BGN: 2191A

P5

Q8

a. Modern SoCs need to perform a wide range of tasks, each of which is best suited for a specific type of processor core. Heterogenous computing allows the programmer of such systems to utilize e.g., the GPU to perform floating point calculations, or to use dedicated hardware accelerators such as video decoders to perform tasks more efficiently. The ability to execute tasks on the hardware best suited for each task (and likewise, having a large range of such hardware available) leads to faster program execution and significantly less power consumption.

b. The von Neumann bottleneck is the notion that the throughput (and by extension, efficiency) of modern computers is limited by the rate at which they can transfer data from one component to another (e.g., the processor to main memory). This idea suggests that while data transfers occur, the processor is idle for some time, and so speeding up the processor will simply increase the duration for which the processor is idle, leading to no net increase in speed.

   Modern SoCs do suffer this to an extent, but the effect has been significantly reduced by introducing a hierarchy of caches. Small caches can take advantage of locality, and since they are small, they can be close to the processor core, which reduces data transfer time.

c. Virtual memory provides isolation by allowing each application to internally use memory addresses for variables which are not the same as the physical addresses in main memory. The program uses these virtual addresses in its code, and they are then translated into physical addresses via the TLB when the program is being executed. This has no impact on the processors as contiguous blocks of memory in the virtual address space remain contiguous when translated (i.e., the translation is a constant shift for the whole virtual address space), so offsets between memory addresses are not affected by the translation.

   This provides isolation because the entire virtual address space is mapped into a specific chunk of physical memory. This means that a program cannot access memory addresses outside of its allocated chunk, which stops programs reading memory from other processes.

d. GPUs are able to hide memory access latency by running threads from other warps during memory access. The GPU has a warp scheduler which selects which warps to execute and when to execute them. It takes into account whether one warp's

threads are blocking due to a memory access and can execute another warp in the meantime.

e. Conditional flow is managed differently because GPUs are optimised to execute the same code across multiple threads in parallel on different data (of the same shape, i.e., different elements in an array). If one thread were allowed to conditionally branch while others did not, then the threads would be executing different instructions, and the performance gain is greatly diminished. Therefore GPUs use a mask to indicate whether or not an instruction should be executed. When a conditional is evaluated, the mask is updated to indicate for which of the threads does the condition hold. Then, such threads execute the body of the conditional block, while the others stall. After the condition block, the mask is flipped in the case of an “else” block, or reset if the conditional is finished. Nested conditionals can be achieved using a stack.

   Since CPUs do not execute multiple instances of the same code in parallel, they would see a performance *decrease* from the masking technique, as conditional branching allows the CPU to simply skip over unnecessary instructions rather than stall for the amount of time it would take them to execute.