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.

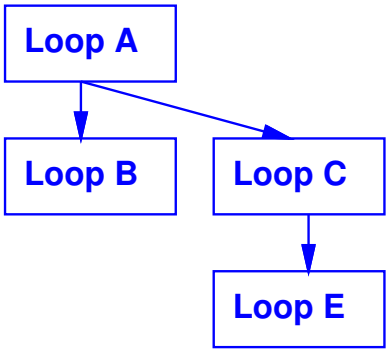
Loop A	×1000 iterations of area_param (4 ops + 1 array memory×5) × 10=90 time_param (5 ops, 2 array memory ×5)×10=150	240,000
Loop B	1000 × (1 (min) +5 (read)) × 2	12,000
Loop C/D	1000×1000 iterations of 6 ops 5 reads ×5 1 write ×1	33,000,000
Loop E	1000 × (2 op + 5 (read))	7,000
Total		33,259,000
	(two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

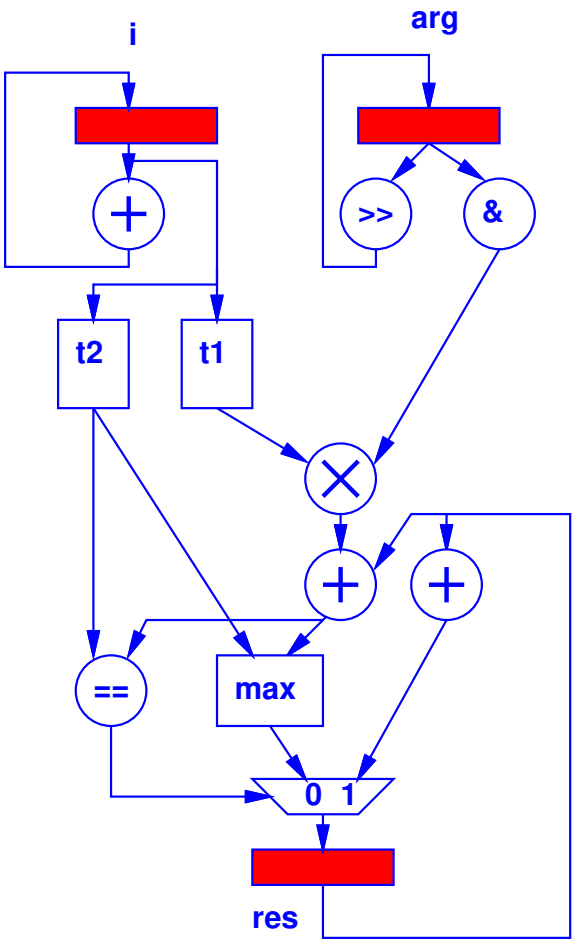
    Total runtime drops from 33M to 8.3M.

    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.]

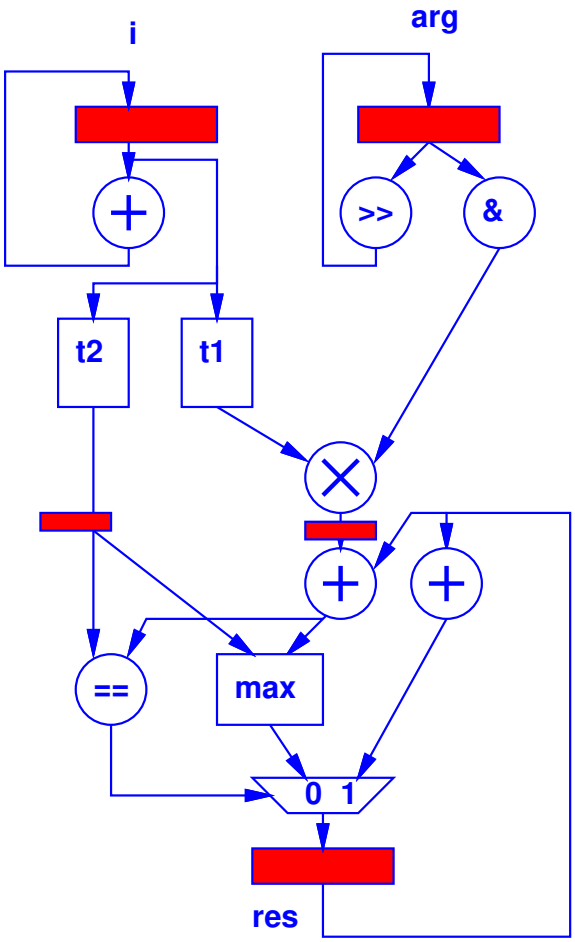
    3



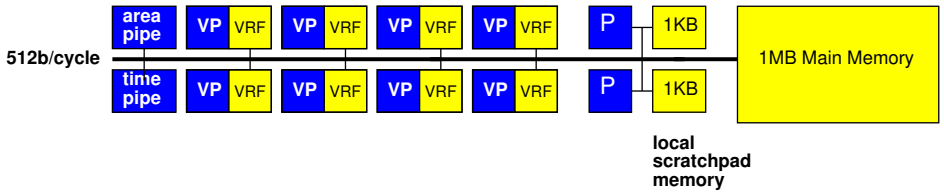
Largest loop is res→add (for b*t1[i]+res)→max→mux→res. So delay of between registers is 3.

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.