial credit consideration.
- 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

1a	1b	2a	2b	3ab	3c	4	5a	5b	5c	6	Total
7	8	5	5	6	4	15	15	5	15	15	100

`warn_collide` is a typo for `warn_collision`.

`ostream[i]` and `tstream[i]` should be `ostream` and `tstream`.

Consider the following code to track objects and predict potential collisions.

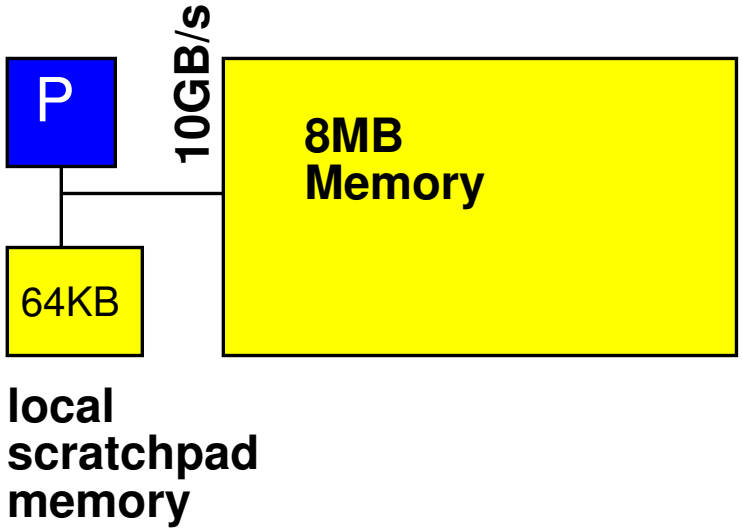
You want to consider `warn_collision` as a separate thread that must execute at 100 frames per second. So, it is called once every 10ms with a new image. For this exam, we are only concerned with this `warn_collision` task.

```
#define XMAX 1024
#define YMAX 1024
#define MAX_OBJECTS 1000
#define LOOKAHEAD 1000
#define XPOS 0
#define YPOS 1
#define WMIN -50
#define WMAX  50
#define MAXDIST 10000
#define OSIZE 30
uint16_t ocare[MAX_OBJECTS][OSIZE][OSIZE], oval[MAX_OBJECTS][OSIZE][OSIZE];
uint16_t dfx[MAX_OBJECTS], dfy[MAX_OBJECTS], x[MAX_OBJECTS], y[MAX_OBJECTS];
int speed[MAX_OBJECTS], angle[MAX_OBJECTS];
uint16_t collide[MAX_OBJECTS];
uint16_t collide_at_f[LOOKAHEAD][MAX_OBJECTS];
void findobj(uint16_t **image, int o, int gx, int gy, int result_xy[2]) {
     for (int y=gy+WMIN; y<gy+WMAX; y++) // Loop A
       for (int x=gx+WMIN; x<gx+WMAX; x++) { // Loop B
         int dist=MAXDIST;
         for (int i=0;i<OSIZE;i++) // Loop C
            for (int j=0;j<OSIZE;j++) // Loop D
                dist+=ocare[o][i][j]*abs(oval[o][i][j]-image[y+i][x+j]);
          if (dist<bestdist) {
               bestx=x; besty=y; bestdist=dist;
            }
          }
      result_xy[XPOS]=bestx;
      result_xy[YPOS]=besty;
  }
int distance(int x1, int x2, int y1, int y2) {
   int dx=x1-x2;
   int dy=y1-y2;
   return(sqrt(dx*dx+dy*dy));
}

// assume sin, cos, getangle, and sqrt
//   can each be performed with 20 multiplies and 20 adds;
//   critical path is 6 instructions long.
```

```
void warn_collisons(uint16_t image[XMAX][YMAX],
                    hls::stream<int> ostream, hls::stream<int> tstream) {
  for (int i=0;i<omax;i++)  // omax<MAX_OBJECTS // Loop E
     collide[i]=LOOKAHEAD;
  for (int f=0;f<LOOKAHEAD;f++) // Loop F
     for (int i=0;i<omax;i++) collide_at_f[f][i]=LOOKAHEAD; // Loop G
  int future[XMAX][YMAX];
  for (int y=0;y<YMAX;x++) // Loop H
     for (int x=0;x<XMAX;x++) future[y][x]=0; // Loop I
  for (int i=0;i<omax;i++) { // Loop J
    int newloc_guess_x=x[i]+dfx[i];
    int newloc_guess_y=y[i]+dfy[i];
    int xyloc[2];
    findobj(image,i,newloc_guess_x,newloc_guess_y,xyloc);
    speed[i]=distance(x[i],xyloc[XPOS],y[i],xyloc[YPOS]);
    angle[i]=getangle(x[i],xyloc[XPOS],y[i],xyloc[YPOS]);
    x[i]=xyloc[XPOS];
    y[i]=xyloc[YPOS];
    }
  for (int i=0;i<omax;i++) { // Loop K
     dfx[i]=speed[i]*cos(angle[i]);
     dfy[i]=speed[i]*sin(angle[i]);
     }
  for (int f=0;f<LOOKAHEAD;f++) { // Loop L
    for (int i=0;i<omax;i++) { // Loop M
       int fx=x[i]+i*dfx[i]; int fy=y[i]+i*dfy[i];
       future[fy][fx]++;
       }
    for (int i=0;i<omax;i++) { // Loop N
       int fx=x[i]+i*dfx[i]; int fy=y[i]+i*dfy[i];
       if (future[fy][fx]>1) collide_at_f[f][i]=f;
       }
    for (int i=0;i<omax;i++) { // Loop O
       int fx=x[i]+i*dfx[i]; int fy=y[i]+i*dfy[i];
       future[fy][fx]=0;
       }
     } // end of L
  for (int i=0;i<omax;i++) { // Loop P
     for (int f=0;f<LOOKAHEAD;f++) // Loop Q
        collide[i]=min(collide[i],collide_at_f[f][i]);
  for (int i=0;i<omax;i++) // Loop R
     if (collide[i]!=LOOKAHEAD)
        { ostream[i].write(i); tstream[i].write(collide[i]); }
}
```

```
// used in a pipeline like:
while(true) {
   getImage(image);
   warn_collisions(image,collision_objects,collision_times);
   steer(collision_objects,collision_times);
   }
```

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



- For simplicity throughout, we will treat non-memory indexing adds (subtracts count as adds) and multplies 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 or add per cycle and runs at 1 GHz.
- Data can be transfered from the 8MB main memory at 10 GB/s when streamed in chunks of at least 256B.
- Non-streamed access to the main memory takes 5 cycles.
- Baseline processor has a local scratchpad memory that holds 64KB of data. Data can be streamed into the local scratchpad memory at 10 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- By default, all arrays (image, future, ocare, oval, x, y, speed, angle, dfx, dfy, collide, collide_at_f) live in the main memory.
- Assume scalar (non-array) variables and xyloc can live in registers.
- Assume all additions are associative.
- Assume adds and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz.

1. Resource Bound

(a) Assuming the scalar processor can perform one add or one multiply on a cycle, what is the compute resource bound in seconds for the time taken by `warn_collide`?

E, F, G, H	(no adds or multiplies here)	0
J:findobj	$MAX\_OBJECTS \times (WMAX - WMIN)^2 \times OSIZE^2 \times 3$ 3 for -, *, + $1000 \times 100^2 \times 30^2 \times 3$	2.7×10^{10}
J:not findobj	$MAX\_OBJECTS \times (2 + 40 + 40 + 5)$	87,000
K	$MAX\_OBJECTS \times 2 \times (40 + 1)$	82,000
L	$LOOKAHEAD \times MAX\_OBJECTS \times (5 + 4 + 4)$	1.3×10^7
P	(no adds or multiplies here)	0
R	(no adds or multiplies here)	0
Total		2.7×10^{10}

$\frac{2.7 \times 10^{10}}{10^9}$=27 seconds

(b) Assuming all array data located in the 8MB memory by default can be streamed at the full 10 GB/s rate, what is the memory resource bound in seconds for the time taken by `warn_collide`?

Here, we assume everything is streamed, so we count memory operations on the data in arrays.

		uint16_t
E	$MAX\_OBJECTS$	1000
F	$LOOKAHEAD \times MAX\_OBJECTS$	10^6
H	$YMAX \times XMAX$	10^6
J:findobj	$MAX\_OBJECTS \times (WMAX - WMIN)^2 \times OSIZE^2 \times 3$ 3 for oval, ocare, image $1000 \times 100^2 \times 30^2 \times 3$	2.7×10^{10}
J:not findobj	$MAX\_OBJECTS \times (10 + 2 \times 2)$ angle and speed are int instead of uint16_t, multiply by 2	14,000
K	$MAX\_OBJECTS \times 6$	6,000
L	$LOOKAHEAD \times MAX\_OBJECTS \times (6 + 6 + 5)$	1.7×10^7
P	$LOOKAHEAD \times MAX\_OBJECTS \times 3$	3×10^6
R	$MAX\_OBJECTS$	1000
Total		2.7×10^{10}

$\frac{(2\text{Bytes/uint16\_t}) \times 2.7 \times 10^{10}\text{uint16\_t}}{10^{10}\text{GB/s}}$=5.4 seconds

2. Amdahl's Law

   (a) Based on resource bound estimates, which outer loop is the bottleneck?

   Circle One:

E	F	H	J	K	L	P	R

   Why?

   J contains both the most data access and the most computational operations.

   (b) Will accelerating the identified outer loop be sufficient to achieve the real-time goal of 10 ms per invocation of `warn_collide`? Explain why or why not.

No.

We can afford 10^7 cycles in the 10 ms budget.

We perform 13 M compute operations in L, which alone exceeds the budget. We perform 4 M accesses to `future` in L that cannot be streamed, accounting for 20 M cycles.

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?
J	Y	Computation for every object is independent.
L	Y	Computation for every future (LOOKAHEAD) time step is independent.
M	N	`future[fx][fy]` could change with every object.

(c) Identify a loop or loop nest that performs data parallel operations followed by an associative reduce.

Loop	Reduce Operation
C,D	add
A,B	min
Q	min

M could be considered an add-reduce in a very sophisticated way. It would require effectively assuming there was a `future_for_object[YMAX][XMAX][MAX_OBJECTS]` array, computing M data parallel for each object, then performing a sum reduce over the object component for each y, x position to get the `future` array that appears in the program.

4. Focusing on the computation, and assuming unlimited computing resources, what is the latency bound for `warn_collide`? (for this problem, assume `min(a,b)` takes 1 ns just like add and multiply).

E, F, G, H	define initial values	0
J	newloc_guess (both, parallel)	1
	AB min reduce $\log_2(100^2)$	+14
	CD sum reduce $\log_2(30^2)$	+10
	D body pipeline of 3 operations	+3
	speed and angle run in parallel distance: subs in parallel, * in parallel, add, 6 for sqrt	+9
K	all objects in parallel dfx and dfy in parallel multiply, 6 for cos or sin critical path	+7
L	all iterations run concurrently	
M	fx, fy compuations all in parallel	+2
	serial future[fy][fx]++ (alternately reduce (see answer 3c))	+1000 (+10 instead)
N	`future[fy][fx]` access must wait for M to complete	
	count 1 for test/assignment	+1
O	defining initial value of `future` for loop	
P	all objects run concurrently	
Q	min reduce $\log_2(1000)$	+ 10
R	must put out in order worst case, all collide	+1000
Total	 or (reduce M case)	2057 (1067)

Latency Bound = 2.1μs (or 1.1μs)

(this page left mostly blank for pagination; can use for calculations or answers)

5. Accelerator

(a) Considering an accelerator target (e.g. FPGA mapping) for the bottleneck outer loop (Problem 2a), which of its loops need to be pipelined and unrolled to achieve the real-time goal of 10 ms per invocation of `warn_collide`? If unrolled less than completely, indicate the unroll factor.
Answer here likely assumes your solution to part (c); you will explain how you support data movement in part (c).

Loop	Unroll? (factor)	Pipeline?	II	Comments
C	Y			
D	Y			
A	N	Y	100	
B	N	Y	1	
J	N	Y	100^2	

`findobj` has 2.7×10^{10} cycles due to compute operations, but we can only afford 10^7 to meet our 10 ms target. Unrolling C and D gives us $3 \times 30^2 - 2700$ operations, to bring `findobj` down to 10^7 sequential cycles. Pipeline A, B allows us to start one of the unrolled C-D computation and reduces per cycle. Pipeline rest of J so can operate concurrently with `findobj` accelerator task.
Pipelining J outside of `findobj` and including in accelerator not strictly necessary. Could run as separate, concurrent thread on a processor.

(b) How many adders and multipliers does this design contain?

Multipliers	$1 \times 30^2 + 1$=901
Adders	$2 \times 30^2 + 1$=1801

+1 for computations in J not in CD. Not necessary for 2 significant figures.

(c) What memory organization do you need to support the accelerator? Assume you have 4KB, 2-port memory blocks (similar to BlockRAMs).

- What local memories do you need to allocate?
- What do these memories hold?
- What is the strategy for moving data into these local memories?
- How do these memories satisfy the data needs for the accelerator to operate as identified above? (answer may involve describing what data lives in pipeline registers as well)
- What bandwidth does this use from the main memory?

`oval` and `ocare` for a single object must be completely partitioned and placed in registers in order to support the C, D unrolling.

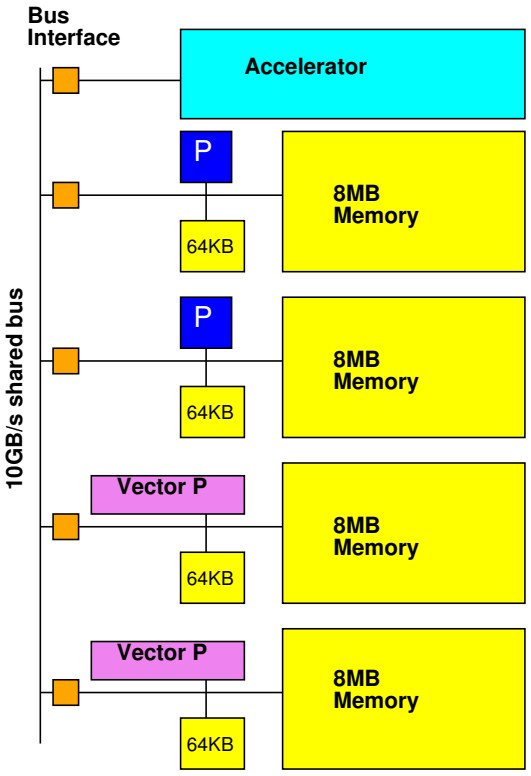
`findobj` needs a (WMAX-WMIN+OSIZE)^2 =130^2 window of the image in order to compute. We use OSIZE=30 local memories as line buffers (maybe 30-1) to hold lines reused in the C-D window filter. We use 30 length 30 shift registers to hold reused values as x increments. Together these provide the 900 image pixels that the unrolled C, D datapath needs on every cycle. With the line buffers and shift registers in place, the `findobj` accelerator needs only one image pixel per cycle during operation.

To setup a findobj call, we need to send $2 \times 2 \times 30^2$ bytes for `oval` and `ocare` and 2×130^2 for the image. 1800 bytes for each of `oval` and `ocare` are larger than 256, so streamable. Each line is $2 \times 130 = 260$ bytes, so also, independently streamable. We need a total of 3,600+33,800=37,400 bytes for every object or 37M bytes every 10 ms, which is 3.7GB/s.

To hit the 10 ms goal with this unrolling, the accelerator must run continuously. So, we must move the `ocare` and `oval` for an invocation of `findobj` during the previous invocation. This means we'll need a second set of `ocare`, `oval` registers to shift in and hold the new values during operation. Similarly, we must move the first 30 lines of the image window during the previous invocation. With care, it might be possible to use the same line buffers, but a simpler solution would be another set of line buffers for alternating invocations.

6. Assuming you use the accelerator from Problem 5 to accelerate the bottleneck loop, how do you implement the remaining loops to achieve the real-time goal of 10 ms per invocation of `warn_collide`?

   - Assume you can have any number of processors like the baseline processor, each with their own pool of 8MB of memory.
   - Assume you can have any number of vector processors that can perform 8 identical operations in a cycle, als with their own pool of 8MB of memory.
     - you can stream full vectors into the vector register file at the full streaming rate for the memory
   - Assume the vector processor is twice the size of the baseline processor.
   - Assume the communication among processors, vector processors, and accelerator is a single bus that can support one 10GB/s transfer at a time.
   - Minimize the Hardware you use.
   - Sample system with two baseline processors, two vector processors, and accelerator shown below.



   (a) Summarize the number of processor of each type you use.

   6 baseline processors

(b) Describe what loops (or portions of loops) run on which hardware.

Stragey: split L data parallel across all 6 processors. Run rest on Processor 0.

			seconds
E	Processor 0	(2×1000)/(10GB/s)	2×10^{-7}
F	All processors for own portion	(2(1000/6)×1000)/(10GB/s)	3.3×10^{-5}
H	All processors for own copy (not really executed each time)		
J	(on accelerator)		
K	Processor 0	$(82 + 4 \times 5) \times 1000$	10^{-5}
	Processor 0 copy data for L	$(2 \times 1024 \times 1024 + 3 \times 2 \times 1000)/(10\text{GB/s})$	0.0002
L	All processors	$(4 \times 1000)/(10\text{GB/s})$ (stream x, y, dfx, dfy)	4×10^{-6}
		$(1000/6) \times 1000 \times (49)$ M(18): 2×5 for `future`, 4 `x`, `y`, `dfx`, `dfy`; 4 +, * L(18): 2×5 for `future`, `collide_at_f`, 4 `x`, `y`, `dfx`, `dfy`; 4 +, * O(13): 5 for `future`, 4 `x`, `y`, `dfx`, `dfy`; 4 +, *	0.0082
P	All processors with per processor reduce	$(1000/6) \times 1000 \times (8)$ 5 `collide_at_f`, 1 for each `collide` and min	0.0014
	Processor 0 stream back per processor collide	$(5 \times 1000 \times 2)/(10\text{GB/s})$	10^{-6}
P	Processor 0	finish collide reduce $1000 \times 6 \times 8$	0.0000048
R	Processor 0	1000×3 (`collide`, `ostream`, `tstream`)	2×10^{-6}
		Total	0.0098562

9.9 ms just under 10 ms requirement.

(c) Describe when and how data is moved (to various memories for these processors).

- Clear E, F, and H as streaming operations. Setup 256B of all zeros in the 64KB local memory (and 256B of LOOKAHEAD) and use streaming to copy over `collide`, `collide_at_f`.
- After K, stream `image`, `x`, `y`, `dfx`, `dfy` to other 5 processors.
- After L, stream per processor `collide` back to first processor.
- Within each processor, stream `x`, `y`, `dfx`, `dfy` into 64KB local memory for operation within L.

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 students 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 persons paper, article, or computer work and submitting it for an assignment, cloning someone elses 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 students 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 ones 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 students 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 ones 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 students responsibility to consult with the instructor to clarify any ambiguities.

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

ESE532, Fall 2019 — Midterm — Wednesday, October 9

- Exam ends at 11:50AM; begin as instructed (target 10:30AM)
  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.
- 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	10	10	20	20	100

Consider the following code to pick a set of nearest-neighbor assignments (e.g., Ride Share Drivers to Passengers or Fire Trucks to Fires).

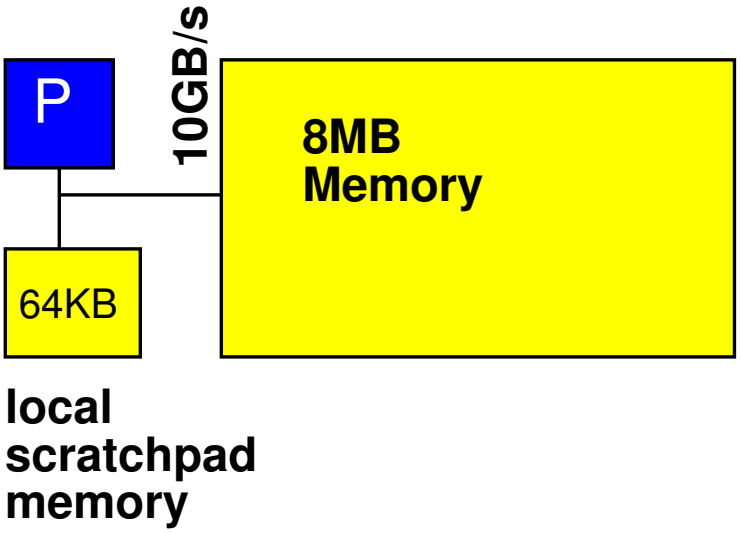
code_assign.c Wed Oct 09 20:33:13 2019 1

```
void assign() {
    // these are all in the main memory
    uint32_t driver_x[AVAILABLE];
    uint32_t driver_y[AVAILABLE];
    uint16_t passenger_x[TARGETS];
    uint16_t passenger_y[TARGETS];
    uint64_t distances[TARGETS][AVAILABLE];
    uint16_t matches[TARGETS][TARGETS];
    uint16_t match[TARGETS];

    get_drivers(driver_x,driver_y); // in 10*AVAILABLE cycles all memory
    get_passengers(passenger_x,passenger_y); // in 10*TARGETS cycles all memory
    compute_distances(driver_x,driver_y,passenger_x,passenger_y,distances);
    best_matches(distances,matches);
    assign_matches(matches,match);
    //TYPO: send_assignments(match); // in 10*TARGET cycles all memory
    send_assignments(match); // in 10*TARGETS cycles all memory
  }
```

code_operators.c Wed Oct 09 20:37:09 2019 1

```
#define TARGETS 100
#define AVAILABLE  1000

uint64_t distance(uint32_t x1, uint32_t y1, uint32_t x2, uint32_t y2) {
     int32_t dx=x1-x2;
     int32_t dy=y1-y2;
     return(dx*dx+dy*dy);
   }

void compute_distances(uint32_t driver_x[AVAILABLE],
                       uint32_t driver_y[AVAILABLE],
                       uint16_t passenger_x[TARGETS],
                       uint16_t passenger_y[TARGETS],
                       uint64_t distances[TARGETS][AVAILABLE]) {
    for (int p=0;p<TARGETS;p++) // loop A
       for (int d=0;d<AVAILABLE;d++) // loop B
          distances[p][d]=distance(driver_x[d],driver_y[d],
                                   passenger_x[p],passenger_y[p]);
    return;
   } // compute_distances

void best_matches(uint64_t distances[TARGETS][AVAILABLE],
                  uint16_t matches[TARGETS][TARGETS])      {

    uint64_t p_distances[AVAILABLE]; // in scratchpad memory
    uint16_t p_matches[TARGETS];     // in scratchpad memory
    for (int p=0;p<TARGETS;p++) {  // loop C
       for (int d=0;d<AVAILABLE;d++) // stream copy distances for p
          p_distances[d]=distances[p][d];
       closest_available(p_distances,p_matches);
       for (int m=0;m<TARGETS;m++) // stream copy matches for p
          matches[p][m]=p_matches[m];
       } // for p
    return;
} // best_matches

void closest_available(uint64_t distances[AVAILABLE],
                       uint16_t matches[TARGETS]) {
   // the matches result is an ordered list (smallest to largest)
   //  of the TARGETS nearest(smallest distance) available resources (drivers)

   // implementation omitted -- we will ask you to supply in Question 7.

   // for Question 1--5, assume
   // closest_available requires 4*TARGETS*AVAILABLE compute cycles
   //                        and 4*TARGETS*AVAILABLE memory cycles
   //                    critical path is TARGETS cycles (or, equivalently, ns)
}

void assign_matches(uint16_t matches[TARGETS][TARGETS], uint16_t match[TARGETS])
{
    // ERROR in exam: uint16_t driver_match[DRIVERS];  // in scratchpad memory
    uint16_t driver_match[AVAILABLE];  // in scratchpad memory
    for (int d=0;d<AVAILABLE;d++) driver_match[d]=0; // assume free
    for (int p=0;p<TARGETS;p++) // loop F
         for (int m=0;m<TARGETS;m++) // loop G
             if (driver_match[matches[p][m]]==0) {
                 match[p]=matches[p][m];
                 driver_match[matches[p][m]]=p;
                 break; // out of the for m loop
              } // if driver available
    return;
}
```

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, and multplies 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, compare, or add per cycle and runs at 1 GHz.
- Data can be transfered from the 8MB main memory at 10 GB/s when streamed in chunks of at least 192B. 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 64KB of data. Data can be streamed into the local scratchpad memory at 10 GB/s. Non-streamed accesses to the local scratchpad memory take 1 cycle.
- By default, all arrays live in the main memory.
- Arrays `p_distances`, `p_matches`, and `driver_match` live in local scratchpad memory.
- Assume scalar (non-array) variables can live in registers.
- Assume all additions are associative.
- Assume comparisons, adds, and multiplies take 1 ns when implemented in hardware accelerator, so fully pipelined accelerators also run at 1 GHz. A compare-mux operation can also be implemented in 1 ns.
- Data can be transfered to accelerator local memory at the same 10 GB/s when streamed in chunks of at least 256B.

1. Simple, Single Processor Resource Bounds

   Give the single processor resource bound time for each function.

function	Compute	Memory
compute_distances	5×10^5	5×10^6
best_matches	4×10^7	$4.0082 \times 10^7+$
assign_matches	10^4	2.2×10^5
assign	4.1×10^7	4.5×10^7

compute_distances compute: TARGETS*AVAILABLE*5

compute_distances memory: TARGETS*AVAILABLE*5*10

best_matches compute: TARGETS*4*TARGETS*AVAILABLE

best_matches memory: TARGETS(AVAILABLE*(8/10) + 4*TARGETS*AVAILABLE + TARGETS*(2/10))
(8/10) and (2/10) terms are for streaming transfers before and after closest_available calls

assign_matches compute: TARGETS*TARGETS*1

assign_matches memory: TARGETS*TARGETS*(2+2*10)
first 2 is readers to driver_match; second is to match and matches assuming only read matches[p][m] once from memory and use 3 times. Otherwise second 2 will be 4.

rest of assign is 10*AVAILABLE+20*TARGETS for 1.2×10^4

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

   (a) What function is the bottleneck? (circle one)

   compute_distances
   (best_matches)
   assign_matches

   (b) What is the Amdahl's Law speedup if you only accelerate the identified function?
   $(8.6 \times 10^7)/(5.7 \times 10^6) = 15$

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	all drivers, targets independent
B	Y	all drivers, targets independent
C	Y	each target closest_available independent
F	N	availability of closest drivers impacted by previous selections; must resolve previous selection before can process target p
G	N	each choice depends on previous availability check; must check each in turn to make selection. We could do parallel-prefix selection to do this in log-time, but that's beyond what we've discussed in class.

4. Identify streaming opportunties between functions. When streaming is possible, what granularity of data can be usefully sent between the functions? [i.e., producer can generate and consumer can operate upon without waiting for additional data from the producer.] Report as a size in bytes and the logical data structure (or part of a data structure) to which this corresponds.

   (a) compute_distances → best_matches

   8*AVAILABLE=8000 Bytes for rows of `distance`.
   While `compute_distances` can produce each distance independently, `best_matches` needs an entire row for a particular passenger to stream to `closest_match`. Each `closest_match` call is independent, so can operate concurrent with `compute_distances` computing the next row.

   (b) best_matches → assign_matches

   2*TARGETS=200 Bytes for rows of `matches`.
   `best_matches` is producing each passenger match row indepdently and streaming them as a group into memory. As each match row is available, `assign_matches` can process it and identify a match. Since matches are assigned in the same order as produced by `best_matches`, `assign_matches`, we can process a passenger assignment while `best_matches` is producing the ordered matches for the next driver.

5. What is the critical path for the entire computation as captured in the `assign` function?

compute_distances: 3 (subracts, multiplies, add)

best_matches: closest_available= TARGETS

assign_matches: $TARGETS^2$

Total critical path: 10,103

Depending on why the times for get_drivers, get_passengers, send_assignmen are what they are, we could include get_drivers, get_passengers, send_assignments for another 10*(AVAILABLE+2*TARGETS)=10,200. Omitting them assumes they are memory bottlenecks that could be avoided with appropriate engineering.

---

assign_matches using parallel-prefix could be $TARGETS * (1 + \log_2(TARGETS)) = 800$, making total around 903

6. Rewrite the body of `compute_distances` to minimize the memory resource bound by exploiting the scratchpad memory.

   - Annotate what arrays live in the local scratchpad
   - Account for total memory usage in the local scratchpad
   - use for loops that only copy data to denote the streaming operations

   Estimate the new memory resource bound for your optimized `compute_distances`.

   Code on facing page.

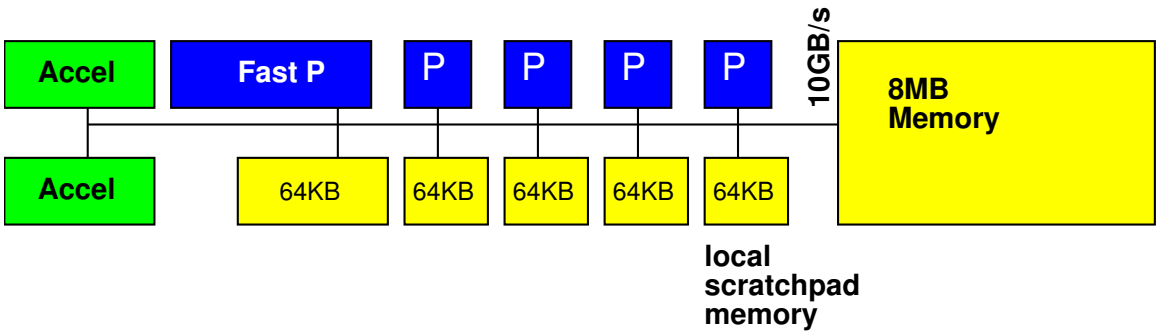
   Uses 16,400 B of scratchpad memory.

   Memory Resource Bound: $4000/10 + 4000/10 + 200/10 + 200/10$ +100*1000*5+100*(8000/10)=580840=5.8×10^5

(This page intentionally left mostly blank for answers.)

```
void compute_distances(uint32_t driver_x[AVAILABLE],
                       uint32_t driver_y[AVAILABLE],
                       uint16_t passenger_x[TARGETS],
                       uint16_t passenger_y[TARGETS],
                       uint64_t distances[TARGETS][AVAILABLE]) {

    uint32_t local_driver_x[AVAILABLE]; // scratchpad 4000 B
    uint32_t local_driver_y[AVAILABLE]; // scratchpad 4000 B
    uint16_t local_passenger_x[TARGETS]; // scratchpad 200 B
    uint16_t local_passenger_y[TARGETS]; // scratchpad 200 B
    uint64_t p_distances[AVAILABLE]; // scratchpad  8000 B

    // streaming read into locals
    for (int d=0;d<AVAILABLE;d++) local_driver_x[d]=driver_x[d];
    for (int d=0;d<AVAILABLE;d++) local_driver_y[d]=driver_y[d];
    for (int p=0;p<TARGETS;p++) local_passenger_x[p]=passenger_x[p];
    for (int p=0;p<TARGETS;p++) local_passenger_y[p]=passenger_y[p];


    for (int p=0;p<TARGETS;p++)  { // loop A
       for (int d=0;d<AVAILABLE;d++) // loop B
          p_distances[d]=distance(local_driver_x[d],local_driver_y[d],
                                  local_passenger_x[p],local_passenger_y[p]);
           // streaming write of distances row
           for (int d=0;d<AVAILABLE;d++) distances[p][d]=p_distances[d];
          }
    return;
   } // compute_distances
```

7. Consider a substrate with 4 simple processors (1 GHz as previously outlined), 1 fast processor (3 GHz, with everything running 3× as fast except data transfer from main memory), and 2 accelerators. The accelerators are pipelined and designed to start one call to `closest_available` each cycle;[1] pipeline depth is TARGETS. (You wlll look at the accelerator in Problem 8, but do not need to know its internal details to solve this problem). Describe how you would map the computation onto these heterogeneous computing resources. Describe how you would use the scratchpad memories as necessary beyond what you've already answered in Problem 6. Estimate the performance your mapping achieves.



Accelerators perform best_match in 100 cycles plus 100 cycles to drain pipeline. Place on two and this runs in 50+100=150 cycles. All but final call to closest_available overlapped with compute_distances.

Consistent with Problem 8, accelerators run in 100,000 cycles. With 2, it takes 50,000 cycles. Only non-overlapped call takes 1000+100 cycles.

Use 3 simple processors and fast processor for compute_distances. With Problem 6, compute_distance total is $(5 + 5.8) \times 10^5$ cycles. Divided by 6 for 3 simple + one 3× processors, 10.8/6=1.8×10^5 cycles.

Use 1 slow processor for everything else.

Use streaming to optimize memory references in assign_matches. Create local for match and a p_matches for one row of matches at a time. Stream in matches[p] into a local p_matches inside each F loop body. Stream out match at end. Memory in assign_matches (assuming read once from local version of matches[p][m] and use 3 times) goes to 100*(200/10)+4*100*100+200/10=4.2×10^4, bringing assign_matches to a total time of 4.2×10^4. Since 4.2×10^4 is less than the time on accelerators or compute_distances processors, this isn't the bottleneck. Streaming overlap means only the final 200/10+(4+1)*100+200/10 cycles for completing assign_matches adds time.

So, total time is 12,000 for get_drivers, get_passengers, and send_assignments, 1.8×10^5 for the compute_distances, and a final 540 cycles for the end of assign_matches.

[1] Error: to be consistent, should be every AVAILABLE cycles.

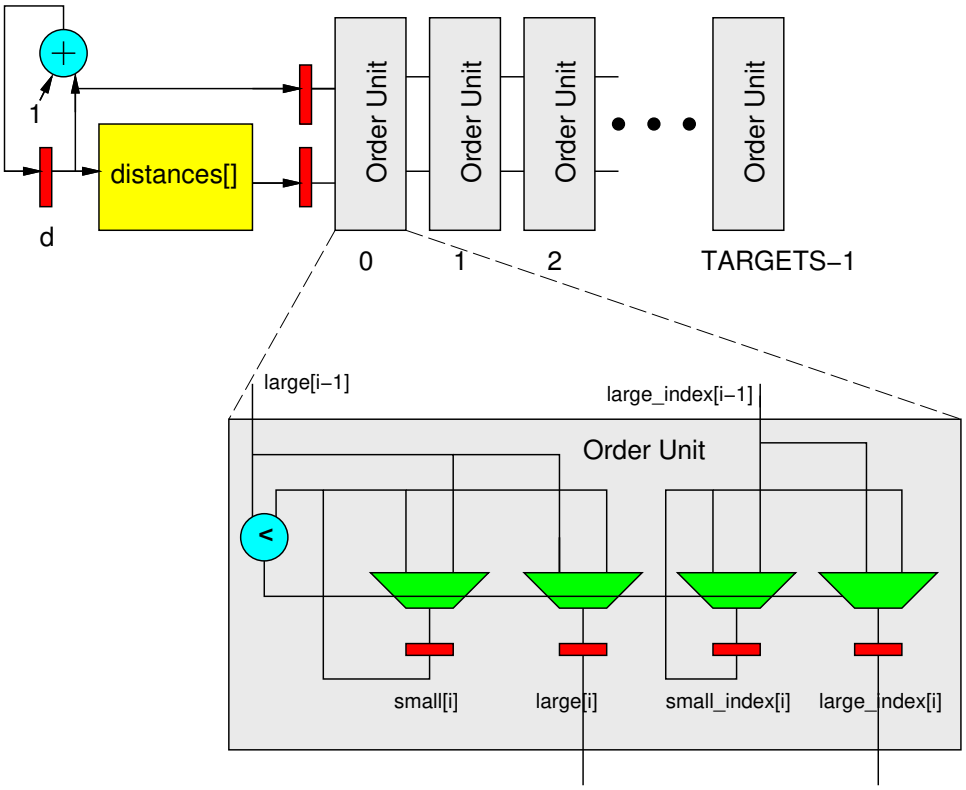
(This page intentionally left mostly blank for answers.)

compute_distances	3 simple + 1 fast	1.8×10^5
best_match	2 accelerators	adds 100 (as stated) adds 1100 (Problem 8)
everything else	1 slow	adds 12,540

Estimate total processing time as 1.8×10^5 cycles.

Same estimate if use consistent Problem 8 time for accelerators.

8. Consider the following accelerator for `closest_available` that processes one distance per cycle:



(Hint: this is performing an insertion sort one distance at a time, but keeping only the smallest TARGETS values.)

Assume:

- distances[] can be streamed from main memory into the local memory shown (with code already shown in best_matches)
- large[i] and small[i] can be reset to a maximum value, `MAX_VALUE`, in one cycle in hardware at the beginning of a call to `closest_available`; code for this reset in the C version is provided.
- small_index[i] and large_index[i] can be reset to a designated empty value, `EMPTY_VALUE`, in one cycle in hardware; code for this reset in the C version is provided.
- small_index[] becomes matches[] when the computation is completed and can be streamed into main memory (with code already shown in best_matches)

(a) Write C code that is consistent with the accelerator.

(b) Annotate the C code to explain how the C code was realized in the accelerator. (As appropriate, explain how unrolling, array partitioning, pipelining, dataflow streaming, and/or inlining were applied to the C code to get the implementation show.)

```
void closest_available(uint64_t distances[AVAILABLE],
                       uint16_t matches[TARGETS]) {


   uint16_t small_index[TARGETS]; // array partition complete
   uint16_t large_index[TARGETS]; // array partition complete
   uint 64_t small[TARGETS]; // array partition complete
   uint 64_t large[TARGETS]; // array partition complete


  for (int i=0;i<TARGETS;i++) small[i]=MAX_VALUE;
  for (int i=0;i<TARGETS;i++) small_index[i]=EMPTY_VALUE;
  for (int i=0;i<TARGETS;i++) large[i]=MAX_VALUE;
  for (int i=0;i<TARGETS;i++) large_index[i]=EMPTY_VALUE;


   for (int d=0;d<AVAILBLE;d++) { // pipeline
         if (distances[d]<small[0]) {
              large[0]=small[0];
              small[0]=distances[d];
              large_index[0]=small_index[0];
              small_index[0]=d;
          }
         else{
              small[0]=small[0];
              large[0]=distances[d];
              small_index[0]=small_index[0];
              large_index[0]=d;
          }
      for (int i=1;i<TARGETS;i++) { // unroll complete, pipeline
         if (large[i-1]<small[i]){
              large[i]=small[i];
              small[i]=large[i-1];
              large_index[i]=small_index[i];
              small_index[i]=large_index[i-1];
          }
         else {
              small[i]=small[i];
              large[i]=large[i-1];
              small_index[i]=small_index[i];
              large_index[i]=large_index[i-1];
          }
        } // i
       } // d
```

```
    for (int i=0;i<TARGETS;i++) matches[i]=small_index[i];
    return;
  }
```

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 students 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 persons paper, article, or computer work and submitting it for an assignment, cloning someone elses 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 students 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 ones 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 students 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 ones 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 students responsibility to consult with the instructor to clarify any ambiguities.

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

ESE532, Fall 2020 — Midterm Solutions — Wednesday, October 7

See exam as given for code, baseline system.

1. I certify that I have complied with the University of Pennsylvania's Code of Academic Integrity and the exam regulations `https://www.seas.upenn.edu/~ese532/fall2020/midterm_details.pdf` in completing this exam.

2. Estimate the time in cycles to run opt() sequentially on the single-processor baseline system described above. Show your work for partial credit consideration.

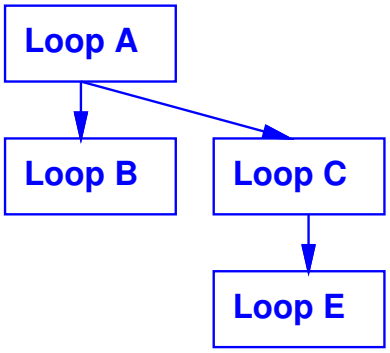
Loop A	×1000 iterations of area_param (4 ops + 1 array memory×5) × 10=90 time_param (5 ops, 2 array memory ×5)×10=150	240,000
Loop B	1000 × (1 (min) +5 (read)) × 2	12,000
Loop C/D	1000×1000 iterations of 6 ops 5 reads ×5 1 write ×1	33,000,000
Loop E	1000 × (2 op + 5 (read))	7,000
Total		33,259,000
	(two significant figures)	33,000,000

3. For the single-processor implementation, identify the bottleneck top-level loop.
   Loop C

4. What is the maximum Amdahl's Law speedup if we only accelerate the loop identified in Question 3. (show work for partial credit consideration)
   $\frac{33{,}259{,}000}{259{,}000} \approx 130$

5. Consider the coarse-grained dataflow graph for the top-level loops (A, B, C, E). What precedence constraints exist among these loops (what producer-¿consumer relationships exist).

Loop A → Loop B, Loop A → Loop C, Loop C → Loop E



6. Identify the loops that are data parallel.

7. Explain why each loop above is data parallel or not.

Which Loop	Data Parallel	Explanation
Loop A	T	independent calculation of each area, time
Loop B		reduce – min is an associative operation
Loop C	T	each dom[i] completely independent
Loop D		reduce – only dependence is add 1; add is an associative operation
Loop E		reduce only dependent is add 1; add is an associative operation
Loop F		reduce – only dependence is sum into res; add is an associative operation
Loop G	F	each iteration sequentially dependent on res from previous iteration

We haven't had a chance to talk in depth about reduce operations. The cleanest case would be to identify them as their own category between data parallel and things that require sequentialization. For the final, we'll likely ask you to make that classification. Here, we will accept a classification of either data parallel or not with an appropriate explanation.

8. What is the Critical Path Latency Bound for the opt() function?

Loop F is a reduce operation, so can happend in 3 cycles for the and, shift, and multiply plus 5 cycles for the read from a, followed by $\log_2(10)$ operations for the sum, or a total of 15 cycles for area_param.

Loop G is serialized on res with an II of 3 (See answer to problem 11). All of the t1, t2 reads (5 cycles), and, shift, and multiplies (3 cycles for those) can occur in 8 cycles before the cyclic portion of the loop. So, it takes $8+3 \times 10=38$ cycles for time_param.

Loop A	max(area_param (F),time_param (G)) since data parallel	38
Loop B	5 (read)+$\log_2(1000)$ (min reduce)	15
Loop C/D	5 (all reads)+ 1 (all compares)+ 2 (ands) + $\log_2(1000)$ (sum reduce)	18
Loop E	5 (read)+1 (compare) + $\log_2(1000)$ (sum reduce)	16
Total		87

For this exam, we will accept sequentialization of reduces since we haven't emphasized it, but we will not on the final. Should identify data parallel things can run in parallel, and that things like area_params and time_params can run in parallel with each other.

9. Accelerate the code on the baseline system by using the scratchpad memory.

   Show your revisions to the code. You only need to show the code you revised.

   Hint: Since we're only asking for performance numbers to two significant figures, don't waste time on speedups that will have an impact of less than 1% .

   Dominant time in Loop C/D so focus on that.

```
for (int i=0;i<NUM_POINTS;i++) // loop C
  {
     uint64_t ai=a[i];
     uint64_t ti=t[i];
     uint64_t ajnext=a[0];
     uint64_t tjnext=t[0];
     for (int j=0;j<NUM_POINTS;j++) // loop D
      {
       aj=ajnext;
       tj=tjnext;
       ajnext=a[j+1];
       tjnext=t[j+1];
       if ((i!=j) && (aj<=ai) && (tj<=ti)) domi++;
      }
     dom[i]=domi;
   }
```

10. Estimate the runtime and speedup for your revised code in Question 9.

    Body of loop now executes in: 2 cycles to issue ajnext and tjnext reads, 3 cycles for comaprisons, 2 cycles for ands, 1 cycle for add = 8 cycles. Note that there are 6 cycles in ops before getting back to the next read; so, there is time for ajnext and tjnext values to come back from memory before they are needed.

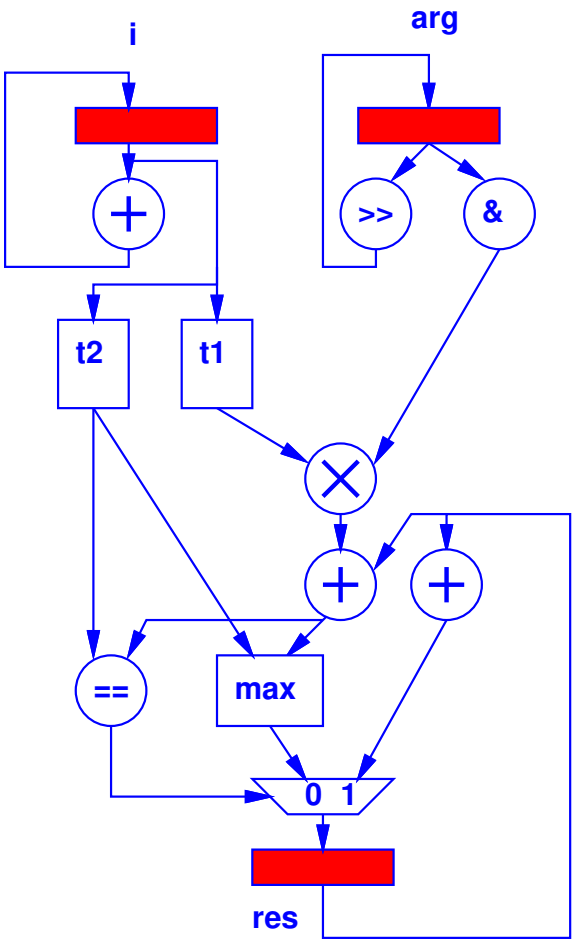
    Total runtime drops from 33M to 8.3M.

    Original case had 33 cycles per loop iteration, new version has 8, so speedup is roughly $\frac{33}{8} = 4.1$. 4.0 using complete cycle estimate.

11. When pipelined, what is the minimum clock cycle time achievable for loop G in time_param()?

    [Do not unroll the loop. Think about pipelining the loop body to start one new iteration of the body on each clock cycle.]

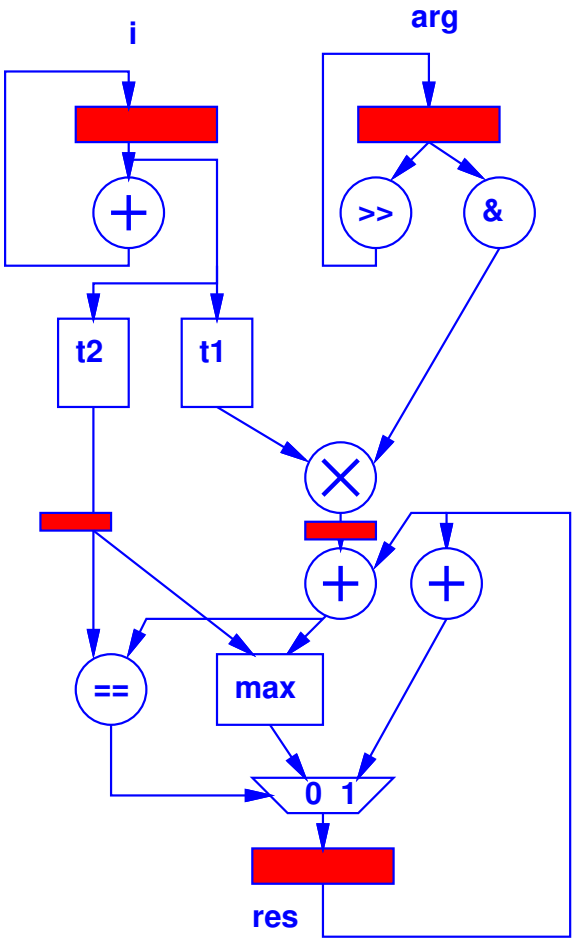
    3



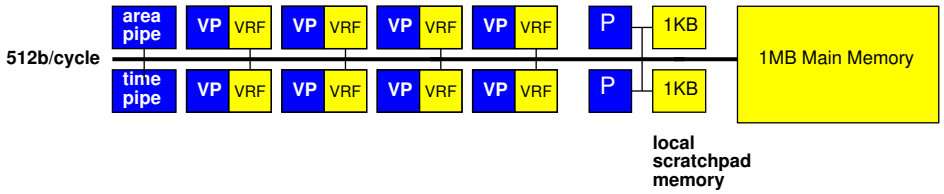
Largest loop is res→add (for b*t1[i]+res)→max→mux→res. So delay of between registers is 3.

In practice, the mux is fast, so a second-order term compared to add, max, or compare. However, we didn't give you a separate time for the mux from the compare-mux operation. In any case, we will take 2, also.

12. Design a pipeline for loop G in the time_param() calculation that achieves the minimum cycle time (in ns) [as identified in the previous question].



13. Describe how to map the multi_opt() computation to the following heterogeneous system to maximize the throughput of opt() calculations.



- 1 instance of 1 GHz, II=1 area_param (loop F) calculation pipeline (assume have local stores for ap1 that can be used for each function invocation)
- 1 instance of time_param (loop G) calculation pipeline (as you designed above) (assume have local stores for tp1, tp2 that can be used for each function invocation)
- 2 single-issues, baseline processors (P), each running at 1 GHz.
- 8 Vector Processors (VP) with 8, 64b-wide vector lanes, each running at 1 GHz. (for the loops that are data-parallel, you may assume computation achieves the resource bound)
- There is a shared, 512b wide path to main memory available to the hardware pipelines and the Vector Processors. It can transfer one contiguous blocks of 512b into a Vector Register File (VRF) or hardware pipe each 1 ns cycle, but it takes 5 cycles of latency before a fetched value is available in the VRF or pipe. The single-issue, baseline processors can only move 64b from the main memory in cycle.

Use respective hardware pipelines for Loop A.

Use one single-issue processor for Loop B and one for Loop E.

Split Loop C into 8 independent blocks of 125 values and assign to each of the 8 VPs.

14. Estimate the throughput of your mapped multi_opt() design in cycles (1 ns cycles) per opt() calculation completion.

    As split, each loop runs on independent resources. The loop graph can be pipelined at the coarse-grain level of loops. So, the throughput limit is for the slowest stage of the coarse-grain pipeline.

    Loop A runs in $3 \times 1000 + 8$ cycles due to the II of 3 for time_param.

    Loop B and E run in 12,000 and 7,000 cycles as established in problem 1.

    Loop C potentially runs in $\frac{125 \times 1000 \times 8}{8}$ =125,000 cycles. However, for that to work, we must deliver $2 \times 8 \times 64$=1024 bits (8 values of a[] and t[]) to each of 8 VPs every 8 cycles. In 8 cycles, the network can deliver 8×512b, or half the bandwidth needed to keep the 8 VPs running at full speed. So, the system is bottlenecked on the memory and will take 250,000 cycles.

    So, the total throughput is one opt() calculation every 250,000 cycles.

15. Estimate the latency of the opt() calculation for your mapped multi_opt() design in cycles (1 ns cycles) to compute each opt() result from when opt() first looks at its ap1, tp1, tp2 inputs.

    For a single input, it needs to compute through Loop A (3008 cycles), Loop C (250,000 cycles) (and Loop B at 12,000 cycles can run in parallel with Loop C), then Loop E at 7,000 cycles, for a latency of 260,008 cycles.

    Assuming we start all stages at the same time, Loop A finishes quickly, but its data must wait for the VPs in Loop C to start. So, it would also be reasonable to say the latency was $3 \times 250,000$=750,000. Or, we could at least recognize that E will produce a value early, so it takes $2 \times 250,000 + 7,000$=257,000.