HW5 Answer

4.10 Vector processor requires:

- (200 MB + 100 MB)/(30 GB/s) = 10 ms for vector memory access +
- 400 ms for scalar execution.

Assuming that vector computation can be overlapped with memory access, total time = 410 ms.

The hybrid system requires:

- (200 MB + 100 MB)/(150 GB/s) = 2 ms for vector memory access +
- 400 ms for scalar execution +
- (200 MB + 100 MB)/(10 GB/s) = 30 ms for host I/O

Even if host I/O can be overlapped with GPU execution, the GPU will require 430 ms and therefore will achieve lower performance than the host.

4.12 a. Reads 40 bytes and writes 4 bytes for every 8 FLOPs, thus 8/44 FLOPs/byte.

b. This code performs indirect references through the Ca and Cb arrays, as they are indexed using the contents of the IDx array, which can only be performed at runtime. While this complicates SIMD implementation, it is still possible to perform the type of indexing using gather-type load instructions. The innermost loop (iterates on z) can be vectorized: the values for Ex, dH1, dH2, Ca, and Cb could be operated on as SIMD registers or vectors. Thus, this code is amenable to SIMD and vector execution.

c. Having an arithmetic intensity of 0.18, if the processor has a peak floating-point throughout > (30 GB/s) × (0.18 FLOPs/byte) = 5.4 GFLOPs/s, then this code is likely to be memory-bound, unless the working set fits well within the processor's cache.

d. The single precision arithmetic intensity corresponding to the edge of the roof is 85/4 = 21.25 FLOPs/byte.

4.13 a. 1.5 GHz × 0.80 × 0.85 × 0.70 × 10 cores × 32/4 = 57.12 GFLOPs/s

b. **Option 1:**
1.5 GHz × 0.80 × 0.85 × 0.70 × 10 cores × 32/2 = 114.24 GFLOPs/s
(speedup = 114.24/57.12 = 2)
Option 2:
1.5 GHz × 0.80 × 0.85 × 0.70 × 15 cores × 32/4 = 85.68 GFLOPs/s
(speedup = 85.68/57.12 = 1.5)
Option 3:
1.5 GHz × 0.80 × 0.95 × 0.70 × 10 cores × 32/4 = 63.84 GFLOPs/s
(speedup = 63.84/57.12 = 1.11)

4.14 a. Using the GCD test, a dependency exists if GCD (2,4) must divide 5 − 4. In this case, a loop-carried dependency does exist.

b. *Output dependencies*
S1 and S3 cause through A[i]
Anti-dependencies
S4 and S3 cause an anti-dependency through C[i]
Re-written code

```
for (i=0;i<100;i++) {
  T[i] = A[i] * B[i]; /* S1 */
  B[i] = T[i] + c; /* S2 */
  A1[i] = C[i] * c; /* S3 */
  C1[i] = D[i] * A1[i]; /* S4 */}
```

True dependencies
S4 and S3 through A[i]
S2 and S1 through T[i]

c. There is an anti-dependence between iteration i and i + 1 for array B. This can be avoided by renaming the B array in S2.

4.15 a. Branch divergence: causes SIMD lanes to be masked when threads follow different control paths.

b. Covering memory latency: a sufficient number of active threads can hide memory latency and increase instruction issue rate.

c. Coalesced off-chip memory references: memory accesses should be organized consecutively within SIMD thread groups.

d. Use of on-chip memory: memory references with locality should take advantage of on-chip memory, references to on-chip memory within a SIMD thread group should be organized to avoid bank conflicts.

4.16 This GPU has a peak throughput of $1.5 \times 16 \times 16 = 384$ GFLOPS/s of single-precision throughput. However, assuming each single precision operation requires four-byte two operands and outputs one four-byte result, sustaining this throughput (assuming no temporal locality) would require 12 bytes/FLOP × 384 GFLOPs/s = 4.6 TB/s of memory bandwidth. As such, this throughput is not sustainable, but can still be achieved in short bursts when using on-chip memory.