**Parallel & Distributed Programming**

Table of Contents

[Coarse Grain parallelism 2](#_Toc130545964)

[Advantages of coarse grain parallelism. 2](#_Toc130545965)

[Disadvantages of coarse grain parallelism. 2](#_Toc130545966)

[MPI used in clusters: 2](#_Toc130545967)

[Fine Grain parallelism 3](#_Toc130545968)

[Advantages of fine-grained parallelism. 3](#_Toc130545969)

[Using TBB intel library for fine grain parallelism: 4](#_Toc130545970)

[The limits of parallel programming. 4](#_Toc130545971)

[Lower Limit of Parallelism. 4](#_Toc130545972)

[Example of lower parallelism: 5](#_Toc130545973)

[Upper limit of parallelism. 5](#_Toc130545974)

[Distribution. 6](#_Toc130545975)

[Parallel Distribution with Ocean Mappers. 6](#_Toc130545976)

[Diagram of Flowchart. 7](#_Toc130545977)

[Screenshot: 8](#_Toc130545978)

[Parallel 8](#_Toc130545979)

[Part A: 8](#_Toc130545980)

[Part B: 8](#_Toc130545981)

[PartC: 8](#_Toc130545982)

[Distributed: 9](#_Toc130545983)

[Part A: 9](#_Toc130545984)

[Part B: 9](#_Toc130545985)

[Part C: 9](#_Toc130545986)

Momodou Ceesay

Coventry University

# **Coarse Grain parallelism**

Coarse grain parallelism is doing parallel task in manual order. This is because it requires human interaction. Coarse grain systems require fewer components when compared to fine grain because fine grain is mostly used for large amounts of work that require parallelism. Coarse grain is more common because it’s cheaper to implement the downside of this however is that it requires more clock cycles since it uses less processors. Coarse grain is commonly used for most programming task that require concurrency. With coarse grain there is less communication between the tasks it’s been delegated to do: meaning more runtime.

## **Advantages of coarse grain parallelism.**

Requires less hardware cost.

Can be easily implemented.

Needs little to no adaptation.

Programs are broken into small tasks.

## **Disadvantages of coarse grain parallelism.**

Does recognise parallelism straight away.

Doesn’t have proper load balancing facilities.

Parallelism can’t be distributed automatically.

Two libraries that implement fine coarse grain parallelism are MPI and Apache Arrow.MPI uses parallelism to distribute tasks in a cluster. An example of the how MPI code works will be exemplified below:

## **MPI used in clusters:**

|  |
| --- |
| #include <iostream>  #include <omp.h>  **int** **main**() {  **int** n = **100**;  **double** sum = **0**;  #pragma omp parallel **for** reduction(+:sum)  **for** (**int** i = **0**; i < n; i++) {  sum += i;  }  std::cout << "Sum: " << sum << std::endl;  **return** **0**;  } |

This code parallelises the code between into multiple tasks by adding 0 to the sum until it reaches 100. It uses multiple cores that process the task differently. The pragma openmp directive is used to parallelise the loop that iterates from 0 to 99. The reduction clause specifies that the variable ‘sum’ should be summed across all threads that are executing the loop in parallel.

## **Another one is the Apache arrow:**

Apache Arrow provides a standard, efficient, and flexible framework for data interchange and processing that can improve the interoperability and performance of data processing systems. It can be used with C++.

# **Fine Grain parallelism**

Fine Grain parallelism is a form of parallelism that’s low-level. Fine Grain parallelism allows programming to be done concurrently using parallelism. Fine grain parallelism has more communication overhead because it uses more clock cycles. Fine Grain implemented systems can do auto-scaling because they have dynamic scheduling meaning they can adjust depending on demand. Fine Grain implementation is much more expensive because more processors and nodes are needed for implementation. Fine Grain parallelism is also the new norm for large scale cloud computing. One of working implementation of fine grain parallelism is load balancing in AWS. The Diagram below shows how it works:

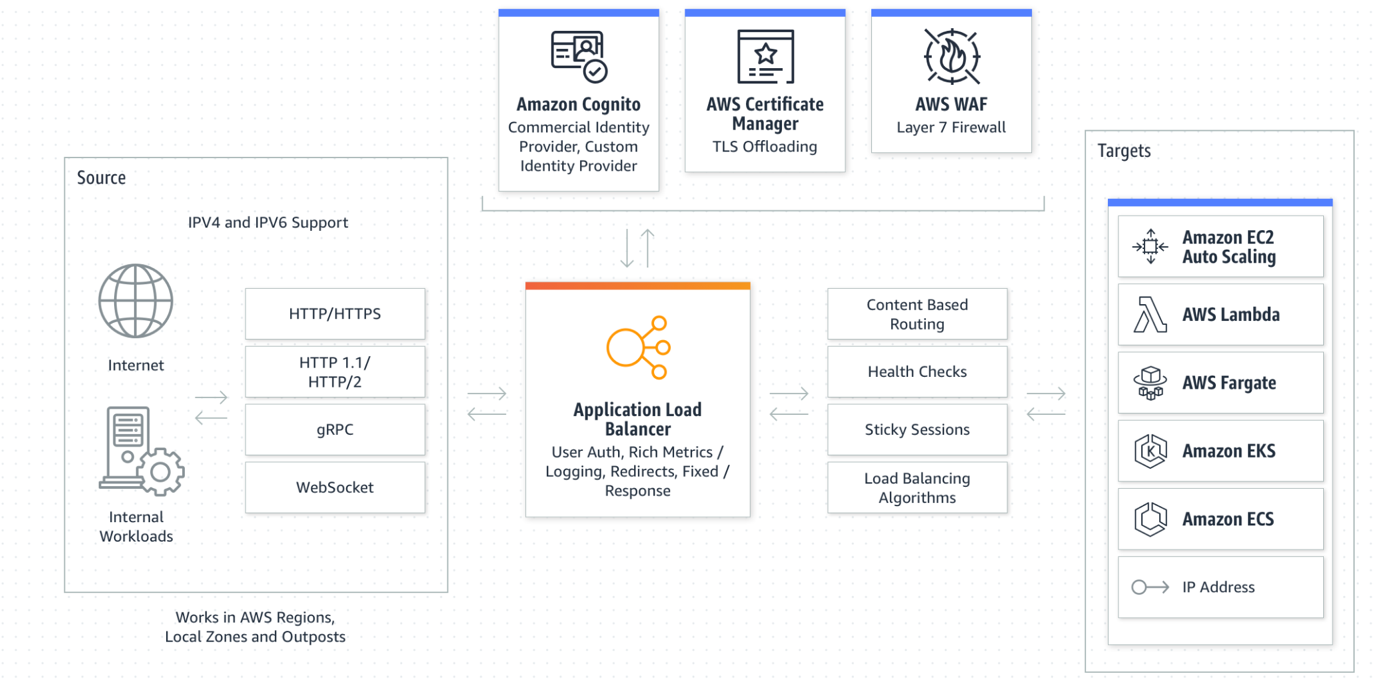


Figure 1: Application Load Balancer, AWS, 2023, https://aws.amazon.com/elasticloadbalancing/,AWS

AWS is a system as a service accessed through the internet. The application load balancer very the certificates and decide which certificate to use at a certain time. The next thing is does is determine which route to use at a certain time. Then it can select multiple targets at will. This process requires many nodes, it also shows that powerful systems are being used.

## **Advantages of fine-grained parallelism.**

Problem is broken into large tasks.

Grain size is over 1000 instructions.

Load balancing is proper.

Parallelism can be detected using a compiler.

## 

## **Using TBB intel library for fine grain parallelism:**

Example code of intel TBB being used:

|  |  |  |
| --- | --- | --- |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | #include <iostream>  #include <vector>  #include "tbb/parallel\_for.h"  **using** **namespace** std;  **using** **namespace** tbb;  **int** **main**() {  // Create a vector of integers to be processed in parallel  vector<**int**> data = {**1**, **2**, **3**, **4**, **5**, **6**, **7**, **8**, **9**, **10**};  // Use TBB's parallel\_for to process the data in parallel  parallel\_for(blocked\_range<**size\_t**>(**0**, data.size()), [&](**const** blocked\_range<**size\_t**>& r) {  **for** (**size\_t** i = r.begin(); i < r.end(); i++) {  // Process each element of the vector independently  data[i] = data[i] \* **2**;  }  });  // Print the results  **for** (**int** i = **0**; i < data.size(); i++) {  cout << data[i] << " ";  }  cout << endl;  **return** **0**;  } | |

## **The limits of parallel programming.**

The potential speedup is defined by the fraction of code (p) that can be parallelised (Amdahl’s law). This means that the number of processors required should be more than the number of tasks to be executed. It works using three things Fraction of the application that is serial. Fraction of the application that is parallelizable and the number of processors. We have two types of limits upper limits and lower limits. This will be discussed in the paragraph below.

## **Lower Limit of Parallelism.**

This totally depends on the amount of work that can be parallelised. When the hardware starts with sequential processing and the amount of work increases it uses threads to help parallelize the work into multiple smaller tasks. This makes use of the total use of the cores that aren’t being used. However, parallelism is effective for repetitive tasks that make use of resources.

## **Example of lower parallelism:**

## **Upper limit of parallelism.**

Upper limit focuses on how many cores can be used by computer to perform parallel computing. (**Amdahl's law**) This law focuses on the fact that not everything can be parallelised. It also states that single component affects the number of threads that can be parallelised. It also limits the amount of parallelisation the application can use. An example of Amdahl’s law can be:

Predicting the weather forecast using the new computer with the same specs as the previous one. This means that the process can be done but because the new one has more spec gains it will perform better than the old one. This is because the new one will be able to use some of its processing power to do the simulation instead of a separate core.

This is a figure of Amdahl law: Graphical user interface, text, letter

Description automatically generated

Chart, line chart

Description automatically generated

Figure 2: Amdahl Law, 2018,<http://rits.github-pages.ucl.ac.uk/intro-hpchtc/morea/lesson2/reading3.html>,ucl

From the diagram above we can see that there are diminishing returns due to Amdahl’s law. As we can see the purple line shows the amount of work that has been done without the use of parallelism. This makes proper use of the processing power of the computer. It shows that the application makes use of a single node. The diagram shows how the level of parallelism is influenced by the number of cores available. More availability higher usage. In summary, the Amdahl law focuses more on using hardware efficiently since most parallel task can be done using less cores. This improves efficiency.

# **Distribution.**

Distributed programming is split into two main parts Clients and servers. It allows for computers to share resources and communicate with each other. The client server architecture is used in mostly distributed programming. The clients are more like the people that use the application. The server is more like the host that has all the main data. The server handles all the backend applications needed to make the program work. An example of distributed programming being used is ocean mapping.

## **Parallel Distribution with Ocean Mappers.**

The ocean mappers are increasingly expected to have a basic understanding of the computational tools that they are using to collect and process data daily (Masetti et al., 2020). This is because the task is divided into many parts.one part of the process could be sensors checking how deep the sea is. Another set is the data collected from the satellite maps. The estimated cost of mapping the entire ocean is 3 billion dollars. This cost can be saved by using parallel programming. With parallel programming the computational tools can communicate with each other and save resources. The sonar works by taking sound and the measurement of time. The sonar sends a beam down the ocean and the reaction of the beam is then sent back to the sonar. The time is takes for the sound to be sent back can be done separately to make it more accurate.

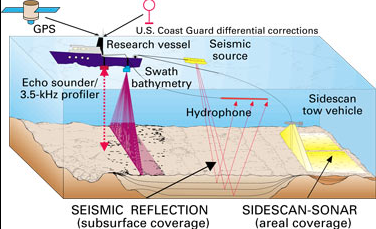
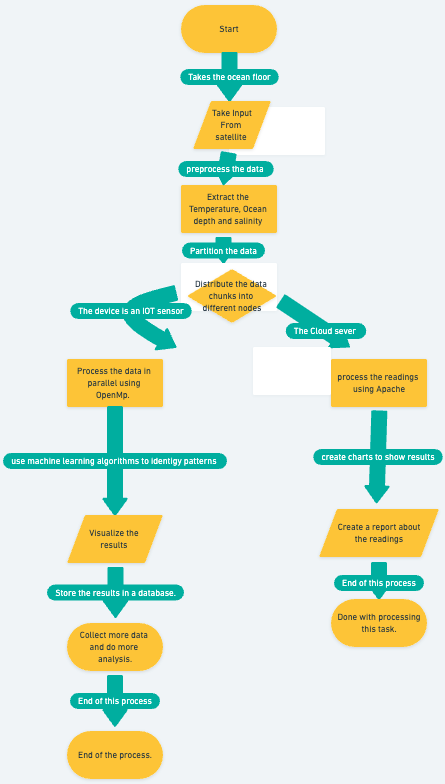


Figure 3: Ocean Mapping, 1992, <https://serc.carleton.edu/eet/seafloor/case_study.html> , William B. F. Ryan

When there’s mapping done for a certain area. The satellite mapper can be configured to store only the data that can be processed the excess amount of data can be used for another time. The thickness of the soil can also be measured using the ground penetrating radar. There will be a GPS satellite which is connected to the main server. The main server has many computers connected to each other to work together. These computers work together with the use of the MPI (Message parsing interface) and connected nodes including the topology it uses. From research this will be a star topology. The data collected by the Echo sound will be recorded by a separate computer. This will save time for the other computations that should use this process. Whilst the results are stored, they will be passed on to another computer using procedural programming. This will avoid other process to continue when there’s failure. The seismic reflection allows for the sound waves to bounce back. For sound transmission the hydrophone is used. There will be a computer present on board to record the electrical signals. These signals will be sent to the main computer for calculations. The ability to program can provide the ocean mapper with a more efficient mechanism to use available tools in new ways, allowing control for every single step of data analysis (Masetti et al., 2019). A more intuitive way will be shown in a flowchart.

## **Diagram of Flowchart.**

Figure 3.



## **The cost of distributed systems in ocean mapping.**

The cost will be higher but its more effective than normal methods. The cost of computation will be very high. This is because the rest of the sea map is simulated. This is because the soundwaves are only collected sometimes and not all the time photos have been taken. The other aspect is the sea drones that are sent to take video and images They are normally modelled. These models give output. Not all models are 100% accurate.

# **Screenshot:**

# **Parallel**

## **Part A:**

Text

Description automatically generated

## **Part B:**

Text

Description automatically generated

## 

## **PartC:**

Text

Description automatically generated

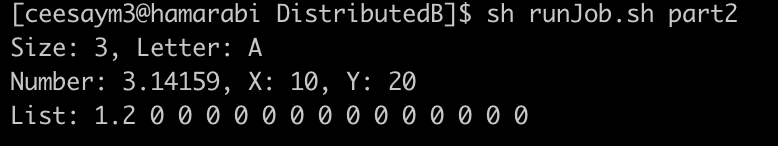
# **Distributed:**

## **Part A:**

Text

Description automatically generated with medium confidence

## **Part B:**



## **Part C:**

# **References:**

No Author. (2017, May 31). *Underwater drones use sound to send snaps of the ocean floor*. New Scientist. https://www.newscientist.com/article/mg23431283-400-underwater-drones-use-sound-to-send-snaps-of-the-ocean-floor/

This investigates ways in which underwater drones work and the cost that’s involved.

*Coarse-grained vs. Fine-grained APIs Explained | Lightboard Series*. (2020, August 21). Www.youtube.com. https://youtu.be/5umsne8512E

Used this to know the use of course-grained parallelism and fine grain parallelism.

*Difference between Fine-Grained and Coarse-Grained SIMD Architecture*. (2020, January 6). GeeksforGeeks. https://www.geeksforgeeks.org/difference-between-fine-grained-and-coarse-grained-simd-architecture/

This is used for defining fine vs coarse grain.

GeekForGeeks. (2017, May 30). *Multidimensional Arrays in C / C++ - GeeksforGeeks*. GeeksforGeeks. https://www.geeksforgeeks.org/multidimensional-arrays-c-cpp/

used this for a matrix template for the part b of the parallel coursework.

IHO. (2022, November 30). *INTRODUCING PROGRAMMING TO OCEAN MAPPING STUDENTS*. IHR. https://ihr.iho.int/articles/introducing-programming-to-ocean-mapping-students/

an essay about ocean mapping. How it works, the programming involved and the resources it takes.