Lecture 8

Parallelization Strategies

Institute of Computer Science & Information Technology,
Faculty of Management & Computer Sciences,
The University of Agriculture, Peshawar, Pakistan.

Parallelization Strategies: Challenges

- Selecting the best parallelization strategy: using proper parallelization strategy is crucial in order to achieve the optimal performance, such that the calculation takes the shortest possible time.

- It is very difficult to give a universal method for finding the optimal strategy, since it depends not only on the platform, but also on the application itself.

Parallelization Strategies

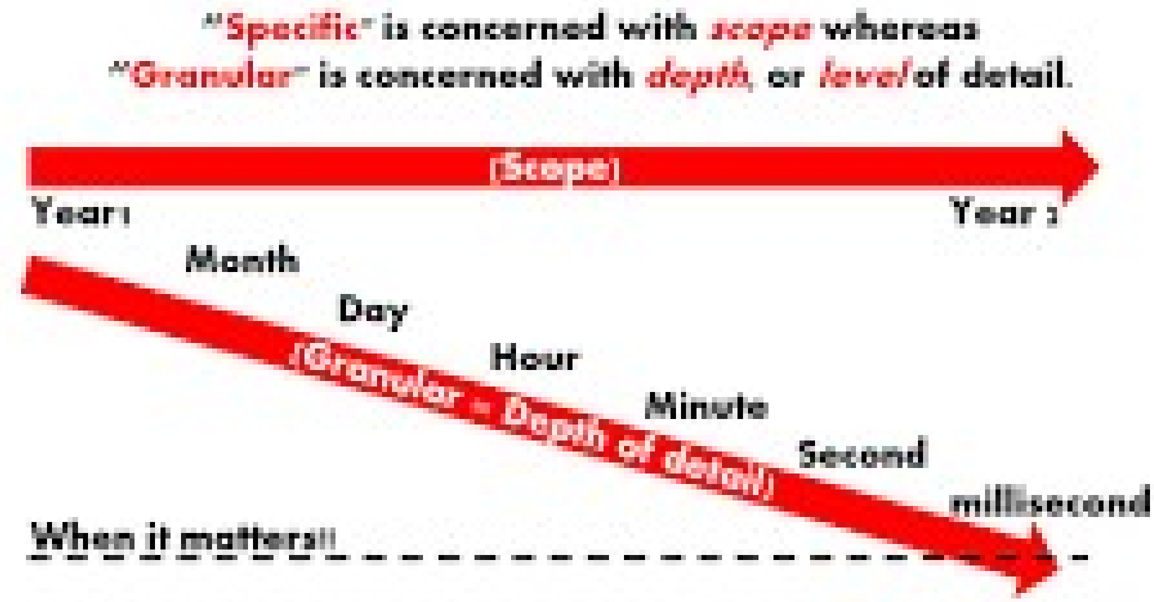
- Parallelization Strategies define how to implement parallelization opportunities.

- How much levels?

- How much granular?

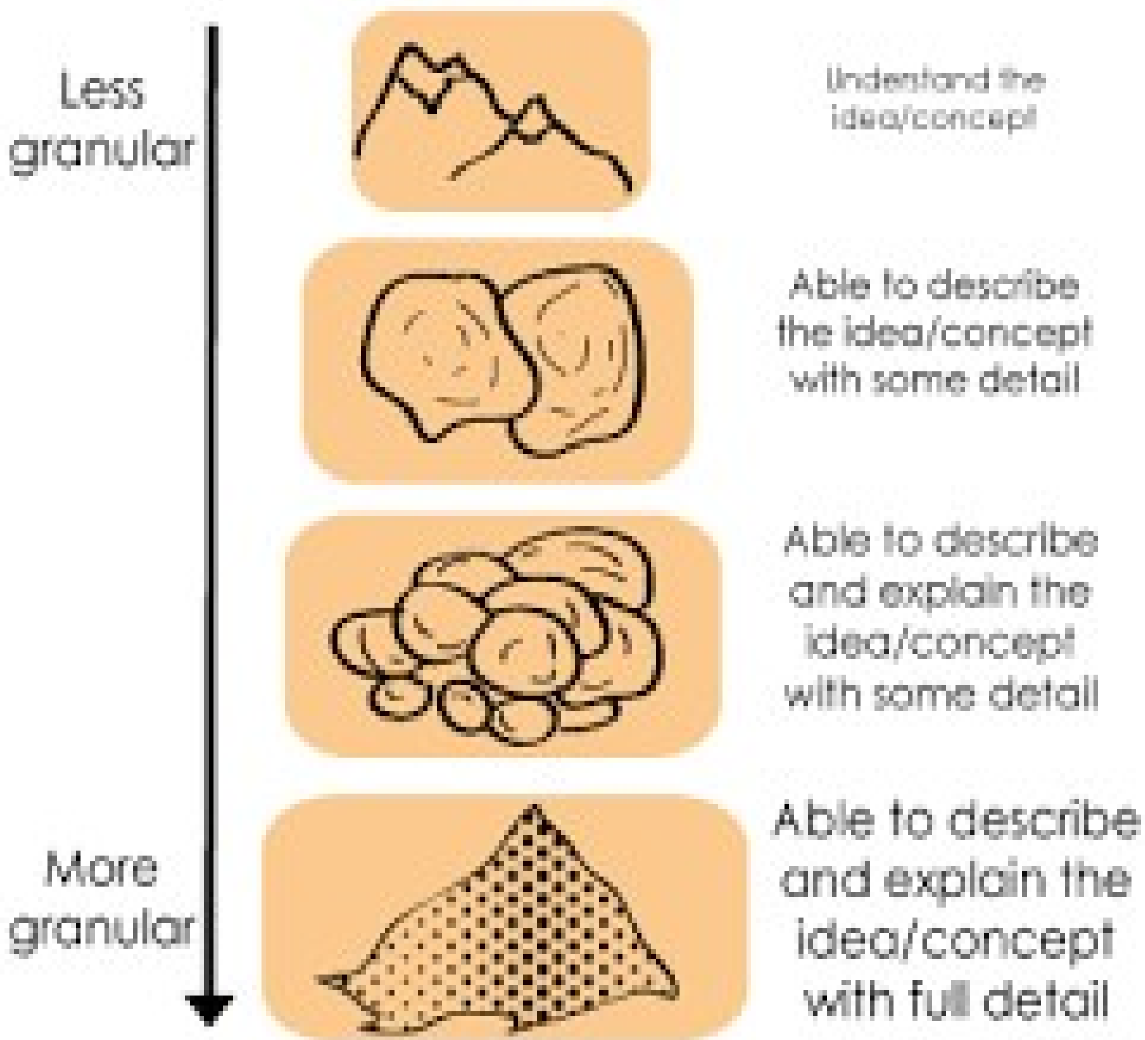
Granularity

- Parallelization granularity refers to the size of parallel entities at which we can divide an application.
- Granularity is concerned with depth or level of details.



Coarse Grain

Fine Grain



Granularity

- For both levels, we may define mainly two granularities:

1. Fine Grain
2. Coarse Grain

- Coarse grained materials or systems have fewer, larger components than fine-grained systems.

- The concepts granularity, coarseness, and fineness are relative; are used when we are decomposing some system.

Coarse Vs. Fine Grain

- Fine-grained parallelism means individual task are relatively small in terms of code size and execution time.
- The data is transferred among processors frequently in amounts of one or a few memory words.
- Coarse-grained is the opposite: data is communicated infrequently, after larger amounts of computation.

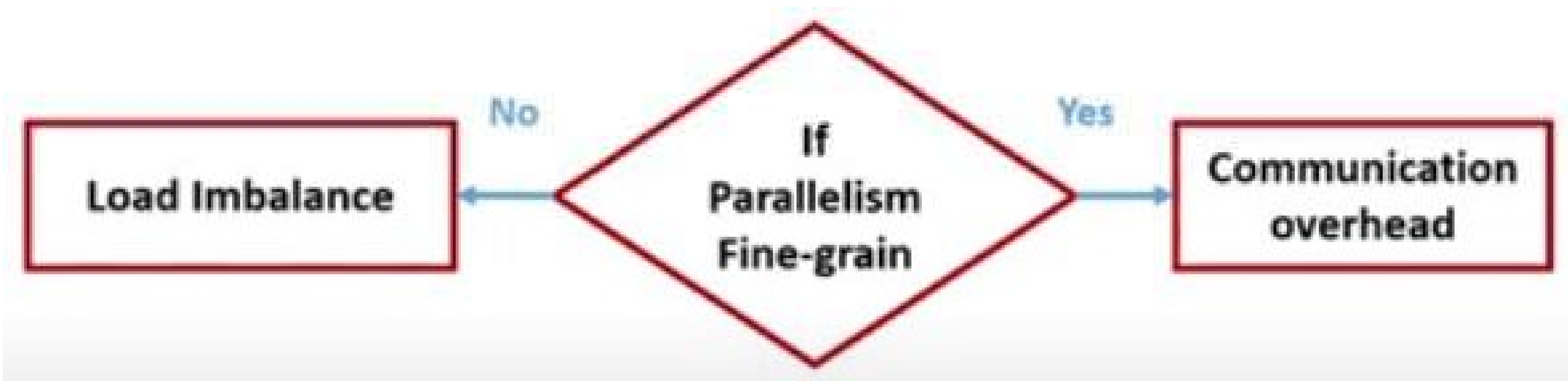
Coarse Vs. Fine Grain (I)

- The finer the granularity, the greater the potential for parallelism and hence speed-up, but the greater the overheads of synchronization and communication.

- On the other side, if the granularity is too coarse, the performance can suffer from load imbalance.

- In order to attain the best parallel performance, the best balance between load and communication overhead needs to be found.

Coarse Vs. Fine Grain (2)



Coarse Vs. Fine Grain (2)

1. Task-level parallelization
2. Data-level parallelization

- For both levels we may define mainly two granularities:

1. Fine grain
2. Coarse grain

Task-level Granularity

- Task-level granularity is directly related to the program decomposition into <u>independent tasks</u>.

- It has two types:

1. Fine grain tasking
2. Coarse grain tasking

Fine Grain Tasking

- Fine-grain tasking consists of dividing the program in fundamental separate tasks and each single task is parallelized apart (separately).

- This process is called fission.

- The benefit of the fine grain parallelization strategy is the high reusability, since each task may be found in more than one algorithm which is the case of most image processing algorithms.

Fine Grain Tasking(1)

- This means that a parallel version of this operation can be reused more than once in different applications without any modification to ensure high portability among different applications.

- However, this strategy may suffer from:

1. Overhead introduced by successive threads launches.
2. Poor temporal data locality.

Coarse Grain Tasking

- Coarse-grain tasking consists in packing a sequence of tasks into a macro task.

- This process is called fusion.

- Each macro task is assigned to a single thread which processes a part of data array.

Coarse Grain Tasking (1)

- The implementation of this parallelization strategy needs additional programming effort to manage dependencies between neighbors data in order to minimize the synchronization barriers and data communication.

- When the implementation of the coarse-grain strategy is optimized, runtime performances may be improved by increasing temporal data locality and by avoiding overhead cause by threads launches and data transfers.

Coarse Grain Tasking (2)

- Temporal locality refers to the reuse of specific data and/or resources within a relatively small time duration.

Data-Level Granularity

- Data level granularity defines the degree of decomposition of initial data into data subsets.

- It has two types:

1. Fine-grain Tiling
2. Coarse-grain Tiling

Fine-grain Tiling

- Fine-grain Tiling consists in decomposing the data into small subsets (small tiles in the case of image processing).

- These small tiles are assigned to small groups of threads.

- This strategy exposes a high-degree of concurrency and takes advantage of architectures supporting a huge number of threads.

Fine-grain Tiling (1)

- In addition, this strategy is usually not constrained by the hardware resources limitations.
- However, it suffers from a significant overhead in processing replicated data at borders to handle boundary dependencies.
- Boundaries are created as per purposes, each boundary must create objects related to a single area of concern, like a Layer, a Feature, etc.

Coarse-grain Tiling

- Coarse-grain Tiling consists in decomposing data into large data subsets and involving large groups of threads.
- This strategy reduces the data replication at borders and offers high space data locality.
- However, large tiles may not fit well with available resources, cache or local memory size, which may degrade performance.

Types of Parallelization

- Types of parallelism or also known as parallelism models, define the way to organize independent workflows.

- We can distinguish three main types of parallelism:

1. Data parallelism
2. Task parallelism
3. Pipeline parallelism

Data Parallelism

- Data parallelism refers to work units executing the same operations on a set of data.

- The data is typically organized into a common structure such as arrays.

- Each work unit performs the same operations as other work units but on different elements of the data structure.

Data Parallelism (1)

- The first concern of this parallelism type is how to distribute data on work units while keeping them independent.

- Data parallelism is generally easier to exploit due to the simpler computational model involved.

Task Parallelism

- Task parallelism is known as functional parallelism or control parallelism.

- This type of parallelism considers the case when work units are executing on different control flow paths.

- Work units may execute different operations on either the same data or different data.

Task Parallelism (1)

- In task parallelism, work units can be known at the beginning of execution or can be generated at runtime.

- Task parallelism could be expressed in the GPU context in two ways. In a multi-GPU environment, each task could be processed by a separate GPU.

Task Parallelism (2)

- In a single GPU environment, the task parallelism is expressed as independent kernel queues called respectively streams and Command queues in CUDA (Compute Unified Device Architecture) and OpenCL.
- Kernel queues are executed concurrently where each kernel queue performs its workload in a data-parallel fashion.

Pipeline Parallelism

- Pipeline parallelism is also known as temporal parallelism.
- This type of parallelism is applied to chains of producers are consumers that are directly connected.
- Each task is divided in a number of successive phases.
- The result of each work unit is delivered to the next for processing.

Pipeline Parallelism (I)

- At the same time, the producer work unit starts to process a give phase of a new task.
- Compared to data parallelism, this approach offers reduced latency, reduced buffering and good locality.
- However, this form of pipelining introduces extra synchronization, as producers and consumers must stay tightly coupled in their execution.