**Discuss pattern depth with sufficient examples to point out the assumptions that the four control flow patterns make, even though they are familiar.**

Parallel Patterns: A recurring combination of task distribution and data access that solves a specific problem in parallel algorithm design.

**Types of serial control patterns: sequence, selection, iteration, recursion**

1. Sequence

A picture containing chart

Description automatically generated

1. Selection

Chart, box and whisker chart

Description automatically generated

1. Iteration

Diagram

Description automatically generated

1. Recursion

A screenshot of a text message

Description automatically generated with low confidence

**Types of parallel control patterns: fork-join, map, stencil, reduction, scan, recurrence**

<http://ipcc.cs.uoregon.edu/lectures/lecture-5-patterns.pdf>

**Discuss the latency and throughput as the primary purpose of parallelization in a computation, how Amdahl's law apply on this?**

* Throughput: number of computing tasks per time unit. ie: 1000 credit card payments in a minute.
* Latency: delay between invoking the operation and getting the response. ie: maximum time taken to process a credit card transaction is 25ms.

When optimizing performance, an improvement in one factor (such as throughput) may lead to the worsening in another factor (such as latency).

[**Amdahl's Law**](https://en.wikipedia.org/wiki/Amdahl%27s_law) states that potential program speedup(theoretical latency) is defined by the fraction of code p that can be parallelized.

Graphical user interface, text, application, email

Description automatically generated

<https://leonardoaraujosantos.gitbook.io/opencl/performance>

 Speedup is limited by the fraction of the work that is not parallelizable, even using an infinite number of processors the speed will not improve, because the serial part will limit.

\*Amdahl's law does not take into account other factors like memory latency.

**Explain the sorting techniques. Serial and parallel sorting are the same?**

A Sorting technique is used to rearrange a given array or list elements according to a comparison operator on the elements. The comparison operator is used to decide the new order of element in the respective data structure.

**For example**: The below list of characters is sorted in increasing order of their ASCII values. That is, the character with lesser ASCII value will be placed first than the character with higher ASCII value.

Input: hello 🡺 output: ehllo

**Serial sorting**

* Uses single thread for the operation
* Takes longer time to perform the operation
* Done sequentially

**Parallel sorting**

* Uses multiple threads for the operation
* Faster when there are a lot of elements whereas slower for lesser elements
* The overhead for parallelization becomes tolerably small on large ones

**Special type of sorting for parallel programming are many. How the merge sort achieve parallelism?**

Sorting is a process of arranging elements in a group in a particular order, i.e., ascending order, descending order, alphabetic order, etc. Sorting a list of elements is a very common operation. A sequential sorting algorithm may not be efficient enough when we have to sort a huge volume of data. Therefore, parallel algorithms are used in sorting.

**Merge Sort**

Merge sort first divides the unsorted list into smallest possible sub-lists, compares it with the adjacent list, and merges it in a sorted order. It implements parallelism very nicely by following the divide and conquer algorithm.

Diagram

Description automatically generated

**OpenCL and OpenGL are the same?  describe the OpenCL specifications in parts.**

OpenCL is an open standard however–meaning anyone can use its functionality in their hardware or software without paying for any proprietary technology or licenses. OpenCL will pass off the information entirely, using the graphics card more as a separate general purpose peer processor. It’s a minor philosophical distinction, but there’s a quantifiable difference in the end. For the programmer, it’s a little bit harder to code for. As a user, you are not tied to any one vendor, and support is so widespread that most programs don’t even mention its adoption.

OpenGL is about drawing pixels or vertices on the screen. It’s the system that allows your graphics card to create 2D and 3D displays for your computer much faster than your CPU could. Like CUDA and OpenCL are alternatives to one another, OpenGL is an alternative to systems like DirectX on Windows. Simply, OpenGL draws everything on your screen really fast, OpenCL and CUDA process the calculations necessary when your videos interact with your effects and other media. OpenGL may place your video within the editing interface and make it play.

OpenCL specification:

Platform model: Specifies that there is one processor coordinating execution (the *host*) and one or more processors capable of executing OpenCL C code (the *devices*). It defines an abstract hardware model that is used by programmers when writing OpenCL C functions (called *kernels*) that execute on the devices.

Execution model: Defines how the OpenCL environment is configured on the host and how kernels are executed on the device. This includes setting up an OpenCL context on the host, providing mechanisms for host–device interaction, and defining a concurrency model used for [kernel execution](https://www.sciencedirect.com/topics/computer-science/kernel-execution) on devices.

Memory model: Defines the abstract memory hierarchy that kernels use, regardless of the actual underlying memory architecture. The memory model closely resembles current GPU memory hierarchies, although this has not limited adoptability by other accelerators.

Programming model: Defines how the concurrency model is mapped to physical hardware.

**Heterogeneous Computing. Process Synchronization applied in GPU? or CPU Justify**

**Diagram

Description automatically generated**

**Heterogeneous computing** refers to systems that use more than one kind of processor or [cores](https://en.wikipedia.org/wiki/Multi-core_processor). These systems gain performance or [energy efficiency](https://en.wikipedia.org/wiki/Electrical_efficiency) not just by adding the same type of processors, but by adding dissimilar [coprocessors](https://en.wikipedia.org/wiki/Coprocessors), usually incorporating specialized processing capabilities to handle particular tasks.

CPU. CPU is able to support synchronization and communication and cannot benefit from GPU implementation.

**Sketch the matrix graph.**

A picture containing text, clock

Description automatically generated

https://towardsdatascience.com/feature-extraction-for-graphs-625f4c5fb8cd

**Distinguish between exclusive and inclusive scan.**

**Text

Description automatically generated**

**Exclusive Scan - The output (sum scan), all elements before, not including current element.**

The exclusive scan operation takes a binary associative operator ⊕ with identity I, and an array of n elements.

An exclusive scan can be generated from an inclusive scan by shifting the resulting array right by one element and inserting the identity. Likewise, an inclusive scan can be generated from an exclusive scan by shifting the resulting array left, and inserting at the end the sum of the last element of the scan and the last element of the input array [1]. For the remainder of this paper we will focus on the implementation of exclusive scan and refer to it simply as scan unless otherwise specified.

**Inclusive Scan – The output (sum scan), all elements before, including current element.**

In other words, the summation operations may be performed in arbitrary order. The behaviour is nondeterministic if binary\_op is not associative.

Inclusive scan often useful for each element j in the results of a scan to contain the sum of all previous elements, but not j itself. This operation is commonly known as an exclusive scan (or pre scan)

**Explain the work efficient implementation of the parallel scan pattern. Write a simple OpenCL code.**

work-efficient scan algorithm avoids the extra factor of log n work performed. The scan algorithm performs O(n) operations (it performs 2\*(n-1) adds and n-1 swaps); therefore it is work efficient and for large arrays, should perform much better. Algorithmic efficiency is not enough; we must also use the hardware efficiently. If we examine the operation of the scan on a GPU running CUDA, we will find that it suffers from many shared memory bank conflicts. These hurt the performance of every access to shared memory, and significantly affect overall performance.

A picture containing text

Description automatically generated

https://www.eecs.umich.edu/courses/eecs570/hw/parprefix.pdf

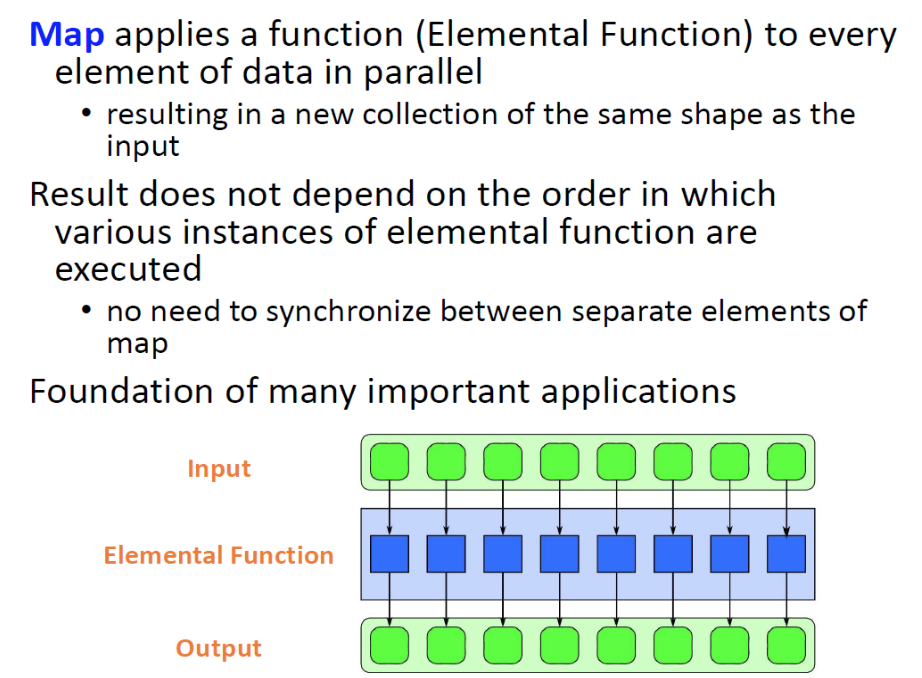
**Briefly explain the technique used in GPUs to avoid pipeline stalls due to long latency accesses to memory. Justify with your answer with suitable diagrams.**

**Diagram

Description automatically generated**

GPU new pipeline uses a unified hardware that combines all shaders under one core and shared memory. In addition, nVidia introduced CUDA (Compute Unified Device Architecture) which added the ability to write general-purpose C code with some restrictions. This meant that a programmer has access to thousands of cores which can be instructed to carry out similar operations in parallel.

**Map, Stencil patterns**

****

**Map**

**-applies a function (elemental function) to every element of data in parallel**

**-resulting in a new collection of the same shape as the input**

**-result does not depend on the order in which various instances of elemental function are executed**

**-no need to synchronize between separate elements of map**

**Sequence of maps**

**-often several map operations of same shape occur in sequence**

**-vector operations are maps using very simple operations multiplication**

**-a naïve implementation may write each intermediate result to memory**

**-convert a sequence of maps into a map of sequences**

**-fuse together the operations to perform them at once inside a single map**

**-adds arithmetic intensity, reduces memory requirements and bandwidth**

**Text, application

Description automatically generated**

**Stencil**

**-A stencil pattern is a map where each output depends on a “neighbourhood” of inputs**

**-These inputs are a set of fixed offsets relative to the output position**

**-A stencil output is a function of a “neighbourhood” of elements in an input collection**

**❍ Applies the stencil to select the inputs**

**-Data access patterns of stencils are regular**

**❍ Stencil is the “shape” of “neighbourhood”**

**❍ Stencil remains the same**

**Example**

**Text, letter

Description automatically generatedGraphical user interface, text, application

Description automatically generated**

(http://ipcc.cs.uoregon.edu/lectures/lecture-8-stencil.pdf)

* **Stencils can operate on one dimensional and multidimensional data**
* **Stencil neighbourhoods can range from compact to sparse, square to cube, and anything else!**
* **It is the pattern of the stencil that determines how the stencil operates in an application**

**Atomic functions.**

When you have a lot of threads reading and writing same memory location, the solution is by using atomic memory operations.

Functions:

-atomic Add()

-atomic Min()

-atomic XOR()

-atomic CAS()

Example:

Graphical user interface, text

Description automatically generated with medium confidence