1 SIMD Architecture

(a) Describe SIMD, one way GPUs exploit parallelism. Apart from performance what are
the other benefits of SIMD execution?

(b) How can a basic processor pipeline be modified to allow SIMD execution?

(c) What are the implications for other parts of the processor for providing SIMD (e.g. the
data cache)?

(d) What is the basic way GPUs allow branching within the lanes SIMD instructions?

   (i) What does this imply about the way you should write code for GPUs?
   (ii) Why does it make sense to do as little work as possible after an infrequently-taken
        branch?

(e) Assuming each line below is a single instruction, what is the efficiency of the code
(ignoring predicate manipulation instructions)?

   \text{ld } X[i:i+3]
   \text{if } (i \% 2 == 0)
      \text{mul a, } X[i:i+3]
   \text{else}
      \text{mul b, } X[i:i+3]
   \text{if } (i \% 4 == 0)
      \text{add c, } X[i:i+3]
      \text{mul 2, } X[i:i+3]
   \text{endif}
   \text{endif}

(f) For example the following loop:

   \text{for(i=1;i<N;i++)}
   \{\text{a[i] = b[i] + c[i];}\}

   could be replaced by a single expression of

   \text{a(1:N) = B(1:N) + C(1:N)}

   given the vector register has the size of N.

   This question uses this idea to do loop vectorization using SIMD architectures.

   Considering the following loop:
for (i=1; i<N-1; i++) {
    a[i] = b[i];
    c[i] = a[i] + b[i];
    d[i] = c[i+1];
}

Based on the example given above, re-write the code after vectorization, you can assume an infinite length of the vector register.

2 GPU Architecture

(a) Why do GPUs rely on multithreading as well as SIMD? What bottleneck does multithreading target?

(b) How can a SIMD processor pipeline be modified to allow multithreading?

(c) The warp scheduler chooses instructions to execute. What are its criteria for choosing and what other schemes could be implemented?

(d) What are the types of memory available within a GPU and what are their relative advantages and disadvantages?

(e) What types of application best suit GPUs and how can you program to take advantage of the various forms of parallelism available?

3 Computer Architecture Futures

(a) Explain the “dark silicon” effect and why couldn’t we indefinitely scale up the number of cores to continue the trend as specified in the Moore’s Law?

(b) Describe ARMs big.LITTLE system.

   (i) What is the point of having two types of core and when might they be used?

   (ii) Would it be useful to have a third core type and, if so, what would its characteristics be?

(c) What types of application domain might benefit from approximate computing?

4 Part II project discussion

Read the part II project description from the computer architecture group: http://www.cl.cam.ac.uk/~tmj32/projects.html

We will discuss the backgrounds and motivations of the following project.

1. Hardware transactional memory simulation
2. Optimising multi-threaded stalls
3. Function call parallelism
4. A fast lock-free software queue
5. Binary parallelisation
6. Ultra-fine-grained parallelisation
Before attending the supervision, please familiarise yourself with the following concepts from the Wikipedia:

- Speculative Execution
- Instruction Prefetch
- Producer-Consumer Problem

End of Problems