# ICA_Supervision_4_Answers_Zhou_Kyra_Micha



## Question 1

Discuss an example of a problem that we can solve efficiently by using CUDA.

- CUDA can only run on NVIDIA hardware but give a better parallel programming performance.
- Developers can use CUDA to write programs in popular languages (C, C++, Java and Python, etc.) and add parallelism to their code with basic keywords.
- So CUDA is suitable for general-purpose computing on NVIDIA's GPU hardware.

💡 Comments:

## Question 2

Discuss an example of a problem that we can solve efficiently by using OpenCL.

- OpenCL is an open-source standard for cross-platform, parallel programming. It is often used to accelerate supercomputers, cloud servers, PCs, mobile devices, and embedded platforms.
- OpenCL is not just for GPUs; it also supports CPUs, FPGAs, etc. It provides an API that enables programs running on a host to load the OpenCL kernel on computing devices.

💡 Comments:

## Question 3

How is it possible that so many different devices, with completely different internal architecture, are compatible with OpenCL and enable acceleration of execution?

- The OpenCL has a platform model; it abstracts away the vendor-specific runtime and allows applications to
- The devices are all independent of each other; for instance, we can have one intel CPU device, two Nvidia GPU devices and one AMD GPU device, which gives us three platforms with different characteristics.

- The acceleration of execution is enabled via the kernel programming model, where concurrency is mapped via work-items and work-groups to execute the kernel.

💡 Comments:

# Question 4

In CUDA, what's the difference between a thread block and a warp?

- A thread block is a set of threads that can access the same shared memory and cooperate in the same kernel; synchronization occurs at the thread block level, so threads in a thread block can communicate.
- A warp is a set of threads that execute together on a single core.
- A thread block is composed of warps.

💡 Comments:

# Question 5

Why does OpenCL require an application to discover the available platforms and devices?

- Because the platform model abstracts away the vendor-specific runtime characteristics, it is an abstraction between the application and the platform/devices.
- OpenCL specifies a programming language for these devices, so to convert the codes into device-specific codes, we have to use an application programming interface (API) to control the platform and execute programs on the specific devices.

💡 Comments:

# Question 6

Compare and contrast the OpenCL programming model with CUDA.

- Similarity:
  - Both are designed for parallel programming
  - Both are supported by various operating systems
  - Both have good performance libraries, although CUDA has a more extensive one
- Differences:
  - CUDA is not a language or an API, it is a platform and programming model for parallel computing, and it accelerates general-purpose computing using GPUs. Developers can still write software in C or C++ and include parallelization using CUDA keywords.
  - OpenCL does not enable writing codes in C++, but you can work in a C programming language environment and directly with GPU resources.
  - CUDA is a platform that only works with NVIDIA hardware; OpenCL is an open-source specification that works for all hardware devices from different vendors.
  - OpenCL allows the choice of hardware and is portable from platform to platform.

💡 Comments:

# Question 7

What types of applications best suit GPUs and how can you program to take advantage of the various forms of parallelism available? Try to describe examples.

- GPU is designed for parallel processing, so applications like graphics and video rendering, particularly gaming software machine learning programs that involve picture manipulations
- Different parallelism includes:
  - Data parallelism: concurrently execute the same task on different data which take place on different cores
    - e.g., summing the contents of an array of size $N$. For a single-core system, one thread would sum the element from $0^{th}$ to $(N-1)^{th}$. For a dual-core system, one thread could sum the elements from $0^{th}$ to $(\frac{N}{2}-1)^{th}$ and the other could sum the element from $\frac{N}{2}^{th}$ to $(N-1)^{th}$, so the two threads can run in parallel on separate cores.
  - Task parallelism: concurrently execute different tasks on different cores
    - e.g., Two threads, each performing a unique statistical operation on the array of elements. One may be summing the contents of an array; the other way be calculating the variance of the values.
  - Instruction parallelism: the simultaneous execution of multiple instructions from a program.
    - e.g., in a for loop, if the loop iterations are independent of each other, then we can execute all instructions concurrently

```
//  assuming x,y,z are three arrays
for (int i = 1; i < 100; i++){
  x[i] = y[i] + z[i];
}
```

💡 Comments:

# Question 8(y2019p5q3)

$(a)$ Describe each of the four models defined by OpenCL's specification. [4 marks]

- OpenCL supports heterogeneous computing, it has:
  - (1) a platform model used to describe how the device looks inside: device (e.g., GPU) divided into computing units/multiprocessors, then divided into threads. A platform abstracts away the vendor-specific runtime applications used to choose the platform/device themselves
  - (2) an execution model used to describe how the hosts interact with devices: a context is the environment of execution (devices + memories + command queues), each device has one command queue, hosts will send commands to the queue for communication with the device, each command has a waitlist containing events of commands it depends on
  - (3) a kernel programming model used to describe how concurrency is mapped: the kernel is the code that runs on a work item (e.g., a C function), a program is collection kernels and other functions.
  - (4) a memory model used to describe the abstract memory kernel sues, defines what happens to each memory operation (e.g., values read, order of operations for memory consistency) as an abstraction between programmers and vendor implementations.

$(b)$ Describe the different types of memory available to OpenCL kernels. [4 marks]

- (1) Private memory per work-item (i.e., thread) has low latency and limited capacity and lasts for the lifetime of the work-item.
- (2) Local memory shared among a work-group (i.e., thread lock), stored off-chip so have large capacity but high latency, used for spilling registers when there aren't sufficient private memory to hold variables.

- (3) constant memory, which is visible to threads for all simultaneous read-only access, usually store constants and textures, can be cached to reduce memory traffic and increase OpenCL performance.
- (4) global memory, which is visible to all work-items, has high latency and high capacity, stored off-chip.

(c) Contrast how calls to a kernel, e.g. DAXPY, are invoked and grouped for execution in OpenCL compared with CUDA. [4 marks]

- For OpenCL, the basic unit of concurrency is work-item. Multiple work-items combine to form a work-group. All work-items in a work-group can access shared memory and cooperate and communicate. Work-groups are combined to form an NDRange, representing the computation space. The kernel is invoked for each work-item.
- For CUDA, the basic unit of concurrency is a thread; multiple threads combine to form a warp where all threads in the warp can execute together on the same core. Multiple threads form a thread block where all threads in a thread block can cooperate, communicate and access shared memory. Several threads blocks are then combined to form a grid which is the space of computation. Programmers can simply add keywords in the original C function to invoke the kernel.

💡 Comments:

## Question 9

Why is energy efficiency the “new fundamental limiter of processor performance”, as Borkar and Chien say?

- The performance improvement could not catch up with Moore's law on resistor density. With a higher density for resistors, the distance between resistors decreases and leads to charge leakage if all resistors work together, hence we are forced to have dark silicon where a proportion of the resistors will be inactive.
- Multi-core doesn't improve, or the improvement is limited, so we need to change the core structure fundamentally.

💡 Comments:

## Question 10

Describe Arm's big.LITTLE system.

- A heterogeneous computing architecture that couples relatively battery-saving and slower processor cores (LITTLE) with relatively more powerful and power-hungry ones (BIG).
- CPUs that run fast are different from CPUs that saves more power because it is a trade-off in CPU designs. So the big.LITTLE system is aimed to create a multi-core processor that can adjust better to dynamic computing needs and use less power.
- At one time, only one side will be active, but all cores have access to the same memory regions so that workloads can be swapped between BIG and LITTLE cores in the runtime.

💡 Comments:

## Question 11

When might it make sense to implement functionality in a specialized accelerator rather than within a general-purpose core?

- A specialized accelerator is designed for a specific task; for instance, Google's TPU is designed for machine learning. It will have a higher performance and be more energy-efficient for that particular task, but at the same time, more expensive and less useful for general tasks.

- A general-purpose core will be helpful in a variety of tasks but not as efficient as the specialized accelerator for that particular task.

💡 Comments:

## Question 12

Discuss a general multi-core CPU vs ASIC vs DSP.

- A multi-core CPU is a general-purpose device that can be adapted to various applications. It can increase performance by running multiple applications concurrently.
- ASIC (application-specific integrated circuit) is custom-designed for a particular application, implementing as many system functionalities as possible on a single die; this optimizes the number of transistors and clock cycles, reducing the unit power consumption at the expense of development time.
- DSP (digital signal processor) hard-wire the basic functions of many signal-processing algorithms optimizes transistor use and clock cycles for the required operations at the expense of flexibility.

💡 Comments:

## Question 13

Discuss approximate computing.

- A computer processor that does not compute a precise result is useful for some applications that do not require a precise result, an approximate result is good enough, and such chips use fewer circuits and much less energy.
- For instance, a search engine with no exact answer for a specific search query may provide many possible answers.
- It is used in machine learning as well, for example, in k-means clustering algorithm, allowing only 5% loss in classification in classification accuracy can provide 50 times energy saving compared to the fully accurate classification.

💡 Comments:

## Question 14(y2020p5q3)

$(d)$ Computational sprinting is a technique to increase the performance of a core for a brief amount of time by temporarily raising its frequency and voltage. Describe the types of workload where computational sprinting on a general-purpose core would out-perform a specialised accelerator. [4 marks]

- A specialized accelerator is designed for a specialized task. It would be better to use computational sprinting on a general-purpose core if the workload involves varied tasks.
- For example, when the responsiveness of multiple applications is essential. A typical example is on a smart device where many interactive applications are running with short bursts of computational demand.

$(e)$ Comment on the advantages and disadvantages of computing within a reconfigurable fabric (e.g. an FPGA) alongside the processor within the same SoC. [5 marks]

- Advantages:
  - FPGAs can be programmed at the logic level, hence they can implement faster and parallel processing of signals; these are hard to be executed by a processor; therefore, FPGAs can be used to execute specialized tasks faster, reducing the computation loads for a processor.
  - FPGA is programmable at the software level at any time so that it can be re-programmed or reused any number of times, even from remote locations.
  - FPGA development is cheaper and uses less costly tools.
- Disadvantages:
  - The programming language of FPGA is not as simple as C programming used in processor-based hardware, so we have to use different languages for the FPGA and processor on the same SoC.
  - The power consumption increases, and programmers cannot control power optimization in FPGA.
  - Putting an FPGA on the SoC limited the size of the processor.

💡 Comments:

# Question 15

Discuss the Spectre bug.

- Spectre is a security vulnerability that relies on speculative execution and tricks victims into loading secreted data. Because of branch prediction, programs may predict a wrong branch and load secreted information into the cache; Spectre uses the cache side channels to leak information.
- We deal with Spectre bug by hiding any speculative state using a filter cache or use a secure processor such as CHERI.

💡 Comments:

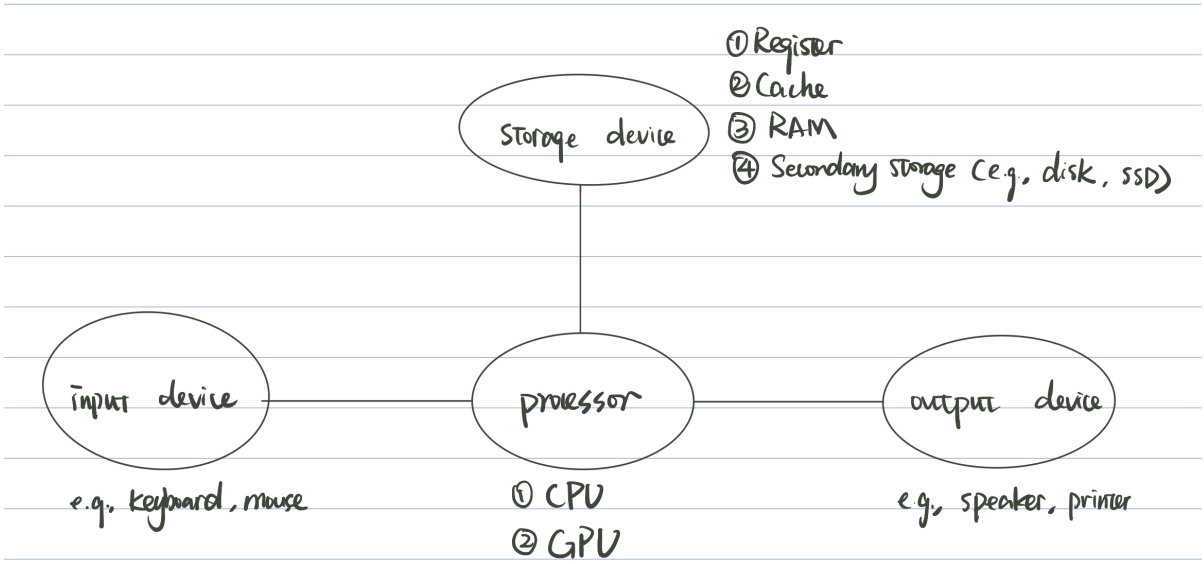
# Question 16

Create a diagram with all discussed elements of a computer system (e.g., cache, RAM). Discuss common units of time for operations that these elements perform (e.g., time to fetch memory from cache and RAM, time to execute an instruction).

- For storage device - time to fetch date:

| | Register | Cache | Main memory (RAM) | SSD | Disk |
|---|---|---|---|---|---|
| # clock cycles | 1 | 10 | 100 | 1M | 10M |

- We have a hierarchy of storage devices that are designed to cope with the trade-off between storage capacity and access latency. For instance, the cache is designed based on temporal and spatial locality to store data that are mostly likely to be fetched in the near future to reduce memory access overhead and improve processor performance.




Comments:

## Question 17

Write a 1-page essay (no longer) describing how topics discussed in different lectures are related and how in your opinion, they relate to software engineering. Focus on identifying conceptual solutions discussed in different lectures and explain how these abstract concepts are applied for different concrete problems, on different abstraction levels.

There are eight great ideas for computer architecture, but each idea faces a related challenge.

Firstly, Moore's law says the transistor density would double every two years; since the power consumption density keeps constant according to the Dennard scaling, the performance of a chip would roughly double every two years. However, it is not true recently as the performance cannot catch up with Moore's law. The main reason behind this is that higher transistor density leads to current leakage, so we are forced to have dark silicon where not every transistor works simultaneously; thus, we cannot have a significant performance improvement with the increasing number transistors. Therefore, energy efficiency becomes a challenge when designing chips. One way to overcome this challenge is to create specialized multi-core chips such as ASIC or specialized accelerators such as Google's Tensor Processing Unit, mainly used for specific machine learning tasks.

The second great idea is about using abstraction to simplify design. One example is using high-level programming languages such as C++, Java, and Python to abstract the complexity of programming with low-level programming languages such as assembly language and machine codes. This helps programmers to focus on their tasks without worrying about low-level details. However, it is always helpful to understand low-level programming to optimize specific tools such as compilers.

Four great ideas are related to improving the performance of the processing, including making the common case fast, processing via parallelism, pipelining and prediction. According to Amdahl's law, the performance of a processor is limited by the sequential part of the program, so new technology such as processing-in-memory is designed to overcome this problem. For parallelism, there are many different levels, including instruction-level, thread-level and data-level. Data-level parallelism is widely used in GPUs (Graphics processing units), which implemented the idea of SIMD (single instruction multiple data) and SIMT (single instruction multiple threads). Pipelining is another strategy to increase the throughput per unit time; it breaks the production line into stages and continuously moves instructions through stages. We also apply prediction during execution, including control-flow prediction, data-value prediction and data-access pattern prediction. If we predicted correct, then these could make the pipeline and memory subsystem more efficient.

The seventh idea is about memory hierarchy; since the access or processing speed between different computer components is not compatible, we are restricted by the slowest operation - accessing the main memory to load data. Hence caches are designed based on spatial locality and temporal locality to reduce memory access time. Since we have limited space on the chip, there is a trade-off between memory capacity and latency; we must construct a memory hierarchy.

The final great idea is related to reliability and security. The variation among transistors introduces reliability challenges, so we implemented dual-core lock-step and redundancy to improve reliability. Security vulnerabilities such as the Spectre bug will

impact the chip's performance, so we added filter cache and secure processors such as the CHERI processor to help avoid them.

💡 Comments:

# Question 18

Summarize the main message from "Lecture 15: Cuda, OpenCL" in 1-3 sentences?

- Both CUDA and OpenCL are GPU programming languages, but CUDA only targets NVIDA's GPUs and are suitable for general-purpose tasks, while OpenCL supports heterogeneous computing is portable and open-source.
- CUDA and OpenCL's programming models are similar. They both have a unit of concurrency(i.e., called thread in CUDA and work-item in OpenCL). They have a similar memory hierarchy (i.e., private memory, local memory, constant memory and global memory).

💡 Comments:

# Question 19

Summarize the main message from Lecture "16: Future directions. Energy efficiency. Performance. Reliability. Security" in 1-3 sentences?

- Energy efficiency is the new fundamental limit of processor performance; the problem is caused by dark silicon. It cannot be solved using multi-core processing, so we designed an accelerator for specific tasks and specialized multi-core chips to lower power consumption.
- According to Amdahl's law, the sequential part limits fast performance, so we designed processing-in-memory and other new memory technology to overcome.
- Reliability challenge is caused by variation among transistors, so we designed dual-core lock-step or embraced poor reliability in some applications that are not affected.
- Security vulnerability such as the Spectre bug causes leakage of secrete via wrong branch prediction are solved using filter cache and CHIRI secure processor.

💡 Comments:

# Question 20(BONUS)

- (a) Create an example of an SoC on which some processes and threads execute concurrently. Discuss sources of non-determinism starting from cache and pipelines to RAM access, network, scheduling, pre-emption, synchronization between threads… Consider there are also some cloud services. Discuss what range of delays non-determinism can cause.
- (b) Discuss the Meltdown bug and compare it with the Spectre bug.
- (c) Discuss Misra C in the context of security.
- (d) Create an example with CUDA and commit it to GitHub.
- (e) Create an example with OpenCL ("fortune favours the bold") and commit it to GitHub.

💡 Comments: