High Performance Optimizations and Dynamic Load Balancing for Computational Aerodynamics Solvers

Blessy Boby TVE15CS020
Gmon K TVE15CS024
Shilpa S TVE15CS055
Robin R LTVE15CS068

Under the guidance of
Prof. Dr. Ajeesh Ramanujan
College of Engineering, Trivandrum
Harichand M. V
Scientist Engineer, Vikram Sarabhai Space Centre

TABLE OF CONTENTS

Introduction

Introduction

Load Balancing

1. It is a challenging and interesting problem that appear in all High Performance Computing(HPC) environments.
2. Most of the applications running in HPC environments use MPI based parallel computing.
3. This approach needs load assignment, before starting the computations, which is difficult in many problems where computational load changes dynamically.

In **VSSC Aeronautics entity**,

1. Parallel computing is used to solve Computational Fluid Dynamics problems.
2. Memory bandwidth utilisation of CFD problems are very low, making them computationally inefficient.

Motivation

The motivation for doing this project was primarily an interest in undertaking a project at VSSC, ISRO. The opportunity to learn about a new area of computing not covered in lectures was appealing.

Problem Statement

Problem Statement

Develop a distributed MPI framework for solving computational problems efficiently. Challenge here is to design data layouts which can effectively utilise memory bandwidth. Multiple re-ordering schemes, along with data duplication can be attempted to improve the memory performance.

While solving computational problems in a distributed MPI environment, it is important to ensure load balancing. But the static load balancing algorithms can fail if the computational loads vary over time. A distributed work queue may be attempted to solve this problem

Objectives

Objectives

Two problems can be attempted as part of this Project

- Improve the dynamic load balancing capability in an MPI environment for solving CFD problems
- Improve data locality of CFD problems

Scope of Project

Scope of project

Develop a frame work for solving Computational Fluid Dynamics Problems keeping in mind

1. Explore graph re-ordering, data layout modification for increased compute efficiency.
2. Dynamic load balancing.
3. Hybrid CPU-GPU computing.
4. Compare the results with default MPI method.

Requirements

Requirements

1. **METIS** software for partitioning graph into sub-graph.
2. **OPENMPI** for implementing MPI i.e, passing messages between sub-graphs.
3. Operating system used are both **ubuntu / Linux.**
4. MPI programs are written in c++ language.
5. Input mesh file is represented in **VTK** format.

Mesh File

- The SU2 mesh format carries an extension of .su2, and the files are in a readable ASCII format.
- As an unstructured code, SU2 requires information about both the node locations as well as their connectivity.
- The connectivity description provides information about the types of elements (triangle, rectangle, tetrahedron, hexahedral, etc.) that make up the volumes in the mesh and also which nodes make up each of those elements.

Input Format Of Mesh File

- The first line of the .su2 mesh declares the dimensionality of the problem (NDIME).
- The next part of the file describes the interior element connectivity, which begins with the NELEM(Number of Elements in the mesh) keyword
- Remaining part contains the information about Element in the format

```
Connectivity node1 node2 etc...
```

- Connectivity and number of nodes depends on the type of element

Element	Connectivity	Number of nodes
Triangle	5	3
Quadrilateral	9	4
Tetrahedral	10	4
Hexahedral	12	8
Prism	13	6
Pyramid	14	5

Metis

The **METIS** is a software package for partitioning large irregular graphs and meshes.Its source code can be downloaded directly from `http://www.cs.umn.edu/metis`. METIS provides a set of stand-alone command-line programs for computing partitionings as well as an application programming interface (API) that can be used to invoke its various algorithms from C/C++ or Fortran programs.

Partitioning Of Graph

- METIS can partition an unstructured graph into a user-specified number k of parts using either the multilevel recursive bisection or the multilevel k-way partitioning paradigms.
- METIS' stand-alone program for partitioning a graph is **gpmetis** and the functionality that it provides is achieved by the **METIS PartGraphRecursive** and **METIS PartGraphKway** API routines.

MPI

- The **message passing interface (MPI)** is a standardized means of exchanging messages between multiple computers running a parallel program across distributed memory.
- It can be implemented using Open MPI which can be download from it's official page: `://www.open-mpi.org/`

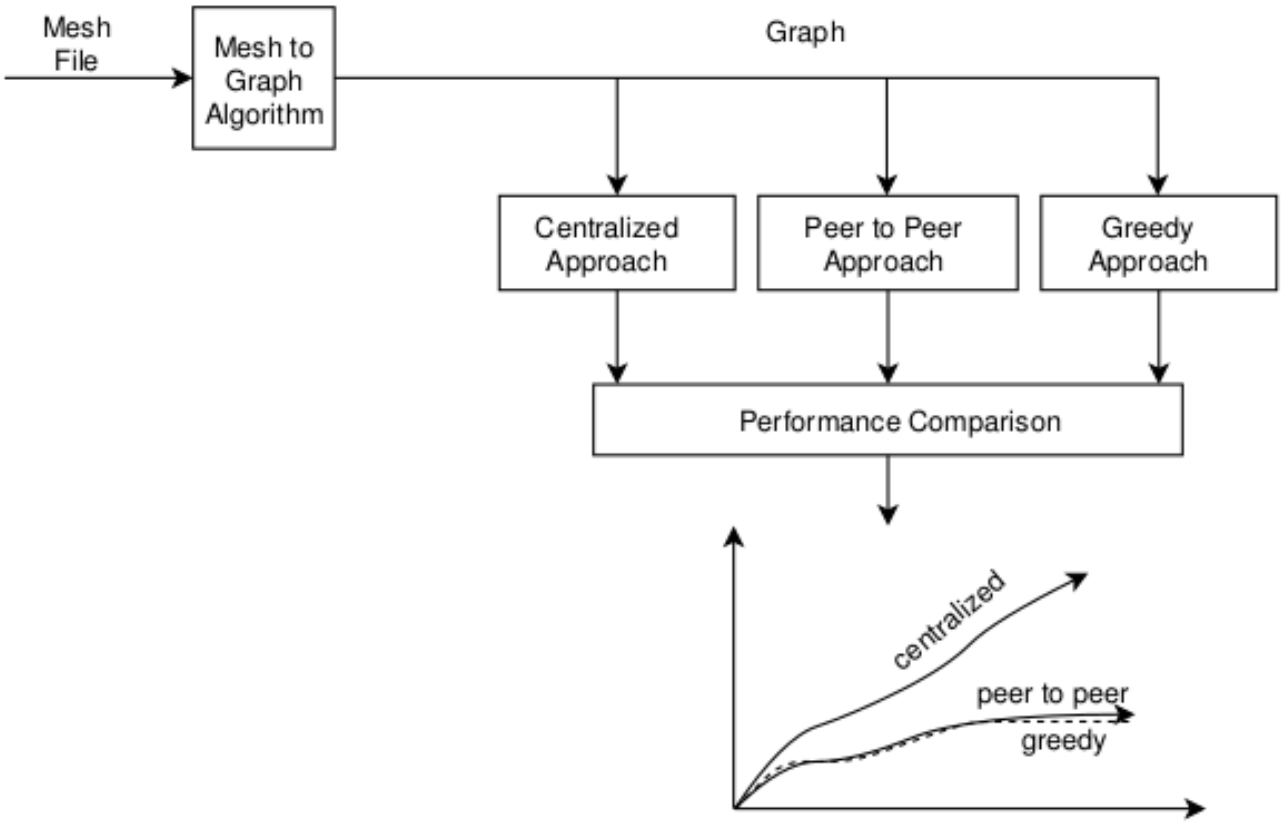
Why We Use MPI?

- After the mesh file is partitioned, each node is allotted into different partitions. These partitions are distributed into different systems.
- Some times the neighbouring node of a particular node in the mesh might not belong to the same partition. So we create ghost nodes which simulate the neighbouring nodes in the partition where the particular node belongs.

- Whenever we are in need of the data from the neighbouring node, we find out the original neighbour node's partition and retrieve the data from the node and then copy it to the ghost node in our partition.
- Here there is a need to communicate between the different partitions so that the values of ghost nodes can be updated as per the need. This is where MPI plays its role.

Proposed System

We are provided with mesh file represented in VTK format as initial input. In order to use MPI communication, we need to convert this mesh file into graph file which is then partitioned using METIS. And this partitioned graph file is further applied to centralized, peer-to-peer and greedy approach and its performance comparison is noted.


Mesh
File
Mesh to
Graph
Algorithm
Graph
Centralized
Approach
Peer to Peer
Approach
Greedy
Approach
Performance Comparison
centralized
peer to peer
greedy

Mesh file in VTK format

```
NDIME= 3
NELEM= 6
10  1  2  4  8
10  1  5  7  8
10  1  2  6  8
10  1  5  6  8
10  1  3  4  8
10  1  3  7  8
```

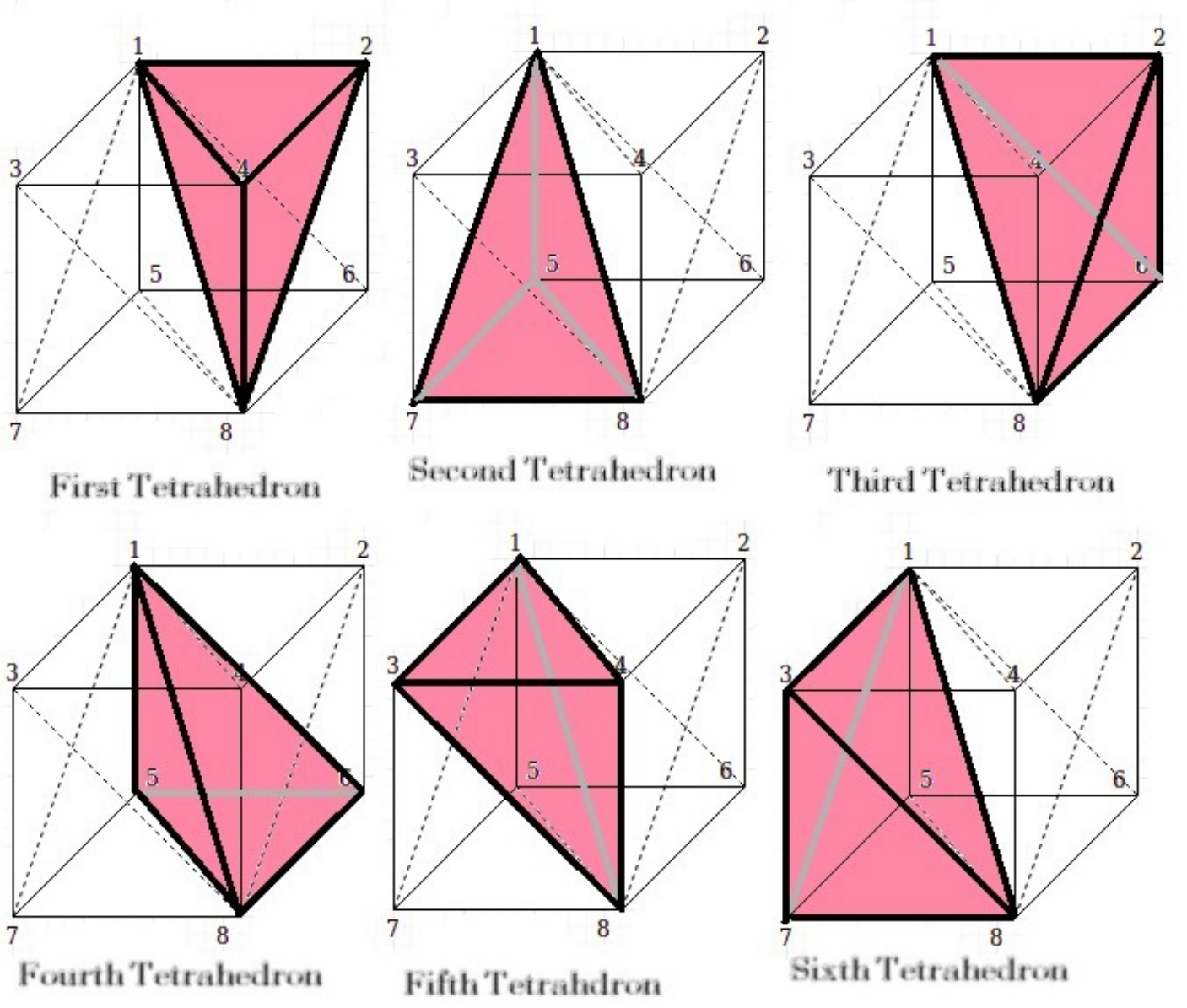
Algorithm for converting 3D mesh file into adjacency matrix

ALGORITHM 1

```
1. Read Mesh file in .su2 format
2. Check the dimension of the input
3. Store the number of vertices
4. For each element in the mesh file do the following
   4.1 Check the type of element
   4.2 Make an element to node mapping
   4.3 Make a node to element mapping
5. From the node to element mapping do the following.
   5.1 For each face for an element do the following
     5.1.1 Make an intersection among all the node to
           element mappings in the current face.
     5.1.2 If the cardinality of the resultant set is >= 2
           then create adjacency list
6. Write the adjacency list to a file.
```

Alternate Algorithm

```
1. Read Mesh file in .su2 format
2. Check the dimension of the input
3. Store the number of vertices
4. For each element in the mesh file do the following
   4.1 Check the type of element
   4.2 Make an element to node mapping
5. From the element to node mapping do the following.
   5.1 Find each face of an element
   5.2 For each face for an element do the following
      5.2.1 If this face is equal to another face of an
            element then make the face as edge and add this
            pair of elements to adjacency list.
      5.2.2 Store the face information.
6. Write the face information to file.
7. Write the adjacency list to a file.
```


1
2
3
4
5
6
7
8
First Tetrahedron
Second Tetrahedron
Third Tetrahedron
Fourth Tetrahedron
Fifth Tetrahdron
Sixth Tetrahedron

First tetrahedron and third tetrahedron are neighbours because they have common face 128. Likewise, first and fifth tetrahedron are neighbours

Output File(Graph File)

```
6 6
3 5
6 4
1 4
2 3
1 6
2 5
```

- The first line of the graph file describes number of vertices and number of edges.

- Remaining part contains the information about neighbouring vertex.
- i.e, in this example, file consists of 6 vertices and 5 edges.
- Vertex 1 has vertex 2 as neighbour, vertex 2 has vertex 1 and 4 as neighbours and so on.

METIS

Input to metis is the graph file which is the output of above algorithm. Output from metis is graph partition file.

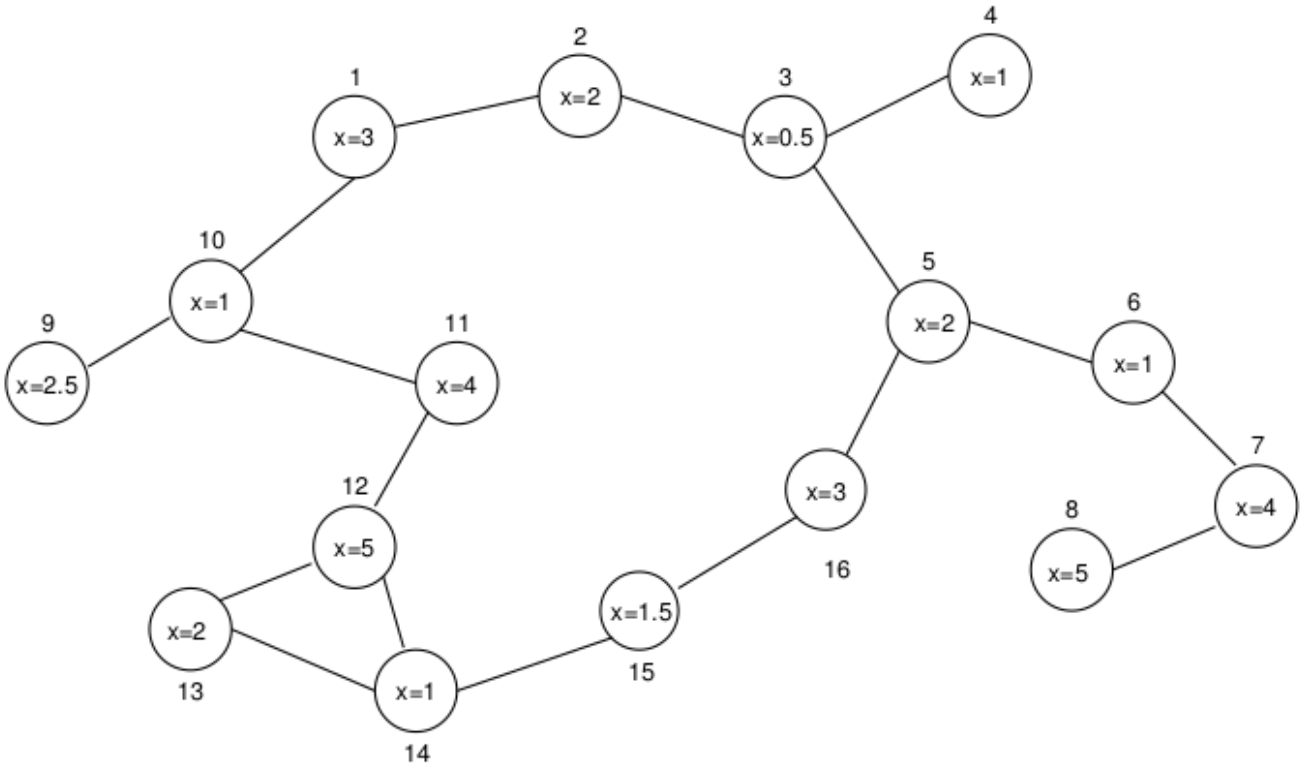
Graph partition file

```
1
1
2
2
0
0
```

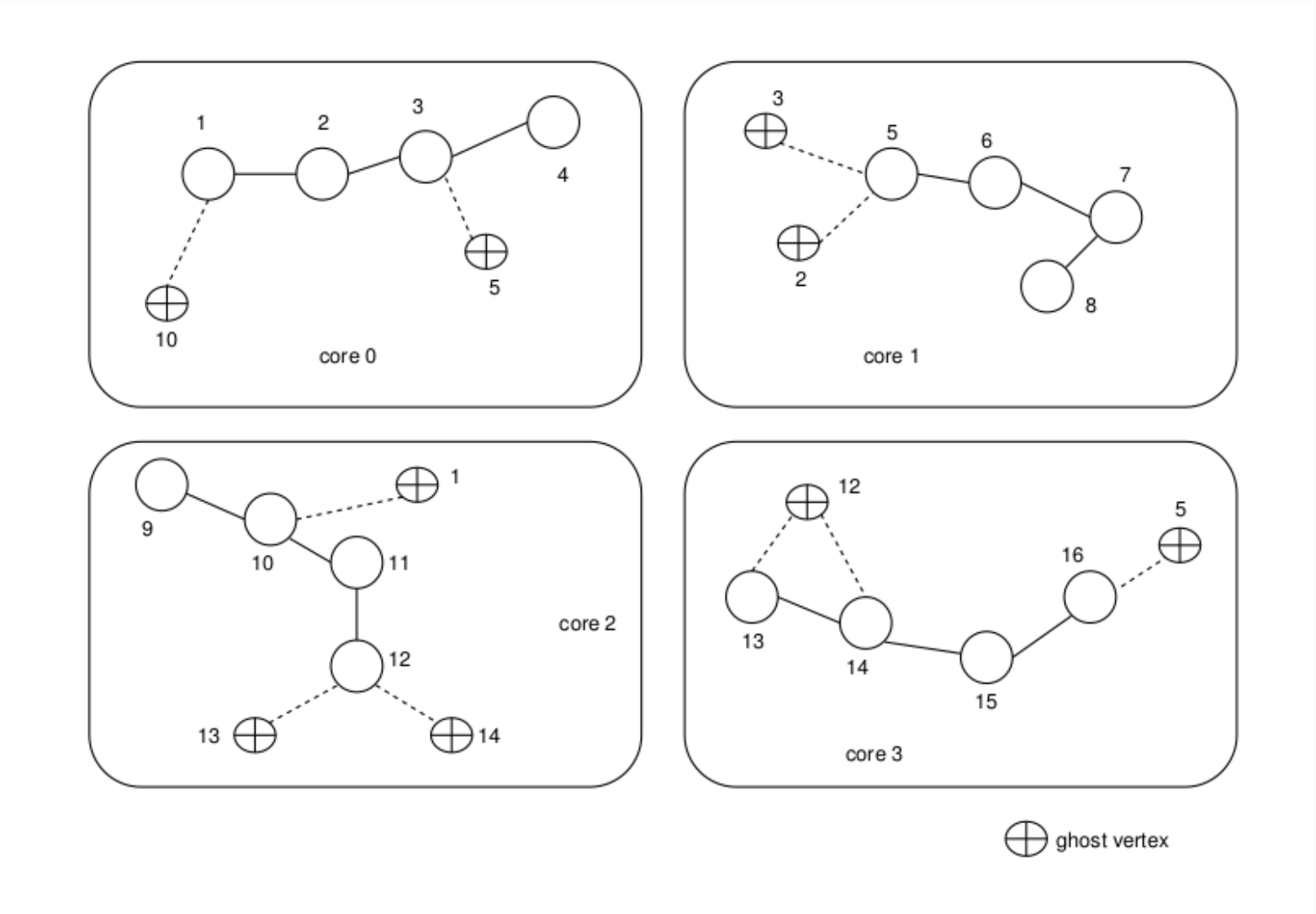
In graph partition file, each line indicates the core in which that vertex belong. In this example, first line indicates that first vertex belongs to core 1.

Communication

Sample Graph



Partitioned Graph



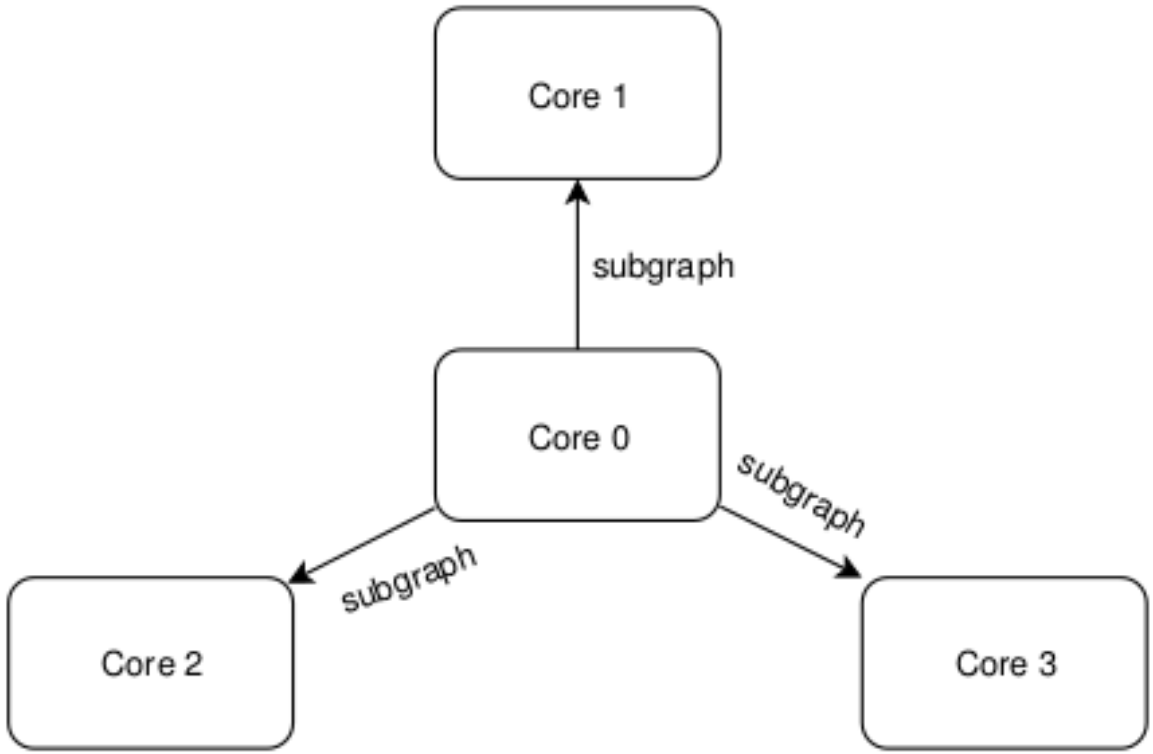
Centralized Approach

1. Taking into consideration the knowledge of the ghost vertices ,we were able to develop a program which is centralized based.
2. In this particular program zeroth core is assigned as the master and other cores as slaves.
3. Core 0 contains information about all the vertices of the graph.
4. Whenever any of the slave core is in need of any details about any vertex, it can communicate with zeroth core.

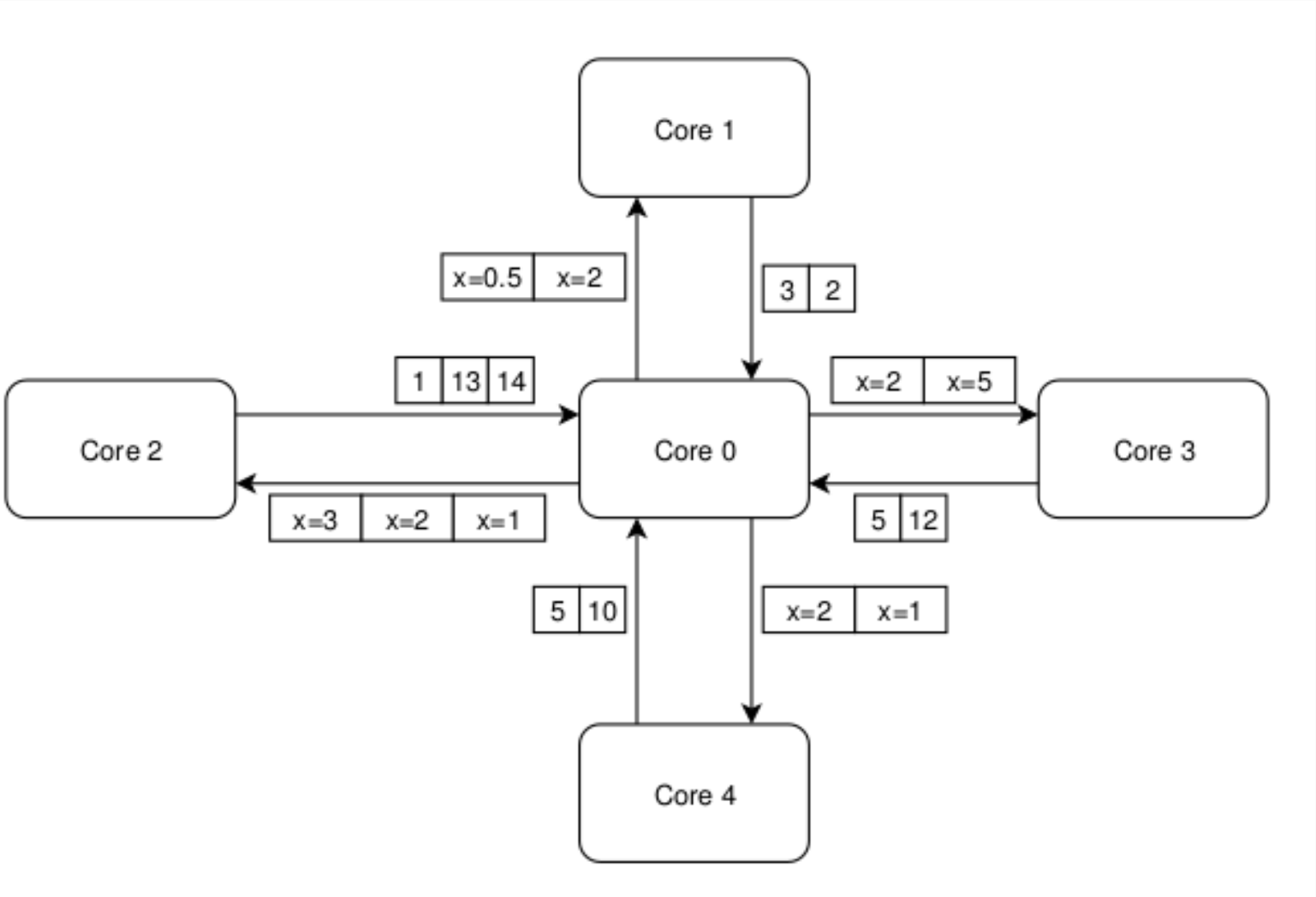
Algorithm

1. Create a record structure for the vertex that contains
   - 1.1 Vertex number
   - 1.2 Vertex data
   - 1.3 Adjacency list
2. Initialize MPI communication environment
3. Do the following in Core 0.
   - 3.1 Read the graph file
   - 3.2 Read the partition file
   - 3.3 For each partition send the vertex corresponding to that partition one by one
4. For each core other than zero , do the following
   - 4.1 Construct the subgraph from the vertex sent by the zeroth core
   - 4.2 Find the ghost vertices from the subgraph
   - 4.3 Pack together the ghost vertices and send to the zeroth core.
5. On the zeroth core do the following
   - 5.1 find the data corresponding to each ghost vertices sent by each other partition
   - 5.2 send the data to corresponding partition

Sending vertex to corresponding core and constructing sub-graph in centralized approach



Sending and requesting ghost data



Shortcoming

1. Causes performance bottleneck at core 0.
2. Each vertex requires 2 MPI call, which in term needs that much amount of system call, which is so expensive and inefficient.
3. In case of large graph file which may contain millions of vertices, zeroth core alone is insufficient in handling such large files.
4. Even if it was possible for core 0 to handle such files, it may cause delay in the communication.
5. Here cores other than core 0 are in idle state until core 0 completes its task hence it can not initiate any communication.

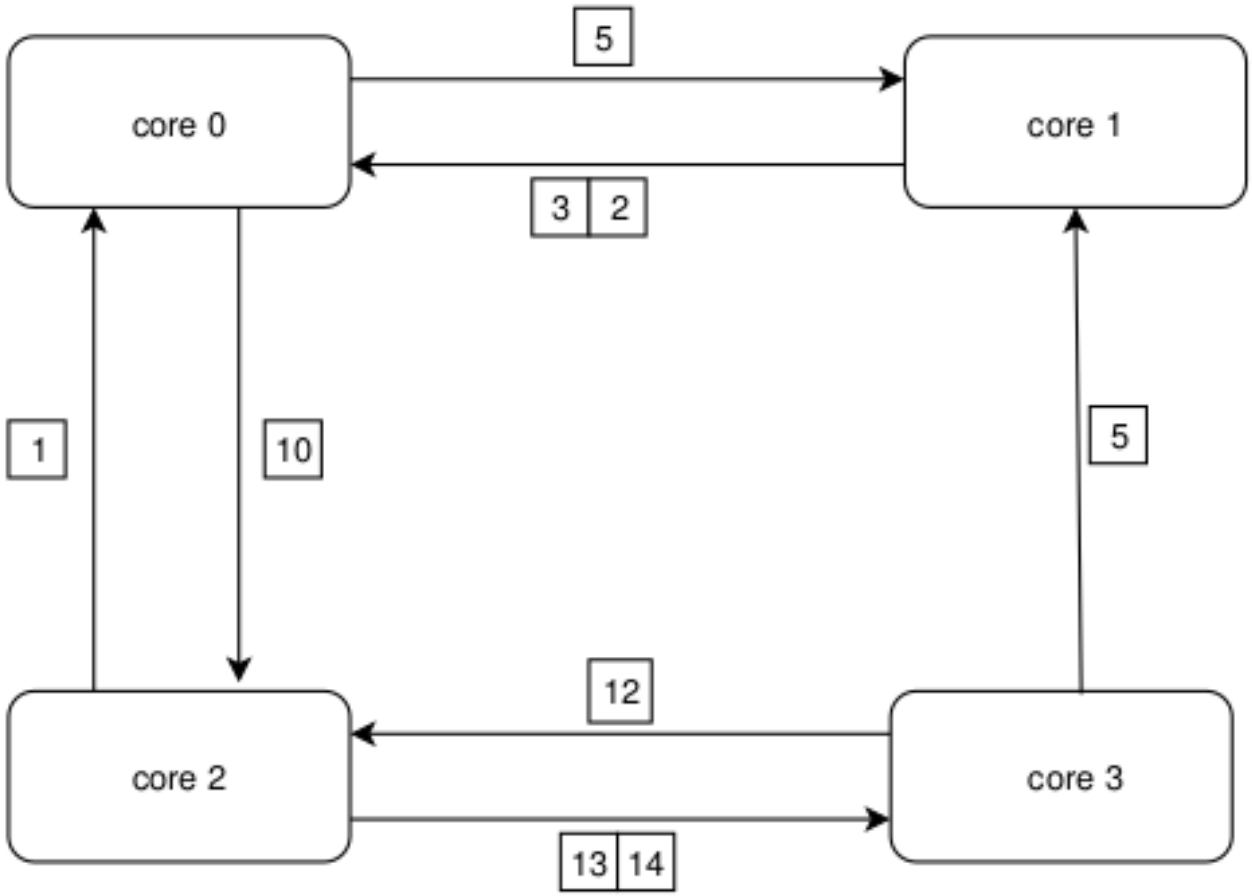
Peer-to-Peer Approach

1. Considering the shortcomings of centralised approach, we moved on to peer to peer approach.
2. In this approach there is no centralised node. Each node is provided with subgraph of its own.
3. Here, each core is parallely constructing subgraph of its own and at that time amount of communication is zero.
4. Hence cores does not suffers from idle state problem.
5. Since there is no unnecessary transfer of information, the issue of bottleneck is resolved.
6. Also because of less transfer, MPI communication is less frequent and thus it is no more expensive.

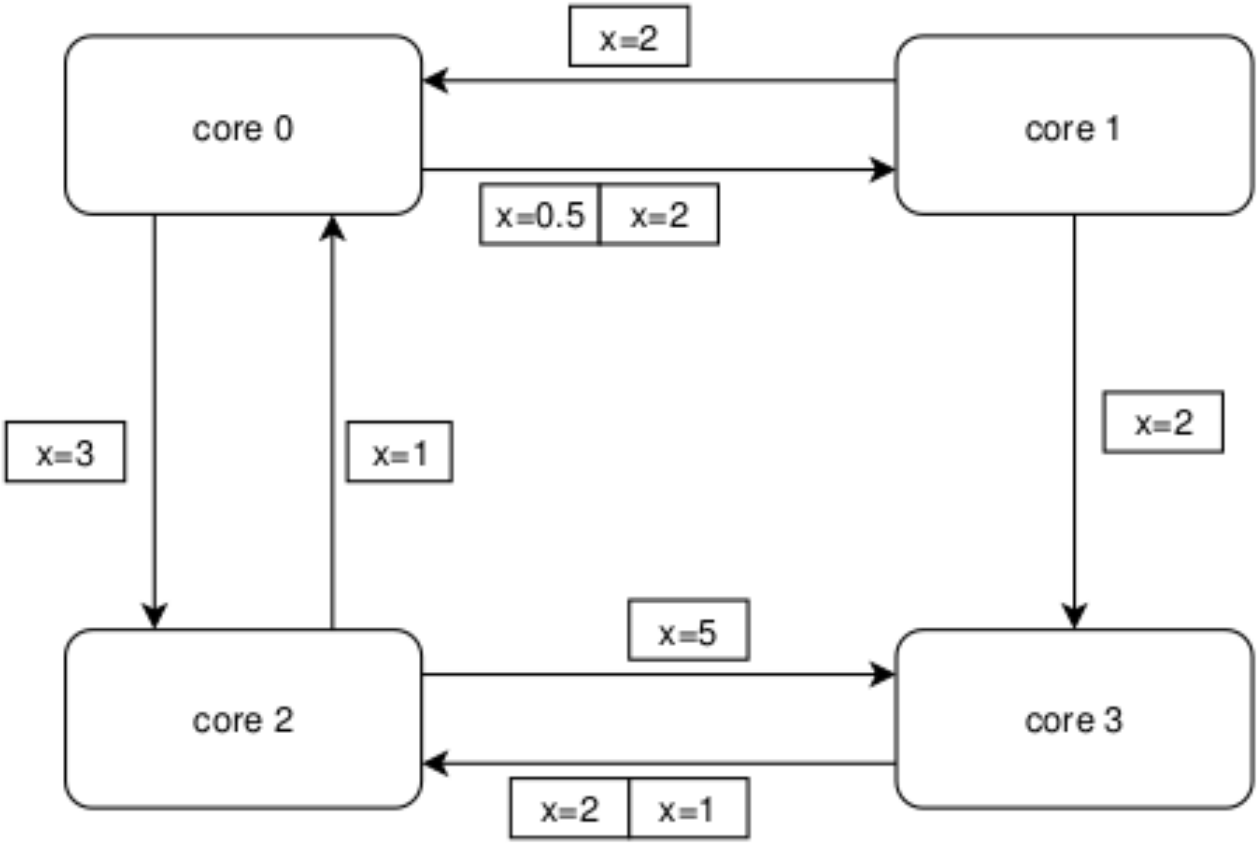
Algorithm

1. Define Vertex structure that contains.
   1.1 Vertex number
   1.2 Vertex data
   1.3 Adjacency list
   1.4 Local indexing list
2. Initialize MPI environment.
3. For each core do the following.
   3.1 Read the adjacency list
   3.2 Read the partition file
   3.3 Create subgraph on each core
   3.4 Create ghost list on each core
4. Create a local index for vertices on each core.
5. Prepare the ghost vertices for other cores as follows.
   5.1 Pack the ghost vertices for other cores into an array
   5.2 Send the ghost vertices as packed to other cores.
6. On the receiving side do the following.
   6.1 Receive the ghost vertices in packed manner
   6.2 Fill the data in the list corresponding to the ghost
   6.3 Send the filled data to the cores from which ghost received
7. Do the required calculations on the cores.
8. Finalize the MPI environment.

Requesting ghost data



Sending ghost data

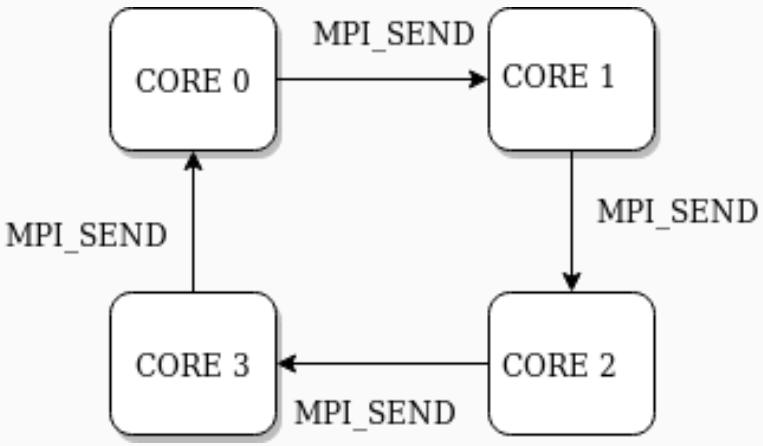


Shortcoming

- Here we are using constant buffer and receiving core is not aware of size of the ghost so that it should allocate a constant buffer size which will lead to inefficient space allocation.
- Algorithm may face deadlock.

Greedy Approach

Peer-to-peer approach may face deadlock in situations where all cores are sending mpi send to all other cores. In such conditions, mpi_send act as a blocking send and blocks communication between cores.



For example, consider a situation where core 0 sends a mpi_send to core 1, core 1 sends a mpi_send to core 2, core 2 sends a mpi_send to core 3 and core 3 sends a mpi_send to core 0. Here communication between core 0 and core 1 is completed only when core 1 receives data that was sent by core 0 however core 1 can initiate mpi_receive only when core 2 had received data that was sent by core 1. hence this will cause deadlock.

- Peer-to-peer approach may incur more delay compared to greedy approach because in peer-to-peer, communicating cores are selected at random without considering any ordering.

However in greedy approach, communicating cores are selected through maximum to minimum communication load ordering and these cores are then arranged in slots depending on their dependency. That is, each slot contains the maximum number of cores, which does not cause delay in the communication between any other cores in that slot.

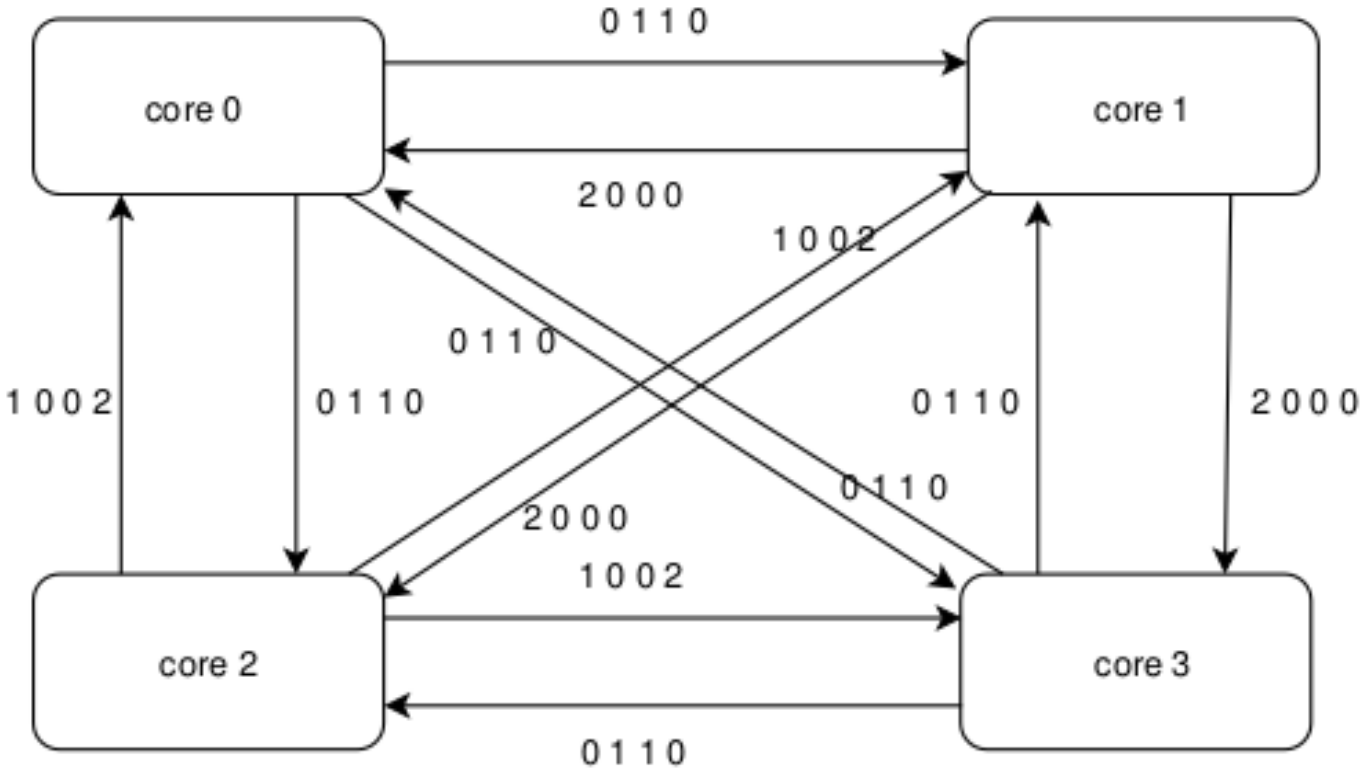
In this approach, initially synchronization message is send between all cores i.e, ghost size.

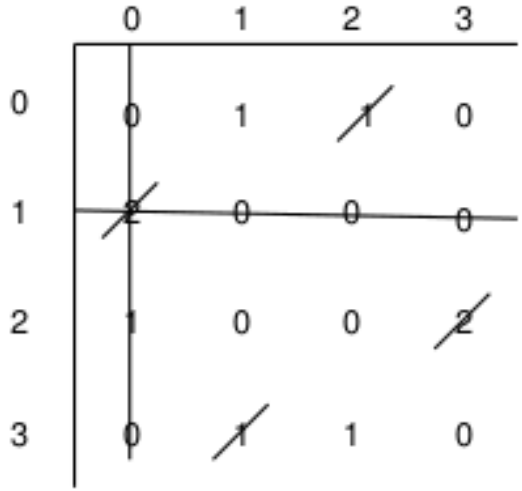
Algorithm

```
1. Define Vertex structure that contains.
  1.1 Vertex number
  1.2 Vertex data
  1.3 Adjacency list
  1.4 Local indexing list
2. Initialize MPI environment.
3. For each core do the following.
  3.1 Read the adjacency list
  3.2 Read the partition file
  3.3 Create subgraph on each core
  3.4 Create ghost list on each core
4. Create a local index for vertices on each core.
5. Create and broadcast synchronization message from each core '
    i' that contains,
  5.1 Size of the ghost from the core 'i' to all other cores.
6. For each core 'i' do the following
  6.1 Receive the synchronization message from all other cores.
  6.2 Construct a matrix 'synch' in which synch[i][j] contains
    size of the ghost to send from i'th core to j'th core.
```

7. Prepare the cores for communication as follows.
   7.1 Take the current largest entry in the 'synch' matrix and assign corresponding 'i' and 'j' to slot 0.
   7.2 Avoid the previously taken 'i' and 'j', now find next largest entry assign the corresponding 'j' and 'j' to slot 0.
   7.3 Continuing in ths way fill out slot 0 and then proceed to slot 1 and so on.
8. Initiate the communication.
   8.1 Pack the ghost vertices for the cores in slot 0 and initiate their communication.
   8.2 After completing all communication in slot 0 initiate the slot 1 and so on.
7. Do the required calculations on the cores.
9. Finalize the MPI environment.

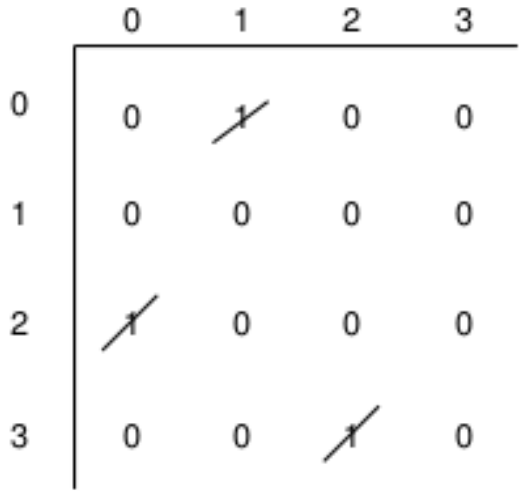
Sending Synchronization message





Slot 0

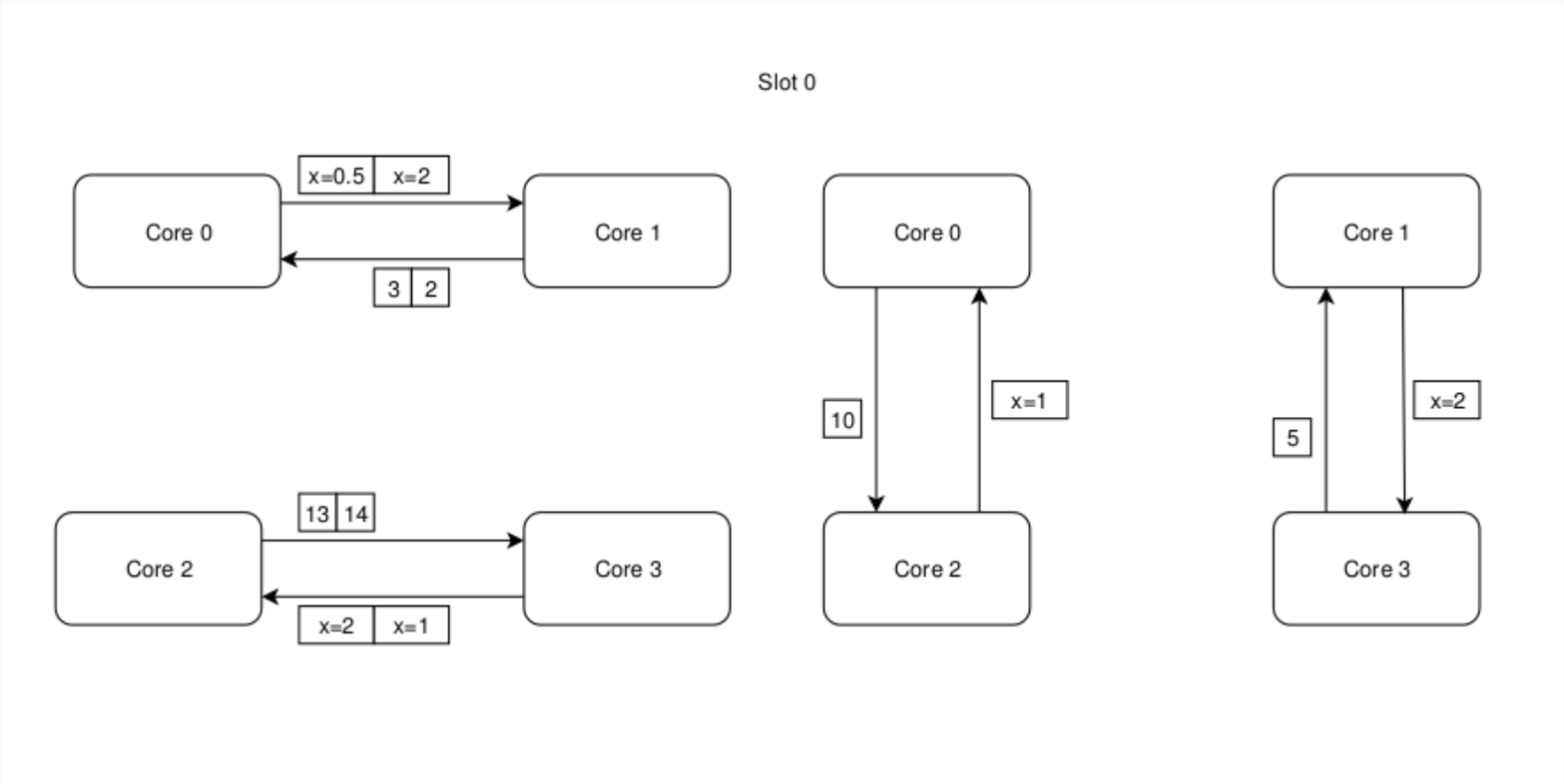
(2,3) (1,0) (3,1) (0,2)

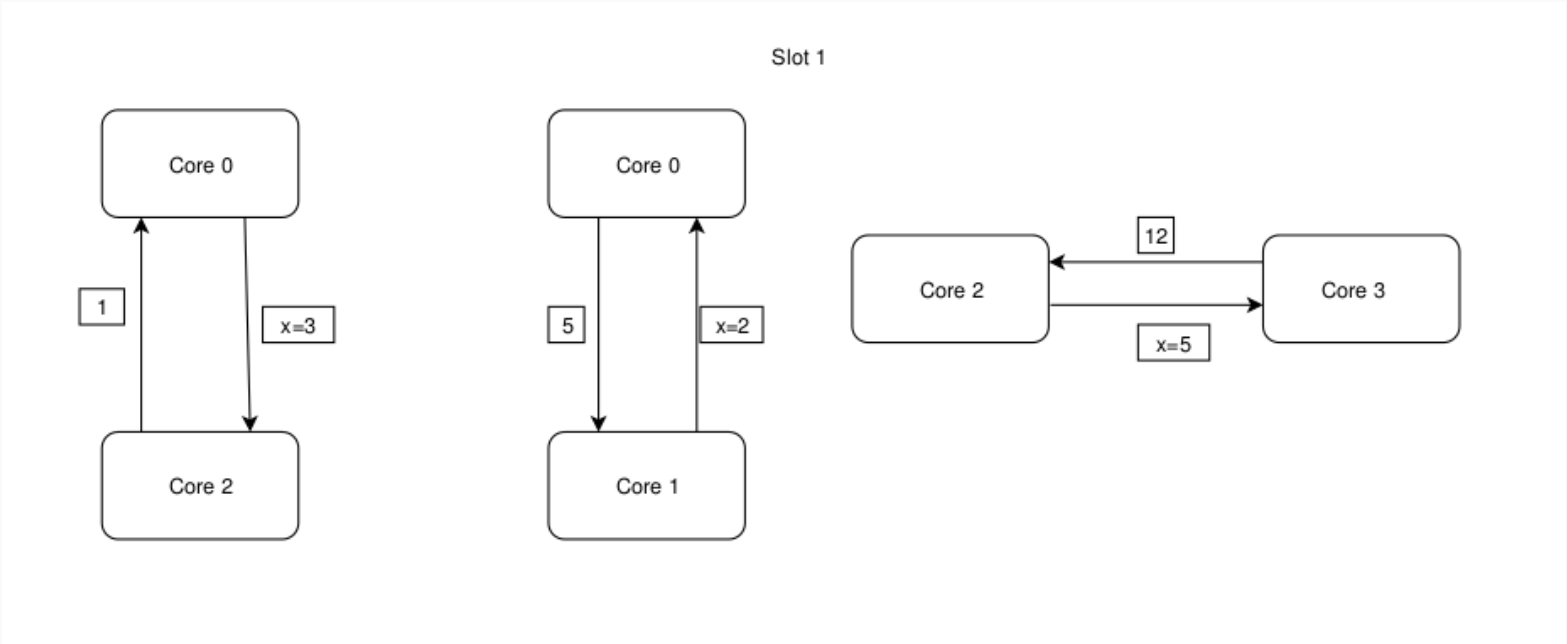


Slot 1

(2,0) (3,2) (0,1)

From the matrix of communication load, communication with maximum load is selected and is added to slot 0 and repeat the process by excluding row and column of selected load.


Slot 0
x=0.5
x=2
Core 0
Core 1
3
2
13
14
Core 2
Core 3
x=2
x=1
Core 0
10
x=1
Core 2
Core 1
5
x=2
Core 3


Slot 1
Core 0
1
x=3
Core 2
Core 0
5
x=2
Core 1
12
Core 2
Core 3
x=5

Theoretical proof

Centralized Approach

- Let number of vertices be **n** and number of cores be **m**.
- Here, core 0 act as **master** and contains all details about every other vertex.
- Therefore for creating subgraph, each vertex requires **2 mpi call.**

- Worst case :
    a. Assume that each core contain only 1 vertex and remaining all vertices are ghost for that vertex.
    b. In such condition, algorithm takes n-1 mpi send for requesting ghost data from 0th core and n-1 mpi receive for receiving ghost details from 0th core.
    c. In total there are m-1 cores, therefore amount of communication will be,

$$2n + 2(m-1) * (n-1)$$

- **General case:** In cases other than worst cases, amount of computation will be proportional to the number of ghost for that core.

$$2n + 2*\sum_{i=1}^{m-1} ki$$

where ki is the number of ghost in ith core

Peer-to-Peer and Greedy Approach

In this approach there is no centralised node. Each node is provided with subgraph of its own. Here each core invokes atmost 2 mpi call with every other core.

- **Worst case:** Each core has to communicate with all other cores. since there are **m** cores, total amount of communication will be

$$2(m-1) * m$$

How to run in a cluster

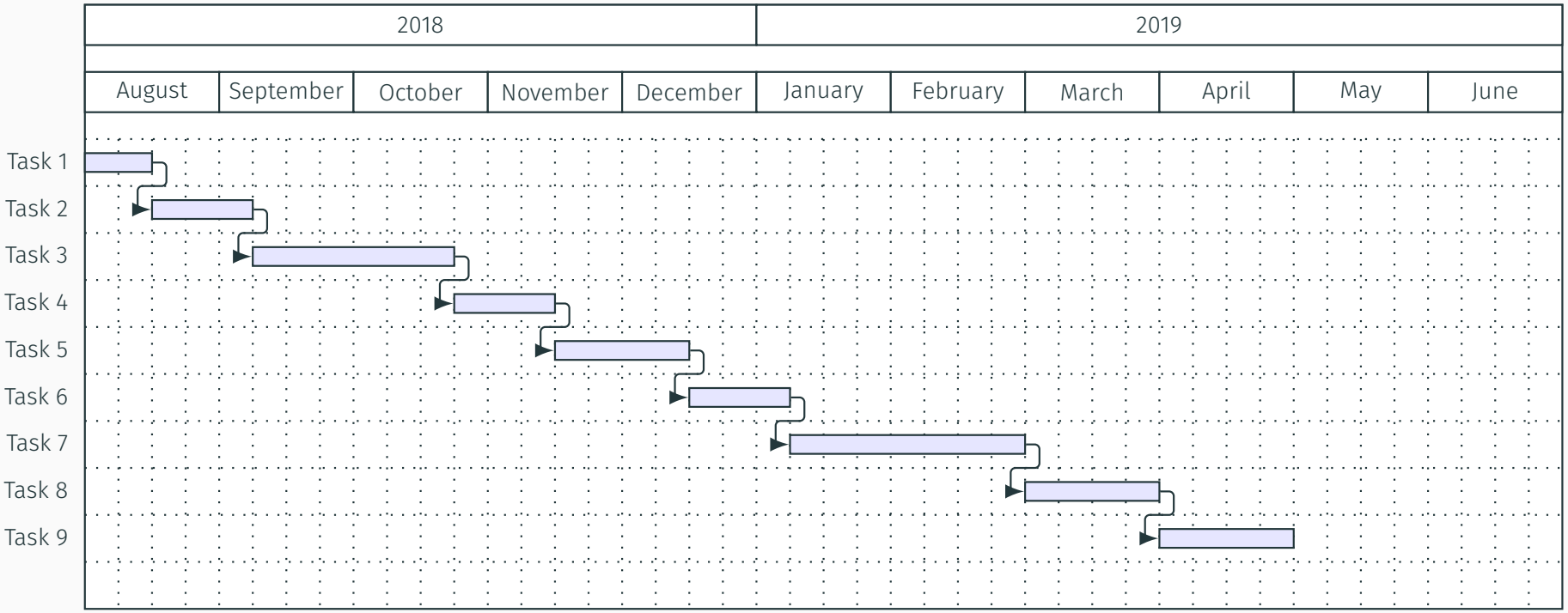
1. Establish physical connection between the systems.
2. Make sure that same version of MPI is installed on all systems.
3. Create a common user account for all systems.
4. Make sure that files are on same path on all the systems.
5. Enable password–less login in all systems.
6. Create a hostfile listing the ip address of all other systems.
7. Execute the mpi program passing the hostfile as argument.

Git-Hub Logs

https://github.com/robinl3680/Final-Year-Project

Project Plan

Project Plan



Tasks

Task 1 : Topic selection and Research.
Task 2 : Study the given input file in mesh format.

- Analyse the structure of .su2 files.
- Discussion about developing an algorithm for conversion to adjacency list.
- Extending the idea from 2D mesh to 3D.

Task 3 : Implementation of the algorithm.

- Node to Element mapping algorithm.
- Face to Element mapping algorithm.

Task 4 :Discussion about partitioning the graph.

- Study about existing ordinary methods for partitioning.
- METIS: an effective way for partitioning graph.

Task 5 : Distribution of subgraphs among multiple processors.

- Discussion about communication methods.
- MPI: an effective interface for the communication.

Task 6 : Effective communication among processors.

- Concept of ghost vertices.
- Reducing parallel asynchronous mpi communication.

Task 7: Developing algorithm for efficient communication.

- Developed centralized approach.
- Identified drawbacks and discussed about a better algorithm.

Task 8: Developed algorithm based on peer-to-peer approach.

- Discussed about amount of communication involved.
- Identified the drawback of deadlock.
- Discussed about avoiding deadlock.

Task 9: Moved to greedy algorithm for communication.

- Established synchronization.
- Developed greedy algorithm which does not contain deadlock.

Project Outcomes

Project Outcomes

PrO1 : Study of the requirements which include analysis of .su2 file , graph file etc.
PrO2 : Construction of adjacency matrix from given mesh file.
PrO3 : Partitioned mesh file.
PrO4 : Developed centralized approach, peer-to-peer approach and optimized it in greedy way
PrO5 : Publish a research paper based on the project

Program Outcomes

Program Outcomes

PO1 : Engineering Knowledge
PO2 : Problem Analysis
PO3 : Design and development of solutions
PO4 : Conduct investigations of complex problems
PO5 : Modern tool usage
PO6 : The Engineer and Society
PO7 : Environment and Sustainability
PO8 : Ethics
PO9 : Individual and team work
PO10 : Communication
PO11 : Project management and finance
PO12 : Life-long learning

Program Specific Outcomes

Program Specific Outcomes

PSO1 : Model computational problems by applying mathematical concepts and design solutions using suitable data structures and algorithmic techniques.
PSO2 : Design and develop solutions by following standard software engineering principles and implement by using suitable programming languages and platforms
PSO3 : Develop simple system solutions involving both hardware and software modules.

Project Outcome to Program Outcome Mapping

PrO	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12	PSO1	PSO2	PSO3
PrO1	3	2	3	3	3	2	3	3	4	4	3	2	3	3	-
PrO2	4	3	3	4	4	3	2	3	3	4	2	1	3	3	3
PrO3	4	3	3	4	3	3	3	3	2	4	2	1	3	3	3
PrO4	4	3	3	4	3	3	3	3	2	4	2	3	3	3	3
PrO5	4	3	3	2	2	3	2	3	2	4	4	3	3	-	-

Queries

- **Would it be feasible to use DFS/ BFS for partitioning instead of using Metis?**
  DFS/BFS technique will not reduce the communication volume(represented as edgecuts in metis). The proper alternative to metis is space filling curves.

- **What is the real computation that is happening in the vertex?**
  Its more of a fluid mechanics computation. Mass, energy and momentum fluxes will be computed on graph edges. Conserved fluid properties will be updated on each vertex looking at the fluxes on its edges. A close computation can be the one which appear in the paper.

- **What are the alternatives to MPI**
  MPI is not only the standard for communication. There are many other communication libraries like ARMCI, Portals etc. which are used for special cases. But here we work on MPI as it was the library which was suggested to us.

- **The final result of the project**
  Final result will be a frame work for solving CFD problems in a distributed MPI manner with dynamic load balancing taken care of. We will also attempt techniques to improve the serial code performance and data layouts for effective memory bandwidth utilization.

THANKYOU!!